Implémentation et apprentissage de réseaux bayésiens dynamiques d'ordre K avec pyAgrum

Jad CHAMSEDDINE Jesus Alejandro GOMEZ URZUA

26 juin 2025

Table des matières

1 Introduction **2**

2 État de l'art **2**

2.1 Réseaux Bayésiens : Concepts Essentiels 2

2.2 Apprentissage des BN 3

2.3 Réseaux Bayésiens Dynamiques 6

3 Modélisation et implantation de la structure KTBN **6**

3.1 Objectif de la classe `KTBN` 6

3.2 Architecture et conception de la classe `KTBN` 7

3.3 Méthodes de base et interface 8

3.4 Choix techniques et considérations d'implémentation 9

4 Algorithmes d'apprentissage dynamiques **10**

4.1 Apprentissage structurel avec k connu 10

4.2 Apprentissage de l'hyperparamètre K 12

4.3 Implémentation dans la classe `Learner` 13

5 Expérimentations **14**

5.1 Méthodologie générale 14

5.2 Évaluation de l'apprentissage avec K fixé 14

5.3 Sélection automatique de K 17

6 Conclusion **20**

1 Introduction

Les réseaux bayésiens (BN) constituent un cadre formel puissant pour la représentation de connaissances et le raisonnement probabiliste. Ils modélisent des relations de dépendance conditionnelle entre variables aléatoires au moyen d'un graphe orienté sans circuit associé à une factorisation de la probabilité jointe. En intelligence artificielle, ils sont utilisés avec succès dans des domaines aussi variés que la classification d'échantillons biologiques[1] la modélisation du risque en agriculture[2] ou encore l'analyse de systèmes de sécurité industrielle[4].

Les réseaux bayésiens dynamiques (dBN), étendent ce cadre aux processus temporels en modélisant l'évolution des variables au cours du temps. Cette extension permet de représenter des phénomènes dynamiques tout en conservant les propriétés de factorisation des BN classiques. De nombreuses applications en séquençage, reconnaissance vocale ou séries temporelles motivent leur étude. [7, 9]

Dans ce contexte, la bibliothèque `pyAgrum` fournit un cadre de développement en Python pour la modélisation et l'inférence dans les réseaux bayésiens. Si elle offre déjà un support partiel pour les réseaux dynamiques d'ordre 2 (2TBN), elle ne propose pas de mécanisme général pour la représentation ou l'apprentissage de modèles dynamiques d'ordre arbitraire K.

Le travail présenté dans ce rapport s'inscrit dans le cadre de l'UE LU2IN013, mené en binôme sous la direction de Pierre-Henri WUILLEMIN, et visant à étendre les fonctionnalités de `pyAgrum` pour prendre en charge les réseaux bayésiens dynamiques d'ordre arbitraire. L'objectif principal a été la conception d'un cadre logiciel pour manipuler de tels objets de manière efficace, ainsi que le développement de méthodes d'apprentissage adaptées à ce contexte.

Ce rapport présente de manière détaillée les résultats de ce projet. Après un rappel de l'état de l'art sur les réseaux bayésiens et leurs extensions dynamiques, nous exposons les choix de conception et d'implémentation de la classe `KTBN`. Nous présentons ensuite les algorithmes d'apprentissage développés, couvrant à la fois l'apprentissage structurel, l'estimation des paramètres, et la sélection automatique de l'ordre K. Une évaluation expérimentale vient conclure le rapport, illustrant les performances des approches proposées sur des données synthétiques.

2 État de l'art

Afin de situer le travail effectué dans un cadre théorique rigoureux, cette section présente les principaux concepts utilisés au cours du projet. Nous introduisons d'abord les réseaux bayésiens. Nous abordons ensuite leur extension temporelle, les réseaux bayésiens dynamiques (dBN). Enfin, nous évoquons brièvement les approches classiques d'apprentissage structurel.

2.1 Réseaux Bayésiens : Concepts Essentiels

Un réseau bayésien (BN) est un modèle graphique probabiliste représenté par un graphe dirigé sans circuit (DAG) [5], où les nœuds correspondent à des variables aléatoires, et les arcs codent des relations de dépendance probabilistes ou causales.

Sous une interprétation purement statistique, un BN constitue une représentation compacte des relations d'indépendance conditionnelle présentes dans les données d'observation et peut être utilisé pour l'inférence des distributions conditionnelles et marginales.

En termes de structure, un nœud A est appelé parent d'un nœud B si une arc $A \to B$ existe, et réciproquement, B est un enfant de A. L'absence d'un arc entre deux nœuds implique une indépendance conditionnelle entre ces variables, étant donné leurs parents respectifs. Cette propriété est formalisée par la propriété de Markov

locale :

Toute variable X_i est indépendante de ses non-descendants, sachant ses parents directs.

Un réseau bayésien encode une décomposition factorisée de la loi de probabilité jointe en exploitant la structure du DAG. Plus précisément, étant donné un ensemble de variables aléatoires $X_1, X_2, \ldots, X_n$, la probabilité jointe se factorise comme suit :

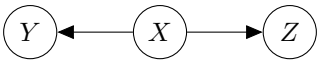
$$\mathbb{P}(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} \mathbb{P}(X_i \mid \text{Parents}(X_i))$$

où $\text{Parents}(X_i)$ désigne l'ensemble des parents de la variable X_i dans le graphe.

Par exemple, considérons une factorisation donnée par :

$$\mathbb{P}(X, Y, Z) = \mathbb{P}(X)\mathbb{P}(Y \mid X)\mathbb{P}(Z \mid X).$$

Le graphe correspondant à cette décomposition est un DAG de la forme :



L'analyse des relations d'indépendance conditionnelle dans un réseau bayésien repose sur le critère de d-séparation [6]. Soient X, Y, Z trois ensembles disjoints de nœuds d'un DAG $\mathcal{G}$. L'ensemble Z est dit d-séparateur de X et Y, noté $\langle X \mid Z \mid Y \rangle_d$, si et seulement si tout chemin non dirigé entre un nœud de X et un nœud de Y est bloqué par Z. Un chemin est dit bloqué s'il existe un nœud W sur ce chemin vérifient l'une des deux conditions suivantes : W est un nœud en collision, c'est-à-dire qu'il existe dans $\mathcal{G}$ deux arcs entrants vers W le long du chemin considéré, autrement dit une structure de la forme $A \to W \leftarrow B$, et qu'aucun nœud parmi W et ses descendants n'appartient à Z, ou bien, W n'est pas un nœud en collision et W appartient à Z.

Cette caractérisation graphique implique l'indépendance conditionnelle dans l'espace probabiliste :

$$Z \text{ d-sépare } X \text{ et } Y \implies X \perp\!\!\!\perp Y | Z$$

2.2 Apprentissage des BN

L'apprentissage d'un réseau bayésien à partir de données est un problème NP-difficile. Cela a été prouvé par Chickering [3]. Cette complexité provient de l'explosion du nombre de structures possibles, qui croît exponentiellement avec le nombre de variables. Malgré cette complexité inhérente, plusieurs approches heuristiques ont été développées pour approximer la solution optimale.

2.2.1 Classes d'Équivalence de Markov

Dans un réseau bayésien, la structure d'un DAG encode un ensemble d'indépendances conditionnelles à travers le critère de d-séparation. Toutefois, une même distribution de probabilités peut être représentée par plusieurs DAGs distincts qui induisent exactement les mêmes relations d'indépendance conditionnelle. Cette propriété conduit à la notion de classe d'équivalence de Markov:

Figure 1: Configurations d'orientation des arêtes entre A, B et C. (a)-(c) sont équivalentes ; (d) est une v-structure.

Deux DAGs $\mathcal{G}_1$ et $\mathcal{G}_2$ sont dits Markov-équivalents si et seulement si ils induisent le même ensemble d'indépendances conditionnelles.

Un élément clé dans l'analyse des classes d'équivalence est la notion de structure en V (v-structure). Soient A, B et C trois variables représentées par des sommets dans un DAG $\mathcal{G}$. Supposons que deux arcs relient A et B, et C et B (dans un sens ou dans l'autre), mais qu'aucun arc ne relie directement A et C. Dans ce cas, il existe exactement quatre configurations possibles pour l'orientation des arcs entre ces trois sommets, illustrées dans la Figure 1. D'un point de vue probabiliste, les trois premières configurations sont indistinguables en ce qu'elles encodent exactement la même relation d'indépendance conditionnelle, à savoir

$$A \perp\!\!\!\perp C \mid B.$$

En revanche, la quatrième configuration se distingue des autres, car elle encode une relation d'indépendance différente, en particulier

$$A \not\perp\!\!\!\perp C \mid B.$$

Dans cette dernière structure, le nœud B possède deux arcs entrants, et il n'existe d'arc entre A et C. Une telle configuration est appelée une v-structure.

Le théorème de caractérisation de Verma et Pearl [6] établit un critère nécessaire et suffisant pour que deux DAGs soient Markov-équivalents. Deux DAGs $\mathcal{G}_1$ et $\mathcal{G}_2$ sont Markov-équivalents si et seulement si ils vérifient simultanément les deux conditions suivantes :

(i) $\mathcal{G}_1$ et $\mathcal{G}_2$ possèdent le même squelette, c'est-à-dire qu'ils admettent le même graphe non orienté sous-jacent, où seules les connexions entre variables sont considérées, indépendamment de l'orientation des arêtes.

(ii) $\mathcal{G}_1$ et $\mathcal{G}_2$ contiennent exactement les mêmes v-structures

Ces deux conditions garantissent que les DAGs appartenant à une même classe d'équivalence induisent exactement les mêmes relations d'indépendance conditionnelle, et sont par conséquent indiscernables d'un point de vue probabiliste. Il en résulte qu'un algorithme d'apprentissage statistique ne peut pas différencier deux réseaux bayésiens issus de la même classe d'équivalence. En effet, ils sont statistiquement équivalents vis-à-vis de l'information accessible dans les observations.

2.2.2 Approches Basées sur les Contraintes

Les méthodes d'apprentissage structurel basées sur les contraintes (*constraint-based structure learning*) s'appuient sur l'identification des indépendances conditionnelles entre variables afin de reconstruire la structure d'un réseau bayésien. Ces méthodes exploitent le fait que la structure d'un DAG Markov-compatible avec une distribution de probabilités est entièrement caractérisée par les relations d'indépendance conditionnelle entre variables.

L'algorithme suit généralement trois phases successives :

1. Construction du squelette

   On détermine d'abord le squelette du graphe, c'est-à-dire son graphe sous-jacent non orienté, en effectuant des tests d'indépendance conditionnelle sur les données. Deux variables sont reliées par une arête dans le squelette si et seulement si elles ne sont pas indépendantes conditionnellement à un sous-ensemble approprié des autres variables observées. Cette phase fournit une première approximation de la structure du réseau sans information directionnelle.

2. Orientation des v-structures

   Une fois le squelette obtenu, il est nécessaire d'identifier et d'orienter les v-structures, qui sont les seules configurations dont l'orientation est déductible directement à partir des relations d'indépendance conditionnelle observées. Ces orientations sont imposées de manière unique afin de préserver les relations d'indépendance et de dépendance induites par les tests effectués.

3. Orientation des arêtes restantes

   Après l'identification des v-structures, les autres arêtes du graphe doivent être orientées en respectant deux contraintes fondamentales :

   (i) éviter la formation de cycles dirigés, et

   (ii) ne pas introduire de nouvelles v-structures non justifiées par les indépendances conditionnelles observées.

   Pour cela, des règles d'orientation supplémentaires sont appliquées de manière itérative, comme la propagation des orientations dérivées des relations d'indépendance.

L'exactitude de ces méthodes dépend fortement de la fiabilité des tests d'indépendance, qui peuvent être affectés par le bruit ou la taille limitée des échantillons. Dans la plupart des cas, elles ne permettent pas d'orienter toutes les arêtes, ce qui conduit à l'obtention d'un DAG partiellement orienté (*partially directed acyclic graph*, PDAG), représentant l'ensemble des structures Markov-équivalentes compatibles avec les données.

2.2.3 Approches Basées sur les Scores

Les méthodes basées sur les scores (*score-based*) modélisent l'apprentissage de la structure comme un problème d'optimisation : il s'agit de trouver la structure qui maximise un score mesurant l'adéquation du modèle aux données. Parmi les scores les plus couramment utilisés figurent le BIC (*Bayesian Information Criterion*), le BDe (*Bayesian Dirichlet equivalent*) et le MDL (*Minimum Description Length*).

Ces scores équilibrent deux objectifs contradictoires : obtenir un bon ajustement aux données observées tout en pénalisant les structures trop complexes, afin d'éviter le sur-apprentissage.

Dans la pratique, la recherche de la structure optimale est rendue difficile par la taille exponentielle de l'espace des graphes possibles. Les algorithmes utilisés sont donc généralement gloutons (greedy), explorant l'espace des DAGs à l'aide d'ajouts, suppressions ou inversions d'arcs qui améliorent localement le score.

2.3 Réseaux Bayésiens Dynamiques

Les réseaux bayésiens dynamiques (dBN) constituent une extension naturelle des réseaux bayésiens aux processus évolutifs, où les variables aléatoires sont observées sur plusieurs instants successifs. Ils permettent de modéliser des systèmes dynamiques probabilistes, dans lesquels les dépendances statistiques entre les variables évoluent dans le temps de manière structurée.

Les dBN s'appuient généralement sur l'hypothèse de Markov d'ordre 1, c'est-à-dire que l'état du système à l'instant $t+1$ ne dépend que de son état à l'instant t. Cette approche inclut notamment les *2-Time-slice Bayesian Networks* (2TBN), déjà présents dans `pyAgrum`.

Dans le cadre de notre projet, nous considérons une généralisation de ce modèle, les réseaux bayésiens dynamiques d'ordre K (KTBN), dans lesquels la dynamique temporelle du système repose sur les $K-1$ dernières tranches temporelles. Plus formellement, un KTBN se compose de deux parties :

1. Une structure initiale décrivant la distribution jointe des variables sur les $K-1$ premières tranches temporelles $(X^{(0)}, X^{(1)}, \ldots, X^{(K-2)})$, modélisée par un réseau bayésien classique ;
2. Un modèle de transition qui encode la distribution conditionnelle $\mathbb{P}(X^{(t)} \mid X^{(t-1)}, \ldots, X^{(t-(K-1))})$ pour tout $t \geq K$, sous l'hypothèse que cette transition reste stationnaire au cours du temps.

Ce modèle repose donc sur deux hypothèses fondamentales :

1. Hypothèse de Markov d'ordre $K-1$: l'état futur est indépendant des états plus anciens, conditionnellement aux $K-1$ derniers états. ;
2. Stationnarité de la transition : la loi de transition est identique pour $t \geq K$, ce qui permet de réutiliser la même structure temporelle à chaque pas.

En pratique, la représentation graphique d'un KTBN peut être réalisée au moyen d'un seul DAG statique, dans lequel chaque nœud représente une variable à un instant donné. Une fois les $K-1$ premières tranches temporelles définies, la K-ième est utilisée comme patron de transition, qui peut être "déroulé" de manière répétée pour modéliser un processus de longueur arbitraire.

Malgré l'expressivité du modèle KTBN, celui-ci reste relativement peu exploré dans la littérature. En dehors de cas particuliers bien étudiés, tels que les modèles de Markov cachés ou les 2TBN, peu de travaux portent sur la modélisation de réseaux bayésiens dynamiques d'ordre K arbitraire. En particulier, les questions liées à l'apprentissage automatique de ces structures demeurent encore ouvertes.

Dans ce contexte, nous explorons des méthodes pour l'apprentissage structurel d'un KTBN à ordre K fixé, ainsi que pour l'estimation de cet ordre à partir des données. Les algorithmes que nous proposons s'appuient sur des critères d'adéquation statistique, tels que le score BIC.

3 Modélisation et implantation de la structure KTBN

3.1 Objectif de la classe `KTBN`

La bibliothèque `pyAgrum` prend en charge les réseaux bayésiens statiques et propose un support partiel pour les 2TBN. Elle ne permet toutefois pas de représenter ou manipuler directement des réseaux bayésiens dynamiques d'ordre K quelconque.

La classe KTBN a pour objectif d'étendre cette fonctionnalité en introduisant une surcouche au-dessus d'un objet BayesNet. Cette abstraction gère explicitement la structure temporelle du modèle, permettant notamment l'accès aux variables temporelles, la manipulation d'arcs inter-tranches et la génération de trajectoires, tout en restant compatible avec l'infrastructure de pyAgrum.

3.2 Architecture et conception de la classe KTBN

3.2.1 Classe d'abstraction et encapsulation

La classe KTBN enveloppe un réseau bayésien standard de pyAgrum (gum.BayesNet) dans une classe adaptée aux réseaux bayésiens dynamiques. Cette approche offre plusieurs avantages :

- Exploitation des fonctionnalités existantes : L'utilisation d'un gum.BayesNet sous-jacent permet de bénéficier de toutes les fonctionnalités robustes et optimisées de pyAgrum (inférence, apprentissage, manipulation des CPT, etc.).
- Compatibilité : La possibilité de convertir un KTBN vers un réseau bayésien standard via la méthode to_bn() permet de réutiliser les fonctionnalités de pyAgrum.
- Abstraction temporelle : La classe KTBN masque la complexité interne et propose des méthodes simples pour les opérations temporelles.

3.2.2 Distinction entre variables temporelles et atemporelles

Un point clé du modèle et de notre implémentation est de séparer deux types de variables :

- Variables temporelles : Variables qui changent selon le temps. Pour un ordre K, chaque variable temporelle est copiée K fois dans le réseau bayésien.
- Variables atemporelles : Variables qui restent fixes dans le temps.

Deux ensembles internes permettent cette séparation :

```
1     self._temporal_variables = set()      # Variables temporelles
2     self._atemporal_variables = set()     # Variables atemporelles
```

3.2.3 Système d'encodage des noms de variables

Pour différencier les variables dans le temps, chaque variable temporelle reçoit un nom unique composé de son nom original, d'un délimiteur (# par défaut) et du time-slice correspondant. Par exemple, Temperature#0, Temperature#1, Humimidity#2. Les variables atemporelles gardent leur nom original. Cette méthode permet d'identifier chaque variable à un moment précis.

Les fonctions encode_name() et decode_name() permettent la conversion entre la représentation logique (nom et time-slice séparés) et la représentation technique (nom encodé) :

```
1     def encode_name(self, variable: str, time_slice: int) -> str:
2     # Convertit (nom, time-slice) en nom_technique
3
4     def decode_name(self, name: str) -> Tuple[str, int]:
5     # Convertit nom_technique en (nom, time-slice)
```

3.3 Méthodes de base et interface

3.3.1 Adaptation de l'API `pyAgrum`

Dans `pyAgrum`, les variables sont manipulées à l'aide de méthodes avec des paramètres simples du type `méthode(nom_variable)`. Cependant, dans un réseau bayésien dynamique, chaque variable est indexée par son time-slice, ce qui nécessite une désambiguïsation temporelle lors de la manipulation.

La classe `KTBN` introduit ainsi une interface adaptée, dans laquelle les méthodes attendent un couple `(nom_variable, time_slice)` comme paramètre, sous la forme `méthode(nom_variable, time_slice)`, avec la convention $t = -1$ pour les variables atemporelles.

Cette extension permet d'ajouter des arcs entre différentes tranches temporelles, par exemple :

```
1    ktbn.addArc(("Humidity", 1), ("Humidity", 2))
```

ce qui crée un arc de la variable `Humidity` à l'instant $t = 1$ vers la même variable à l'instant $t = 2$.

3.3.2 Méthodes d'interaction avec `pyAgrum`

La classe KTBN expose plusieurs méthodes permettant l'interaction directe avec le réseau bayésien sous-jacent :

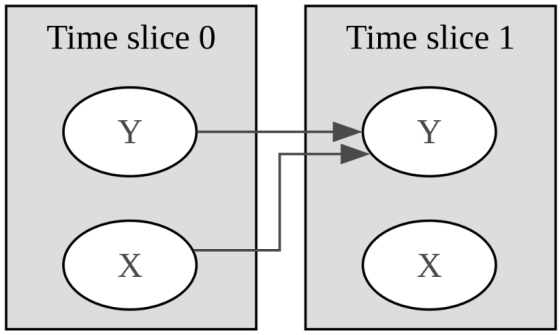
- `addVariable(variable, temporal)` : Ajout d'une variable (temporelle ou atemporelle)
- `addArc(tail, head)` : Ajout d'un arc avec spécification temporelle
- `cpt(variable, time_slice)` : Accès aux tables de probabilité conditionnelle (CPT)
- `to_bn()` : Conversion vers un `gum.BayesNet` standard
- `from_bn(bn, delimiter)`: Création d'un KTBN à partir d'un réseau bayésien existant

Ces méthodes permettent de modifier et d'accéder au réseau bayésien sous-jacent tout en respectant les contraintes temporelles et la logique de la classe `KTBN`.

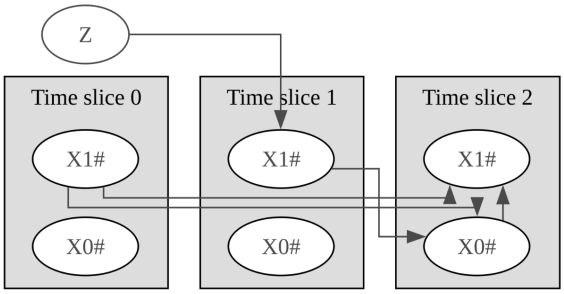
3.3.3 Méthodes spécifiques aux réseaux dynamiques

Au-delà de l'adaptation de l'interface de `pyAgrum` à la temporalité, la classe `KTBN` introduit plusieurs méthodes spécifiques aux réseaux bayésiens dynamiques :

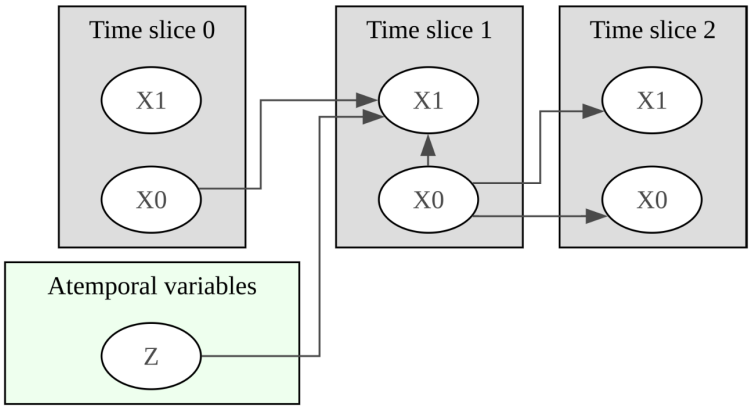
- `unroll(n)` : méthode permettant de dérouler le `KTBN` sur n tranches temporelles. Elle génère un BN statique équivalent, compatible avec les outils standards de `pyAgrum`. L'algorithme :
  1. duplique les variables temporelles pour chaque nouvelle tranche ;
  2. réplique la structure des arcs intra et inter-tranches selon le schéma de transition ;
  3. ajuste les CPT en conséquence.
- `sample(n_trajectories, trajectory_len, processes)` : génère un ensemble de trajectoires aléatoires à partir du modèle, en exploitant la parallélisation via le module `multiprocessing` de Python.
- `random(k, n_vars, n_mods, n_arcs)` : crée un KTBN aléatoire respectant les contraintes structurelles imposées, pratique pour les phases de test et de validation expérimentale.
- `log_likelihood(trajectories)` : calcule la log-vraisemblance d'un ensemble de trajectoires données, une mesure essentielle pour l'évaluation et l'apprentissage du modèle.



(a) 2TBN standard



(b) KTBN affiché par les outils existants



(c) KTBN affiché par `KTBN_notebook`

Figure 2: Comparaison des différentes représentations graphiques d'un KTBN.

3.3.4 Visualisation des KTBN

Afin de permettre la visualisation claire et structurée des réseaux KTBN dans un environnement Jupyter, nous avons développé un module dédié, `KTBN_notebook`, assurant la prise en charge explicite de ces modèles. Inspirée du module `pyAgrum.lib.dynamicBN`, conçu pour organiser les variables des 2TBN par tranches temporelles, notre classe adapte le rendu graphique aux spécificités des KTBN : elle regroupe les variables par tranche temporelle, positionne les nœuds de manière cohérente, et distingue visuellement les variables atemporelles lorsqu'elles sont présentes. La figure 2 compare ces représentations : (a) montre un 2TBN standard ; (b), l'affichage par défaut d'un KTBN via les outils existants ; (c), le rendu structuré produit par notre adaptation.

3.4 Choix techniques et considérations d'implémentation

3.4.1 Parallélisation de l'échantillonnage

La génération de trajectoires dans un KTBN peut être parallélisée efficacement du fait de l'indépendance entre les trajectoires. Cette propriété garantit qu'aucune synchronisation ni communication inter-processus n'est nécessaire durant l'échantillonnage, ce qui rend le problème naturellement adapté à une exécution concurrente. L'implémentation repose sur le module `multiprocessing` de Python, où un pool de processus est utilisé pour répartir la génération des trajectoires. Chaque processus exécute indépendamment une fonction d'échantillonnage, et les résultats sont collectés à l'issue de l'exécution. Cette approche permet de tirer parti des ressources multicœurs pour réduire significativement le temps de calcul.

Chaque trajectoire est générée en deux étapes : les $K-1$ premières tranches sont échantillonnées selon la distribution jointe du modèle, puis les tranches suivantes sont produites de manière séquentielle à l'aide du modèle de transition. Contrairement à une approche naïve qui consisterait à dérouler intégralement le KTBN au préalable, au prix d'une empreinte mémoire considérable, cette méthode itérative permet une consommation mémoire faible, rendant possible l'échantillonnage de trajectoires très longues.

3.4.2 Validation et contraintes temporelles

La classe KTBN vérifie automatiquement la cohérence du réseau. La méthode `_validate_variable()` contrôle que les variables existent, que leur type (temporelle ou atemporelle) correspond à leur utilisation, et que les time-slices sont valides.

La classe KTBN bloque aussi la création d'arcs qui violeraient les contraintes temporelles, comme les arcs allant du futur vers le passé. En cas d'erreur, des exceptions claires informent l'utilisateur du problème rencontré.

3.4.3 Tests unitaires et validation

L'implémentation de la classe `KTBN` est accompagnée d'une suite de tests unitaires conçue avec le module standard `unittest` de Python. Ces tests couvrent les principales fonctionnalités : initialisation, gestion des variables temporelles et atemporelles, ajout d'arcs avec vérification des contraintes temporelles, opérations d'unrolling, sérialisation et désérialisation, conversion depuis et vers un objet `BayesNet`, ainsi que l'accès aux CPT. Les cas testés incluent aussi bien des situations standards que des cas d'erreur (arcs invalides, tranches hors bornes, noms absents), permettant de vérifier la conformité de l'API à ses spécifications et la détection correcte des usages incorrects. Cette couverture garantit que les contraintes structurelles spécifiques aux KTBN sont bien respectées, et que les erreurs sont détectées de manière fiable.

4 Algorithmes d'apprentissage dynamiques

4.1 Apprentissage structurel avec k connu

4.1.1 Principe algorithmique et décomposition du problème

L'apprentissage d'un KTBN à partir d'un ensemble de trajectoires temporelles constitue un défi particulier, car les données sont ordonnées dans le temps et les variables présentent des dépendances à travers plusieurs time-slices. Notre approche repose sur une décomposition du problème en deux sous-problèmes plus simples :

1. Apprentissage des k premiers pas de temps : Apprentissage des relations entre variables dans les k premières time-slices
2. Apprentissage de la transition : Apprentissage des relations entre l'état t+1 et les k états précédents

Cette décomposition s'explique par le fait qu'un KTBN n'est qu'un réseau bayésien classique qui modélise la distribution des k premiers pas de temps et de la transition. Cette perspective permet donc de réutiliser les algorithmes d'apprentissage existants de `pyAgrum` pour apprendre le KTBN.

4.1.2 Transformation des données et construction des bases d'apprentissage

L'implémentation de cette approche nécessite une transformation des trajectoires originales en bases de données adaptées à l'apprentissage de réseaux bayésiens. Cette transformation est réalisée par la fonction `_create_sequences()`.

Les données sont représentées sous forme de `pd.DataFrame` via la bibliothèque `pandas`, choisie pour ses performances, sa souplesse de manipulation et sa compatibilité directe avec les fonctions d'apprentissage de `pyAgrum`.

Processus de transformation :

1. Identification des types de variables :
   - Variables temporelles : Variables dont les valeurs évoluent au cours du temps.
   - Variables atemporelles : Variables qui restent fixes dans le temps, détectées automatiquement. par `_is_atemporal()`
2. Génération des séquences avec décalages : Pour chaque variable temporelle, nous créons K colonnes : une pour la variable à l'instant présent, une pour la variable à l'instant précédent, et ainsi de suite jusqu'à K instants. Cela permet d'avoir simultanément sur une même ligne les valeurs de la variable à K instants consécutifs.

Nous obtenons ainsi deux bases de données qui capturent respectivement les conditions initiales et l'évolution temporelle :

- `lags` : Contient toutes les séquences de k tranches temporelles consécutives.
- `first_rows` : Contient uniquement les k premières tranches de chaque trajectoire.

4.1.3 Implémentation de la classe Learner

La classe `Learner` sert à apprendre la structure d'un KTBN. Elle utilise deux instances de `gum.BNLearner`, une pour chaque partie du problème.

Processus d'apprentissage avec k connu (`learn_ktbn()`) :

1. Configuration des contraintes :
   - Interdiction de parents pour les variables atemporelles (contrainte `addNoParentNode`).
   - Définition de l'ordre des tranches temporelles via `setSliceOrder`.
2. Apprentissage :
   - `lags_learner` : Apprentissage des arcs de transition.
   - `first_learner` : Apprentissage des structures internes aux $k-1$ premières tranches.
3. Combinaison des résultats :
   - Création du réseau à partir des premières time-slices.
   - Ajout des variables de la dernière tranche temporelle.
   - Ajout des arcs de transition appris.
   - Copie des CPT.

Cette approche permet d'exploiter les algorithmes standards de `pyAgrum` tout en prenant en compte les dépendances temporelles propres aux KTBN.

4.1.4 Paramétrage de l'apprentissage

La classe `Learner` reproduit l'interface de `gum.BNLearner`, ce qui permet à l'utilisateur de configurer librement l'apprentissage d'un KTBN. Elle applique automatiquement les mêmes paramètres aux deux learners internes (`first_learner` et `lags_learner`).

Fonctionnalités exposées :

- Contraintes structurelles : `addMandatoryArc()`, `addForbiddenArc()`, `setPossibleEdges()`
- Algorithmes d'apprentissage : `useGreedyHillClimbing()`, `useMIIC()`, `useLocalSearchWithTabuList()`
- Scores : `useScoreBIC()`, `useScoreAIC()`, `useScoreBDeu()`, `useScoreK2()`
- Priors : `useBDeuPrior()`, `useDirichletPrior()`, `useSmoothingPrior()`

Cette approche permet une configuration flexible tout en masquant les détails liés à la gestion des tranches temporelles. Des méthodes internes comme `_verify_timeslice()` assurent que toute modification respecte les contraintes temporelles et structurelles du modèle, garantissant ainsi la validité du réseau.

4.2 Apprentissage de l'hyperparamètre K

4.2.1 Trouver la valeur optimale de k

En pratique, la valeur de K n'est pas toujours connue à l'avance. Il est donc nécessaire de la sélectionner automatiquement, en recherchant un compromis entre complexité du modèle et capacité à capturer les dépendances temporelles présentes dans les données.

Défis liés à la sélection de K :

- Sous-ajustement (K trop faible) : Le modèle ne capture pas les dépendances temporelles suffisamment riches, conduisant à une représentation trop simplifiée du processus dynamique.
- Sur-ajustement (K trop élevé) : Le modèle devient trop complexe, apprenant des structures spécifiques aux données d'entraînement qui ne se généralisent pas.
- Coût computationnel : L'espace de recherche croît exponentiellement avec K, rendant l'exploration exhaustive impraticable sans heuristiques ou stratégies d'optimisation efficaces.

4.2.2 Critère BIC et justification théorique

La sélection du meilleur ordre K dans un KTBN consiste à choisir le modèle qui offre le meilleur compromis entre complexité et capacité à modéliser les dépendances temporelles dans les données. Pour cela, nous utilisons le Critère d'Information Bayésien (BIC), un estimateur fondé sur une approximation asymptotique de l'évidence bayésienne, particulièrement adapté à la comparaison de réseaux bayésiens appris à partir de données[8].

Étant donné un KTBN B_k, appris à partir d'un ensemble de données D, le score BIC est défini par :

$$\mathrm{BIC}(k) = \dim(\mathcal{B}_k) \cdot \ln N - 2 \cdot \ell(\mathcal{D} \mid \mathcal{B}_k)$$

où :

$\dim(\mathcal{B}_k)$ est le nombre de paramètres du KTBN ;
N est la taille effective des données, égale à la somme des longueurs des trajectoires observées ;
$\ell(\mathcal{D} \mid \mathcal{B}_k)$ Log-vraisemblance calculée sur toutes les trajectoires via `ktbn.log_likelihood(trajectories)`

L'enjeu central est de trouver un modèle suffisamment riche pour capturer les relations temporelles, mais pas trop complexe pour éviter le sur-apprentissage. Le score BIC pénalise explicitement la complexité du modèle tout en valorisant son pouvoir explicatif, ce qui permet :

- de limiter le sur-apprentissage : un grand K peut augmenter la vraisemblance, mais aussi la pénalité de complexité ;
- d'éviter le sous-apprentissage : un petit K réduit la complexité, mais diminue souvent la vraisemblance ;
- de comparer plusieurs modèles appris avec des valeurs de K différentes, en les ramenant à un même critère.

Ainsi, le K retenu est celui qui minimise le score BIC parmi les candidats évalués, offrant un compromis satisfaisant entre simplicité et qualité d'ajustement du modèle.

4.3 Implémentation dans la classe `Learner`

Le constructeur de la classe `Learner` accepte en paramètre optionnel l'ordre K du KTBN à apprendre. Si ce paramètre n'est pas spécifié, la méthode `learn_ktbn()` procède à une recherche itérative sur des valeurs de K allant de 2 à une valeur maximale `max_k` (fixée par défaut à 10). À chaque itération, le score BIC est calculé, et le modèle finalement retenu est celui dont le score BIC est minimal. La classe `Learner` permet d'accéder aux scores BIC et aux vraisemblances pour chaque valeur de k testée, facilitant l'analyse des résultats.

Architecture et algorithme principal :

```
    class Learner:
        def learn_ktbn(self, max_k=10):
        # Retourne directement le KTBN si k est connu.
        if self._k != -1:
            return self._learn()


        best_bic_score = float('inf')

        for k in range(2, max_k + 1):
            # Apprentissage avec k fixe
            self._init_learners(k)
            ktbn = self._learn()

            # Evaluation BIC
            bic_score = self._calculate_bic_score(ktbn)

            # Selection du meilleur modele
            if bic_score < best_bic_score:
                best_bic_score = bic_score
                best_k = k
                best_ktbn = ktbn
```

Cette approche offre une solution pratique pour trouver le k optimal, bien que la recherche exhaustive puisse être coûteuse pour de grandes valeurs de `max_k`. À notre connaissance, il s'agit du premier algorithme d'apprentissage dynamique de l'ordre K.

5 Expérimentations

5.1 Méthodologie générale

Afin d'évaluer empiriquement les performances des algorithmes développés, il est essentiel de disposer d'un grand volume de données représentatives, couvrant une large diversité de cas, dans le but de limiter les biais liés à des configurations particulières. Pour cela, nous avons adopté une approche fondée sur la génération de données synthétiques.

La procédure expérimentale s'effectue en deux étapes :

1. Génération d'un KTBN aléatoire : Un modèle est généré à l'aide de la méthode `KTBN.random()`, qui permet de spécifier des paramètres tels que le nombre de variables par tranche temporelle, l'ordre temporel k, le nombre de modalités par variable, ainsi que la densité (nombre d'arcs).
2. Échantillonnage de trajectoires : Le modèle ainsi obtenu sert à générer un ensemble de trajectoires via la méthode `sample()`, suivant la distribution jointe du KTBN.

Cette méthode présente l'avantage de fournir un modèle de référence connu (le KTBN générateur), ce qui permet une évaluation rigoureuse des performances de l'apprentissage. En effet, il devient possible de comparer le modèle appris à partir des trajectoires, obtenu avec la classe `Learner`, au modèle d'origine, en utilisant différents critères quantitatifs.

5.2 Évaluation de l'apprentissage avec K fixé

Lorsque l'ordre temporel K est connu, la classe `Learner` délègue l'apprentissage de la structure et des paramètres du KTBN aux algorithmes standards proposés par `pyAgrum`. Pour évaluer la qualité du modèle appris, nous avons comparé le KTBN estimé au modèle générateur en mesurant à la fois la similarité de leurs structures et la proximité de leurs distributions de probabilité. Nous avons analysé l'évolution de ces mesures en fonction de la taille de la base d'apprentissage.

5.2.1 F-Score

Le F-score est une métrique classique d'évaluation structurelle qui combine la précision (proportion d'arcs prédits corrects parmi tous les arcs prédits) et le rappel (proportion d'arcs prédits corrects parmi tous les arcs réels). Nous avons considéré deux variantes :

- F-score squelette : basé sur les graphes non orientés (squelettes), ignorant l'orientation des arcs.
- F-score orienté : prenant en compte l'orientation des arcs dans le graphe.

Protocole expérimental

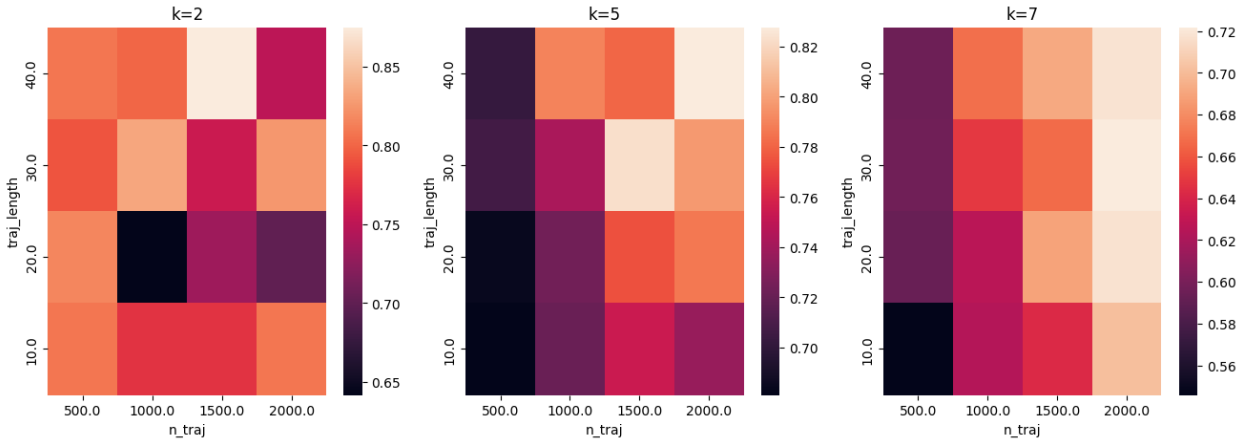
L'expérience a été conduite selon les paramètres suivants :

- Nombre de variables par tranche temporelle : fixé à $n = 4$
- Nombre de modalités par variable : $m = 3$
- Densité du graphe (nombre d'arcs) : 0.1 (du nombre total d'arcs)

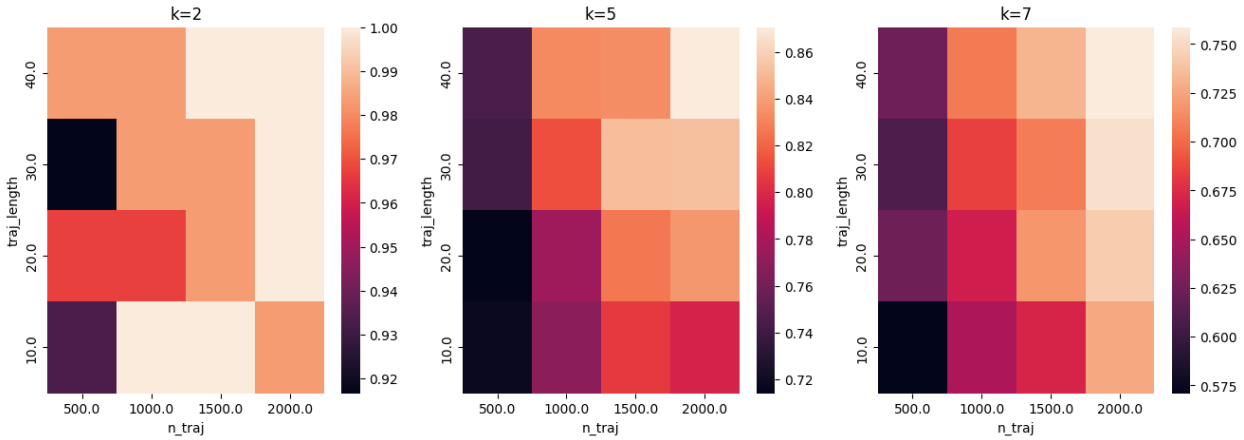
- K : valeurs testées $K \in \{2, 5, 7\}$
- Nombre de trajectoires : valeurs variables entre 500 et 2000, avec un pas de 500
- Longueur de chaque trajectoire : valeurs variables entre 10 et 40, avec un pas de 10

Les résultats ont été moyennés sur 20 tirages aléatoires afin d'assurer la robustesse des mesures.

Résultats



(a) Heatmap f-score orienté



(b) Heatmap f-score squelette

Figure 3: F-score (squelettique et orienté)

La figure 3 présente l'évolution du F-score (squelettique et orienté) en fonction du nombre de trajectoires et de leur longueur, pour différentes valeurs de K.

De manière générale, les résultats confirment l'intuition: plus la base d'apprentissage est grande, meilleure est la qualité de la structure apprise. Le F-score augmente de façon notable lorsque ces deux dimensions augmentent. Cependant, on observe que cette amélioration est moins marquée lorsque K augmente. En effet, pour des valeurs plus élevées de K, la complexité du modèle sous-jacent augmente, ce qui nécessite davantage de données pour estimer correctement la structure. À taille de base égale, les performances tendent donc à diminuer avec K, suggérant une exigence plus forte en volume de données pour maintenir une qualité d'apprentissage équivalente.

5.2.2 Divergence de Kullback-Leibler

Afin d'évaluer la qualité de l'apprentissage probabiliste, nous avons mesuré la divergence de Kullback-Leibler entre la distribution jointe générée par le KTBN appris et celle du KTBN générateur. Cette mesure, notée $D_{\mathrm{KL}}(P \parallel Q)$, quantifie la perte d'information induite lorsque le modèle appris Q est utilisé à la place de la distribution réelle P. Elle est définie par la formule suivante :

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x} P(x) \log \frac{P(x)}{Q(x)}$$

où la somme porte sur tous les états x du système.

Protocole expérimental

Le protocole expérimental reste identique à celui de la section précédente. Le nombre de variables par tranche temporelle, la densité des arcs et le nombre de modalités par variable sont fixés aux mêmes valeurs. L'ordre K est également fixé, avec des valeurs testées de $K = 2, 3, 4$, tandis que la taille de la base d'apprentissage est progressivement augmentée, à la fois en nombre de trajectoires et en longueur de trajectoires, afin d'observer l'impact de la quantité de données sur la qualité de l'estimation des probabilités.

Résultats

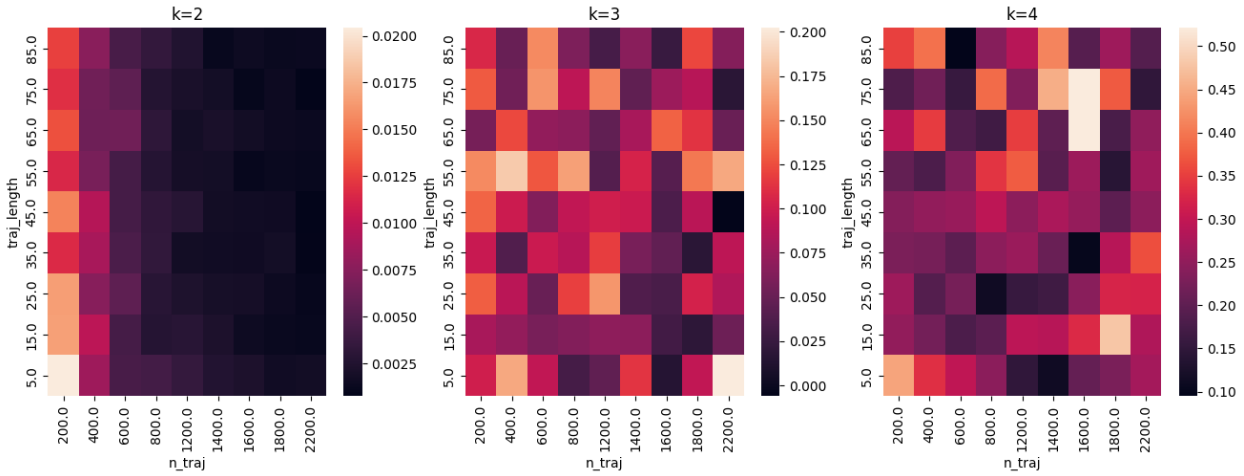


Figure 4: Heatmap divergence Kullback-Leibler

Les résultats, présents dans la figure 4 mettent en évidence une diminution globale de la divergence D_{KL} à mesure que la taille de la base d'apprentissage augmente, signe d'un meilleur ajustement du modèle aux données. Pour $K = 2$, cette amélioration est nette et régulière. En revanche, pour des ordres temporels plus élevés ($K = 3$ et $K = 4$), les courbes sont significativement plus bruitées. Bien qu'une tendance à la baisse soit observable, la convergence est moins stable et nécessite des volumes de données plus importants pour devenir significative. Ce comportement s'explique par la difficulté croissante à estimer correctement les distributions conditionnelles dans des espaces de dépendance temporelle plus larges.

5.2.3 Temps d'apprentissage

Cette expérience vise à évaluer la scalabilité de notre implémentation en mesurant le temps nécessaire pour apprendre un KTBN à partir de bases de données de taille variable.

Protocole expérimental

Nous avons fixé le KTBN générateur à $K = 3$, avec 4 variables discrètes à 3 modalités, et une densité de graphe de 0.1. L'algorithme apprend les paramètres et la structure à partir d'un ensemble de trajectoires synthétiques générées à partir de ce modèle. Deux expériences ont été menées indépendamment. Dans la première, la longueur des trajectoires est fixée à 10, et le nombre de trajectoires varie de 10 à 1960, par pas de 50. Dans la seconde, le nombre de trajectoires est fixé à 500, et la longueur de chaque trajectoire varie de 10 à 145, par pas de 5. Chaque configuration a été testé 20 fois.

Résultats

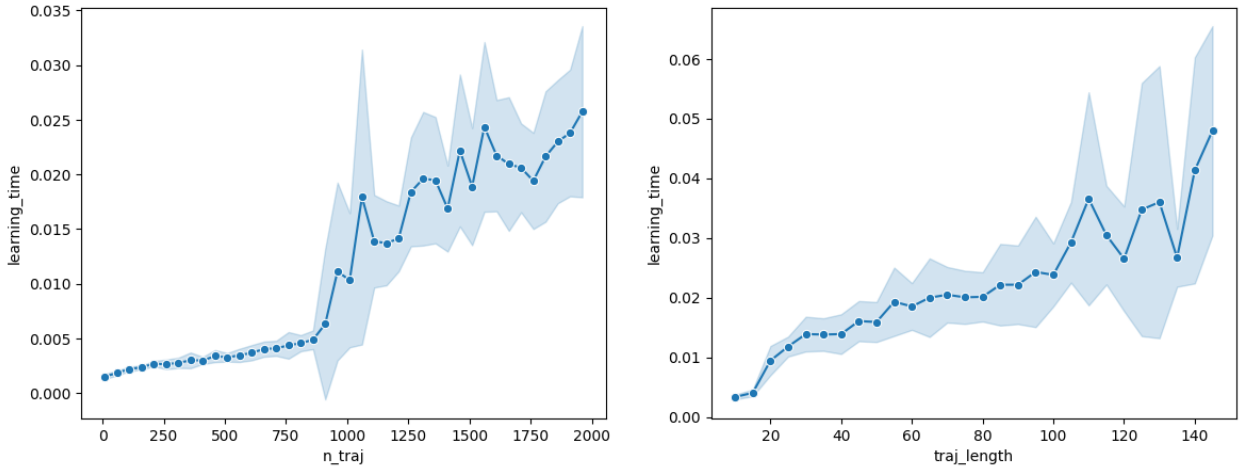


Figure 5: Temps d'apprentissage moyen en fonction de la taille de la base de données. À gauche, le nombre de trajectoires varie ; à droite, c'est la longueur des trajectoires qui varie. Chaque point représente la moyenne de 20 répétitions, l'ombre indique l'écart-type.

Les résultats, dans la figure 5, montrent que le temps d'apprentissage croît de manière globalement linéaire avec la taille de la base, aussi bien lorsque le nombre de trajectoires augmente que lorsque la longueur des trajectoires s'allonge. Des fluctuations sont observées, notamment pour les petites tailles de base, mais la tendance globale reste stable.

Ce comportement s'explique par le fait que la complexité algorithmique est principalement dominée par des facteurs structurels, tels que le nombre de variables et la valeur de K, plutôt que par le volume brut de données. Autrement dit, bien que le modèle sous-jacent soit de nature exponentielle, notamment en ce qui concerne l'espace de recherche structurel, l'impact de la taille de la base est contenu dans une croissance modérée du coût algorithmique.

Cela confirme que notre implémentation reste efficace et applicable à des jeux de données de grande taille, ce qui constitue une propriété essentielle dans des contextes applicatifs modernes.

5.3 Sélection automatique de K

Lorsque l'ordre temporel K n'est pas spécifié, la classe `Learner` tente d'estimer automatiquement une valeur adaptée à partir des données, en s'appuyant sur le critère BIC comme indicateur d'équilibre entre complexité et qualité d'ajustement.

Pour évaluer empiriquement cette capacité, nous avons mené deux expériences complémentaires :

- une analyse de l'évolution du score BIC en fonction de K, pour observer l'argument d'optimalité utilisé par `Learner` ;

- une comparaison directe entre la valeur de K estimée et la valeur réelle utilisée lors de la génération des données.

5.3.1 Évolution du score BIC

Protocole expérimental

- $k\_true = 3$
- $max\_k = 12$
- 1200 répétitions indépendantes
- Taille de base : 24,000 points (2000 trajectoires de longueur 12)

Résultats et validation

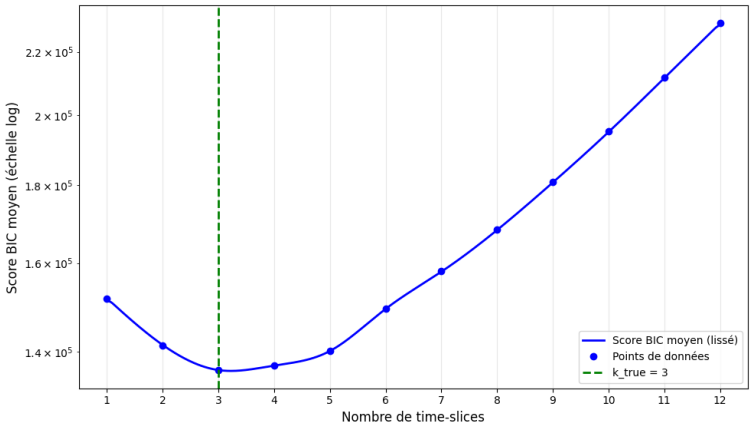


Figure 6: Évolution du score BIC en fonction de la valeur testée de K. Le minimum est atteint pour $K = 3$, valeur correspondant au KTBN générateur.

Notre hypothèse initiale était que le score BIC devrait présenter un pattern en cloche répétitif modulo 3, c'est-à-dire des minima locaux à $k = 3, 6, 9, 12$, etc., qui correspondent aux multiples de k_true. Cette hypothèse reposait sur l'idée que le modèle pourrait capturer des patterns temporels plus longs en "empilant" plusieurs cycles de l'ordre véritable (par exemple, un modèle d'ordre 6 pourrait théoriquement représenter deux cycles d'ordre 3 consécutifs).

Cependant, les résultats expérimentaux révèlent un comportement différent : après le minimum net à $k = 3$, le score BIC croît de manière monotone sans présenter de minima secondaires aux multiples de k_true. Cette divergence s'explique par la nature même du critère BIC et sa formulation $\text{BIC}(k) = \dim(\mathcal{B}_k) \cdot \ln N - 2 \cdot \ell(\mathcal{D} \mid \mathcal{B}_k)$, où le terme de pénalisation $\dim(\mathcal{B}_k) \cdot \ln N$ joue un rôle déterminant. Dans notre contexte, la dimension de l'espace des paramètres $\dim(\mathcal{B}_k)$ croît rapidement avec K, et avec $N = 24,000$ observations, le facteur $\ln N \approx 10.09$ impose une pénalisation très forte de la complexité structurelle.

Concrètement, même si un modèle d'ordre $K = 6$ pouvait théoriquement offrir une légère amélioration de log-vraisemblance $\ell(\mathcal{D} \mid \mathcal{B}_6)$ en capturant des patterns sur deux cycles de longueur 3, cette amélioration marginale (multipliée par 2) ne compense pas la pénalisation drastique imposée par le doublement de la dimension paramétrique. Le terme de pénalisation domine ainsi largement le terme de log-vraisemblance pour $k > k\_true$, empêchant l'émergence de minima secondaires et confirmant que le BIC privilégie naturellement la parcimonie structurelle.

Cette observation renforce finalement la pertinence du critère BIC : il identifie efficacement l'ordre véritable tout en évitant la sur-complexification du modèle.

5.3.2 Écart entre K estimé et K réel pour une base fixe

La matrice de confusion de la figure 7 quantifie la précision de `Learner`.

Protocole expérimental :

- $k\_true \in 2, 3, 4$
- 800 expériences totales
- $max\_k = 5$

Résultats :

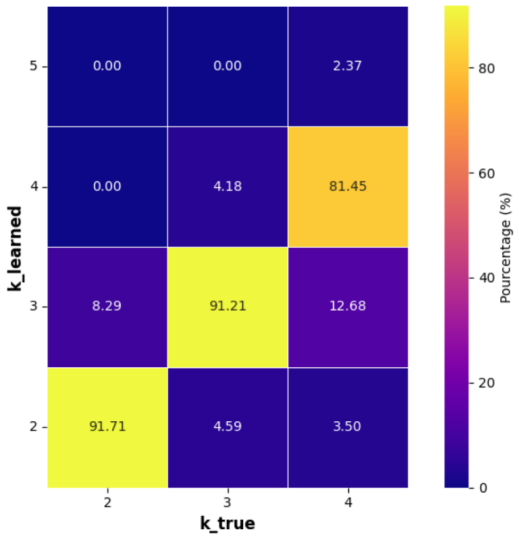


Figure 7: Heatmap K estimé vs K réel

Les 800 expériences révèlent une précision globale de 88.6% pour l'identification automatique de K. Cette performance démontre la capacité de `Learner` à identifier correctement le k optimal dans la majorité des cas testés.

Analyse des erreurs :

Les erreurs de `Learner` sont principalement des sous-évaluations de K plutôt que des sur-évaluations, ce qui est cohérent avec la nature pénalisante du score BIC. L'absence d'erreurs importantes témoigne de la robustesse de l'algorithme.

5.3.3 Influence de la taille de la base sur l'estimation de K

Protocole expérimental

Afin d'évaluer la robustesse de l'estimation automatique de K, nous avons étudié l'impact de la taille de la base d'apprentissage sur la capacité de l'algorithme à retrouver la valeur correcte de K. Nous avons fixé

successivement la valeur réelle de K à 3, 5 et 7, et fait varier la taille de la base en modifiant le nombre de trajectoires et leur longueur. Les autres paramètres (nombre de variables, modalités, densité) sont restés constants.

Pour chaque configuration, `Learner` a estimé la valeur optimale de K en minimisant le score BIC. Cette valeur a ensuite été comparée à la valeur réelle du modèle générateur.

Résultats

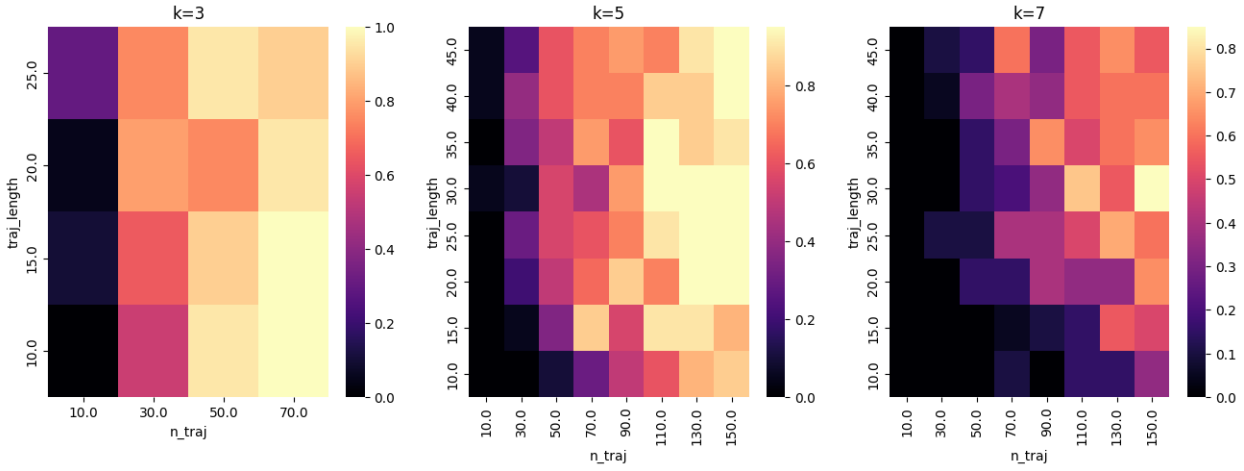


Figure 8: Heatmap K estimé vs K réel

Les résultats dans la figure 8 obtenus montrent qu'une base d'apprentissage suffisamment grande permet une estimation fiable de K. Pour $K = 3$, l'estimation est correcte dès que la base dépasse un certain seuil de taille modérée. Pour $K = 5$, une taille plus importante est requise pour converger vers la bonne valeur, tandis que pour $K = 7$, les performances restent bonnes, mais nécessitent des volumes de données significatifs pour éviter une sous-estimation.

6 Conclusion

Ce projet avait pour objectif la conception, l'implémentation et la validation d'un module complet pour la modélisation et l'apprentissage de réseaux bayésiens dynamiques d'ordre K arbitraire, dans le cadre de la bibliothèque `pyAgrum`. Deux composantes principales ont été développées : la classe `KTBN`, permettant de représenter explicitement ce type de modèle temporel, et la classe `Learner`, qui automatise l'apprentissage de la structure, des paramètres, et de l'ordre temporel à partir de données séquentielles catégorielles.

Lorsque l'ordre K est fourni, `Learner` s'appuie sur les algorithmes d'apprentissage de `pyAgrum` adaptés au contexte temporel pour construire un modèle fidèle à la distribution sous-jacente. En l'absence d'information sur K, l'algorithme explore plusieurs valeurs et sélectionne celle minimisant le critère BIC, assurant ainsi un bon équilibre entre adéquation aux données et simplicité du modèle.

Les expérimentations, menées sur des données synthétiques générées à partir de KTBN aléatoires, ont permis d'évaluer la qualité de l'apprentissage tant au niveau structural que probabiliste. Les résultats confirment la robustesse de notre approche, qui parvient à reconstruire efficacement les modèles d'origine dès que la taille de la base d'apprentissage est suffisante. En particulier, la capacité à estimer correctement l'ordre K montre l'utilité du critère BIC dans ce contexte.

L'ensemble du code développé sera intégré de manière officielle dans `pyAgrum`, suite à la validation de notre encadrant. Il constitue une contribution concrète à la modélisation probabiliste de phénomènes dynamiques,

offrant aux utilisateurs de la bibliothèque un outil flexible et automatisé pour l'apprentissage de réseaux bayésiens temporels.

Références

[1] Axelson, D. E., Standal, I. B., Martinez, I. and Aursand, M. [2009], 'Classification of wild and farmed salmon using bayesian belief networks and gas chromatography-derived fatty acid distributions', *Journal of Agricultural and Food Chemistry* **57**(17), 7634–7639. PMID: 19655799.
URL: *https://doi.org/10.1021/jf9013235*

[2] Bressan, G. M., Oliveira, V. A., Hruschka, E. R. and Nicoletti, M. C. [2009], 'Using bayesian networks with rule extraction to infer the risk of weed infestation in a corn-crop', *Engineering Applications of Artificial Intelligence* **22**(4), 579–592.
URL: *https://www.sciencedirect.com/science/article/pii/S0952197609000621*

[3] Chickering, D. M. [1996], 'Learning bayesian networks is np-complete', *Learning from data: Artificial intelligence and statistics V* pp. 121–130.

[4] Kannan, P. R. [2007], 'Bayesian networks: Application in safety instrumentation and risk reduction', *ISA Transactions* **46**(2), 255–259.
URL: *https://www.sciencedirect.com/science/article/pii/S0019057807000328*

[5] Mihajlovic, V. and Petkovic, M. [2001], 'Dynamic bayesian networks: A state of the art'.

[6] Pearl, J. [1988], *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[7] Perrin, B.-E., Ralaivola, L., Mazurie, A., Bottani, S., Mallet, J. and d'Alché Buc, F. [2003], 'Gene networks inference using dynamic bayesian networks', *Bioinformatics-Oxford* **19**(2), 138–148.

[8] Schwarz, G. [1978], 'Estimating the dimension of a model', *The annals of statistics* pp. 461–464.

[9] Zweig, G. and Russell, S. [1998], 'Speech recognition with dynamic bayesian networks'.