Énumération de motifs temporels

Hugo Hourcade

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par
Hugo Hourcade

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Sujet de la thèse :

Énumération de motifs temporels

Manuscript envoyé le 12/05/2023 au jury composé de :

Binh-Minh Bui-Xuan	Sorbonne Université & CNRS, France	Encadrant
Emmanuel Chailloux	Sorbonne Université, France	Directeur de thèse
Roland Grappe	Université Sorbonne Paris Nord, France	Rapporteur
Cédric Miachon	LesFurets, Courtanet, France	Encadrant
Maria Potop-Butucaru	Sorbonne Université, France	Examinatrice
Laurent Viennot	INRIA, France	Rapporteur, Président du jury

Résumé

Un graphe temporel G peut être représenté comme une suite de graphes statiques $(G_t)_{t\in\mathbb{N}}$ indexés par un ensemble d'instants temporels. Ce formalisme permet de modéliser le comportement et les interactions d'agents divers ainsi que la temporalité de ces interactions. La recherche de motifs temporels dans ce formalisme correspond alors à l'identification des similitudes de comportement ou à la recherche de comportements spécifiques dans la temporalité de ces interactions. Ces motifs temporels peuvent revêtir divers aspects et leur identification et énumération peut correspondre à différents problèmes.

Dans cette thèse, je considère deux de ces problèmes, l'énumération de Δ-jumeaux et l'énumération de sous-graphes temporels isomorphes à un motif, avec pour objectif de caractériser des populations par leurs interactions dans le cadre de cycles d'achat. Les graphes sur lesquels je réalise mes expériences, tirés de données d'exploitation réelle, présentent une taille d'historique, c'est-à-dire un nombre d'instants temporels entre le premier instant t_0 pour lequel G_{t_0} n'est pas vide et le dernier instant t_f pour lequel G_{t_f} n'est pas vide, de l'ordre de plusieurs millions d'instants temporels. Du fait de cette taille importante, je me concentre particulièrement sur l'influence de la taille de l'historique du graphe temporel sur le temps de calcul des algorithmes proposés.

Le premier problème traité par cette thèse est le problème d'énumération des Δ-modules d'un graphe. Étant donné un entier Δ, un Δ-module est un ensemble de sommets A ayant le même voisinage en dehors de A pour Δ instants consécutifs. Un tel sous-ensemble peut par exemple s'avérer utile pour identifier des populations ayant des comportements similaires sur des périodes données mais pouvant se comporter différemment en dehors de ces périodes. Cette thèse traite également de trois cas spécifiques du problème d'énumération de Δ-modules. Le cas spécifique de l'énumération des modules éternels est tel que $\Delta = \infty$. Le cas spécifique de l'énumération des Δ-jumeaux est tel que $\|A\| = 2$. Le cas spécifique de l'énumération des jumeaux éternels est tel que $\|A\| = 2$ et $\Delta = \infty$. En utilisant une structure de données basée sur les arbres rouge-noir, je donne une solution au problème d'énumération de Δ-modules en temps quadratique de la taille de l'historique et aux cas spécifiques de ce problème en temps logarithmique de l'historique.

Le deuxième problème consiste en l'énumération des sous-graphes temporels de G isomorphes à un deuxième graphe temporel appelé "motif". Étant donné un deuxième graphe temporel P, l'isomorphisme de sous-graphes temporels est une paire constituée d'un instant temporel de G et d'une bijection des sommets de P vers un sous-ensemble de sommets de G. Cette bijection doit être telle qu'à partir de l'instant temporel en question, les images des sommets de P par la bijection présentent le même comportement interactif entre eux que les sommets dont ils sont l'image. Une telle bijection permet par exemple d'identifier des motifs comportementaux permettant d'appréhender des relations de contingence ou de causalité au sein de cycles d'achat. Cette thèse traite également le cas spécifique de l'isomorphisme de sous-graphes temporels pour lequel P est une forêt temporelle linéaire et G une forêt temporelle. En utilisant le lien entre isomorphisme de sous-graphes statiques et la recherche de cliques de taille donnée, je donne une solution au problème d'énumération des sous-graphes temporels isomorphes, avec une complexité temporelle linéaire en la taille de l'historique.

L'étude expérimentale des solutions des problèmes d'énumération de Δ-modules et d'énumération de sous-graphes temporels isomorphes confirment une limite, anticipée par l'étude théorique, aux tailles des graphes temporels traitables par ces algorithmes. Selon l'analyse numérique, et sachant que sur les jeux de données que je traite dans cette thèse, une seconde de temps réel sépare deux instants temporels successifs du graphe, mes expérimentations sur des graphes obtenus à partir de données du monde réel passent à l'échelle dans les cas spécifiques de l'énumération des Δ-jumeaux et des jumeaux éternels jusqu'à une taille d'historique de 10^8 instants temporels, soit 3 ans et 2 mois, et dans le cas spécifique des modules éternels ainsi que pour l'énumération des sous-graphes temporels isomorphes jusqu'à une taille d'historique de 10^6 instants temporels, qui équivaut à 11 jours et demi d'historique.

Abstract

A temporal graph G can be represented as a sequence of static graphs $(G_t)_{t\in\mathbb{N}}$ indexed by a set of temporal instants. This formalism allows one to model multiple agents interaction behaviours along with the temporal evolution of said interactions. The search for temporal patterns in this formalism is then akin to the identification of behaviour similarities or the search for specific behaviours in the chronology of said interactions. Those temporal patterns can be modeled in many ways and their identification and enumeration can correspond to various problems.

In this thesis, I consider two of those problems, the Δ-modules enumeration and the enumeration of sub-graphs isomorphic to a given pattern, the goal being to characterize populations by their interactions in the course of a purchase cycle. The graphs on which I conduct my experiments, extracted from real-world data, display a history length, which is the number of temporal instants between the first instant t_0 for which G_{t_0} is not empty and the last instant t_f for which G_{t_f} is not empty, ranging up to several million temporal instants. Given this huge history length, I focus on the influence of this history length on the computation time of the algorithms presented in this thesis.

The first problem considered in this thesis is the Δ-modules enumeration problem. Given an integer Δ, a Δ-module is a subset of vertices A which have the same neighbourhood outside of A for Δ consecutive instants. Such a subset can, for instance, be useful in the identification of populations displaying similar behaviours on temporal periods of a given length while behaving differently outside of those periods. This thesis also consider three specific cases of the Δ-modules enumeration problem. The specific case of eternal modules enumeration is such that $\Delta = \infty$. The specific case of Δ-twins enumeration is such that $\|A\| = 2$. The specific case of eternal twins enumeration is such that $\|A\| = 2$ and $\Delta = \infty$. Using data structures based on red-black trees, I give a solution to the Δ-modules enumeration problem with a computation time quadratic in the history length, and to the specific cases of this problem with a computation time logarithmic in the history length.

The second problem consists in the enumeration of all temporal sub-graphs of G isomorphic to a given temporal graph called "pattern". Given a second temporal graph P, a temporal sub-graph isomorphism is a pair, composed of a temporal instant of G and of a bijection from the set of vertices of P to a subset of vertices of G. Said bijection must be such that, starting from the aforementioned temporal instant, images of vertices of P by this bijection display the same interactive behaviour between them than the vertices they are images of. Such a bijection allows, for instance, the identification of behaviour patterns displaying contingency or causality relations inside a purchase cycle. This thesis also consider the specific case of the temporal subgraph isomorphism for which P is a linear forest and G a forest. Using a method solving static sub-graph isomorphism problems by way of a search of cliques of a given size, I give a solution to the isomorphic temporal sub-graphs enumeration with a temporal complexity linear in the history length.

Experiments on the solutions of Δ-modules enumeration problem and isomorphic temporal sub-graphs enumeration problem confirm a theoretically foreseen limit to the size of the temporal graphs those algorithms can be run on. According to those experiments, and given that, on the datasets I consider for this thesis, two consecutive time instants are one real-time second apart from each other, my experiments on graphs extracted from real-world data scale up to 10^8 temporal instants for the specific cases of Δ-twins enumeration and eternal twins enumeration, which is to say 3 years and 2 months, and up to 10^6 temporal instants for eternal modules enumeration and isomorphic temporal sub-graphs enumeration, which represent 11 days and a half of history.

Remerciements

Je remercie Emmanuel Chailloux, professeur des universités au LIP6 et directeur de ma thèse pour m'avoir donné l'opportunité de conduire cette recherche, pour les discussions et indications fondamentales dans la progression de mon travail, pour ses retours très importants et fondamentaux sur mes travaux, notamment en terme de rédaction scientifique, et pour avoir encadré les démarches administratives nécessaires.

Je remercie Binh-Minh Bui-Xuan, maître de conférence au LIP6 et au CNRS, pour son encadrement, pour m'avoir dirigé dans ce travail de recherche, m'avoir donné des indications fondamentales pour surmonter certains obstacles, pour son aide et sa pédagogie dans la rédaction scientifique, et pour avoir joué un rôle important dans la spécification du sujet au début de mon doctorat.

Je remercie Cédric Miachon, Head of Data de Courtanet, pour son encadrement au sein de l'entreprise, les discussions sur les liens entre le travail de recherche conduit et les applications industrielles de mes résultats, et pour avoir porté et défendu le projet auprès de Courtanet pendant toute la durée de ma thèse.

Je remercie Courtanet pour m'avoir fait confiance et avoir accepté de financer cette thèse, et pour m'avoir donné l'opportunité de mener cette thèse.

Je remercie Laurent Viennot, chercheur à l'INRIA, pour avoir participé à mon comité de mi-parcours, pour avoir été rapporteur de cette thèse et pour avoir présidé mon jury de soutenance, de même que pour les indications utiles, notamment quant à des erreurs scientifiques que j'avais pu commettre et qui ne méritaient pas de figurer sur le présent mémoire.

Je remercie Roland Grappe, maître de conférence au LIPN, pour avoir été rapporteur de cette thèse, pour sa participation au jury de soutenance, et pour les discussions et indications utiles et formatrices, notamment en terme de rédaction scientifique, lors de la phase de correction du présent mémoire.

Je remercie Maria Potop-Butucaru, professeur des universités au LIP6, pour avoir participé à mon comité de mi-parcours et à mon jury de soutenance et pour les indications intéressantes.

Je remercie Fatemeh Hamissi, stagiaire au LIP6, pour ses travaux de stage qui m'ont servi de base pour établir l'algorithme d'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire présenté dans cette thèse.

Je remercie Eleni Pistiloglou, stagiaire au LIP6, pour la bibliographie de ses travaux de stage, qui m'a servi dans le cadre de mon étude de l'énumération des sous-graphes temporels isomorphes par suite d'arêtes.

Je remercie l'ensemble de l'équipe APR du LIP6, et l'ensemble des employés de Courtanet, qui ont participé à me fournir un environnement de travail sain et agréable, et pour avoir été des soutiens moraux, et des interlocuteurs intéressés, pendant toute la durée de ma thèse, soutien bienvenu, notamment pendant toute la période de la pandémie de Covid-19.

Et merci à mes amis et ma famille pour ce même soutien.

Sommaire

Chapitre 1

Introduction

Dans le cadre de l'étude des cycles d'achat d'utilisateurs d'un site internet d'e-commerce, on désigne sous le nom d'"exitters" les populations qui quittent le site avant d'avoir réalisé un achat. Courtanet, propriétaire du site internet d'e-commerce lesfurets.com, qui co-encadre cette thèse dans le cadre d'un contrat CIFRE, s'intéresse à l'utilisation de la théorie des graphes afin d'identifier ces populations d'"exitters". Le but est de repérer chez les utilisateurs des comportements caractéristiques des populations d'"exitters", un tel comportement mettant en exergue un frein à la poursuite du parcours du site par l'utilisateur, tel qu'une absence, un manque d'information ou un défaut dans la conception de l'expérience utilisateur, en vue de déclencher dynamiquement et préventivement une action pour pallier ce frein avant que l'utilisateur ne quitte le site. Dans le cadre de cette thèse, je cherche donc à identifier des comportements d'"exitters", c'est-à-dire des comportements reflétant un motif comportemental connu comme étant caractéristique d'une population d'"exitters" ou un comportement semblable à celui d'une population connue comme faisant partie des "exitters".

La recherche de tels motifs peut être formalisée par divers concepts de théorie des graphes qui peuvent être regroupés sous le terme de recherche de motifs temporels.

Lors de la récupération des informations d'un jeu de données à historique, l'instant temporel auquel une information est enregistrée en base de données peut être aussi important que la donnée elle-même. Afin de prendre en compte cette temporalité, je travaille avec le formalisme des graphes temporels. Considérons un graphe temporel comme une suite de graphes statiques indexés par des instants temporels $G = (G_t)_{t \in T}$. Cette thèse se concentre sur deux problèmes de recherche de motifs temporels.

Le premier problème est la recherche d'un comportement spécifique représenté par une partie de l'évolution temporelle du voisinage d'un sommet s donné.

Dans sa forme la plus simple, la détection d'un comportement donné, tel que celui dont le sommets s est acteur, équivaut à une énumération de tous les sommets présentant le même voisinage que s à chaque instant temporel du jeu de données. Je définis formellement une paire de *jumeaux éternels* $\{u, w\} \in V^2$, où V est l'ensemble de sommets de G, comme une paire de sommets dont les voisinages instantanés $N_t(u)$ et $N_t(w)$ sont égaux dans $V \backslash \{u, w\}$ pour chaque instant temporel $t \in T$. Un voisinage instantané $N_t(u)$ d'un sommet u est l'ensemble de tous les sommets adjacents à u dans le graphe G_t.

Le problème d'énumération associé au cas temporel est défini comme suit :

EnumerationDesJumeauxEternelsDUnSommetDonne
Soit un graphe temporel $G(V, E, T)$ et un sommet $s \in V$.
On cherche la liste de tous les sommets formant une paire de jumeaux éternels avec s dans G.

Les sommets formant une paire de jumeaux éternels avec le sommet choisi ont exactement le même comportement que lui à chaque instant temporel. En réalité, ce cas de figure apparaît très peu souvent dans les jeux de données réels. Les paires de jumeaux éternels ont plus de chance de représenter des duplications de données que de réelles similitudes de comportement entre deux

agents du système étudié.

Afin d'obtenir des informations pertinentes dans un jeu de données réel, je considère le sujet plus général de l'énumération des Δ-modules. Cette notion correspond à une ressemblance partielle entre voisinages de sommets qui ne sont pas des duplicats. Énumérer les Δ-modules peut donc aider à détecter des motifs comportementaux sur des périodes temporelles spécifiques qui permettront de caractériser les populations d'un même groupe.

Pour un entier Δ donné, un Δ-*module* est un ensemble $A \subseteq V$ pour lequel il existe un instant temporel t_0 tel que, pour tout $\{u, w\} \in A^2$, $N_t(u)$ et $N_t(w)$ sont strictement égaux dans $V \backslash A$ pour Δ instants temporels consécutifs, $t_0 \leqslant t < t_0 + \Delta$ avec $[t_0, t_0 + \Delta[\subseteq T$. Quand $\|A\| = 2$, j'appelle le Δ-module une *paire de Δ-jumeaux*. Naturellement, en dehors de la fenêtre temporelle de Δ instants temporels consécutifs, les comportements de deux Δ-jumeaux peuvent être différents. Je note τ la taille de l'historique de G, c'est-à-dire le nombre d'instants temporels séparant le premier instant pour lequel le graphe n'est pas vide et le dernier instant pour lequel le graphe n'est pas vide. Quand $\Delta = \tau$, j'appelle le τ-module un *module éternel*. Les comportements de $u \in A$ et $w \in A$ peuvent être très différents au sein du module A, ce qui permet d'identifier des similitudes entre des agents qui diffèrent pourtant dans leurs interactions internes.

Les problèmes d'énumération associés sont définis comme suit :

EnumerationDesΔ-ModulesContenantUnSommetDonne
Soit un graphe temporel $G(V, E, T)$, un sommet $s \in V$ et un entier $\Delta \in \mathbb{N}$.
On cherche la liste de tous les Δ-modules de G contenant s.

EnumerationDesModulesEternelsContenantUnSommetDonne
Soit un graphe temporel $G(V, E, T)$ et un sommet $s \in V$.
On cherche la liste de tous les modules éternels de G contenant s.

EnumerationDesΔ-JumeauxDUnSommetDonne
Soit un graphe temporel $G(V, E, T)$, un sommet $s \in V$ et un entier $\Delta \in \mathbb{N}$.
On cherche la liste de tous les sommets formant une paire de Δ-jumeaux avec s dans G.

Dans un graphe temporel représentant l'activité d'utilisateurs sur un site internet d'e-commerce, les paires de Δ-jumeaux peuvent par exemple caractériser des utilisateurs présentant un comportement d'achat similaire pour une période donnée, soulignant un motif dans le cycle d'achat, ce qui permet des recommandations d'achat basées sur cette ressemblance locale. De même, les modules éternels peuvent représenter une ressemblance entre des sommets qui ne sont pas des duplicats, leur comportement différant au sein du module. Ces modules peuvent donc représenter une population qui présente toujours le même comportement envers l'extérieur, sans égard pour les disparités internes qui peuvent exister. De tels ensembles peuvent ainsi regrouper les membres d'une communauté ayant les mêmes interactions avec l'extérieur de leur groupe, sans considération pour les interactions internes au groupe. Dans le contexte d'un réseau social, les modules éternels peuvent donc caractériser des sous-cultures ou des groupes d'opinion comme des groupes extrémistes ou conspirationnistes (par le phénomène de la caisse de résonnance). Il est à noter que dans un graphe biparti, qui est un graphe temporel représenté par deux ensembles de sommets colorés différemment, tel qu'il n'existe aucune arête entre deux sommets de même couleur, les modules éternels unicolores représentent des ensembles de duplicats de données, aucune interaction n'existant à l'intérieur du module. Enfin, les Δ-modules combinent les deux notions précédemment mentionnées, caractérisant des ensembles de sommets qui présentent le même comportement envers l'extérieur pour une période de temps tout en différant à la fois dans leurs interactions internes et dans leur comportement en dehors de la fenêtre temporelle de Δ instants temporels consécutifs considérée. Cette dernière notion plus laxe puisqu'elle permet de regrouper des populations ayant temporairement un comportement identique vis-à-vis de l'extérieur, permet la caractérisation de populations qui appartiennent temporairement au même groupe sans que l'analyse soit polluée par leurs spécificités internes. Par exemple, dans un graphe représentant des échanges de courriers électroniques (emails), si on considère la période de temps qui suit la réception d'un email d'hameçonnage (phishing) par l'ensemble des utilisateurs, les Δ-modules caractérisent une réponse identique, regroupant les

utilisateurs crédules au sein d'un module et les utilisateurs qui signalent la fraude à leur fournisseur au sein d'un autre module, sans prendre en considération les différences qui peuvent intervenir en dehors de cette période ou les interactions entre membres d'une même population.

Pour le deuxième problème étudié, on définit un second graphe temporel $P(V', E', T')$, appelé motif, qui caractérise le comportement que l'on souhaite repérer dans notre graphe temporel. Le comportement recherché est alors représenté par l'ensemble d'arêtes temporelles E' de P sur une période temporelle T', et l'ensemble des sommets V' représente les acteurs se livrant à ce comportement.

La détection de ce comportement équivaut à une énumération des sous-graphes temporels de G présentant le même comportement que les sommets de P pendant un intervalle de temps de T de durée égale à $\|T'\|$. Je définis formellement un *isomorphisme de sous-graphes temporels* $\{f : V' \to A \subseteq V, t_0 \in T\}$, où V est l'ensemble de sommets de G, V' l'ensemble de sommets de P, et f est une bijection, des sommets de V' vers un sous-ensemble de V, qui conserve les relations d'adjacence pour une certaine période d'instants temporels $[t_0, t_0 + \|T'\|[$. La conservation des adjacences internes signifie que pour toute paire$\{u, v\} \in V'^2$, il existe une arête entre u et v à l'instant $t \in T'$ dans E' si et seulement s'il existe une arête entre $f(u)$ et $f(v)$ à l'instant temporel $t_0 + t$ dans E. T' joue ici le même rôle que la fenêtre temporelle de taille Δ dans le problème d'énumération des Δ-modules comme représentation de la période temporelle où le comportement recherché apparaît. Cela signifie que le comportement du sous-ensemble de sommets de G qui sont images des sommets de P par f, en dehors de la fenêtre temporelle $[t_0, t_0 + \|T'\|[$, est ignoré, permettant la recherche d'un comportement temporaire. La différence entre la notion de Δ-modules et celle de sous-graphe temporel isomorphe est que là où le Δ-module permettait de modéliser une similitude de comportement vis-à-vis de l'extérieur sans prendre compte des interactions internes, la notion de sous-graphe isomorphe permet de modéliser des comportements spécifiques en interne, sans égard au comportement des agents concernés vis-à-vis de l'extérieur.

Le problème d'énumération associé au cas temporel est défini comme suit.

EnumerationDesSousGraphesTemporelsIsomorphesAUnMotif
Soit deux graphes temporel $G(V, E, T)$ et $P(V', E', T')$.
On cherche la liste de toutes les paires composées d'un instant temporel $t_0 \in T$ et d'une bijection f qui associe à chaque sommet de V' une image dans V, telles qu'il existe une arête entre deux sommets de V' à un instant $t \in T'$ dans P si et seulement si il existe une arête entre leurs images par f à l'instant $t + t_0$ dans L.

Dans un graphe temporel représentant l'activité d'utilisateurs sur un site internet d'e-commerce, tel que le graphe `LesFurets` que j'utilise dans le cadre de cette thèse, qui représente les utilisateurs du site et les événéments systèmes de celui-ci par des sommets, et une interaction entre un utilisateur et un événement système par une arête temporelle, les sous-graphes isomorphes permettent de caractériser des successions d'actions liées par une relation de causalité ou de contingence. Ils permettent par exemple de repérer des cycles d'achat dans lesquels plusieurs produits sont consultés avant que l'un d'entre eux soit acheté, ce qui permet de détecter des liens d'intérêts similaires entre divers produits et de réaliser des recommandations d'achat pour les utilisateurs qui commencent à s'intéresser à certains de ces produits, avec une compréhension de la temporalité associée à l'évolution de cet intérêt commun. Ils permettent également de repérer des comportements comme signes d'un blocage du parcours de l'utilisateur, de sorte à pouvoir apporter une modification au parcours de cet utilisateur qui lui permet de contourner ce blocage.

Dans cette thèse, je m'intéresse également à la deuxième formalisation de ce problème qui peut être trouvée dans les publications scientifiques (comme dans l'article de Paranjape et Pardalos [40]). Dans ce formalisme, P est un graphe temporel pour lequel, à tout instant temporel, P n'est pas un graphe vide. Je définis la compression à l'infini d'un graphe temporel G comme le graphe statique ayant le même ensemble de sommets que G et tel qu'il existe une arête entre deux sommets u et v dans ledit graphe statique s'il existe un instant temporel t tel qu'une arête lie u et v à l'instant t dans G. Étant donné un entier Δ, le problème d'énumération des isomorphismes de

sous-graphes temporels par suite d'arêtes consiste à énumérer les paires composées d'une bijection $f : V' \to A \subseteq V$ et d'une suite $(e_n)_{n\in\mathbb{N}}$. f est ici un isomorphisme de sous-graphes statiques entre la compression à l'infini de L et la compression à l'infini de P. Les arêtes de $(e_n)_{n\in\mathbb{N}}$ appartiennent à E et sont contenues dans une fenêtre temporelle de Δ instants temporels consécutifs de T. $(e_n)_{n\in\mathbb{N}}$ est de plus classée par instants temporels croissants et pour tout $n \in \mathbb{N}$, e_n est l'image de la n-ième arête de E', E' étant également triée par instants temporels croissants.

Ce formalisme, que je rebaptise, afin de différencier les deux formalismes, sous le nom de *l'isomorphisme de sous-graphes temporels par suite d'arêtes*, permet d'identifier des successions d'interactions au sein d'un groupe sur un intervalle de temps, sans faire cas des comportements parasitaires qui pourraient également intervenir dans l'intervalle de temps et sans que l'intervalle de temps entre chaque interaction soit strictement contraint.

Le problème d'énumération associé à ce formalisme est défini comme suit :

EnumerationDeSousGraphesTemporelsIsomorphesParSuiteDAretesAUnMotif
Soit deux graphes temporels $G(V, E, T)$ et $P(V', E', T')$ et un entier Δ.
On cherche la liste de toutes les paires composées d'une bijection f qui associe à chaque sommet de V' une image dans V, telle qu'il existe un instant temporel tel qu'une arête temporelle lie deux sommets de V' dans P si et seulement si il existe un instant temporel tel qu'une arête temporelle lie les images de ces sommets par f dans G, et d'une suite d'arêtes temporelles de E, ordonnée par instants temporels croissants, telles que ses arêtes apparaissent dans un intervalle temporel de taille maximale Δ, et que la k-ième arête de cette suite soit l'image de la k-ième arête de E'.

Appliqué au cas d'utilisation d'un site internet d'e-commerce, ce problème permet l'identification de comportements d'utilisateurs constitués d'une succession d'actions réalisées dans un temps borné tout en autorisant une certaine flexibilité quant à la temporalité au sein de cet intervalle de temps, le tout en ignorant les actions qui ne participent pas au comportement étudié.

Une forêt temporelle est un graphe temporel G tel qu'à chaque instant temporel $t \in T$, G_t est une forêt. Une forêt temporelle linéaire est un graphe temporel G tel qu'à chaque instant temporel $t \in T$, G_t est une forêt linéaire. Quand G est une forêt temporelle et P est une forêt temporelle linéaire, j'appelle $\{f, t_0\}$ un *isomorphisme de sous-forêts temporelles pour un motif de forêt temporelle linéaire.*

Je m'intéresserai également à ce cas particulier du problème de sous-graphes isomorphes :

EnumerationDesSousForetsTemporellesIsomorphesAUneForetLineaire
Soit deux graphes temporel $G(V, E, T)$ et $P(V', E', T')$, G étant une forêt temporelle et P une forêt temporelle linéaire.
On cherche la liste de toutes les paires composées d'un instant temporel $t_0 \in T$ et d'une bijection f qui associe à chaque sommet de V' une image dans V et telle qu'il existe une arête entre deux sommets de V' à un instant $t \in T'$ dans P si et seulement si il existe une arête entre leurs images par f à l'instant $t + t_0$ dans L.

Cette thèse traite les questions suivantes : *peut-on obtenir une réponse en temps raisonnable au problème d'énumération des Δ-ModulesContenantUnSommet pour les graphes temporels issus de récolte de données réelles, c'est à dire que le temps de calcul reste de l'ordre de quelques minutes au maximum ? Peut-on obtenir une réponse en temps raisonnable au problème d'énumération des IsomorphismeDeSousGraphesTemporels sur ces mêmes graphes temporels ? Cela peut-il être confirmé par les expérimentations sur les implémentations des algorithmes présentés ?* En particulier, la thèse se concentre sur de longs historiques (un τ fort, pouvant s'élever à plusieurs dizaines de millions d'instants temporels), un faible nombre de sommets ($n = \|V\|$ faible, de l'ordre de quelques centaines au maximum) et un bon nombre d'arêtes temporelles ($m = \|E\|$ moyen, quelques dizaines de milliers d'arêtes).

Du fait des propriétés de ces graphes, notamment des longs historiques de ces graphes temporels, mon principal but sera de limiter autant que possible l'influence de la taille de l'historique τ sur les complexités spatiales et temporelles des algorithmes que je présente pour résoudre tous ces problèmes d'énumération. Dans la mesure du possible, je chercherai donc à établir que les

algorithmes présentés dans cette thèse ont des complexités indépendantes de τ, logarithmiques en τ ou dans le pire des cas linéaires de τ. Si j'essaie de diminuer également au maximum l'influence des autres paramètres du problème, à savoir le nombre de sommets du graphe temporel et le nombre d'arêtes temporelles, sur les complexités spatiales et temporelles, c'est principalement l'influence de τ sur ces complexités qui m'intéressera.

Pour ce qui est du problème d'énumération des Δ-modules, j'ai commencé par m'intéresser aux techniques d'affinage de partitions présentes dans la littérature pour les variantes de ces problèmes dans les graphes statiques, pour aboutir à la conclusion que les implémentations basées sur les listes, telles que celles décrites par Habib et al. dans leur article *Partition refinement techniques : an interesting algorithmic tool kit* [23] ou dans l'article *A survey of the algorithmic aspects of modular decomposition* de Habib et Paul [22, Lemma 10] dépendent linéairement, a minima, de la taille de l'historique, puisque le calcul devrait être réalisé sur chaque graphe statique G_t, ce qui ajoute au temps de calcul total un facteur multiplicatif en τ.

Afin de diminuer l'influence de τ sur la complexité temporelle, j'ai préféré privilégier des algorithmes itérant sur la liste des sommets ou des arêtes temporelles du graphe plutôt que sur l'historique. En effet, une itération sur l'historique entraînerait de facto une complexité temporelle a minima linéaire en τ, là où, dans la mesure du possible, je recherche une dépendance logarithmique en τ des complexités temporelles. Cette approche disqualifie donc l'utilisation des solutions aux problèmes statiques à chaque instant temporel. Il faut donc, pour chaque problème, trouver une propriété structurelle du problème qui puisse être exploitée lors d'un parcours de la liste des arêtes temporelles, ou de celle des sommets du graphe temporel traité.

Pour les problèmes d'énumération des Δ-modules et de leurs cas particuliers, la propriété structurelle que j'utilise est une propriété de permutation triangulaire des splitters. Un splitter d'une paire de sommet est un sommet mettant en exergue une différence entre les voisinages des deux sommets constituant la paire, prouvant que cette paire n'est pas une paire de jumeaux. Revisitant une structure de données d'arbres rouge-noir, j'ai conçu un calcul en temps logarithmique en la taille de l'historique du graphe en entrée pour l'énumération des Δ-jumeaux et des Δ-modules, et en temps indépendant en la taille de l'historique pour l'énumération des jumeaux éternels et des modules éternels. Les implémentations de ces algorithmes seront expérimentalement testées sur un jeu de données généré et trois jeux de données issus de données d'exploitation du monde réel. Pour chaque cas particulier du problème, deux variantes des algorithmes définis sont présentées et comparées. Les deux variantes diffèrent par l'utilisation de la suite de matrices d'adjacence de chaque graphe G_t pour chaque instant temporel $t \in T$, la version sans suite de matrice permettant d'éviter des problèmes de manque de mémoire pour les problèmes d'énumération des jumeaux. De plus, l'utilisation d'arbres rouge-noir permet aux algorithmes présentés dans cette thèse d'énumérer correctement les modules, même dans le cas où les arêtes des graphes ne sont pas ordonnées par instant temporel croissant. Cette caractéristique permet d'opérer sur des données récoltées de manière asynchrone, ce qui permet de mettre en place des versions streaming de ces algorithmes.

En ce qui concerne les sous-graphes temporels isomorphes, j'établis la nécessité, dans le cadre des jeux de données considérés, d'utiliser des propriétés structurelles du graphe temporel plutôt que d'exécuter à chaque instant temporel un calcul du problème statique associé. Je commence donc par chercher une solution au problème d'EnumérationDeSousForetsTemporellesIsomorphesAUneForetTemporelleLineaire en utilisant des solutions au problème statique d'énumération des sous-graphes statiques isomorphes à un motif et en appliquant cette solution à chaque instant temporel $t \in T$. Ces solutions s'appuient sur des algorithmes de Breadth First Search pour énumérer des chemins de taille n dans des arbres.

Puis, afin d'établir la supériorité d'une approche structurelle utilisant les degrés des sommets, je traite le cas général et utilise des résultats publiés par Paranjape et al. [40] établissant un lien entre l'isomorphisme de sous-graphes temporels par suite d'arêtes et l'isomorphisme de sous-graphes statiques, puis les résultats publiés par Kozen [32] établissant un lien entre l'isomorphisme de sous-graphes statiques et l'énumération de cliques de taille n et enfin les résultats publiés par Carraghan et Pardalos [12] pour établir un algorithme apportant une solution au problème

d'EnumérationDeSousGraphesTemporelsIsomorphesParSuiteDAretesAUnMotif.

Enfin en m'appuyant sur cette démarche, je modifie l'algorithme pour apporter une solution au problème d'EnumérationDeSousGraphesTemporelsIsomorphesAUnMotif. Je compare finalement les complexités de cette solution au problème général de l'énumération de sous-graphes temporels isomorphes à un motif à celles de la solution proposée pour le cas particulier de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire, afin d'établir la supériorité de la démarche structurelle sur la démarche statique itérative.

Les résultats théoriques obtenus sur les problèmes d'énumération des Δ-modules et d'énumérations des sous-graphes temporels isomorphes sont présentés, puis une étude expérimentale permet de valider les résultats théoriques. Si des résultats expérimentaux positifs ont pu être obtenus pour les problèmes d'énumération des jumeaux éternels, des Δ-jumeaux et des modules éternels, le problème d'énumération des Δ-modules dans le cas général ne donnera pas lieu à des résultats expérimentaux satisfaisants sur les jeux de données considérés, du fait de complexités trop importantes des algorithmes proposés. Les résultats expérimentaux des implémentations des algorithmes présentés pour l'énumération des sous-graphes temporels isomorphes sont également présentés et je discuterai des limites de l'utilisation pratique de ces algorithmes.

Tous les algorithmes présentés dans cette thèse ont été implémentés en Java, sont diffusés et mis à disposition sous licence libre (pour les Δ-modules : `https://github.com/DaemonFire/deltaModules`, pour les sous-forêts temporelles isomorphes à une forêt temporelle linéaire : `https://github.com/DaemonFire/PathIsomorphicTree` et pour les sous-graphes temporels isomorphes à un motif : `https://github.com/DaemonFire/SequentialSubgraphIsomorphism`).

Cette thèse se concentrant principalement sur l'influence de la taille de l'historique sur la complexité des algorithmes, je confronte mes implémentations à des données générées pour confirmer l'influence de τ sur les complexités de ces algorithmes, telle que théoriquement établie. Je confronte ensuite ces implémentations à des jeux de données réelles, avec deux collections de données issues de précédentes expériences, le jeu de données Enron, introduit par Klimt et Yang [31], qui s'étend sur une longue période temporelle et présente une faible densité d'arêtes temporelles représentant les échanges de courriers électroniques au sein d'une entreprise, et le jeu de données Rollernet présenté par Tournoux et al. [44], représentant la proximité deux à deux de rollerbladers circulant dans Paris, qui est au contraire plus limité dans le temps mais présente une densité d'arêtes temporelles importante, ce qui permettra de tester les limites des présents algorithmes, conçus pour résoudre le problème sur des graphes moins denses. J'utilise également un jeu de données, que j'introduis et livre à disposition, appelé `LesFurets` qui présentera une faible densité, s'étendra sur une durée d'historique plus faible que celle d'Enron mais présentera un grand nombre de sommets, et qui représentera les interactions entre les utilisateurs du site et différents agents du système. Les utilisateurs seront représentés par des sommets, de même que les actions système et une arête lie un sommet représentant un utilisateur à un sommet représentant une action système si l'utilisateur a réalisé l'action système. Les caractéristiques intéressantes de ce dernier jeu de données pour l'étude expérimentale sont le caractère asynchrone de la collecte de données qui entraîne un graphe temporel dont les arêtes ne sont pas triées par instants temporels croissants, et le fait que le graphe considéré est biparti.

Le temps de calcul des algorithmes pour l'énumération des Δ-jumeaux est en moyenne de 12 secondes pour tous les jeux de données sauf un, pour lequel la moyenne est de 70 secondes. L'algorithme pour l'énumération des ModulesEternels a un temps de calcul de même ordre mais la complexité spatiale s'avère trop importante et des problèmes de mémoire surviennent sur des jeux de données pour lesquels l'énumération des Δ-jumeaux est possible. L'algorithme d'énumération des sous-graphes isomorphes présente un temps de calcul moyen de 30 secondes pour des motifs de 2 sommets et de 18 minutes pour des motifs de 3 sommets, un des calculs culminant à un temps total de 4 heures.

Dans le cadre de cette thèse, mon but est d'énumérer les sous-ensembles recherchés en temps raisonnable, que je définis comme un temps de calcul inférieur ou de l'ordre de l'heure. Obtenir un résultat en moins d'une heure pour la détection de comportements dans des graphes temporels qui

représentent l'activité des agents pendant une période de 11 jours et demi dans le cas de l'énumération de sous-graphes temporels isomorphes, et de 3 ans et 2 mois dans le cas de l'énumération de Δ-jumeaux et de modules éternels, permet de réaliser des analyses de comportements sur le long terme dans un délai suffisamment court pour une étude a posteriori de l'historique de l'activité dans le système considéré. Dans le cas spécifique qui motive cette thèse, à savoir l'identification de comportements des utilisateurs d'un site d'e-commerce et la caractérisation de comportements n'emmenant pas à une complétion du parcours de l'utilisateur, une fois l'historique analysé, l'adaptation de ces algorithmes en version "au fil de l'eau" (dite algorithme online) est également souhaitable, pour traiter l'enregistrement de nouvelles données au fur et à mesure qu'elles sont enregistrées.

Liste des publications lors de cette thèse
(Publication dans une conférence [8] : B.M. Bui-Xuan, H. Hourcade, and C. Miachon. Computing temporal twins in time logarithmic in history length. In *9th International Conference on Complex Networks and their Applications*, volume 944 of SCI, pages 651-663, 2020.)
(Publication dans une revue [9] : B.M. Bui-Xuan, H. Hourcade and C. Miachon. Computing small temporal modules in time logarithmic in history length. In *Social Network Analysis and Mining*, volume 12, 2022.)
(Présentation orale dans une conférence : B.M. Bui-Xuan, F. Hamissi, H. Hourcade and C. Miachon. Behaviour Recognition Algorithms In Temporal Graphs. At *Conference on Complex Systems*, Lyon, 2021. Pas d'article publié)

En résumé de cette introduction : je présente des algorithmes énumérant des motifs temporels en cherchant à limiter au maximum l'influence de la taille de l'historique du graphe temporel sur les complexités spatiales et temporelles. Deux problèmes sont étudiés, l'un dans lequel je cherche des occurrences d'un motif connu, l'énumération de sous-graphes temporels isomorphes, l'autre dans lequel je cherche les sommets présentant des similitudes de comportement avec une population connue, l'énumération de Δ-modules.

Plan de la thèse

La thèse est organisée de la manière suivante. Un aperçu de l'état de l'art sur les sujets traités ainsi que les définitions des termes et notations et la formalisation des problèmes étudiés dans cette thèse seront donnés en Chapitre 2. Dans le Chapitre 3, je définirai la notion de splitters et j'étudierai une méthode d'énumération des splitters, nommée Edge Iteration, avant de définir la structure de données des arbres rouge-noir utilisés pour atteindre une complexité temporelle logarithmique en la taille de l'historique pour l'énumération des Δ-Jumeaux et des Δ-Modules. Dans le Chapitre 4, je présente un algorithme pour l'énumération des Sous-forêts temporelles isomorphes à une forêt temporelle linéaire, avant d'étudier les liens entre isomorphisme par suite d'arêtes de sous-graphes temporels et isomorphisme de sous-graphes statiques, puis entre isomorphisme de sous-graphes statiques et recherche de cliques de taille n, ce qui me permettra de concevoir un algorithme énumérant les Sous-graphes temporels isomorphes par suite d'arêtes, et un algorithme énumérant les Sous-graphes temporels isomorphes. L'étude expérimentale des implémentations est présentée dans le Chapitre 5, où je me concentre sur les temps de calcul lors des expérimentations sur les jeux de données traités, avant la conclusion de cette thèse et des remarques sur les perspectives d'ouverture pour de futurs travaux.

Ce plan est résumé sous forme graphique dans la figure 1.1 qui représente également le calendrier de cette thèse et la chronologie des mes publications.

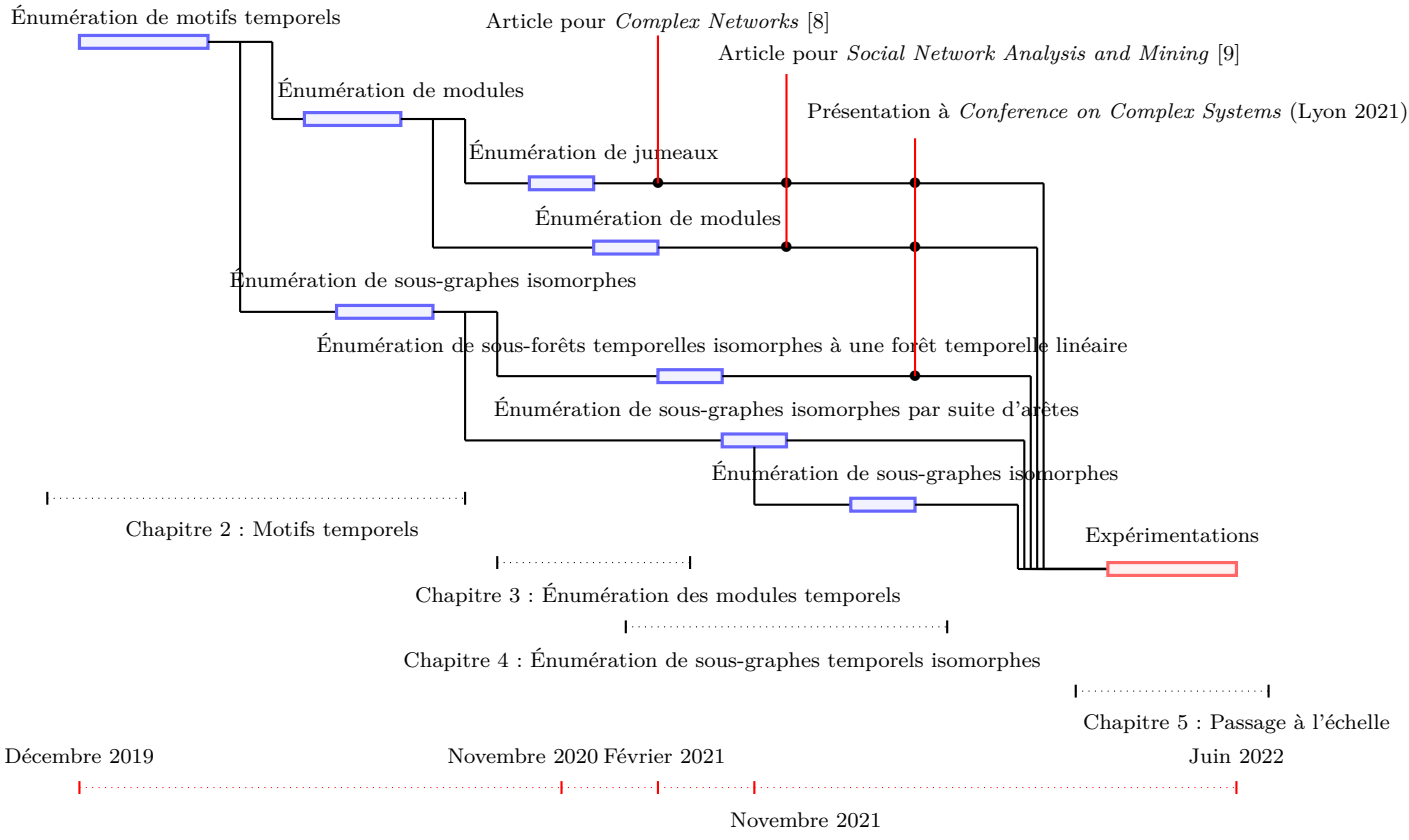

Figure 1.1 – Représentation graphique du plan de thèse avec le calendrier de la réalisation de celle-ci et la chronologie des publications

Chapitre 2

Motifs temporels

Avant de présenter les travaux théoriques et expérimentaux que j'ai réalisés dans le cadre de cette thèse, je vais commencer par introduire le sujet en bonne et due forme. Il est important que dans un premier temps 2.1, je présente l'état de l'art des recherches actuelles sur les problèmes d'énumération des modules temporels et sur les problèmes relatifs à l'isomorphisme de sous-graphes dans les graphes temporels, afin de fournir une base de l'état actuel de la littérature scientifique qui se rapporte à cette thèse. Une fois cet état de l'art établi, je poserai de manière formelle les définitions de tous les termes et notations utilisés dans cette thèse 2.2. Un langage commun ayant été mis en place, je finirai cette partie introductive en définissant formellement les deux problèmes principaux que je vais développer dans cette thèse, ainsi que les cas particuliers de chacun de ces problèmes. Je définirai ainsi le problème d'énumération des Δ-modules 2.3 puis finirai par la définition du problème d'énumération des sous-graphes temporels isomorphes 2.4.

2.1 État de l'art

La littérature contient un certain nombre de formalismes de graphe définissant une notion de temporalité.

L'un de ces formalismes, le graphe évolutif, est décrit par Bui-Xuan et al. [7] comme une paire constituée d'un graphe statique orienté dont les arêtes sont valuées par des ensembles d'instants temporels, et d'une suite de graphes statiques orientés correspondant aux graphes instantanés indexés par des instants temporels.

L'article *Expressivity of Time-Varying Graphs* de Casteigts et al. [13] présente un deuxième formalisme, celui du graphe variant dans le temps, composé d'un graphe statique et de deux fonctions pour chaque arête du graphe, l'une définissant les instants temporels de présence de l'arête et l'autre le temps mis pour parcourir l'arête en fonction de l'instant de début du parcours.

Ces deux premiers formalismes sont très utiles pour traiter les problèmes de recherche de chemin temporel ou de flot temporel mais ne sont pas de très grande utilité dans les problèmes qui m'intéressent dans cette thèse, pour lesquels nous nous intéressons à des connexions instantanées bilatérales entre agents d'un système.

Le graphe temporel est quant à lui défini par Erlebach et al. [17] comme étant une suite de graphes statiques ayant le même ensemble de sommets mais des ensembles d'arêtes qui varient d'une instance à l'autre.

Enfin, l'article *Stream Graphs and Link Streams for the Modeling of Interactions over Time* de Latapy et al. [34] propose une définition du flot de graphes, défini par quatre ensemble, un ensemble de sommets, un ensemble d'instants temporels, un ensemble de sommets temporels et un ensemble d'arêtes temporelles. L'ensemble de sommets temporels, composé de paires regroupant un sommet et un instant temporel, permet de caractériser l'existence à un instant donné d'un sommet. L'ensemble des arêtes temporelles, quant à lui, composé d'un triplet formé de deux sommets et d'un instant temporel, indique des connexions entre des sommets à un instant donné, à condition que les deux sommets existent à cet instant temporel, c'est-à-dire que chacun des deux sommets correspond

à un sommet temporel lorsqu'associé à l'instant temporel considéré. Cet article considère également le cas particulier où tous les sommets existent à chaque instant temporel du graphe, ce qui mène à la définition du flot de lien, caractérisé par un ensemble de sommets, un ensemble d'instants temporels et un ensemble d'arêtes temporelles.

Les formalismes du graphe temporel et du flot de lien permettent de caractériser les problèmes que je cherche à traiter et je me contenterai de ces deux formalismes dans le reste de cette thèse. En effet, ces deux formalismes permettent la modélisation de graphes en prenant en considération des interactions instantanées, ce qui me permettra de modéliser les problèmes de détection de motif.

Ces différents formalismes trouvent des applications dans divers secteurs. En 2018, Dibbelt et al. [15] publient un article à propos du Connection Scan Algorithm. Foschini et al. [19] traitent de la recherche de plus court chemin dans un graphe dépendant du temps, tout comme Dean et al. [14]. Kempe et al. [30] se concentrent sur les problèmes de connectivité et d'inférence dans des graphes temporels. Ros et al. [42] traitent le problème d'ensemble dominant connexe dans des graphes évolutifs. Ces cinq articles se rapportent à l'organisation des horaires de transport.

Les échanges d'e-mail ont également été étudiés, notamment par Klimt et al. [31] où le jeu de données Enron a été rendu accessible au monde de la recherche.

Les interactions de proximité font également partie des cas d'étude, comme dans l'article *The accordion phenomenon : analysis, characterization, and impact on DTN Routing* de Tournoux et al. [44] qui étudie le phénomène accordéon dans un graphe connecté de manière intermittente, à savoir le jeu de données Rollernet que j'utiliserai moi-même dans le cadre de cette thèse.

Les problèmes d'énumération de jumeaux et de sous-graphes isomorphes ont quant à eux été étudiés en détail dans le cas des graphes statiques, comme je vais le présenter dans les paragraphes suivants, même si les variantes temporelles de ces problèmes ne font pas l'objet d'autant d'études. Cependant, plusieurs concepts utilisés pour ces études ne tiennent plus ou doivent être modifiés lorsque l'étude porte sur des graphes temporels, Holme et Saramaki indiquant en 2012 [27] que la modification de la structure du graphe pour prendre en compte la temporalité des interactions entraîne des changements structurels fondamentaux, les arêtes temporelles étant intermittentes par exemple.

Notons que les jumeaux dans un graphe statique représentent le cas de base de la décomposition modulaire, qui, entre autres, aide à réduire les complexités spatiales et temporelles des problèmes de graphe, comme présenté par Ehrenfeucht et al. [16], par Habib et al. [22] et par Spinrad [43] qui, à eux trois, présentent une étude complète de la décomposition modulaire.

L'application la plus notable de la décomposition modulaire est la recherche de sous-graphes libres de toute intersection entre eux, établie par Gallai [21, , traduit par Maffray et Preissmann [36]].

Ces formalismes trouvent également des applications dans le domaine de la biologie, que ce soit dans l'identification d'équipes de gènes par Béal et al. [10], qui se basent sur le formalisme d'affinage de partition d'Hopcroft [28], ou dans l'étude de la conservation des structures combinatoires dans les scénarios de réarrangement de génome menée par Bérard et al. [11], ou encore dans la comparaison de génome, comme le présentent Bergeron et Stoye [4], qui s'appuient sur les travaux de Bergeron et al. de 2005 [3]. Enfin, Gagneur et al. utilisent la décomposition modulaire pour décomposer les interactions protéine-protéine afin d'identifier les facteurs de transcription de tumeurs nécrosées [20].

En l'état actuel des publications, la meilleure performance d'un algorithme au fil de l'eau (dit algorithme online) d'énumération de jumeaux appliqué à un graphe temporel est au pire cas en $O(m + n)$ ou en $O(\tau + n)$, où n est le nombre de sommets et m le nombre d'arêtes du graphe. Cependant, cette complexité est atteinte dans le cas de l'énumération des vrais jumeaux éternels d'un sommet donné, une paire de vrais jumeaux éternels étant une paire de sommets $\{u, v\}$ ayant le même voisinage et tels qu'il n'y ait pas d'arête liant u et v, si bien que ces deux sommets présentent exactement le même voisinage à chaque instant temporel du graphe temporel dans l'ensemble des sommets V, là où les faux jumeaux sont tels qu'il existe une arête entre u et v, ce qui ne permet de garantir l'égalité des voisinages que dans $V \backslash \{u, v\}$. n est en général petit comparé à m ou à τ. La comparaison entre m et τ est en revanche plus compliquée. Je peux considérer que $m \geqslant \tau$ si je supprime tous les instants temporels t où G_t ne présente aucune arête. Néanmoins, si je considère

des Δ-modules avec Δ qui représente une fenêtre temporelle de temps réel, par exemple 51 secondes, ce type de pré-traitement oblige à réaliser de nombreux calculs supplémentaires pour préserver la correspondance entre les intervalles d'instants temporels consécutifs de L et les véritables intervalles de temps réel qu'ils représentent, un 51-module pouvant alors n'être module que pendant un nombre plus faible d'instants temporels, nombre qui ne sera pas uniforme sur T puisque des intervalles de temps réel de même durée pourront être représentés par des intervalles d'instants temporelles de taille différente. Pour éviter de se soucier de ces calculs supplémentaires, je préfère alors me dispenser de ce pré-traitement, ce qui ne permet pas de garantir de relation entre m et τ.

Dans un graphe statique, les approches standards pour trouver des sommets jumeaux incluent la décomposition modulaire, comme présenté par Spinrad [43] dans l'article précédemment cité, et l'affinage de partition, détaillé par Habib et al. dans leur article *Partition refinement techniques : an interesting algorithmic tool kit* [23].

Plus globalement, le problème d'énumération des JumeauxEternels peut être résolu en utilisant les algorithmes de décomposition modulaire, comme présenté par Habib et Paul cité plus haut [22] et dans l'article de Spinrad déjà cité [43] sur chaque graphe G_t pour tout $t \in T$, pour une complexité temporelle en $O(\tau \times n + m)$,

Le problème est bien plus complexe avec les Δ-jumeaux, pour lesquels on ne sait pas quand la fenêtre de Δ instants temporels commence. Pour ce problème, les techniques d'affinage de partition telles que décrites dans l'article de Habib et al. précédemment cité [23] peuvent être utilisées consécutivement Δ fois à chaque graphe instantané G_t pour une complexité temporelle polynomiale en $\tau \times \Delta$. Dans des configurations où la taille de l'historique τ est importante, un nouvel algorithme en temps logarithmique en τ est souhaitable. Notons qu'il est possible que $\tau >> m$ s'il y a de nombreux instants temporels t pour lesquels G_t est un graphe vide, ce qui arrive très fréquemment sur les jeux de données que je considère dans le cadre de cette thèse, les jeux de données étant peu denses, c'est-à-dire que le nombre d'arêtes est bien inférieur à $\binom{n}{2} \times \tau$ qui est le nombre maximal d'arêtes d'un graphe temporel.

Nombre d'études des cas temporels se concentrent sur des fenêtres temporelles de Δ instants consécutifs. Ces études estiment en effet que l'identification d'un comportement sur un intervalle de temps qui ne serait pas borné perd de la cohérence et que pour que l'information soit pertinente et ne relève pas de coïncidences, les interactions constituant le motif recherché doivent se succéder dans un intervalle temporel limité, comme expliqué par Zhao et al. [49]. Les problèmes de fenêtre de Δ instants temporels coulissante ont récemment fait l'objet de nombreux travaux *e.g.* la couverture de sommets dans l'article *Temporal vertex cover with a sliding time window* d'Akrida et al. [1], l'énumération de cliques maximales par Viard et al. [46] ou la recherche de chemin le plus court et le plus rapide par Tsalouchidou [45]. En revanche, les jumeaux temporels n'ont pas été étudiés.

Le problème de l'énumération de sous-graphes statiques isomorphes a également fait l'objet d'un grand nombre d'études. En 1966, Hedetniemi formalise le problème de décision d'isomorphisme de graphes statique, sous le nom d'homomorphisme de graphe [25], qui consiste à établir l'existence ou non d'un isomorphisme de graphes entre deux graphes G et P. Le livre *Graph Theory with application* de Bondy et Murty [5] cite l'article *The graph isomorphism problem is polynomially equivalent to the legitimate deck problem for regular graphs* de Harary et al. [24] qui établit que l'isomorphisme de graphes est polynomialement équivalent au problème du deck légitime dans le cas spécifique des graphes réguliers, confirmant les travaux de Mansfield publiés la même année [37].

Le problème de l'isomorphisme de sous-graphes est NP-complet, comme indiqué dans le *Computers and Intracibility* de Garey et Johnson [29] ou dans le livre de Köbler, Schöning et Toran spécifiquement dédié à ce problème [33]. Mathon [38] propose une preuve de l'équivalence à un facteur polynomial près entre le problème de décision de l'isomorphisme de graphes et le problème de dénombrement associé. Le problème de dénombrement de l'isomorphisme de graphes consiste à établir le nombre d'isomorphismes de graphes existant entre deux graphes G et P. A. Lubiw tente en 1981 d'approcher une preuve de NP-complétude du problème de décision de l'isomorphisme de graphes en notant dans son article *Some NP-complete problems similar to graph isomorphism* [35] les similitudes entre ce problème et divers problèmes NP-complets.

Il apparaît que l'isomorphisme de sous-graphes statiques est NP-complet dans le cas général mais que dans le cas de l'isomorphisme de graphes, pour lequel le motif a la même taille que le graphe considéré, il pourrait être de complexité polynomiale.

Miller soumet des résultats sur le lien entre la complexité du problème de décision de l'isomorphisme de graphes et le degré maximal du graphe [39]. Basin confirme l'existence de cas spécifiques du problème de décision de l'isomorphisme de graphes pour lesquels le problème est de complexité polynomiale en établissant une équivalence du problème avec un problème de décision d'égalité de termes [2]. Booth établit également une équivalence entre le problème de décision de l'isomorphisme de graphes, l'isomorphisme de semi-groupes et l'isomorphisme d'automates finis [6]. Fiala et Paulusma synthétisent tous ces travaux sur la complexité de ce problème, traité dans l'optique de l'étude des relations sociales [18]. White et Reitz [48] utilisent également ces résultats dans ce même contexte. L'ensemble des solutions à des cas spécifiques est répertorié dans le livre *A non-factorial algorithm for the graph isomorphism problem* de Goldberg [26].

Ces travaux sur l'isomorphisme de sous-graphes statiques m'intéressent dans la mesure où la démarche présentée dans cette thèse impliquera l'utilisation de résultats sur l'énumération des sous-graphes statiques isomorphes comme pré-traitement à l'énumération des sous-graphes temporels isomorphes et isomorphes par suite d'arêtes.

Les études de la version temporelle du problème sont en revanche moins nombreuses. Redmond et Cunningham [41] proposent une étude du problème d'isomorphisme par suite d'arêtes de sous-graphes temporels en comparant des adaptations des algorithmes d'Ulmann et de VF2, notamment sur des extraits du jeu de données Enron, pour des motifs P petits (moins de 10 sommets, une dizaine d'arêtes), et cette étude est la seule, à ce jour, à avoir étudié le problème d'énumération de sous-graphes temporels isomorphes en détail. Cependant, l'influence de la taille de l'historique sur le temps de calcul n'y est pas précisée, le focus de ce papier portant plus sur l'influence du nombre de sommets sur le temps de calcul.

2.2 Définition des notations et termes

La cardinalité d'un ensemble X sera notée $\|X\|$. Un **graphe statique** $G(V, E)$ est défini par son ensemble de sommets V et son ensemble d'arêtes E, chaque arête étant une paire de sommets distincts. On dira qu'une arête (u, v) **lie** les sommets u et v. Un **graphe temporel** $L(V, E, T)$ est défini par son ensemble de sommets V, un intervalle $T \subseteq \mathbb{N}$ d'instants temporels et son ensemble d'arêtes temporelles $E \subseteq V^2 \times T$, chaque arête temporelle liant deux sommets à un instant temporel donné.

Un graphe temporel L peut également être défini comme une suite $(L_t)_{t \in T}$ de graphes statiques indexés par des instants temporels appartenant à un intervalle $T \subseteq \mathbb{N}$.

Pour une meilleure compréhension graphique, je représente les graphes temporels en utilisant le formalisme des flots de lien. Un exemple de ce formalisme est présenté en figure 2.1 pour lequel

— $V = \{A, B, C, D\}$.
— $T = [0, 5]$.
— $E = \{(A, B, 0), (A, C, 0), (A, B, 1), (A, C, 1), (A, B, 2), (A, C, 2), (A, B, 4), (A, C, 4), (A, B, 5), (A, C, 5)\}$

Soit $G(V, E)$ un graphe statique. Le **voisinage** $N(u) \subseteq V$ d'un sommet $u \in V$ est l'ensemble des sommets liés à u dans G. Pour $A \subseteq V$, on note $E_A \subseteq E$ l'ensemble des arêtes liant deux sommets de A, et on l'appelle l'ensemble des arêtes de G **restreint** à A. La **matrice d'adjacence** M de G est la matrice 0/1 de taille $\|V\| \times \|V\|$ telle que $M[u][v] = 1$ si $(u, v) \in E$ et 0 si $(u, v) \notin E$. Le **splitter** d'une paire de sommets est un sommet lié à l'un des sommets de la paire, mais pas à l'autre. Deux sommets $\{u, v\} \in V^2$ de G sont **jumeaux** si et seulement si $N(u) = N(v)$. Un sous-ensemble $A \subseteq V$ de sommets de G est un **module** si et seulement si tous les sommets de A ont le même voisinage hors de A. Le **module minimal** contenant $\{u, v\} \in V^2$ est le module $A \subseteq V$ contenant u et v de cardinalité minimale. Un sommet $v \in V$ dont le voisinage est vide est dit **isolé**. Un **parcours de longueur k** d'un sommet x vers un sommet y est un ensemble d'arêtes $v_0v_1, v_1v_2, \ldots, v_{k-1}v_k$ où

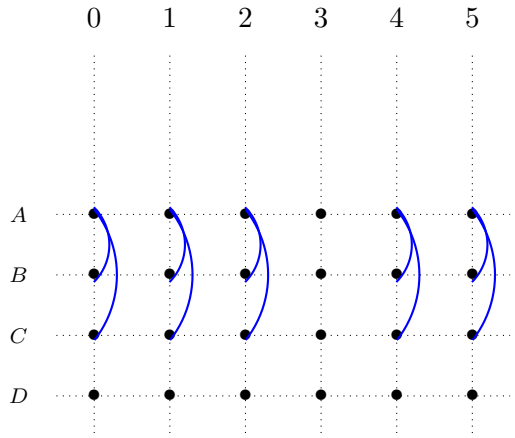

FIGURE 2.1 – Exemple de flot de lien

tous les v_i sont distincts. Une **composante connexe** de G est un sous-ensemble $A \subseteq V$ tel qu'il existe un parcours de tout sommet de A vers tout autre sommet de A mais qu'il n'en existe pas d'un sommet de A vers un sommet extérieur à A. Un **arbre** est un graphe connexe dans lequel il existe un unique parcours d'un sommet appelé **racine** vers tout autre sommet de l'arbre, et qu'il n'existe aucun parcours de tout sommet vers lui-même. Une **forêt** est un graphe statique dont chacune des composantes connexes est un arbre. Une **forêt linéaire** est une forêt dont les sommets ont un voisinage de cardinalité au plus 2. Si une composante connexe A est un chemin, on appelle **longueur du chemin** A la taille du parcours d'une extrêmité du chemin à l'autre. Si une composante connexe A est un arbre, j'appelle **diamètre d'un arbre** A la taille de parcours maximale entre deux sommets de A.

Soit $L(V, E, T)$ un graphe temporel. Pour $t \in T$, on note $E_t \subseteq E$ l'ensemble des arêtes à l'instant t, et on l'appelle l'**ensemble des arêtes instantanées** de L à l'instant t. Pour $t \in T$ et $A \subseteq V$, on note $E_{A_t} \subseteq E$ l'ensemble des arêtes liant des sommets de A à l'instant t dans L, et on l'appelle **l'ensemble des arêtes instantanées restreint à** A **à l'instant** t de L. Le **voisinage instantané** de $u \in V$ à l'instant $t \in T$ est l'ensemble des sommets liés à u à l'instant t dans L. Un **splitter instantané** d'une paire de sommets à l'instant $t \in T$ est un sommet lié à l'un des sommets de la paire à l'instant t, mais pas à l'autre. La **suite de matrices d'adjacences** $(M_t)_{t \in T}$ de L est la suite des matrices d'adjacence des graphes statiques $L_t(V, E_t)$.

Dans le cadre de l'étude des jumeaux et modules dans le cadre des graphes temporels, j'introduis les notions suivantes.

Deux sommets $\{u, v\} \in V^2$ sont **jumeaux éternels** si et seulement si ils sont jumeaux à chaque instant $t \in T$. Un exemple de jumeaux éternels est présenté en figure 2.2. Dans cette figure, la paire de sommets $\{B, C\}$ constitue une paire de jumeaux éternels. Étant donné $\Delta \in \mathbb{N}$, deux sommets $\{u, v\} \in V^2$ de L sont **Δ-jumeaux** si et seulement si il existe un intervalle $I \subseteq T$ de taille au moins Δ tel que u et v sont jumeaux à chaque instant de I dans L. Un exemple de Δ-jumeaux est présenté en figure 2.3. Dans cette figure, la paire de sommets $\{B, C\}$ constitue une paire de Δ-jumeaux pour tout $\Delta \leqslant 3$ entre les instants 1 et 3. Une paire de jumeaux éternels est donc une paire de Δ-jumeaux pour $\Delta = \|T\|$. Un sous-ensemble $A \subseteq V$ est un **module éternel** si et seulement si A est module à chaque instant $t \in T$ dans L. Un exemple de module éternel est présenté en figure 2.4. Dans cette figure, le sous-ensemble de sommets $\{A, B, C\}$ constitue un module éternel. Étant donné $\Delta \in \mathbb{N}$, $A \subseteq V$ est un **Δ-module** si et seulement si il existe un intervalle $I \subseteq T$ tel que A est module pour tout instant de I dans L. Un exemple de Δ-module est présenté en figure 2.5. Dans cette figure, le sous-ensemble de sommets $\{A, B, C\}$ constitue un Δ-module pour tout $\Delta \leqslant 4$ entre les instants 0 et 3. Un module éternel est donc un Δ-module pour $\Delta = \|T\|$.

Un graphe temporel $L(V, E, T)$ est dit **biparti** si chaque sommet peut être coloré d'une couleur parmi deux et qu'il n'existe aucune arête entre deux sommets de même couleur, à aucun instant temporel. Exemple : la figure 2.4 représente un graphe temporel biparti avec $\{A\}$ d'une couleur et

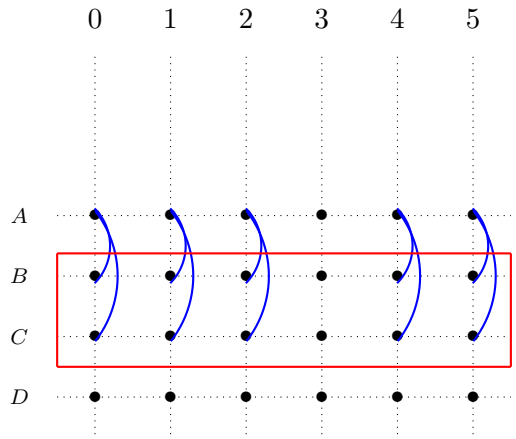

FIGURE 2.2 – Exemple de jumeaux éternels

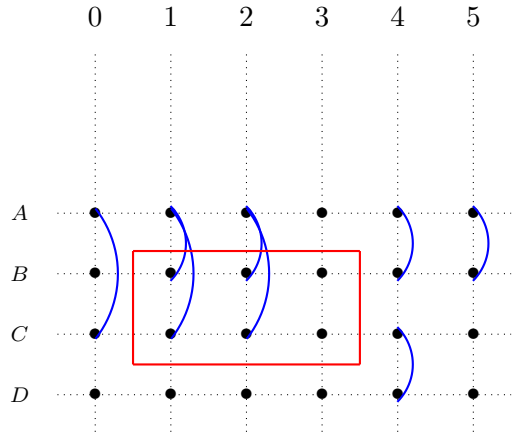

FIGURE 2.3 – Exemple de Δ-jumeaux pour $\Delta \leqslant 3$.

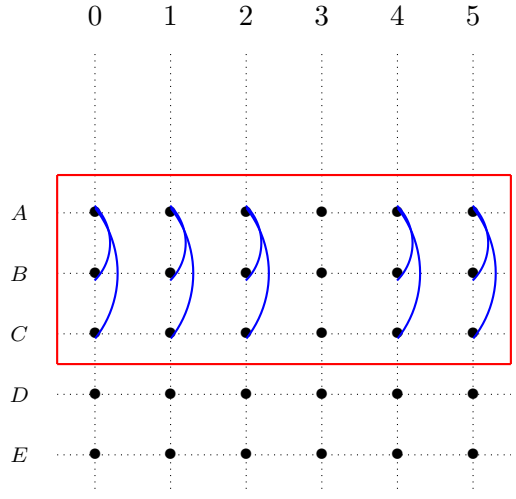

FIGURE 2.4 – Exemple de module éternel.

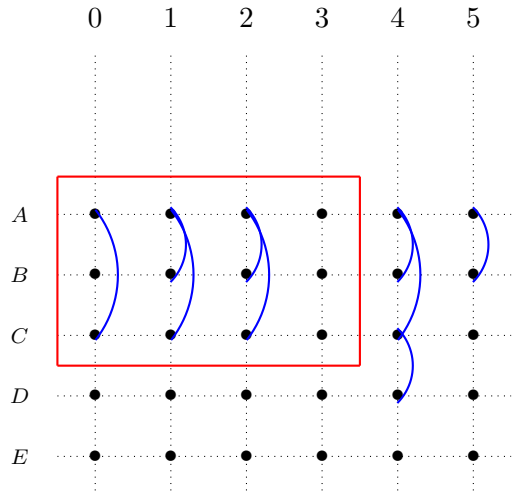

FIGURE 2.5 – Exemple de Δ-module pour $\Delta \geqslant 4$.

$\{B, C\}$ de l'autre.

Le **module éternel minimal** contenant $\{u, v\} \in V^2$ est le module éternel $A \subseteq V$ contenant u et v de cardinalité minimale. Le **module instantané minimal** contenant $\{u, v\} \in V^2$ à l'instant $t \in T$ est le module $A \subseteq V$ à l'instant t, contenant u et v, de cardinalité minimale. Soit $\Delta \in \mathbb{N}$ et $I \subseteq T$ un intervalle de temps de taille Δ. Le **Δ-module minimal** contenant $\{u, v\} \in V^2$ sur l'intervalle I est le Δ-module $A \subseteq V$ sur I contenant u et v de cardinalité minimale. L est une **forêt temporelle linéaire** s'il est une forêt linéaire à tout instant temporel $t \in T$. L est une **forêt temporelle** s'il est une forêt à tout instant temporel $t \in T$.

$f : A \to B$ est une **bijection** si tout élément de A a une image unique dans B et tout élément de B est image d'une unique élément de A. Soit $f : A \to B$ une bijection. Je note $f^{-1} : B \to A$ la **bijection inverse** de f. Soit $f : A \to B$ et $f' : A' \to B'$ deux bijections, avec A et A' deux ensembles disjoints. Je nomme **prolongement de f par f'** la bijection $g : A \cup A' \to B \cup B'$ telle que $g = f$ sur A et $g = f'$ sur A'.

Soit $P(V', E', T')$ un graphe temporel. Un sous-graphe $S(U, E_U, I)$ de L est un sous-graphe **isomorphe** à P à partir de l'instant $t_0 \in T$ si il existe une bijection $f : V' \to U$ telle qu'il existe une arête entre deux sommets x et y dans P à l'instant $t \in T'$ si et seulement si il existe une arête entre $f(x)$ et $f(y)$ dans L à l'instant $t + t_0$. Un exemple de sous-graphe temporel isomorphe est donné en figure 2.6, où

— $U = \{A, B, C, D\}$
— $f(A') = A$, $f(B') = B$, $f(C') = C$ et $f(D') = D$
— $t_0 = 1$;

2.3 Définition du problème d'énumération des Δ-modules

Le problème d'énumération des Δ-modules dans un graphe temporel est défini comme :

Étant donné un graphe temporel $L(V, E, T)$ et un entier naturel $\Delta \in \mathbb{N}$, lister tous les ensembles $[A, t_0, t_1]$ avec $A \subseteq V$ et $[\![t_0, t_1]\!] \subseteq T$ tels que $t_1 - t_0 \geqslant \Delta$ et à chaque instant temporel entre t_0 et t_1, les sommets de A ont le même voisinage hors de A.

Pour le bien de cette thèse, j'ai également travaillé sur des cas particuliers. L'énumération des modules éternels est le cas particulier pour lequel $\Delta = \|T\|$. L'énumération des Δ-jumeaux est le cas particulier pour lequel $\|A\| = 2$. L'énumération des jumeaux éternels est le cas particulier pour lequel $\Delta = \|T\|$ et $\|A\| = 2$.

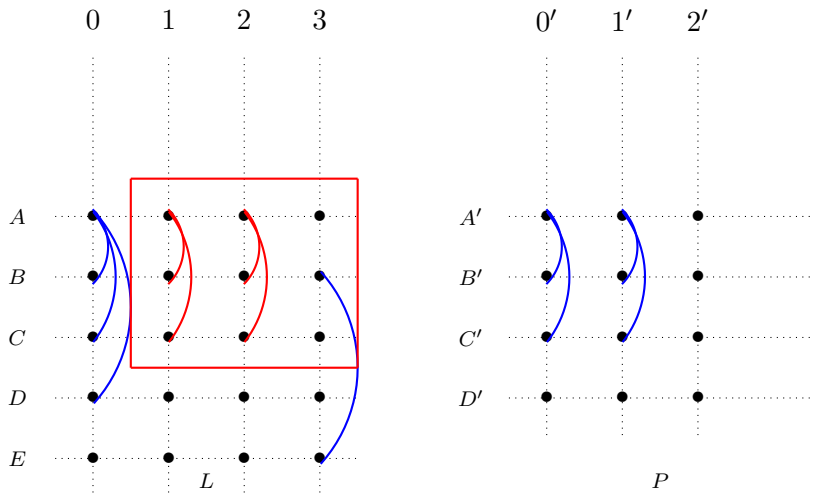

FIGURE 2.6 – Exemple de sous-graphe temporel isomorphe

2.4 Définition du problème d'isomorphisme de sous-graphe temporel

Le problème d'énumération des sous-graphes temporels isomorphes à un motif est défini comme :

Étant donné un graphe temporel $L(V, E, T)$ et un graphe temporel $P(V', E', T')$ désigné comme motif, lister toute paire $\{f, t_0\}$ tel que $t_0 \in T$ et $f : V' \to A \subseteq V$ bijective, tel que pour tout instant temporel $t \in T'$, il existe une arête temporelle entre u et v deux sommets de P à l'instant t si et seulement si il existe une arête entre $f(u)$ et $f(v)$ à l'instant $t_0 + t$ dans L.

Pour le bien de cette thèse, j'ai également travaillé sur un cas particulier. L'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire est un cas particulier où L est une forêt temporelle et P une forêt temporelle linéaire.

Maintenant que j'ai présenté les travaux relatifs aux problèmes qui m'intéressent dans le cadre de l'énumération de motifs temporels, introduit les notations et termes que j'utilise dans cette thèse et défini les problèmes auxquels je cherche à apporter une solution, je définis dans le chapitre 3 des algorithmes capables d'énumérer les Δ-modules et les cas spécifiques de modules temporels tels que définis dans la section 2.3, avant de définir dans le chapire 4 des algorithmes capables d'énumérer les sous-graphes temporels isomorphes et les cas spécifiques associés tels que définis dans la section 2.4.

Chapitre 3

Énumération des modules temporels

Le premier problème que je traite dans cette étude est le problème d'énumération des Δ-modules au sein d'un graphe temporel. Les jeux de données qui m'intéressent présentant des tailles d'historique $\tau = \|T\| = \max\{t : L_t \text{ non vide}\} - \min\{t : L_t \text{ non vide }\}$ importantes (jusqu'à plusieurs dizaines de millions d'instants temporels), je m'intéresserai particulièrement à mettre au point des algorithmes dont les complexités temporelles et spatiales dépendront le moins possible de τ. L'idéal serait d'obtenir des complexités indépendantes de τ ou logarithmiques en τ mais je tolérerai des complexités proportionnelles à τ.

La définition des modules temporels repose sur le concept d'égalité de voisinages. Par conséquent, pour résoudre le problème d'énumération de modules temporels, je dois énumérer les splitters instantanés de toute paire de sommets du graphe donné. Une fois ces splitters instantanés énumérés, ils permettront de déterminer quels voisinages ou quelles restrictions de voisinages sont égaux. La section 3.1 détaillera le travail théorique concernant l'énumération des splitters instantanés de toute paire de sommets de V pour chaque instant temporel de T. Puis, en utilisant ce résultat et l'algorithme que j'aurais décrit pour résoudre ce problème, j'utiliserai ces splitters instantanés en section 3.2 pour énumérer les jumeaux éternels du graphe temporel en disqualifiant toute paire de sommets $\{u, v\} \in V^2$ de la solution s'il existe un instant temporel $t \in T$ tel qu'il existe un splitter instantané entre u et v à l'instant t, puis pour énumérer les Δ-jumeaux en utilisant une structure de données basée sur les arbres rouge-noir pour stocker en mémoire, pour chaque $\{u, v\} \in V^2$, tous les instants temporels pour lesquels il existe un splitter instantané entre u et v. En section 3.3, je détaillerai le travail théorique sur les modules, utilisant les résultats sur l'énumération des splitters pour construire les modules éternels minimaux contenant chaque paire de sommets $\{u, v\} \in V^2$ puis en utilisant des propriétés de construction par fermeture des modules éternels pour construire tous les modules éternels du graphe, avant d'utiliser les résultats sur les splitters et une version modifiée de la structure d'arbre rouge-noir pour bâtir les modules instantanés minimaux contenant chaque paire de sommets $\{u, v\} \in V^2$ avant d'utiliser des propriétés de fermeture pour en déterminer tous les Δ-jumeaux du graphe temporel. Enfin, une fois ce travail théorique présenté, je présenterai dans la section 3.4 une validation expérimentale des complexités temporelles théoriques, et principalement de l'influence de τ sur celles-ci.

3.1 Énumération des splitters : l'algorithme Edge Iteration

Je cherche dans cette section à répondre à un problème sous-jacent, l'énumération de splitters instantanés.

EnumerationDesSplittersInstantanesDeChaquePaireDeSommetsDUnGraphe

Soit un graphe temporel $L(V, E, T)$.

On cherche, pour chaque paire de sommets $\{u, v\} \in V^2, u \neq v$ la liste de tous les splitters instantanés $\{z, t\} \in V \times T$ de $\{u, v\}$.

En effet, dans les sections suivantes, ces listes de splitters me permettront de construire des réponses aux problèmes d'énumération des modules et jumeaux.

Selon la définition du splitter instantané, un sommet $z \in V$ et un instant temporel $t \in T$ constituent un splitter instantané $\{z, t\}$ pour une paire de sommets $\{u, v\} \in V^2$ si et seulement z est dans $N_t(u)$ ou dans $N_t(v)$ mais pas dans les deux. Je sais également que pour que $z \in V$ appartienne à $N_t(u)$ pour $t \in T$ et $u \in V$, il faut et il suffit qu'il existe une arête temporelle $(u, z, t) \in E$. Par conséquent, pour toute arête temporelle $(u, z, t) \in E$, si je trouve un sommet $v \in V$ tel que $(v, z, t) \notin E, \{z, t\}$ est un splitter instantané de $\{u, v\}$.

En utilisant ces propriétés, je peux construire les listes de splitters instantanés pour chaque paire de sommets en réalisant une itération sur la liste d'arêtes temporelles E, et pour chaque $(u, z, t) \in E$, en itérant sur la liste de sommets V pour vérifier, pour chaque sommet $v \in V$ si $(v, z, t) \notin E$. Ce résultat m'intéresse particulièrement car il signifie qu'aucune itération sur T ne sera nécessaire, diminuant l'influence de τ sur la complexité temporelle d'un algorithme utilisant cette méthode d'action par rapport à une méthode qui itèrerait sur T. J'ai baptisé l'algorithme qui utilise cette propriété pour réaliser une énumération exhaustive de tous les splitters instantanés de toutes les paires de sommets du graphe temporel : algorithme Edge Iteration. Il est décrit dans l'algorithme 1.

Data: Graphe temporel $L : (V, E, T)$
Result: Liste des tous les splitters instantanés $\{z, t\}$, avec $z \in V$ et $t \in T$, de chaque paire $\{u, v\} \in V^2$
for *Toute arête temporelle* $(u, z, t) \in E$ **do**
  for *Tout sommet* $v \in V \backslash \{u, z\}$ **do**
    if $(v, z, t) \notin E$ **then**
      Lister $\{z, t\}$ comme splitter instantané de $\{u, v\}$
    end
    if $(u, v, t) \notin E$ **then**
      Lister $\{u, t\}$ comme splitter instantané de $\{v, z\}$
    end
  end
end
return La liste des splitters instantanés.

Algorithm 1: Algorithme Edge Iteration

Je cherche à prouver que l'algorithme Edge Iteration énumère bien tous les splitters instantanés de toute paire de sommets $\{u, v\} \in V^2$ et qu'il n'énumère que des splitters instantanés.

Propriété 3.1.1. *La liste de splitters instantanés fournie comme résultat par l'algorithme Edge Iteration est exhaustive, obtenue en temps indépendant de la taille de l'historique et l'utilisation de mémoire est indépendante de la taille de l'historique.*

Démonstration. Montrons que tout splitter instantané est bien renvoyé par l'algorithme Edge Iteration. Soit $\{c, t\}$ un splitter instantané de a et b. Sans perte de généralité, on peut supposer $(c, a, t) \in E$ et $(c, b, t) \notin E$. Ce splitter est bien renvoyé par l'algorithme pour $c = z$, $u = a$ et $v = b$.

On s'occupe maintenant des complexités temporelles et spatiales de l'algorithme Edge Iteration. La complexité temporelle de cet algorithme est égale au produit de la complexité des boucles imbriquées et de la complexité de la vérification que $(v, z, t) \notin E$. La complexité de l'algorithme Edge Iteration est donc proportionnelle à $O(n \times m \times$Complexité de vérification que $(v, z, t) \notin E)$ avec $n = \|V\|$ et $m = \|E\|$.

La vérification que $(v, z, t) \notin E$ peut être réalisée de deux manières différentes. Si la suite de matrice d'adjacence $(M_t)_{t \in T}$ est donnée en entrée, cette vérification revient à accéder à $M_t[u][w]$ pour vérifier si la valeur contenue dans cette case mémoire est 1 ou 0. Cet accès mémoire se fait en temps constant, si bien que la complexité temporelle totale de la variante de l'algorithme Edge Iteration utilisant les matrices d'adjacence, appelée algorithme Matrix Edge Iteration (MEI), est en $O(n \times m)$. En revanche, le stockage de cette suite de matrices d'adjacence en mémoire entraîne

une complexité spatiale en $O(n^2 \times \tau)$ avec $\tau = \|T\|$. La complexité spatiale de cette version de l'algorithme Edge Iteration est donc proportionnelle à τ. Si la suite de matrices d'adjacence $(M_t)_{t\in T}$ n'est pas donnée en entrée, la vérification peut être effectuée en parcourant E à la recherche de la possible arête (v, z, t). Ce parcours d'E entraîne une complexité temporelle en $O(m)$. La complexité temporelle totale de la variante de l'algorithme Edge Iteration n'utilisant pas de matrice d'adjacence, appelée algorithme Matrix-Less Edge Iteration (MLEI), est donc en $O(n \times m^2)$. Cette version de l'algorithme présente une complexité spatiale en $O(n + m)$, correspondant à la taille du graphe.

En utilisant une HashMap $E(T)$ ayant pour clé les instants temporels $t \in T$ et pour objet E_t l'ensemble des arêtes instantanées de L à l'instant t, l'algorithme MLEI n'est plus obligé de parcourir l'ensemble E pour vérifier si $(z, v, t) \in E$ ou non, mais peut se contenter de parcourir E_t, entraînant une complexité temporelle de MLEI en $O(n \times m \times m_T)$ avec $m_T = max(\|E_t\|)_{t\in T}$.

La complexité spatiale de cette variante de l'algorithme est alors celle de la HashMap $E(t)$ qui est en $O(m)$. □

J'ai ainsi mis au point un algorithme, l'algorithme Edge Iteration, dont j'ai démontré théoriquement la correction et la complétude, capable d'énumérer les splitters instantanés de toute paire de sommets $\{u, v\} \in V^2$. Je m'intéresserai principalement dans la suite de cette thèse à la version Matrix-Less Edge Iteration ou MLEI de cet algorithme, raffinée par l'utilisation d'une $HashMap$ listant pour chaque instant temporel t la liste E_t des arêtes temporelles de E existant à l'instant t. Ainsi défini, l'algorithme MLEI présente une complexité temporelle en $O(n \times m \times m_T)$ et une complexité spatiale en $O(m)$, où n est le nombre de sommets de L, m le nombre d'arêtes temporelles de L et m_T le nombre maximal d'arêtes temporelles de E_t pour tout $t \in T$.

3.2 Énumération des jumeaux temporels

Maintenant que je suis capable d'énumérer tous les splitters instantanés de toutes les paires de sommets d'un graphe temporel, je peux utiliser ces splitters pour énumérer les jumeaux temporels d'un graphe temporel.

Énumération des jumeaux éternels Le problème d'énumération des jumeaux éternels est le cas spécifique du problème d'énumération des Δ-modules le plus simple puisque je me limite à des modules de taille 2 et à $\Delta = \|T\|$.

Si je suis la définition du problème des Δ-modules donnée en section 2.3 avec ces données spécifiques, le problème d'énumération des jumeaux éternels est défini comme :

Étant donné un graphe temporel $L(V, E, T)$, lister toutes les paires $\{u, v\} \in V^2$ telles que u et v ont le même voisinage à chaque instant de T.

Les splitters instantanés vont me permettre d'apporter une réponse à ce problème grâce à la propriété suivante :

Propriété 3.2.1. *Une paire $\{u, v\} \in V^2$ de sommets de L est une paire de jumeaux éternels si et seulement si il n'existe aucun splitter instantané $\{z, t\} \in V\backslash\{u, v\} \times T$ de la paire $\{u, v\}$.*

La démonstration de cette propriété est triviale puisqu'elle découle directement des définitions de splitters et de jumeaux.

Ainsi, en appliquant l'algorithme Edge Iteration qui me permet de lister tous les splitters instantanés de toute paire de sommets $\{u, v\} \in V^2$, je peux supprimer de la liste des jumeaux éternels chaque paire de sommets $\{u, v\}$ pour laquelle je trouve un splitter instantané $\{z, t\} \in V\backslash\{u, v\} \times T$.

Je définis donc une matrice $Candidate$ de taille $n \times n$, telle que $Candidate[u][v] = 1$ si u et v peuvent être jumeaux éternels et $Candidate[u][v] = 0$ si u et v ne peuvent être jumeaux éternels. J'initialise cette matrice tel que $Candidate[u][v] = 1$ pour tout $\{u, v\} \in V^2$. Puis j'applique l'algorithme Edge Iteration et quand un splitter instantané est trouvé pour $\{u, v\} \in V^2$, je passe $Candidate[u][v]$ à 0. Après exécution de l'algorithme Edge Iteration, $\{u, v\} \in V^2$ seront jumeaux éternels si et seulement si $Candidate[u][v] = 1$.

Le problème d'énumération des jumeaux éternels a donc une complexité temporelle en $O(n \times m)$ si j'accepte une complexité spatiale en $O(n^2 \times \tau)$ (version MEI de l'algorithme Edge Iteration) et une complexité temporelle en $O(n \times m \times m_T)$ sinon (version MLEI de l'algorithme Edge Iteration). La matrice *Candidate* entraîne une complexité spatiale en $O(n^2)$.

En utilisant l'algorithme MLEI, je peux donc énumérer tous les jumeaux éternels d'un graphe temporel avec une complexité temporelle en $O(n \times m \times m_T)$ et une complexité spatiale en $O(n^2)$, où n est le nombre de sommets de L, m le nombre d'arêtes temporelles de L et m_T le nombre maximal d'arêtes temporelles de E_t pour tout $t \in T$.

Énumération des Δ-jumeaux : Introduction des arbres de temps discontinu Maintenant que le problème de l'énumération des jumeaux éternels a été résolu, j'élargis un peu le problème en considérant toujours des modules de taille 2 mais en considérant maintenant $\Delta \in \mathbb{N}$ quelconque.

Avec ces nouvelles données, le problème des Δ-jumeaux est défini comme suit :

Étant donné un graphe temporel $L(V, E, T)$ et un entier naturel $\Delta \in \mathbb{N}$, lister tous les quadruplets $[u, v, t_0, t_1]$ avec $\{u, v\} \in V^2$ et $[\![t_0, t_1]\!] \subseteq T$ tels que t_0 et t_1 soient distants de Δ instants et que u et v sont jumeaux pour tous les instants $t \in [\![t_0, t_1]\!]$.

Les splitters instantanés vont me permettre d'apporter une réponse à ce problème grâce à la propriété suivante :

Propriété 3.2.2. *Étant donné $\Delta \in \mathbb{N}$, une paire $\{u, v\} \in V^2$ constitue une paire de Δ-jumeaux sur l'intervalle $[\![t_0, t_1]\!] \subseteq T$ si et seulement si il n'existe aucun splitter instantané $\{z, t\} \in V \backslash \{u, v\} \times [\![t_0, t_1]\!]$ de la paire $\{u, v\}$.*

La démonstration de cette propriété est triviale car découlant directement des définitions de splitters et de Δ-jumeaux.

Du fait de cette propriété, j'utilise l'algorithme Edge Iteration pour lister l'intégralité des splitters instantanés $\{z, t\} \in V \times T$ de chaque paire de sommets $\{u, v\} \in V^2$, ce qui me permettra de certifier qu'un paire de sommets $\{u, v\} \in V^2$ ne peut constituer une paire de Δ-jumeaux sur tout intervalle de temps contenant t.

Mais si je garde en mémoire une liste de tous les splitters instantanés de chaque paire de sommets sous forme de listes, ces listes occasionneront une complexité spatiale en O(nombre de splitters instantanés). Or, dans le pire cas, dans lequel chaque paire de sommets aurait à chaque instant temporel, un nombre de splitters de l'ordre de $\|V\|$, la complexité spatiale serait au pire cas en $O(n^3 \times \tau)$. Il convient donc d'utiliser une structure de données optimisée pour le stockage de ces splitters instantanés.

Afin d'enregistrer les splitters instantanés de manière optimisée en terme d'espace mémoire, j'introduis donc la notion d'arbre de temps discontinu.

Un arbre de temps discontinu $TimeTree(D)$ enregistre des informations concernant l'intervalle $D \subseteq T$ est défini par :

- Un intervalle d'instants temporels $D = [\![t_i, t_f]\!] \subseteq T$.
- Un intervalle d'instants temporels $P \subseteq D$ sur lequel une propriété est vérifiée à chaque instant temporel de P (ici, l'existence d'au moins un splitter instantané).
- Un nœud fils gauche $TimeTree(B)$ avec B l'ensemble des instants de D qui précèdent P.
- Un nœud fils droit $TimeTree(A)$ avec A l'ensemble des instants de D qui succèdent à P.

tels que $D = B \cup P \cup A$ et B, P et A sont des intervalles deux à deux disjoints.

Un exemple d'arbre de temps discontinu est présenté en figure 3.1. Dans cette figure, l'arbre de temps discontinu racine couvre l'intervalle d'instants temporels $D = [\![0, 16]\!]$, notant que la propriété concernée est vérifiée pour l'intervalle $P = [\![6, 9]\!]$ et dispose de deux nœuds fils, l'un couvrant l'intervalle de temps $B = [\![0, 5]\!]$ pour lequel la propriété est vérifiée pour l'instant 3 et l'autre couvrant l'intervalle de temps $A = [\![10, 16]\!]$ pour lequel la propriété est vérifiée sur $[\![12, 13]\!]$. Cet arbre de temps discontinu permet donc de noter que la propriété est vérifiée pour les instants temporels $3, 6, 7, 8, 9, 12, 13$.

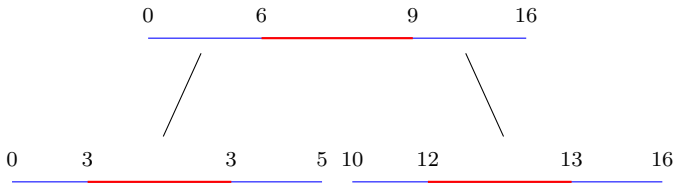

FIGURE 3.1 – Exemple d'arbre de temps discontinu.

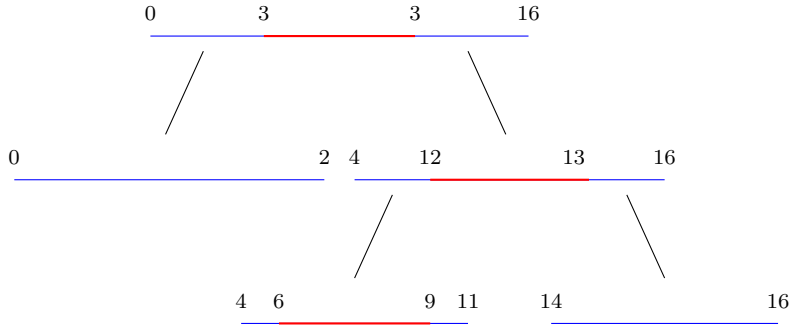

FIGURE 3.2 – Arbre temporel contenant les mêmes informations mais avec une profondeur supérieure d'un niveau.

Il est à noter que ces arbres sont des arbres binaires, c'est-à-dire que chaque nœud a au maximum deux nœuds fils.

Cette structure de données permet de stocker en mémoire un intervalle de temps discontinu avec une complexité spatiale en $O(i)$, où i est le nombre d'intervalles d'instants temporels disjoints pour lesquels la propriété est vérifiée, tout en permettant un parcours optimisé en dichotomie.

Dans l'utilisation présentée ici, à savoir l'énumération des instants temporels $t \in T$ pour lesquels une paire de sommets donnée connaisse au moins un splitter instantané, i est égal au nombre d'intervalles d'instants temporels disjoints pour lesquels il existe un splitter de la paire de sommets. Pour qu'il existe un splitter temporel de cette paire à un instant donné, il faut qu'il existe une arête entre ce splitter et l'un des sommets de la paire à cet instant. Par conséquent, dans ce problème, i est borné par m et par τ. i est donc en toute situation inférieur à $min(m, \tau)$.

En figure 3.2, l'arbre de temps discontinu présente les mêmes informations que celui de la figure 3.1 mais avec une profondeur supérieure d'un niveau. Par conséquent, puisque l'information peut être conservée malgré des rotations de l'arbre, je vais introduire des opérations permettant d'équilibrer un arbre de temps discontinu, de sorte à maintenir la profondeur de ces arbres au minimal. De cette manière, lorsque je souhaiterai parcourir un arbre pour en extraire une information, la complexité temporelle de ces opérations sera plus faible que si je les réalisais sur un arbre déséquilibré.

J'introduis l'opération de rotation gauche de l'arbre de temps discontinu, l'opération de rotation droite étant identique par symétrie :

Je réduis l'intervalle D_{racine} de la racine pour qu'il prenne fin à l'instant précédant le premier instant de son fils droit. J'étends l'intervalle $D_{filsdroit}$ pour qu'il commence au premier instant de la racine. En notant R la racine, S le fils droit de R et U le fils gauche de S, on fait de U le fils droit de R, de R le fils gauche de S et de S la nouvelle racine de l'arbre.

Propriété 3.2.3. *L'opération de rotation gauche d'un arbre de temps discontinu conserve exactement l'information des intervales sur lesquels la proriété étudiée est vérifiée.*

Démonstration. On considère un arbre de temps discontinu $TimeTree(D)$ et l'ensemble de son arborescence, qui contient l'information d'intervales P_i sur lesquels la propriété étudiée est vérifiée.

L'intervale P_{racine} n'est pas altéré par l'opération de rotation gauche puisque lorsqu'on diminue D pour qu'il prenne fin à l'instant précédant A, on ne diminue pas P_{racine}, P_{racine} et A étant, par définition de l'arbre de temps discontinu, disjoints. Hormis cette réduction de D, l'opération de rotation consiste en une permutation de nœuds, qui ne supprime aucun P_i de l'arborescence. □

Je peux donc exécuter des opérations de rotation sans perte d'information. Ceci me permet de m'assurer que les arbres de temps discontinu sont équilibrés à tout moment. Or ces arbres sont binaires, et les arbres binaires équilibrés ont une profondeur bornée par $log(nombre\_noeuds)$. Or, j'ai établi que le nombre de noeuds était inférieur ou égal à $min(m, \tau)$. La profondeur est donc en $O(log(min(m, \tau)))$.

Maintenant que je sais que je peux équilibrer ces arbres, j'introduis les opérations permettant l'insertion d'un instant temporel $t \in T$ pour lequel la propriété est vérifiée dans cet arbre et le maintien de cet arbre.

- ***Initialisation_de_fils*** Si ce nœud ne contient pas d'intervalle P, je rends P égal à $[t, t]$ et créé un nœud gauche couvrant l'intervalle $\{a \in T, a < t\}$ et le nœud droit couvrant l'intervalle $\{a \in T, a > t\}$.
- ***Extension_de_noeud*** Si ce nœud contient un intervalle P, t$\notin P$ et si $\exists t_0 \in P$, tel que $t_0 - 1 = t$ ou $t_0 + 1 = t$, j'ajoute t à P.
- ***Recursion*** Si $t \in B$, B étant l'intervale qui précède P, j'insère l'instant temporel t dans le fils gauche de ce nœud. Respectivement, si $t \in A$, A étant l'intervale qui succède à P, j'insère l'instant temporel t dans le fils droit de ce nœud.
- ***Consolidation*** Une fois l'instant temporel t inséré dans le nœud $TimeTree(D)$, je reparcours les nœuds $(N_n)_{n\in[\![0,d-1]\!]}$ (avec d la profondeur parcourue) déjà parcourus dans l'arborescence de $TimeTree(D)$ pour l'insertion de cet instant temporel. $\forall n \in [\![0, d-1]\!]$, si $P(N_n) \cup P(N_{n+1})$ est connexe, je rends $P(N_n)$ égal à $P(N_n) \cup P(N_{n+1})$ et je supprime le nœud N_{n+1}. Cette opération permet de diminuer la taille de l'arbre pour assurer la profondeur en $O(log(\tau))$.

L'opération d'***Initialisation_de_noeud*** est réalisée en temps constant.

L'opération d'***Extension_de_noeud*** est réalisée en temps constant.

L'opération de ***Recursion*** va parcourir l'arborescence de l'arbre en plongeant jusqu'au nœud auquel l'instant $t \in T$ doit être inséré. Au pire cas, cette opération va donc parcourir l'arbre jusqu'à la feuille la plus profonde de l'arbre. Grâce aux opérations de rotation, j'assure que la profondeur de l'arbre est au pire cas en $O(log(min(m, \tau)))$. À chaque nœud parcouru, je réalise l'opération en temps constant. Par conséquent, l'opération de ***Recursion*** a une complexité en $O(log(min(m, \tau)))$.

Suite à l'insertion d'un instant $t \in T$ dans l'arbre, j'applique l'opération de ***Consolidation*** sur chaque nœud parcouru au cours de la récursion. J'opère donc au pire cas $O(log(min(m, \tau)))$ opérations, chacune de ces opérations étant réalisée en temps constant.

Une optimisation implémentée, dans le cadre de ce problème, est la suivante : Pour tout nœud Q dans l'arborescence d'un arbre de temps discontinu, si l'intervale Q_B précédant l'intervale P_Q contenu dans Q est de taille inférieure à Δ, j'insère tous les instants temporels $t \in Q_B$ dans Q et je supprime le fils gauche de Q. Par symétrie, si l'intervalle Q_A succédant à P_Q est de taille inférieure à Δ, j'insère tous les instants temporels $t \in Q_A$ dans Q et je supprime le fils droit de Q.

En effet, les instants temporels insérés dans un arbre de temps discontinu $TimeTree(D)$ pour une paire de sommets $\{u, v\} \in V^2$ correspondent aux instants temporels $t \in T$ pour lesquels $\{u, v\}$ connait au moins un splitter instantané. Donc tous les intervalles pour lesquels $\{u, v\}$ sont jumeaux correspondent aux instants temporels qui n'ont pas été insérés dans $TimeTree(D)$. Or, pour que $\{u, v\}$ soient Δ-jumeaux sur un intervalle $I \subseteq T$, il est nécessaire que I soit a minima de taille Δ. Donc, si un intervalle de temps Q_B pour lequel aucun instant temporel n'a été inséré est de durée inférieure à Δ, $\{u, v\}$ ne pourront être Δ-jumeaux sur cet intervale et je peux diminuer l'utilisation de mémoire en insérant tous les instants temporels $t \in Q_B$ dans Q (ce qui est fait, en terme d'implémentation, simplement en rendant P_Q égal à $P_Q \cup Q_B$).

La structure des données des arbres de temps discontinu me permet donc de stocker, pour chaque paire de sommets $\{u, v\} \in V^2$ tous les instants temporels $t \in T$ pour lesquels $\exists z \in V, \{z, t\}$ est splitter instantané de $\{u, v\}$ avec une complexité spatiale en $O(n^2 \times m)$.

Afin de créer un algorithme utilisant cette structure de données et l'algorithme Edge Iteration pour énumérer les Δ-jumeaux d'un graphe temporel, j'utilise donc une matrice $Twins$ de taille $n \times n$

qui enregistre, pour chaque paire de sommet $\{u, v\}$, un arbre de temps discontinu $TimeTree(T)$ listant tous les instants temporels pour lesquels $\{u, v\}$ ont au moins un splitter instantané.

En appliquant l'algorithme Edge Iteration, à chaque itération de boucle, si un splitter instantané $\{z, t\} \in V \times T$ existe pour la paire $\{u, v\}$, j'insère l'instant temporel t dans l'arbre de temps discontinu $Twins[u][v]$.

L'algorithme Edge Iteration appliqué au problème de l'énumération des Δ-jumeaux est donc décrit dans l'algorithme 2.

Data: Graphe temporel $L : (V, E, T)$ et $\Delta \in \mathbb{N}$
Result: Tableau des arbres de temps discontinus listant les intervalles discontinus de temps pour lesquels chaque paire de sommets n'est pas une paire de Δ-jumeaux
for *Toute arête temporelle* $(u, z, t) \in E$ **do**
  for *Tout sommet* $v \in V \backslash \{u, z\}$ **do**
    if $(v, z, t) \notin E$ **then**
      Insérer l'instant temporel t dans l'arbre de temps discontinu $Twins[u][v]$
    end
    if $(u, v, t) \notin E$ **then**
      Insérer l'instant temporel t dans l'arbre de temps discontinu $Twins[v][z]$
    end
  end
end
return La matrice $Twins$ des arbres de temps discontinu listant les instants temporels pour lesquels une paire de sommets $\{u, v\}$ compte au moins un splitter.

Algorithm 2: Algorithme Edge Iteration spécifique au problème de l'énumération de Δ-jumeaux

Maintenant que cette structure est remplie, je peux m'en servir pour énumérer la liste des Δ-jumeaux par un simple parcours de chaque arbre en utilisant l'algorithme 3.

Propriété 3.2.4. *L'algorithme Edge Iteration appliqué à l'énumération des Δ-jumeaux présente une complexité temporelle proportionnelle au logarithme de la taille de l'historique et une complexité spatiale indépendante de la taille de l'historique.*

Démonstration. On a déjà établi que l'algorithme Matrix-Less Edge Iteration énumère les splitters instantanés avec une complexité temporelle en $O(n \times m \times m_T)$. L'insertion d'un instant temporel t dans un arbre de temps discontinu $TimeTree(T)$ a une complexité en $log(min(m, \tau))$ donc dans le pire cas en $O(log(\tau))$ puisqu'il s'agit d'appliquer une opération de ***Recursion*** parcourant $O(log(\tau))$ nœuds pour plonger jusqu'à une feuille, et pour chaque nœud, soit de plonger plus profond en temps constant, soit d'appliquer ***Initialisation_de_noeud*** en temps constant, soit de réaliser une ***Extension_de_noeud*** en temps constant, puis de remonter les nœuds parcourus en réalisant une rotation en temps constant afin de rééquilibrer l'arbre de temps discontinu après chaque opération.

La complexité temporelle de l'opération d'extraction des Δ-jumeaux est en $O(n^2 \times nbr(Twins[u][v]))$ où $nbr(Twins[u][v])$ est le nombre de nœuds de $Twins[u][v]$ et comme on a borné le nombre de nœud dans chaque arbre de temps discontinu par $O(min(m, \tau))$, donc par $O(m)$, on a une complexité temporelle de l'opération en $O(n^2 \times m)$.

La complexité temporelle totale de l'énumération des Δ-jumeaux est donc en $O(n \times m \times m_T \times log(\tau) + n^2 \times m)$.

Reste à calculer l'ajout en terme de complexité spatiale de la structure de données permettant le stockage des arbres de temps discontinu permettant le stockage des solutions. Chaque arbre de temps discontinu a une complexité spatiale en $O(m)$. La structure de données pour stocker la solution utilise un arbre de temps discontinu pour chaque paire de sommets donc la complexité spatiale introduite par cette structure et en $O(n^2 \times m)$.

Par conséquent, la complexité spatiale totale est en $O(n^2 \times m)$. □

Data: Un tableau d'arbres de temps discontinu $Twins$
Result: Liste des tous les Δ-jumeaux (u, v, I) avec $\{u, v\} \in V^2$ et $I \subseteq T, \|I\| \geqslant \Delta$
for *Tout sommet* $u \in V$ **do**
  for *Tout sommet* $v \in V \backslash \{u\}$ **do**
    if $Twins[u][v]$ *a un fils gauche* **then**
      On exécute cet algorithme sur ce fils gauche et on ajoute la liste des Δ-jumeaux obtenus à la solution.
    end
    else
      if *durée*$(B) \geqslant \Delta$ **then**
        Ajouter (u, v, B) à la solution
      end
    end
    if $Twins[u][v]$ *a un fils droit* **then**
      On exécute cet algorithme sur ce fils droit et on ajoute la liste des Δ-jumeaux obtenus à la solution.
    end
    else
      if *durée*$(A) \geqslant \Delta$ **then**
        ajouter (u, v, A) à la solution
      end
    end
  end
end
return La liste des Δ-jumeaux.

Algorithm 3: Opération d'extraction des Δ-jumeaux de la structure de données

J'ai donc présenté un algorithme bâti sur l'algorithme MLEI et utilisant la structure de données des arbres de temps discontinu qui permet l'énumération des Δ-jumeaux d'un graphe temporel. Cet algorithme présente une complexité temporelle en $O(n \times m \times m_T \times log(\tau) + n^2 \times m)$ et une complexité spatiale en $O(n^2 \times m)$, où n est le nombre de sommets de L, m le nombre d'arêtes temporelles de L, m_T le nombre d'arêtes instantanées maximal de E_t pour $t \in T$ et τ la taille d'historique de L. J'ai donc réussi à limiter l'influence de τ sur les complexités de cet algorithme puisque la complexité temporelle est logarithmiquement proportionnelle à τ et cette complexité spatiale est indépendante de τ.

3.3 Énumération des modules temporels

Maintenant que j'ai traité le cas spécifique des jumeaux, je m'attaque au problème plus général des modules de taille quelconque. Les splitters seront également utiles pour énumérer les modules temporels car ils vont me permettre de bâtir des modules minimaux contenant chaque paire de sommets $\{u, v\}$. En effet, si $\{u, v\}$ appartiennent à un module $A \subseteq V$, alors A doit également contenir tout splitter instantané de $\{u, v\}$ pour satisfaire la définition de module. Je vais commencer par présenter ces principes dans le cas des graphes statiques.

Quelques notions supplémentaires en graphe statique : le module minimal contenant une paire de sommets Je me propose de calculer le module minimal contenant une paire de sommets donnée à partir des splitters des différentes paires de sommets. Pour ce faire, j'utilise l'algorithme 4.

Maintenant que cet algorithme a été proposé, je cherche à démontrer que la solution de cet algorithme est bien le module minimal contenant la paire de sommets donnée en entrée dans le graphe statique donné en entrée.

Je démontre la propriété suivante :

Data: Graphe statique $G : (V, E)$ et $\{u, v\} \in V^2$
Result: Module minimal contenant $\{u, v\}$
Je pose $Atemp = \{u, v\}$
Je pose $A = \varnothing$
while $A \neq Atemp$ **do**
  Je pose $A = Atemp$
  for *Tout sommet* $z \in V \backslash A$ **do**
    for *Tout sommet* $x \in A$ **do**
      for *Tout sommet* $y \in A$ **do**
        if z *est un splitter de* $\{x, y\}$ **then**
          ajouter z à $Atemp$
        end
      end
    end
  end
end
return A.

Algorithm 4: Calcul de module minimal contenant une paire de sommets

Propriété 3.3.1. *Soit $G(V, E)$ un graphe statique et $A \subseteq V$. A est un module si et seulement si, pour toute paire $\{u, v\} \in A^2$, tous les splitters de $\{u, v\}$ sont inclus dans A.*

Démonstration. Soit $G(V, E)$ un graphe statique et $A \subseteq V$.

($\Leftarrow$) Si A est un module, deux sommets u et v de A ont le même voisinage en dehors de A. Donc u et v ne connaissent pas de splitter hors de A. Tous les splitters de u et v sont donc inclus dans A.

($\Rightarrow$) Pour toute paire $\{u, v\} \in A^2$, si tous les splitters de $\{u, v\}$ sont inclus dans A, u et v ont le même voisinage hors de A. Donc A est un module.

Propriété 3.3.2. *Soit $G(V, E)$ un graphe statique. Le sous-ensemble $A \subseteq V$ construit par l'algorithme 4 de calcul de module minimal contenant une paire de sommets appliqué à la paire de sommets $\{u, v\} \in V^2$ est un module et contient $\{u, v\}$.*

Démonstration. De manière triviale, $\{u, v\} \in A^2$.

Prouvons que A est un module.

L'algorithme prend fin lorsque $A = Atemp$ à l'issue d'une itération de boucle, ce qui se produit lorsqu'aucun nouveau splitter n'a été ajouté pendant l'itération. Cela signifie que pour toute paire $\{a, b\} \in A^2$, aucun splitter n'a été trouvé pour la paire $\{a, b\}$ en dehors de A. Donc par définition, A est un module. □

Je note $MinimalModule(u, v)$ le sous-ensemble $A \subseteq V$ produit par l'algorithme 4 de calcul de module minimal contenant une paire de sommets appliqué à $\{u, v\}$

Propriété 3.3.3. *Soit $G(V, E)$ un graphe statique et $\{u, v\} \in V^2$ une paire de sommets, le module minimal contenant $\{u, v\}$ est unique et est égal à $MinimalModule(u, v)$.*

Démonstration. Soit $G(V, E)$ un graphe statique et $\{u, v\} \in V^2$ une paire de sommets.

On cherche à prouver que pour chaque module B contenant u et v, $\|B\| > \|MinimalModule(u, v)\|$, et pour ce faire, il suffit de montrer $MinimalModule(u, v) \subseteq B$.

Par l'absurde, on suppose qu'il existe un sommet z de $MinimalModule(u, v)$ qui n'appartient pas à B. z a été ajouté lors d'une itération de boucle de l'algorithme 4 car il existait $\{a, b\} \in MinimalModule(u, v)^2$ tel que z est un splitter de $\{a, b\}$. Donc puisque B est un module, si z n'appartient pas à B, l'un des deux sommets $\{a, b\}$ n'appartient pas à B. Le même raisonnement peut être suivi à nouveau pour le sommet de la paire $\{a, b\}$ qui donc n'appartient pas à B.

Or $MinimalModule(u, v)$ a été initialisé comme contenant u et v. Donc, si on répète le raisonnement utilisé pour l'extraction du sommet z de B, on se retrouve à terme à extraire un

splitter de $\{u, v\}$, ce qui nous oblige à extraire u ou v de B pour conserver le caractère de module de B, ce qui contredit mes hypothèses de départ.

Donc si B est un module contenant u et v, $MinimalModule(u, v) \subseteq B$.

□

On a donc établi que le résultat de l'algorithme de calcul de module minimal contenant une paire de sommets est bien le module minimal contenant cette paire de sommets. Maintenant que je suis capable de calculer le module minimal contenant chaque paire de sommets, je vais utiliser ces modules minimaux pour construire par union tous les modules du graphe statique G grâce à la propriété de construction par union suivante.

Des ensembles de sommets $X_0, X_1, \ldots, X_k$ **se chevauchent** si pour tout $I \subsetneq [0, k]$, il existe $j \notin I$ tel que $\cup_{i \in I} X_i$ intersecte X_j.

Propriété 3.3.4. *Dans un graphe statique, un ensemble de sommets est un module si et seulement s'il est une union de modules minimaux qui se chevauchent.*

Démonstration. Soit $G(V, E)$ un graphe statique.

($\Leftarrow$) L'union de modules qui se chevauchent est un module.

($\Rightarrow$) Soit $B \subseteq V$ un module de G. Si B est un singleton, il est bien l'union de modules minimaux qui se chevauchent. Désormais, supposons $\|B\| \geqslant 2$.

Pour tout x et y dans B, distincts, soit M_{xy} le module minimal contenant x et y. Par définition des modules minimaux, $\cup_{\{x,y\} \in B^2} M_{xy} \subseteq B$. Cette inclusion est en fait une égalité, car tout z dans B est dans au moins un M_{xy}. De plus, ces M_{xy} se chevauchent, car si une union X de M_{xy} est différente de B, alors X intersecte M_{uv} pour u dans X et v dans $B \backslash X$. □

Dans la propriété précédente, remarquons que si B n'est pas un singleton, alors tous les modules minimaux qu'il contient sont de taille au moins 2. Ainsi, une fois tous les modules minimaux contenant chaque paire de sommets du graphe statique énumérés, je peux bâtir la liste de tous les modules du graphe statique en construisant itérativement des modules de taille croissante et la propriété de construction de la liste de tous les modules d'un graphe par union de modules minimaux contenant des paires de sommets garantit que tous les modules seront énumérés par cette construction. Maintenant que ces notions ont été introduites dans un cadre statique, je vais m'intéresser à leur application dans un cadre temporel.

Énumération des modules éternels Le problème d'énumération des modules éternels est le cas spécifique du problème d'énumération des Δ-modules où je me limite à $\Delta = \|T\|$.

Si je suis la définition du problème des Δ-modules donnée en section 2.3 avec ces données spécifiques, le problème d'énumération des modules éternels est défini comme :

Étant donné un graphe temporel $L(V, E, T)$, lister tous les sous-ensembles de sommets $A \subseteq V$ tels que tous les sommets de A ont le même voisinage en dehours de A à chaque instant de T.

Ayant introduit les notions de modules minimaux dans le cas du graphe statique, je vais commencer par présenter ces notions dans le cadre des modules éternels afin de proposer un algorithme d'énumération des modules éternels.

Je me propose donc de calculer le module éternel minimal contenant une paire de sommets donnée à partir des splitters instantanés des différentes paires de sommets.

Pour assurer le contrôle dans l'instruction If, j'exécute au préalable l'algorithme Edge Iteration pour lister les splitters instantanés de chaque paire de sommets.

J'utilise cet algorithme pour remplir un tableau $Twins$ de taille $n \times n$ où pour chaque $\{u, v\} \in V^2$, $Twins[u][v]$ contient la liste des sommets z tels qu'il existe un instant temporel t pour lequel z est splitter instantané de $\{u, v\}$. En effet, un tel z doit appartenir à tout module éternel contenant $\{u, v\}$ pour satisfaire la définition de module éternel.

Data: Graphe temporel $L : (V, E, T)$ et $\{u, v\} \in V^2$
Result: Module éternel minimal contenant $\{u, v\}$
Je pose $Atemp = \{u, v\}$
Je pose $A = \varnothing$
while $A \neq Atemp$ **do**
  Je pose $A = Atemp$
  for *Tout sommet* $z \in V \backslash A$ **do**
    for *Tout sommet* $x \in A$ **do**
      for *Tout sommet* $y \in A$ **do**
        if $\exists t \in T, \{z, t\}$ *est un splitter instantané de* $\{x, y\}$ **then**
          ajouter z à $Atemp$
        end
      end
    end
  end
end
return A.

Algorithm 5: Calcul de module éternel minimal contenant une paire de sommets

Grâce à ce tableau, le contrôle de l'instruction If de l'algorithme de calcul de module éternel minimal contenant une paire de sommets 5 revient à vérifier si $z \in Tw[x][y]$, ce qui est réalisé en temps constant.

Je cherche maintenant à démontrer la correction de l'algorithme de calcul du module éternel minimal contenant une paire de sommets présenté dans cette thèse. Je démontre la propriété suivante :

Propriété 3.3.5. *Soit $L(V, E, T)$ un graphe temporel et $A \subseteq V$. A est un module éternel si et seulement si tous les splitters instantannés de toute paire de sommet $\{u, v\} \in A^2$ sont inclus dans A.*

Propriété 3.3.6. *Soit $L(V, E, T)$ un graphe temporel. Le sous-ensemble $A \subseteq V$ construit par l'algorithme 5 de calcul du module éternel minimal contenant une paire de sommets appliqué à la paire de sommets $\{u, v\} \in V^2$ est un module éternel et contient $\{u, v\}$.*

Je note $MinimalEternalModule(u, v)$ le résultat de l'algorithme 5 de calcul du module éternel minimal contenant une paire de sommets appliqué à $\{u, v\}$.

Propriété 3.3.7. *Soit $L(V, E, T)$ un graphe temporel et $\{u, v\} \in V^2$ une paire de sommets, le module éternel minimal contenant $\{u, v\}$ est unique et est égal à $MinimalEternalModule(u, v)$.*

Les démonstrations de ces trois propriétés sont triviales puisqu'il suffit d'appliquer la propriété 3.3.1 d'inclusion de tous les splitters d'une paire de sommets dans tout module statique contenant cette paire, la propriété 3.3.2 de module statique de la solution de l'algorithme de calcul du module statique minimal contenant une paire de sommets, ou la propriété 3.3.3 de minimalité de cette solution, à chaque instant temporel du graphe pour prouver chacune de ces trois propriétés sur le module éternel minimal.

Maintenant que j'ai défini les modules éternels minimaux contenant une paire de sommets et un algorithme permettant des les calculer, je peux utiliser ces modules minimaux pour énumérer de manière exhaustive l'intégralité des modules éternels du graphe temporel $L(V, E, T)$. Je procède en effet de façon semblable à la démarche que j'adopte dans le cas des graphes statiques et je démontre que la liste de tous les modules éternels d'un graphe temporel peut être construite récursivement par union de modules éternels minimaux contenant des paires de sommets.

Propriété 3.3.8. *Soit $L(V, E, T)$ et un ensemble de sommets $B \subseteq V$. B est module éternel si et seulement si il est l'union de modules éternels minimums qui se chevauchent.*

La démonstration de cette propriété est triviale puisqu'il suffit d'appliquer la propriété 3.3 à chaque instant temporel du graphe.

Ainsi, j'ai démontré que tous les modules éternels d'un graphe temporel, à l'exception des singletons contenant chaque sommet $v \in V$, pouvaient être construits itérativement par union de modules minimaux contenant chaque paire de sommets du graphe. Ayant établi qu'une telle construction permettait une énumération exhaustive de tous les modules éternels du graphe temporel, je définis l'algorithme 6 qui utilise ces propriétés pour énumérer tous les modules éternels du graphe temporel.

Propriété 3.3.9. *L'algorithme Edge Iteration appliqué à l'énumération des modules éternels présente des complexités temporelles et spatiales au pire cas indépendantes de la taille de l'historique.*

Démonstration. La première boucle de l'algorithme a une complexité en $O(n)$ où $n = \|V\|$. La deuxième boucle de l'algorithme correspond à Edge Iteration Algorithm et a une complexité temporelle en $O(n \times m \times m_T)$ où $m = \|E\|$ et $m_T = max(\|E_t\|)_{t \in T}$. La troisième boucle itère sur V^2 et pour chaque paire de sommets, la boucle $While$ va itérer jusqu'à ce que $Atemp$ soit un module. Dans le pire cas, le module minimal A contenant une paire de sommets donnée est V et un unique sommet est ajouté à chaque itération de boucle $While$. Donc la boucle $While$ itère au pire cas n fois. A chaque itération de boucle, la double boucle for itère sur A^2. On peut optimiser cette double boucle for en itérant, pour la deuxième boucle, uniquement sur les sommets ajoutés à la $n-1^{ième}$ itération. Dans le pire cas, cette double boucle a une complexité en $O(n)$. La troisième boucle principale a donc une complexité en $O(n^3)$. La dernière boucle $While$ itère tant que l'ensemble $Solution$ n'est pas entièrement construit. A chaque itération, on réalise une double boucle sur l'ensemble temporaire $Temp$. On peut optimiser la deuxième boucle de cette double boucle en n'itérant que sur les modules éternels ajoutés à la $n-1^{ième}$ itération. La vérification $M \cap N \neq \varnothing$ est réalisée en parcourant M et N jusqu'à trouver un sommet présent dans les deux ensembles de taille au pire cas en $O(n)$. La complexité au pire cas de cette vérification est donc en $O(n^2)$, bien que des optimisations comme le tri des ensembles par ordre lexicographique permettent d'accélérer la détection de sommets communs, ce qui permet d'interrompre le calcul plus rapidement pour les paires d'ensembles qui ont effectivement des sommets en commun. On note S la taille de la solution, à savoir le nombre de modules éternels dans L. Dans le pire cas, où chaque itération de boucle ajoute un unique module éternel, la dernière boucle principale de l'algorithme a donc une complexité en $O(n \times S^2)$.

La complexité temporelle totale de l'algorithme est donc en $O(n + n \times m \times m_T + n^3 + n \times S^2)$ soit en $O(n^3 + n \times m \times m_T + n \times S^2)$.

La complexité spatiale de l'algorithme, quant à elle, est entraînée par la structure qui maintient en mémoire, pour chaque paire de sommets, le module éternel minimal contenant cette paire. Dans le pire cas, un module éternel minimal est borné par n, le nombre de sommets de L. Donc au pire cas, la complexité spatiale de cet algorithme est en $O(n^3)$. □

Les complexités de cet algorithme rendent son application sur des graphes contenant un grand nombre de sommets compliquée, puisqu'occasionant un temps de calcul et une utilisation mémoire qui dépendent cubiquement de $\|V\|$. Mais comme cette thèse porte sur des graphes tels que $\|V\|$ est faible, $\|E\|$ est moyen et τ important, je gagne en temps de calcul par rapport aux algorithmes itérant sur l'historique, pour lesquels l'influence de τ sur la complexité temporelle est importante.

Dans un graphe biparti, un ensemble de sommets est dit **monocolore** si tous ses sommets sont de la même couleur. Dans les graphes bipartis, je vais également démontrer que les modules éternels minimaux monocolores contenant une paire de sommets $\{u, v\} \in V^2$ sont soit l'ensemble V soit l'ensemble $\{u, v\}$ si u et v sont jumeaux éternels. De sorte que je puisse utiliser l'algorithme d'énumération des jumeaux éternels présenté dans la section 3.2 pour remplacer la première boucle de l'algorithme 6 d'énumération des modules éternels et ainsi diminuer la complexité temporelle sur cette classe de graphe pour le cas particulier des modules éternels monocolores.

Clairement, l'union de jumeaux éternels de même couleur forme un module éternel monocolore. Il se trouve que la réciproque est vraie.

Data: Graphe temporel $L : (V, E, T)$
Result: Liste de tous les modules éternels
for *Tout sommet* $x \in V$ **do**
  Ajouter $\{x\}$ à *Solution*.
end
for *Toute arête temporelle* $(u, v, t) \in E$ **do**
  for *Tout sommet* $w \in V \backslash \{u, v\}$ **do**
    if $(u, w, t) \notin E$ **then**
      Ajouter u à $Twins[v][w]$
    end
    if $(v, w, t) \notin E$ **then**
      Ajouter v à $Twins[u][w]$
    end
  end
end
for *Sommet* $u \in V$ **do**
  for *Sommet* $v \in V \backslash \{u\}$ **do**
    Je pose $Atemp = \{u, v\}$
    Je pose $A = \varnothing$
    while $A \neq Atemp$ **do**
      Je pose $A = Atemp$
      for *Tout sommet* $x \in A$ **do**
        for *Tout sommet* $y \in A$ **do**
          $Atemp \leftarrow Atemp \cup Tw[x][y]$
        end
      end
    end
    Ajouter A à *Solution*.
  end
end
$Temp = \varnothing$
while $Solution \neq Temp$ **do**
  $Temp \leftarrow Solution$
  for *Module* $M \in Temp$ **do**
    for *Module* $N \in Temp \backslash M$ **do**
      if $M \cap N \neq \varnothing$ **then**
        Ajouter $M \cup N$ à *Solution*.
      end
    end
  end
end
return *Solution*.

Algorithm 6: Algorithme énumérant les modules éternels

Propriété 3.3.10. *Dans un graphe temporel biparti, tout module éternel monocolore est constitué de jumeaux éternels.*

Démonstration. Par définition d'un module, toute paire de sommets d'un module éternel a le même voisinage hors du module à chaque instant. Si de plus le module est monocolore, le voisinage de chacun de ces sommets au sein du module est vide à chaque instant. Les sommets d'un module éternel monocolore sont donc tous des jumeaux éternels. □

Par conséquent, dans un graphe biparti, l'énumération des modules éternels monocolores peut être accomplie par énumération des jumeaux temporels suivie d'une union des modules deux à deux.

La complexité temporelle de l'énumération des modules éternels monocolores dans un graphe biparti en utilisant l'algorithme Edge Iteration est donc en $O(n \times m \times m_t + M^2)$. La complexité spatiale, quant à elle, est limitée par le fait que les modules éternels minimaux sont désormais de taille bornée par 2 si on décide de considérer qu'on marque par une constante arbitraire que le module éternel minimal contenant une paire de sommets est l'ensemble V entier. La complexité spatiale de l'algorithme d'énumération des modules éternels monocolores d'un graphe biparti est donc au pire cas en $O(n^2)$.

Ainsi, en utilisant des propriétés de construction par union des modules éternels et l'algorithme Edge Iteration, j'ai conçu un algorithme permettant l'énumération des jumeaux éternels d'un graphe temporel. Dans le cas général, cet algorithme a une complexité temporelle en $O(n^3 + n \times m \times m_T + n \times S^2)$ et une complexité spatiale en $O(n^3)$, où n est le nombre de sommets de L, m le nombre d'arêtes temporelles de L, m_T le nombre maximal d'arêtes instantanées de L_t pour tout $t \in T$ et S le nombre de modules éternels dans L. Dans le cas des graphes bipartis, en utilisant une propriété de lien entre jumeaux éternels et modules éternels monocolores dans un tel graphe, la version modifiée de l'algorithme d'énumération des jumeaux éternels a une complexité temporelle en $O(n \times m \times m_T + S^2)$ et une complexité spatiale en $O(n^2)$. Si ces complexités sont indépendantes de τ, la dépendance cubique de la complexité temporelle en n dans le cas général rend dans les faits l'application de cet algorithme pour l'énumération de modules éternels dans les jeux de données qui nous intéressent limitée. Cependant, pour une partie de nos jeux de données, je réussis à obtenir une énumération des modules éternels dans un temps raisonnable (de l'ordre de quelques dizaines de minutes), comme je le présente dans la section de validation expérimentale 3.4.

Énumération des Δ-modules Maintenant que j'ai traité tous les cas particuliers intéressants du problème d'énumération des Δ-modules, je m'attaque au problème général. Pour ce faire, je vais de nouveau chercher à prouver que les propriétés sur les modules minimaux contenant une paire de sommets peuvent s'appliquer à ce problème et apporter une définition des Δ-modules minimaux contenant une paire de sommets sur un intervalle de temps. Mais pour en arriver là, il faut déjà pouvoir calculer et stocker en mémoire les modules instantanés minimaux contenant une paire de sommets.

Le module instantané minimal contenant une paire de sommets $\{u, v\} \in V^2$ à l'instant t est le module minimal contenant la paire de sommets $\{u, v\}$ dans le graphe statique L_t.

Je peux donc construire ce module instantané minimal comme un module minimal dans un graphe statique.

Je me propose de calculer le module instantané minimal contenant une paire de sommets donnée à partir des splitters instantanés des différentes paires de sommets.

Pour assurer le contrôle dans l'instruction If de l'algorithme 4 de calcul du module minimal contenant une paire de sommets, j'exécute au préalable l'algorithme Edge Iteration pour lister les splitters instantanés de chaque paire de sommets.

Afin de stocker les splitters instantanés de chaque paire de sommets, j'utilise les arbres de temps discontinus définis dans la section sur les Δ-jumeaux, pour lesquels je définis quelques modifications. En effet, pour l'énumération des Δ-jumeaux, il me suffisait de garder en mémoire que pour un instant temporel, il existait $z \in V$ tel que $\{z, t\}$ était splitter instantané d'une paire de sommets

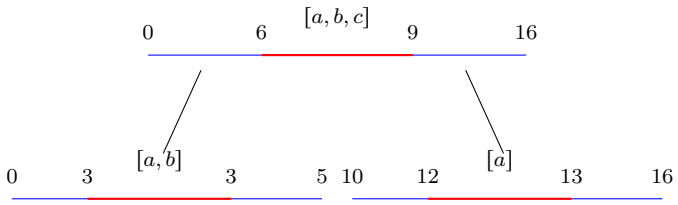

FIGURE 3.3 – Exemple d'arbre de temps discontinu appliqué au problème de l'énumération des Δ-modules.

$\{u, v\} \in V^2$. Mais désormais, je souhaite construire le module instantané minimal contenant une paire de sommets $\{u, v\} \in V^2$ à chaque instant $t \in T$. Il faut donc que je conserve également en mémoire, pour chaque $t \in T$, la liste des $z \in V$ tels que $\{z, t\}$ est un splitter instantané de $\{u, v\}$.

Un arbre de temps discontinu $TimeTree(D)$ avec $D \in T$ appliqué au problème des Δ-modules est défini par :

— Un intervalle d'instants temporels $D = [\![t_i, t_f]\!] \subseteq T$.
— Un intervalle d'instants temporels $P \subseteq D$ sur lequel une propriété est vérifiée (ici, une liste donnée de splitters instantanés).
— Un ensemble de sommets $S \subseteq V$ chaque sommet de S est un splitter instantané de la paire de sommets concernée à chaque instant de P.
— Un nœud fils gauche $TimeTree(B)$ avec $B \subseteq D$ l'intervale de temps qui précède P.
— Un nœud fils droit $TimeTree(A)$ avec $A \subseteq D$ l'intervale de temps qui succède à P.

Chaque nœud de l'arbre de temps discontinu va enregistrer une période connexe de temps pour laquelle la liste de splitters instantanés de la paire de sommets considérée ne change pas.

Un exemple d'arbre de temps discontinu est présenté en figure 3.3.

Dans cette figure, l'arbre de temps discontinu racine couvre l'intervalle d'instants temporels $D = [\![0, 16]\!]$, notant que la liste de sommets $S = \{a, b, c\}$ est la liste des splitters instantanés pour l'intervalle $P = [\![6, 9]\!]$ et dispose de deux nœuds fils, l'un couvrant l'intervalle de temps $B = [\![0, 5]\!]$ pour lequel la liste de sommets $\{a, b\}$ est la liste de splitters instantanés à l'instant 3 et l'autre couvrant l'intervalle de temps $A = [\![10, 16]\!]$ pour lequel la liste de sommets $\{a\}$ est la liste des splitters instantanés sur $[\![12, 13]\!]$. Cet arbre de temps discontinu permet donc de noter que a est splitter instantané pour les instants temporels $3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$ et 16, b est splitter instantané pour les instants temporels $3, 6, 7, 8, 9$ et c est splitter instantané pour les instants $6, 7, 8, 9$.

Cette structure de donnée permet de stocker en mémoire l'intégralité des splitters instantanés d'une paire de sommets donnée avec une complexité spatiale en $O(n \times i)$ où $n = \|V\|$ et i est le nombre d'intervalles d'instants temporels disjoints pour lesquels la liste des splitters instantanés pour la paire de sommets ne change pas, tout en permettant un parcours optimisé en dichotomie.

Pour qu'il existe un splitter temporel $\{z, t\} \in V \times T$ d'une paire de sommets $\{u, v\} \in V^2$, il faut que $(u, z, t) \in E$ ou $(v, z, t) \in E$. Par conséquent, i est en toute situation borné par $min(m, \tau)$.

L'ajout à la structure de données d'un ensemble de sommets correspondant à la liste des splitters instantanés de la paire de sommets étudiée ne modifie pas la possibilité d'équilibrer l'arbre sans perte d'information.

Je conserve donc les opérations de rotation de l'arbre de temps discontinu définies dans la section sur les Δ-jumeaux.

Ceci me permet de m'assurer que les arbres de temps discontinu sont équilibrés à tout moment, assurant une profondeur en $O(log(i))$. Dans le cas présent, la profondeur sera donc toujours bornée par $O(log(min(m, \tau)))$.

Je rappelle les opérations sur les arbres temporels et en introduis des nouvelles pour introduire $\{z, t\} \in V \times T$ un splitter instantané d'une paire de sommets $\{u, v\} \in V^2$. En effet, désormais, même si des splitters instantanés existent pour la paire de sommets concernée sur un intervalle temporel connexe, si la liste exhaustive des splitters instantanés est différente entre deux instants temporels

de cet intervalle, il faut que je puisse conserver cette différence en mémoire. Par conséquent, il faut que je sois capable de casser un nœud pour conserver en mémoire une différence instantanée de liste de splitters instantanés.

- ***Initialisation_de_noeud*** Si $S = \varnothing$, je rends P égal à $[t, t]$, $S = \{z\}$ et créé un nœud gauche couvrant l'intervale précédant P et le nœud droit couvrant l'intervalle succédant à P.
- ***Insertion_de_splitter*** Si $S \neq \varnothing$ et $P = [t, t]$, j'ajoute z à S.
- ***Casse_de_noeud*** Si S n'est pas vide mais ne contient pas z et P contient t mais est de taille supérieure à 1, je créé une nouvelle racine N_0 qui couvrira D, enregistrera un intervale P_{N_0}, limité à t, pour lequel S_{N_0} est égal à $S \cup \{z\}$. Le fils gauche de N_0 est un nœud N_1 couvrant l'intervale $D_{N_1} \subset D$ qui précède t, qui enregistrera l'intervale $P_{N_1} \subset P$, qui est le sous-intervale de P précédant t, pour lequel S_{N_1} est égal à S. Le fils droit de N_0 est un nœud N_2 couvrant l'intervale $D_{N_2} \subset D$ qui succède à t, qui enregistrera l'intervale $P_{N_2} \subset P$, qui est le sous-intervale de P qui succède à t, pour lequel S_{N_2} est égal à S. Le fils gauche de N_1 est le fils gauche du nœud original (resp. le fils droit de N_2 est le fils droit du nœud original).
- ***Extension_de_noeud*** Si $S = \{z\}$ et si $\exists t_0 \in P$, tel que $t_0 - 1 = t$ (resp. $t_0 + 1 = t$), j'ajoute t à P.
- ***Recursion*** Si $t \in B$, j'ajoute le splitter instantané $\{z, t\}$ dans le fils gauche du nœud (respectivement, si $t \in A$, j'ajoute le splitter instantané $\{z, t\}$ dans le fils droit du nœud).

L'opération d'***Initialisation_de_noeud*** est réalisée en temps constant.

L'opération d'***Insertion_de_splitter*** est réalisée en temps constant

L'opération de ***Casse_de_noeud*** est réalisée en temps constant.

L'opération d'***Extension_de_noeud*** est réalisée en temps constant.

L'opération de ***Recursion*** va parcourir l'arborescence de l'arbre en plongeant jusqu'au nœud auquel le splitter instantané $\{z, t\} \in V \times T$ doit être inséré. Au pire cas, cette opération va donc parcourir l'arbre jusqu'à la feuille la plus profonde de l'arbre. Grâce aux opérations de rotation, j'assure que la profondeur de l'arbre est au pire cas en $O(log(min(m, \tau)))$. À chaque nœud parcouru, je réalise l'opération en temps constant. Par conséquent, l'opération de ***Recursion*** a une complexité en $O(log(min(m, \tau)))$.

Mais puisque je suis désormais capable de casser des nœuds, il faut également que je sois capable de consolider deux nœuds pour n'en former qu'un si l'addition d'un splitter instantané $\{z, t\}$ rend égales les deux listes de splitters instantanés enregistrées dans ces deux nœuds. Après addition d'un splitter instantané $\{z, t\} \in V \times T$ dans un nœud N, j'utilise une opération de consolidation qui permet de fusionner deux nœuds N_1 et N_2 si $N_1.S = N_2.S$ et $N_1.P \cup N_2.P$ est connexe. L'opération de consolidation est défini dans l'algorithme 7.

L'opération consiste en un parcours jusqu'à la feuille la plus à droite du sous-arbre dont la racine est le fils gauche de N_1 (que j'enregistre dans N_2) car $N_2.P$ est l'intervalle de temps précédant $N_1.P$ le plus proche. Si $N_1.S = N_2.S$ et $N_1.P \cup N_2.P$ est connexe, j'opère l'opération de consolidation. $N_1.P \leftarrow N_1.P \cup N_2.P$ et je détruis N_2.

J'opère de la même manière en enregistrant dans N_2 la feuille la plus à gauche du sous-arbre dont la racine est le fils droit de N_1.

Une fois le nœud correct trouvé, l'opération de consolidation est réalisée en temps constant. La complexité vient du parcours de l'arbre jusqu'à la feuille N_2, dont la complexité sera en tout cas inférieure à $O(log(min(m, \tau)))$ et dans la vérification de $N_1.S = N_2.S$, qui peut être réalisée au pire cas en $O(n)$ puisqu'au pire cas, $N_1.S$ et $N_2.S$ ont une cardinalité en $O(n)$.

Cette opération de consolidation permet de diminuer la profondeur de l'arbre en s'assurant que le nombre de nœuds soit égal au nombre d'intervalles d'instants temporels pour lesquels la liste de splitters est inchangée.

Équilibrer l'arbre après l'addition d'un splitter et la consolidation d'un arbre requièrent $O(log(min(m, \tau)) + n)$ opérations, ce qui signifie qu'ajouter un splitter dans un arbre équilibré et le rééquilibrer après insertion revient à une complexité totale en $O(log(min(m, \tau)) + n)$.

Data: Un arbre de temps discontinu T
Plonger jusqu'à N_2 le fils le plus à droite de $T.TimeTree(B)$;
if $N_2.t_f + 1 = T.t_i \wedge N_2.S = T.S$ **then**
  $T.t_i \leftarrow N_2.t_i$;
  Détruire N_2;
end
Plonger jusqu'à N_2 le fils le plus à gauche de $T.TimeTree(A)$;
if $N_2.t_i - 1 = T.t_f \wedge N_2.S = T.S$ **then**
  $T.t_f \leftarrow N_2.t_f$;
  Détruire N_2;
end

Algorithm 7: Opération de consolidation

Maintenant que cette version modifiée de l'arbre de temps discontinu a été introduite, je peux l'utiliser pour stocker en mémoire de façon optimisée la liste exhaustive de tous les splitters instantanés de chaque paire de sommets $\{u, v\} \in V^2$ à chaque instant temporel $t \in T$.

J'utilise une matrice $Twins$ de taille $n \times n$ qui enregistre, pour chaque paire de sommet $\{u, v\}$, un arbre de temps discontinu $TimeTree(T)$ listant tous les splitters instantanés $\{z, t\} \in V \times T$ de $\{u, v\}$.

Une fois ces splitters instantanés calculés, pour chaque nœud de chaque arbre de temps discontinu, je peux calculer le module instantané minimal contenant chaque paire de sommets sur chaque intervalle de temps concerné par chaque nœud.

J'appelle $MinimalInstantModule(u, v, t)$ le module minimal instantané contenant u et v à l'instant t.

Ces modules instantanés minimaux permettent de calculer les Δ-modules minimaux contenant des paires de sommets. Afin de le prouver, il faut tout d'abord que je prouve qu'un tel Δ-module minimal englobe nécessairement tous les modules instantanés minimaux contenant toute paire de sommets du Δ-module pour chaque instant temporel de l'intervalle de temps.

Propriété 3.3.11. *Soit $L(V, E, T)$ un graphe temporel, $\{u, v\} \in V^2, \Delta \in \mathbb{N}$ et $D \subseteq T$ un intervalle de temps connexe avec $\|D\| + 1 \geqslant \Delta$. Soit A le Δ-module minimal contenant $\{u, v\}$ sur l'intervalle D. $\forall t \in D$, soit le module instantané minimal $MinimalInstantModule(u, v, t)$ contenant $\{u, v\}$ à l'instant t. Alors $\cup_{t \in D} MinimalInstantModule(u, v, t) \subseteq A$.*

La démonstration de cette propriété est triviale puisqu'un Δ-module A sur un intervale $D \subseteq T$ est module à chaque instant $t \in D$, et inclut donc chaque module minimum instantané contenant chaque paire de sommets $\{u, v\} \in A^2$ à chaque instant t.

Cependant, il n'est pas garanti que $A = \cup_{t \in D} MinimalInstantModule(u, v, t)$. En effet, dans la figure 3.4, à l'instant 0, le module instantané minimal contenant A et B est $\{A, B, D\}$, à l'instant 1, le module instantané minimal contenant A et B est $\{A, B, C\}$. Sur l'intervalle $[\![0, 1]\!]$, $\cup_{t \in [\![0,1]\!]} MinimalInstantModule(A, B, t) = \{A, B, C, D\}$. Pourtant pour $\{E, 0\}$ est splitter instantané de $\{A, B, C, D\}$. Donc $\{A, B, C, D\}$ n'est pas un Δ-module sur $[\![0, 1]\!]$.

Par conséquent, lors de la construction d'un Δ-module minimal contenant une paire de sommets $\{u, v\}$ sur un intervalle $D \subseteq T$ de temps donné, à chaque ajout de sommet w dans l'ensemble solution A, il faut s'assurer que l'ensemble ainsi créé reste module sur tous les instants temporels $t \in D$, c'est à dire qu'il n'existe aucun splitter instantané pour aucune paire de sommets contenant w, à aucun instant temporel de D hors de A.

Je définis donc l'opération décrite dans l'algorithme 8 afin de construire le Δ-module minimal contenant une paire de sommets $\{u, v\} \in V^2$ donnée sur un intervalle de temps $D \subseteq T$ donné avec $\|D\| \geqslant \Delta$.

Propriété 3.3.12. *L'algorithme Edge Iteration appliqué à l'énumération des Δ-modules présente une complexité temporelle au pire cas proportionnelle au logarithme de la taille de l'historique et*

Data: Graphe temporel $L : (T, V, E), \Delta \in \mathbb{N}, D \subseteq T$ avec $\|D\| \geqslant \Delta$ et $\{u, v\} \in V^2$
Result: $\Delta-$module minimal contenant $\{u, v\}$ sur l'intervalle D
Je pose $Atemp = \{u, v\}$
Je pose $A = \varnothing$
while $A \neq Atemp$ **do**
  Je pose $A = Atemp$
  for *Tout sommet* $x \in A$ **do**
    for *Tout sommet* $y \in A$ **do**
      Instant temporel $t = D.t_i$.
      while $t \leqslant D.t_f$ **do**
        if *Il existe un nœud* N *dans* $Twins[x][y]$ *tel que* $N.P$ *contient* t. **then**
          Ajouter $N.S$ à $Atemp$.
          $t \leftarrow N.P.t_f$.
        end
        else
          On incrémente t.
        end
      end
    end
  end
end
$A = Atemp$
return A.

Algorithm 8: Calcul de Δ-module minimal contenant une paire de sommets sur un intervalle de temps donné

une complexité spatiale indépendante de la taille de l'historique.

Démonstration. L'opération de calcul du Δ-module minimal contenant une paire de sommets sur un intervale de temps donné est constituée d'une boucle $While$ qui boucle tant que le Δ-module n'est pas obtenu. Dans le pire cas, où le Δ-module est de taille n, avec un unique sommet ajouté à chaque tour de boucle, j'opère n itérations de boucle. Au sein de cette boucle $While$, on itère sur chaque paire de sommet de A, pour un total au pire cas de $\|A\|^2$ itérations, avec $\|A\| \leqslant n$. On itère ensuite sur une deuxième boucle $While$ dans laquelle on commence par chercher un nœud avec un parcours en profondeur pour une complexité en $O(p)$ où p est la profondeur maximale du nœud, qui est bornée par $log(min(m, \tau))$. Chaque itération de boucle entraîne une incrémentation de t et on parcourt autant de nœuds que nécessaire pour couvrir D. Or $D \subseteq T$ donc le nombre de nœuds parcourus est inférieur au nombre total de nœuds, que l'on a borné par $min(m, \tau)$. On obtient donc une complexité temporelle au pire cas pour cette opération en $O(n^3 \times m \times log(\tau))$, la complexité de l'algorithme d'Edge Iteration devenant négligeable devant la complexité de l'opération de calcul du Δ-module.

En appliquant l'opération définie dans l'algorithme 8 de calcul du Δ-module minimal contenant une paire de sommets sur un intervalle de temps sur les n^2 paires de sommets du graphe temporel, on peut lister tous les Δ-modules minimaux contenant toutes les paires de sommets du graphe sur un intervalle D donné. La complexité temporelle s'élève alors à $O(n^5 \times m \times log(\tau))$.

À partir de cette liste de Δ-modules minimaux sur l'intervalle D, on peut énumérer l'intégralité des Δ-modules sur l'intervalle D en les construisant par unions récursives, comme on a pu le faire pour les modules éternels et les modules sur les graphes statiques. Cette opération entraîne une complexité temporelle supplémentaire en $O(S^2)$ où S est le nombre de Δ-modules existant sur l'intervalle de temps D.

Pour chaque instant temporel $t \in T$, on peut appliquer l'opération pour chaque $\delta \in [\Delta, \tau - t]$, $D = [t, t + \delta]$ afin d'énumérer l'intégralité des Δ-modules dans L.

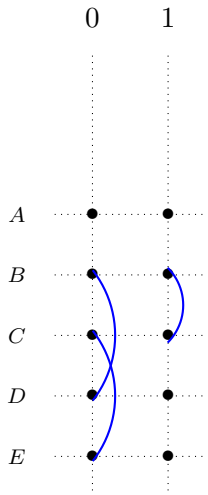

FIGURE 3.4 – Contre-exemple à l'égalité entre unions des modules instantanés minimaux et Δ-module minimal

La complexité temporelle totale est donc en $O(n^5 \times m \times log(\tau) \times \tau^2 + M^2)$ avec M le nombre de Δ-modules dans L.

On peut optimiser l'opération en commençant par traiter $\delta = \tau - t$ et en décrémentant jusqu'à $\delta = \Delta$ de sorte que les Δ-modules qui restent modules sur les plus longues périodes soient ajoutées à la liste de solution en priorité. Ainsi, aux itérations suivantes, on peut interrompre plus tôt les boucles en ne traitant pas les modules qui ont déjà été ajoutés à la solution.

En supposant qu'on réinitialise cette structure de données contenant des arbres de temps discontinu pour chaque paire de sommets du graphe pour chaque intervalle de temps considéré, on obtient une complexité spatiale en $O(n^2 \times$ la complexité spatiale d'un arbre de temps discontinu). La complexité spatiale d'un arbre de temps discontinu est égal au nombre de nœuds (bornée par m) multiplié par la taille maximal du module instantané minimal contenant une paire de sommets à un instant donné, c'est à dire au pire cas n. La complexité spatiale est donc en $O(n^3 \times m)$. □

Cette complexité temporelle trop importante a empêché la mise en application expérimentale de cette opération sur des jeux de données conséquents mais à des fins de progrès théorique, j'ai tout de même conçu les opérations permettant de lister tous les Δ-modules d'un graphe temporel.

Cependant, comme dans le cas des modules éternels, sur des graphes bipartis, je démontre que les Δ-modules monocolores peuvent être construits à partir de la liste des Δ-jumeaux. En utilisant notre algorithme d'énumération des Δ-jumeaux présenté dans la section 3.2, je pourrais ainsi, sur cette classe de graphe, énumérer les Δ-modules monocolores avec une complexité temporelle moindre.

Propriété 3.3.13. *Soit $L(V, E, T)$ un graphe temporel biparti. Soit $\Delta \in \mathbb{N}$. Soit $D \subseteq T$ un intervale de taille au moins Δ. Les paires de sommets de tout Δ-module monocolore sur D sont Δ-jumeaux sur D.*

Démonstration. Par définition d'un module, toute paire de sommets d'un Δ-module sur $D \subseteq T$ a le même voisinage hors du module à chaque instant de D. Si de plus le module est monocolore, le voisinage de chacun de ces sommets au sein du module est vide à chaque instant de D. Les sommets d'un Δ-module monocolore sur D sont donc tous des Δ-jumeaux sur D. □

Par conséquent, dans un graphe biparti, l'énumération des Δ-modules monocolores peut être accomplie par énumération des Δ-jumeaux temporels suivie d'une union des modules deux à deux.

La complexité temporelle de l'énumération des Δ-modules monocolores dans un graphe biparti en utilisant l'algorithme Edge Iteration est donc en $O(n \times m \times m_T \times log(\tau) + M^2)$. La complexité spatiale, quant à elle, est diminuée par le fait que les modules instantanés minimaux contenant une

paire de sommets est bornée par 2 (si on considère qu'on enregistre par une constante arbitraire le fait que le module instantané minimal contenant une paire de sommets est égal à l'ensemble V).

Grâce à une construction par union de Δ-modules minimaux contenant des paires de sommets sur des intervalles de temps, en modifiant les arbres de temps discontinus introduits dans la section 3.2 et en utilisant l'algorithme d'Edge Iteration, j'ai donc mis au point un algorithme permettant l'énumération des Δ-modules d'un graphe temporel. Dans le cas général, cet algorithme a une complexité temporelle en $O(n^5 \times m \times log(\tau) \times \tau^2 + M^2)$ et une complexité spatiale en $O(n^3 \times m)$, où n est le nombre de sommets de L, m le nombre d'arêtes temporelles de L, τ la taille de l'historique de L et M le nombre de Δ-modules de L. Dans le cas particulier des graphes bipartis, en utilisant des propriétés de lien entre Δ-jumeaux et Δ-modules monocolores, j'ai mis au point une variante de l'algorithme présenté plus tôt permettant d'énumérer les Δ-modules monocolores avec une complexité temporelle en $O(n \times m \times m_T \times log(\tau) + M^2)$ et une complexité spatiale en $O(n^2 \times m)$. Ces complexités rendent l'utilisation expérimentale de ces algorithmes compliquées, notamment la complexité spatiale qui risque d'avoir tendance à occasionner des problèmes de manque de mémoire sur les jeux de données traités par cette thèse. Bien qu'intéressant sur un plan théorique, le travail sur le cas général d'énumération des Δ-modules semble donc ne pas pouvoir passer à l'échelle dans une utilisation pratique.

3.4 Validation expérimentale de l'influence de τ

Afin de s'assurer de la validité expérimentale des calculs de complexités théoriques présentés dans ce chapitre, et principalement de l'influence de la taille de l'historique $\tau = \|T\|$ sur celles-ci, j'ai commencé par réaliser quelques expérimentations sur un jeu de données créé de manière artificielle.

Le jeu de données $TimeProgression$ est composé d'une centaine de graphes temporels et a été conçu de sorte que le nombre de sommets et le nombre d'arêtes soient constants entre tous les graphes et que seule varie la taille de l'historique, afin que les temps de calcul et l'utilisation de la mémoire ne varient que sous la seule influence de τ.

Afin d'assurer cette propriété, la démarche pour établir un graphe temporel de ce jeu de données est de créer une arête temporelle entre deux sommets à l'instant 0, une autre à l'instant τ voulu, puis de tirer $m-2$ arêtes (v_j, v_k, t_i) aléatoirement avec $i \in [\![0, \tau]\!], \{j, k\} \in [\![1, n]\!]^2$ choisis de manière uniforme. Dans le cadre de cette étude expérimentale, je fixe $n = 50$ et $m = 10^4$. τ varie de $5 \cdot 10^3$ à $9,95 \cdot 10^5$, avec un incrément, d'une instance à l'autre, de $5 \cdot 10^3$ instants temporels.

Voici les propriétés que je cherche à valider par ces expériences :

- *Propriété 1* : Le temps de calcul pour l'énumération des jumeaux éternels est indépendant de τ.
- *Propriété 2* : L'utilisation de mémoire pour l'énumération des jumeaux éternels est indépendante de τ.
- *Propriété 3* : Le temps de calcul pour l'énumération des Δ-jumeaux est proportionnel à $log(\tau)$.
- *Propriété 4* : L'utilisation de mémoire pour l'énumération des Δ-jumeaux est indépendante de τ.
- *Propriété 5* : Le temps de calcul pour l'énumération des Δ-modules est quadratique en τ.
- *Propriété 6* : L'utilisation de mémoire pour l'énumération des Δ-modules est indépendante de τ.

Les expériences ont été réalisées sur une implémentation en Java sur un ordinateur portable standard tournant à $2,7$ GHz. Je fixe arbitrairement pour ces expériences $\Delta = 102$. Du fait des limites théoriquement établies des version MEI des algorithmes présentés dans cette thèse, qui présentent une forte complexité spatiale induite par le stockage en mémoire de la suite de matrices d'adjacence du graphe temporel, je concentre mes expériences sur les variantes MLEI, qui n'utilisent pas ces suites de matrices d'adjacence. Afin d'établir le temps de calcul, je me base sur l'horloge du système, en réalisant la différence entre l'instant de fin de calcul et l'instant de début de calcul. Pour

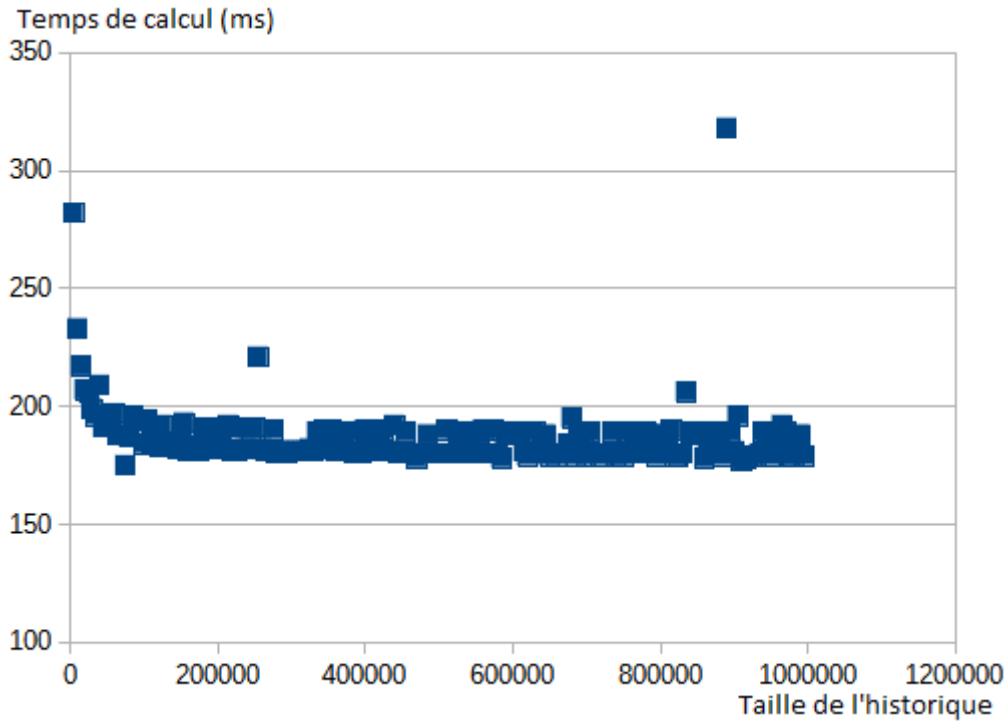

FIGURE 3.5 – Temps de calcul pour énumération des JUMEAUX ÉTERNELS en fonction de la taille de l'historique sur le jeu de données `Timeprogression`.

établir l'utilisation mémoire, j'utilise l'outil Java VisualVM qui permet de visualiser l'utilisation mémoire en cours d'exécution. Pour tout autre détail sur l'implémentation, se référer à la section 5.

Propriété 1 (Validation de l'influence de τ sur la complexité temporelle de l'énumération de jumeaux éternels telle qu'établie théoriquement) À propos de la propriété 1, de validation de l'indépendance en τ du temps de calcul pour l'énumération des jumeaux éternels, j'analyse les résultats présentés en figure 3.5 et je constate que le temps de calcul pour l'énumération des jumeaux éternels reste constant avec l'augmentation de la taille de l'historique à quelques exceptions près. Pour les points isolés présentant des temps de calcul très différents, ces disparités peuvent s'expliquer par une utilisation du processeur par d'autres applications lors du calcul. En ce qui concerne les points correspondant aux graphes dont la taille de l'historique est faible, le temps de calcul plus important peut s'expliquer par le fait que le graphe est plus dense, rendant le parcours pour vérifier l'existence d'une arête donnée à un instant donné plus long.

La propriété 1 d'indépendance en τ de la complexité temporelle de l'énumération de jumeaux éternels est donc validée par l'expérience sur `Timeprogression`.

Propriété 2 (Validation de l'influence de τ sur la complexité spatiale de l'énumération de jumeaux éternels telle qu'établie théoriquement) Pour évaluer la validité de la propriété 2, de validation de l'indépendance en τ de l'utilisation mémoire pour l'énumération des jumeaux éternels, j'analyse les résultats présentés en figure 3.6. Je constate que si l'utilisation mémoire varie entre 30 et 70 méga-octets sur le jeu de données `Timeprogression`, cette variation ne semble pas liée de manière évidente à l'évolution de τ puisqu'on retrouve des graphes de τ élevé sur lesquels la mémoire utilisée est faible et des graphes de τ faible pour lesquels la mémoire utilisée est importante. Par conséquent, τ ne semble pas avoir d'impact sur l'utilisation mémoire de l'algorithme MLEI pour l'énumération des jumeaux éternels.

Propriété 3 (Validation de l'influence de τ sur la complexité temporelle de l'énumération de Δ-jumeaux telle qu'établie théoriquement) À propos de la propriété 3, j'analyse les résultats présentés en figure 3.7 et je constate que le temps de calcul de l'énumération des Δ-jumeaux présenté en fonction du nombre d'instants temporels (τ) décrit une progression logarithmique, validant la

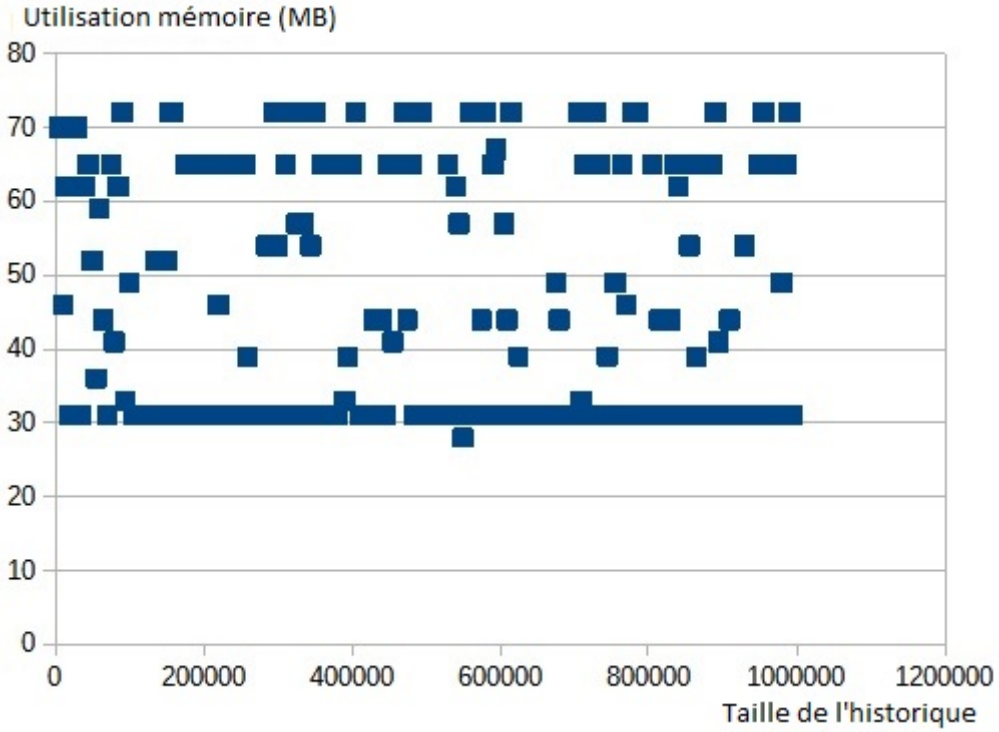

FIGURE 3.6 – Utilisation mémoire pour énumération des JUMEAUX ÉTERNELS en fonction de la taille de l'historique sur le jeu de données `Timeprogression`.

propriété 3 de dépendance logarithmique en τ de la complexité temporelle de l'énumération de Δ-jumeaux.

La propriété 3 de proportionnalité en $O(log(\tau))$ du temps de calcul de l'énumération de Δ-jumeaux est donc validée par l'expérience sur `Timeprogression`.

Propriété 4 (Validation de l'influence de τ sur la complexité spatiale de l'énumération de Δ-jumeaux telle qu'établie théoriquement) À propos de la propriété 4, j'analyse les résultats présentés en figure 3.8 qui représente l'utilisation mémoire par l'algorithme MLEI pour l'énumération des Δ-jumeaux en fonction de la taille de l'historique sur le jeu de données `Timeprogression`. Je constate sur cette figure une forte variance de l'utilisation mémoire sur ce jeu de données, qui peut s'expliquer par la disparité des tailles des arbres de temps discontinu. Si les arêtes temporelles tirées aléatoirement lors de la construction d'une instance sont telles que nombre de splitters instantanés existent pour nombre de paires de sommets mais que, pour chaque paire individuelle de sommets, les divers splitters instantanés ne sont pas adjacents deux à deux, l'arbre de temps discontinu pour cette paire sera en effet composé de nombreux nœuds et l'utilisation mémoire va se révéler importante. En revanche, je constate que malgré l'augmentation de τ, l'utilisation mémoire continue à varier entre un peu plus de $2,5$ giga-octets et un peu plus de 7 giga-octets, sans évolution notable due à l'évolution de τ. Par conséquent, bien que l'utilisation mémoire présente une forte variance, le paramètre τ ne semble pas influer sur celle-ci.

La propriété 4 d'indépendance en τ de l'utilisation mémoire de l'énumération de Δ-jumeaux est donc validée par l'expérience sur `Timeprogression`.

La propriété 2 d'indépendance en τ de l'utilisation mémoire de l'énumération de jumeaux éternels est donc validée par l'expérience sur `Timeprogression`.

Propriété 5 (Validation de l'influence de τ sur la complexité temporelle de l'énumération de Δ-modules telle qu'établie théoriquement) À propos de la propriété 5, la structure de données des arbres de temps discontinu utilisée pour lister les modules instantanés minimaux contenant chaque paire de sommets présente une complexité spatiale trop importante et provoque des erreurs de défaut de mémoire vive lors de l'énumération des Δ-modules sur le jeu de données `Timeprogression`. Il

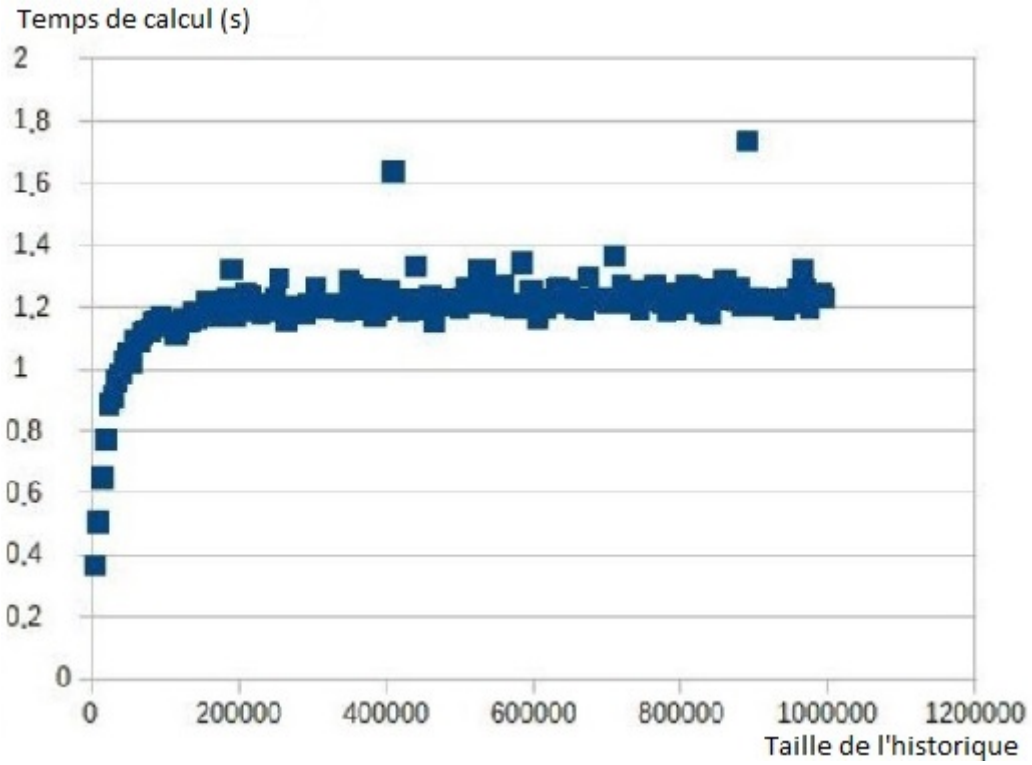

FIGURE 3.7 – Temps de calcul pour énumération des Δ-JUMEAUX en fonction de la taille de l'historique sur le jeu de données `Timeprogression`.

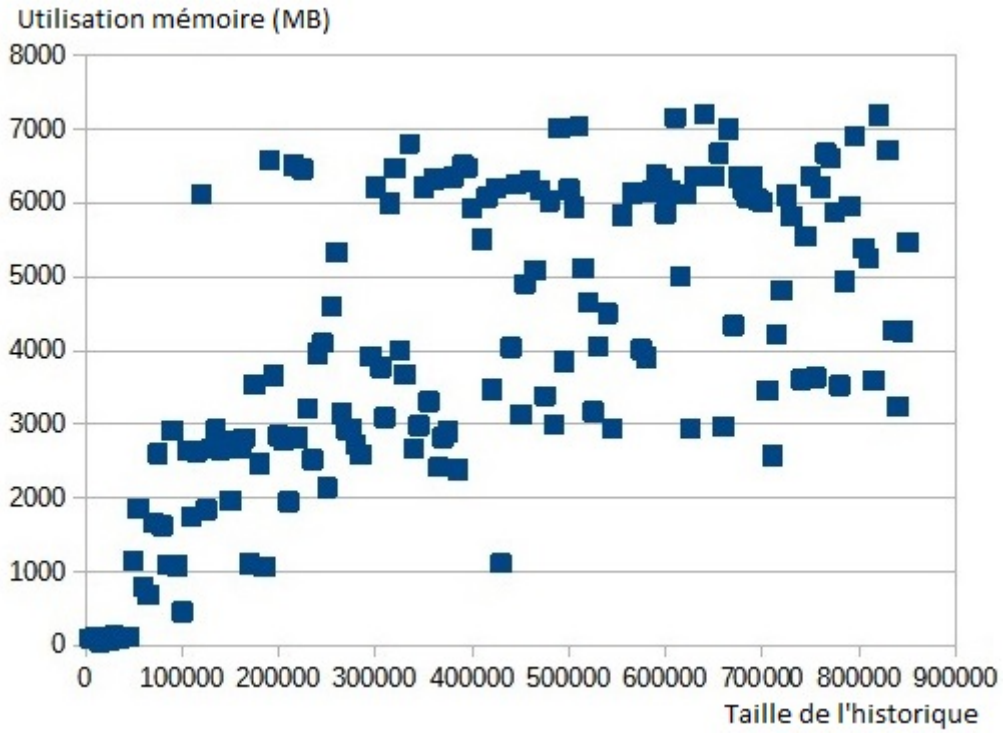

FIGURE 3.8 – Utilisation mémoire pour énumération des Δ-JUMEAUX en fonction de la taille de l'historique sur le jeu de données `Timeprogression`.

n'a donc pas été possible de réaliser suffisamment d'expériences pour obtenir une courbe pertinente de l'évolution du temps de calcul de cet algorithme en fonction de la taille de l'historique. Impossible donc de confirmer la propriété 7 de dépendance quadratique en τ de la complexité temporelle de l'énumération de Δ-modules. Cette expérience met néanmoins en lumière le fait que cet algorithme n'est pas utilisable pour une application pratique. L'algorithme pourrait être utilisé sur des graphes plus petits mais pour les graphes qui m'intéressent dans cette thèse, cet algorithme ne permettra pas d'énumérer les Δ-modules.

La propriété 5 de proportionnalité quadratique en τ du temps de calcul de l'énumération de Δ-modules n'est donc pas validée par l'expérience sur `Timeprogression`.

Propriété 6 (Validation de l'influence de τ sur la complexité spatiale de l'énumération de Δ-modules telle qu'établie théoriquement) À propos de la propriété 6, comme indiqué dans le paragraphe précédent, la structure de données des arbres de temps discontinu utilisée pour lister les modules instantanés minimaux contenant chaque paire de sommets présente une complexité spatiale trop importante. Cette sur-utilisation de la mémoire vive m'empêche d'obtenir suffisamment de résultats sur le jeu de données `Timeprogression` pour établir une courbe expérimentale exploitable de l'utilisation mémoire lors de l'énumération des Δ-modules. Il m'est donc impossible en l'état de confirmer ou d'infirmer l'influence de τ sur cette utilisation mémoire, du fait de problème de manque de mémoire vive, la taille trop importante de cette structure pouvant être expliquée par l'influence trop importante du nombre de sommets et du nombre d'arêtes temporelles du graphe sur la complexité spatiale de cet algorithme.

La propriété 6 d'indépendance en τ de l'utilisation mémoire de l'énumération de Δ-modules n'est donc pas validée par l'expérience sur `Timeprogression`.

En conclusion de cette section de validation théorique,

- Les propriétés de dépendance quadratique en τ de la complexité temporelle de l'énumération de Δ-modules et de l'indépendance en τ de la complexité spatiale de ce même algorithme n'ont pas pu être validées car la complexité spatiale trop importante a entraîné des défauts de mémoire, ne permettant pas aux calculs d'aboutir, sur aucun des jeux de données utilisés dans cette première étude expérimentale.
- Les propriétés d'indépendance en τ des complexités spatiale et temporelle de l'énumération de jumeaux éternels sont validées expérimentalement.
- Les propriétés de dépendance logarithmique en τ de la complexité temporelle de l'énumération de Δ-jumeaux et d'indépendance en τ de la complexité spatiale de ce même algorithme sont expérimentalement validées. Quand à l'influence linéaire de τ sur la complexité spatiale, elle reste dans les limites acceptables définies dans le cadre de cette thèse

L'application de l'algorithme MLEI à l'énumération de Δ-modules semble donc ne pas pouvoir être mise en pratique sur des jeux de données issus d'exploitation réelle. En revanche, les algorithmes d'énumération de jumeaux éternels et de Δ-jumeaux satisfont à toutes les spécifications fixées dans le cadre de cette thèse puisqu'ils parviennent à énumérer les jumeaux en temps, au pire cas, logarithmique en τ pour une utilisation de mémoire indépendante de τ. Le passage à l'échelle pour ces algorithmes devrait donc être réalisable. De plus, il est à noter que sur des graphes bipartis, ces algorithmes d'énumération de jumeaux permettent d'énumérer les Δ-modules et modules éternels monocolores en temps dépendant au pire cas logarithmiquement de τ, ce qui me permet d'énumérer les modules en temps raisonnable sur cette classe de graphe.

Chapitre 4

Énumération de sous-graphes temporels isomorphes

La recherche de motifs dans un graphe est intimement liée au problème d'isomorphisme de sous-graphes. Le problème de l'isomorphisme de sous-graphes statiques est NP-complet, comme l'établit Wegener en 2005 [47]. Passer de ce problème au problème d'énumération de sous-graphes temporels isomorphes entraîne l'ajout d'une dimension supplémentaire, la dimension temporelle, ce qui ne fait qu'augmenter la complexité. Les jeux de données qui m'intéressent dans le cadre de cette thèse présentant de grandes tailles d'historique τ, je cherche à diminuer la dépendance en τ des complexités spatiales et temporelles des algorithmes que j'élabore. L'objectif recherché est une indépendance des complexités des algorithmes en τ ou une dépendance logarithmique en τ. Si des dépendances linéaires en τ sont établies pour les algorithmes que je présente, ces algorithmes seront néanmoins jugés satisfaisants.

Dans un premier temps, j'ai commencé par étudier un cas particulier du problème, en vue de diminuer la complexité. Le problème d'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire est un cas particulier du problème d'énumération des sous-graphes temporels isomorphes pour lequel le graphe temporel considéré est une forêt temporelle et le motif traité est une forêt temporelle linéaire. L'étude théorique de ce cas particulier sera l'objet de la section 4.1. Une fois ce cas particulier traité, je m'attaquerai au cas plus général. Mais pour énumérer les sous-graphes temporels isomorphes, je commencerai par utiliser les travaux de Paranjape et al. [40], de Kozen [32] et de Carraghan et Pardalos [12]. En combinant les travaux présentés dans ces trois articles, je définirai une solution au problème de l'énumération de sous-graphes temporels isomorphes par suite d'arêtes 4.2. En effet, à partir de cette solution, je pourrai ensuite présenter des modifications et démontrer théoriquement leur correction afin de mettre au point un algorithme énumérant les sous-graphes temporels isomorphes. La validation expérimentale de l'influence de la taille de l'historique τ sur les complexités spatiales et temporelles des algorithmes présentés dans ce chapitre sera réalisée dans le chapitre de passage à l'échelle 5.

4.1 Énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire

Dans ce cas particulier du problème de l'énumération de sous-graphes temporels isomorphes, je me limite au cas où le graphe temporel $L(V, E, T)$ est une forêt temporelle et le motif recherché $P(V', E', T')$ est une forêt temporelle linéaire.

Dans le cas d'un graphe s'étendant sur un long historique mais peu dense, il n'est pas déraisonnable de s'attendre à ce que les graphes instantanés présentent peu d'arêtes, diminuant la probabilité d'apparition de cycles dans les graphes instantanés L_t, si bien que pour chaque $t \in T$, on peut émettre l'hypothèse que L_t est une forêt de petite taille.

Dans ce contexte, je décide de résoudre le problème d'énumération de sous-graphes temporels

isomorphes en énumérant les sous-graphes statiques isomorphes pour chaque graphe instantané. A partir de ces isomorphismes, il convient ensuite de calculer quels isomorphismes se maintiennent du graphe instantané P_t vers L_{t_0+t} pour $t \in [\![0, \|T'\|]\!]$. Un tel isomorphisme constituerait alors par définition un isomorphisme de sous-graphes temporels et serait ajouté à la solution de l'algorithme.

Je crée une structure de données sous forme de tableau à double entrée, tel que pour chaque valeur $t \in T$ et $t' \in T'$, le tableau listera toutes les bijections possibles entre $(L_n)_{n\in[\![t-t',t]\!]}$ et $(P_m)_{m\in[\![0,t']\!]}$.

Pour chaque $t \in T$, pour chaque $t' \in T'$, je calcule tous les isomorphismes statique de $P_{t'}$ vers L_t. Ceci est réalisé de la manière suivante, en itérant sur les composantes connexes A de L_t :

— Si A est un arbre, j'itère sur les composantes connexes B de $P_{t'}$. Par définition de la forêt temporelle linéaire, ces B sont des chemins. Je vérifie alors que $longueur(B) \leqslant diametre(A)$. Si cette condition est vérifiée, j'itère sur $v \in A$, et pour chaque sommet de la composante, je réalise un parcours Breadth First Search pour énumérer tous les chemins C de taille $longueur(B)$ ayant v comme extrémité. Pour chaque chemin C, je crée une bijection $f : B -> C$.

— Une fois la liste des isomorphismes partiels Sol_{A_k} calculés pour chaque composante annexe A_k, j'itère sur celles-ci et $\forall\{f_1, f_2, ...f_k\} \in Sol_{A_1} \times Sol_{A_2} ... \times Sol_{A_k}$, je vérifie que les différents f_k sont définis sur des ensembles de sommets $B_k \subseteq V'$ deux à deux disjoints, et si c'est le cas, j'ajoute h le prolongement de f_1 par f_2, ... et par f_k à la liste d'isomorphismes statiques de L_t vers $P_{t'}$.

Une fois les bijections entre L_t et $P_{t'}$ calculés, j'enregistre dans le tableau à double entrée, dans la case correspondant à t et t', l'intersection de cette liste et de la liste d'isomorphismes enregistrée dans la case de la structure de données correspondant à $t-1$ et $t'-1$. Si $t' = 0$, j'enregistre directement la liste de bijections dans la case correspondant à t et 0 sans réaliser d'intersection puisqu'il s'agit du premier instant d'existence des isomorphismes concernés. Si $t' = \|T'\| - 1$, les isomorphismes appartenant à l'intersection ont satisfait la définition d'isomorphisme de sous-graphes temporels à partir de l'instant $t - t'$ et doivent donc être ajoutés à la liste des solutions au problème.

Ce procédé permet d'énumérer, par itération sur T et T', tous les isomorphismes de sous-graphes temporels de P vers L.

Afin d'améliorer le temps de calcul de cet algorithme, je peux ajouter une étape de pré-calcul visant à déterminer la longueur du plus long préfixe-suffixe de $L_0...L_t$ pour chaque valeur $t \in T$.

La **longueur du plus long préfixe-suffixe** de L pour une valeur $t \in T$ est la valeur $\delta \in T$ telle que les graphes instantannés L_k pour les $\delta - 1$ instants temporels précédant t et L_t soient identiques aux δ premiers graphes instantanés de L. La motivation de ce pré-traitement est de limiter le nombre de calculs, puisque si on a la même succession de δ graphes instantanés, la liste des isomorphismes de la suite des δ premiers graphes statiques de P vers chacune de ces deux suites de δ graphes statiques de L sera la même et que je n'aurais pas à réitérer ces calculs.

Pour réaliser ce pré-traitement, je parcours le graphe $L(V, E, T)$ et remplit un tableau Tab indiquant, pour chaque valeur $t \in T$ la longueur du plus long préfixe-suffixe $\delta \in T$.

Afin de remplir ce tableau Tab, j'initialise une valeur arbitraire 0 associée à L_0, et j'itère sur $t \in T\backslash\{0\}$. Pour chaque $t \in T\backslash\{0\}$, je parcours E_t et $E_{Tab[t-1]}$.

— Si E_t et $E_{Tab[t-1]}$ sont égaux, $Tab[t] \leftarrow Tab[t-1] + 1$.

— Sinon, si E_t et E_0 sont égaux, $Tab[t] \leftarrow 1$.

— Sinon, $Tab[t] = 0$.

La vérification d'égalité de E_t et $E_{Tab[t-1]}$ peut être réalisée comme suit :

— Vérifier que $\|E_t\| = \|E_{Tab[t-1]}\|$.

— Si la taille est identique, dupliquer $\|E_t\|$ dans une liste temporaire et y soustraire tous les $e \in E_{Tab[t-1]}$.

— E_t et $E_{Tab[t-1]}$ étant de taille identique, si en soustrayant tous les $e \in E_{Tab[t-1]}$ de E_t j'obtiens une liste vide, c'est que $E_t = E_{Tab[t-1]}$. Si la liste n'est pas vide, c'est que $E_t \neq E_{Tab[t-1]}$.

Cette vérification d'égalité est donc réalisée en itérant sur $E_{Tab[t-1]}$ et en soustrayant chaque arête temporelle ainsi trouvée de E_t, une opération de complexité $O(m_T)$, où $m_T = max(E_t)_{t\in T}$.

Cette vérification d'égalité est donc réalisée avec une complexité en $O(m_T^2)$ où $m_T = max(E_t)_{t \in T}$.

J'aboutis donc à l'algorithme de pré-traitement décrit dans l'algorithme 9.

Data: Graphe temporel $L : (V, E, T)$
Result: Tableau Tab des tailles de préfixe-suffixe
Je pose $Tab[0] \leftarrow 0$
for *Tout instant temporel* $t \in T \backslash \{0\}$ **do**
  Je pose $equals \leftarrow false$
  if $\|E_t\| = \|E_{Tab[t-1]}\|$ **then**
    Je pose $Temp \leftarrow E_t$
    for *Toute arête temporelle* $e \in E_{Tab[t-1]}$ **do**
      Je soustrais e à $Temp$.
    end
    if $\|Temp\| = 0$ **then**
      $equals \leftarrow true$
    end
  end
  if $equals = true$ **then**
    $Tab[t] \leftarrow Tab[t-1] + 1$
  end
  else
    On pose $equals \leftarrow false$
    if $\|E_t\| = \|E_0\|$ **then**
      On pose $Temp \leftarrow E_t$
      for *Toute arête temporelle* $e \in E_0$ **do**
        On soustrait e à $Temp$.
      end
      if $\|Temp\| = 0$ **then**
        $equals \leftarrow true$
      end
    end
    if $equals = true$ **then**
      $Tab[t] \leftarrow 1$
    end
    else
      $Tab[t] \leftarrow 0$
    end
  end
end
return Tab.

Algorithm 9: Calcul du tableau Tab enregistrant la taille maximale du préfixe-suffixe de L pour chaque valeur $t \in T$

Le tableau Tab est donc construit avec une complexité temporelle en $O(m_T^2 \times \tau)$ où $\tau = \|T\|$.

Cette complexité n'est pas idéale vis-à-vis de l'objectif que je me suis fixé dans le cadre de cette thèse, mais elle reste dans les limites de l'acceptable définies plus tôt dans cette thèse.

Maintenant que ce pré-traitement est défini, je vais formaliser l'algorithme d'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire. Afin de définir l'algorithme d'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire de manière digeste pour le lecteur, je définis en amont les opérations suivantes : **Calcul de la longueur d'un chemin depuis l'une de ses extrémités** (algorithme 10), **Breadth First Search pour calculer tous les chemins de longueur n partant d'un sommet d'un arbre** (algorithme 11), **Construction d'une bijection à partir de deux chemins de même longueur** (algorithme 12), **Énumération des prolongements de deux listes d'isomorphisme** (algorithme 13), et **Calcul de l'intersection de deux**

listes d'isomorphisme (algorithme 14).

Data: Un graphe statique $G(V, E)$, un sommet $u \in V$ extrêmité d'un chemin de G
Result: Longueur du chemin dont u est une extrémité
if $\|voisinage(u)\| \neq 0$ **then**
  On note v le voisin de u.
  On crée un graphe statique $H(V_H, E_H)$ qu'on rend égal à G.
  On supprime de E_H l'arrête (u, v).
  On note l la solution de cet algorithme appliqué à H et v.
  $longueur \leftarrow l + 1$.
end
else
  $longueur \leftarrow 1$.
end
return $longueur$.

Algorithm 10: Calcul de la longueur du chemin dont un sommet est une extrêmité

Je m'intéresse maintenant à la complexité temporelle de ces différentes opérations.

L'algorithme 10 de calcul de longueur d'un chemin depuis l'une de ses extrémités procède à un parcours de tous les sommets du chemin dont u est une extrêmité. Donc le calcul de la longueur d'un chemin depuis l'une de ses extrémités est réalisé en temps linéaire $O(longueur(c))$ où c est le chemin en question.

Data: Un graphe statique $G(V, E)$, un sommet $v \in V$, un entier $n \in \mathbb{N}$ et le précédent sommet traité $u \in V$
Result: Liste des chemins de taille n dont v est une extrémité
Je pose $Solution$ une liste de chemins initialement vide
if $n = 1$ **then**
  Ajouter v à Solution
end
else
  if $voisinage(v) = \{u\}$ **then**
    $Solution \leftarrow null$
  end
  else
    for $z \in voisinage(v) \backslash \{u\}$ **do**
      Je pose $Temp$ la solution de cet algorithme appliqué à G, z, $n-1$, et v).
      for *tout chemin* $c \in Temp$ **do**
        J'ajoute v à c
        J'ajoute c à $Solution$
      end
    end
  end
end
return $Solution$.

Algorithm 11: Algorithme de Breadth First Search pour rechercher tous les chemins de taille n dont un sommet v est une extrémité

Pour chaque itération de l'algorithme 11 de Breadth First Search pour calculer tous les chemins de longueur n partant d'un sommet d'un arbre, j'itère donc sur le voisinage du sommet concerné pour une complexité en $O(dmax)$ où $dmax$ est le degré maximal des sommets du graphe. Au sein de cette boucle, j'itère sur les chemins calculés lors de l'itération suivante du BFS. Au niveau 0, ce nombre de chemins est égal à 1. Puis au niveau $k \in N$, ce nombre de chemins sera égal au nombre de chemins calculés à l'itération précédente multiplié par la taille du voisinage du sommet concerné.

Donc au niveau $k \in N$, j'itère au pire cas sur $dmax^{k-1}$ chemins. Je réaliserai un nombre d'itérations égal à la longueur du chemin recherché, donc au pire cas n' le nombre de sommets dans P. J'obtiens donc une complexité au pire cas pour une itération en $O(dmax^{n'})$ où $dmax$ est le degré maximal des sommets de l'arbre traité. Ce degré est en tout instant borné par m_T, le nombre d'arêtes du graphe instantané traité. Au pire cas, je parcoure l'ensemble de l'arbre, chaque itération traitant un sommet et chaque sommet n'était traité que lors d'une itération. La complexité totale du BFS est donc en $O(n \times m_T^{n'})$ où n est le nombre de sommets dans L, n' le nombre de sommets dans P et $m_T = max(\|E_t\|)_{t \in T}$.

Data: Deux chemins c et c'
Result: Une bijection $f : c- > c'$
Je pose $f : c- > c'$ une bijection
for *i entre 0 et longueur$(c[0])$* **do**
  Je pose $f(c'[i]) \leftarrow c[i]$
end
return f.

Algorithm 12: Construction d'une bijection entre les sommets de deux chemins de même taille

L'algorithme 12 de construction d'une bijection à partir de deux chemins de même longueur est réalisé en temps linéaire $O(longueur(c))$

Data: Deux graphes temporels $L(V, E, T)$ et $G(V', E', T')$, et deux listes F_0 et F_1 d'isomorphismes
Result: Une liste de paires $\{f, t\}$ issue du prolongement des fonctions de F_0 et F_1 cohérentes 2 à 2
Je pose *Solution* une liste de paires $\{f, t\}$ initialement vide
for *toute paire* $\{f, t\} \in F_1$ **do**
  for *toute paire* $\{f', t\} \in F_0$ **do**
    Je pose *coherent* $\leftarrow$ *true*
    for *tout sommet* $v' \in V'$ *dont l'image par* f *est définie* **do**
      if *l'image de* v' *par* f' *est définie* $\wedge$ $f(v') \neq f'(v')$ **then**
        coherent $\leftarrow$ *false*
      end
    end
    if *coherent* = *true* **then**
      Je pose $h : V' \rightarrow V$
      for *tout sommet* $v \in V'$ *dont l'image par* f' *est définie* **do**
        Je pose $h(v) \leftarrow f'(v)$
      end
      for *tout sommet* $v \in V'$ *dont l'image par* f *est définie* **do**
        Je pose $h(v) \leftarrow f(v)$
      end
      J'ajoute $\{h, t\}$ à *Solution*
    end
  end
end
return *Solution*.

Algorithm 13: Algorithme d'énumération des prolongements de deux listes d'isomorphisme

L'algorithme 13 d'énumération des prolongements de deux listes d'isomorphisme itère sur les deux listes d'isomorphisme, et pour chaque paire d'isomorphisme, parcourt les sommets ayant une image par l'un de ces isomorphismes pour déterminer s'ils ont une image par l'autre isomorphisme. Cette vérification et la vérification de cohérence de ces deux images sont réalisées en temps constant. Donc pour établir la cohérence entre deux isomorphismes, la complexité est linéaire en le nombre de

sommets de V' ayant une image par ces isomorphismes. Puisque cet isomorphisme repose sur une bijection entre l'ensemble V' des sommets de P et un sous-ensemble de l'ensemble V des sommets de L, le nombre de sommets ayant une image par ces isomorphismes est borné par $n' = \|V'\|$. Puis, si la cohérence est établie, j'itère sur les sommets dont l'image est définie par l'un des isomorphismes puis sur les sommets dont l'image est définie par l'autre. La complexité totale de cette opération est donc en $O(i^2 \times n')$ où i est le nombre maximal d'isomorphismes dans chaque liste, qui au pire cas sera en $O(\binom{n}{n'})$ avec n le nombre de sommets de V et n' le nombre de sommets de V'. La complexité de cette opération est donc en $O(\binom{n}{n'}^2 \times n')$

Data: Deux graphes temporels $L(V, E, T)$ et $P(V', E', T')$ et deux listes F_0 et F_1 d'isomorphismes
Result: La liste de paires $\{f, t\}$ appartenant aux deux listes simultanément
Je pose $Temp$ une liste de paires $\{f, t\}$ initialement vide
for *toute paire* $\{f, t\} \in F_0$ **do**
  Je pose $exists \leftarrow false$
  for *toute paire* $\{f', t'\} \in F_1$ **do**
    On pose $coherent \leftarrow true$
    for *tout sommet* $v' \in V'$ *ayant une image par* f **do**
      if $f(v') \neq f'(v')$ **then**
        $coherent \leftarrow false$
        break
      end
    end
    if $coherent = true$ **then**
      $exists \leftarrow true$
      break
    end
  end
  if $exists = true$ **then**
    Ajouter $\{f, t'\}$ à $Temp$
  end
end
return $Temp$.

Algorithm 14: Algorithme d'intersection de deux listes d'isomorphisme

L'opération d'intersection de deux listes d'isomorphisme détaillée dans l'algorithme 14 itère sur les deux listes d'isomorphismes et pour chaque paire d'isomorphismes, itère sur les sommets de V' pour une vérification en temps constant. La complexité de cette opération est donc en $O(\binom{n}{n'}^2 \times n')$.

Avec l'ajout du pré-traitement visant à calculer un tableau indiquant la longueur maximale du préfixe-suffixe de L pour chaque $t \in T$, de la structure de données listant les bijections possibles entre L_t et $P_{t'}$ pour tout $\{t, t'\} \in T \times T'$ et la définition des cinq opérations présentées ci-dessus, je peux définir un algorithme d'énumération de sous-forêts temporelles isomorphes à des forêts temporelles linéaires tel que décrit dans l'algorithme 15.

Propriété 4.1.1. *L'algorithme d'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire présente des complexités temporelles et spatiales linéaires en la taille de l'historique.*

Démonstration. Cet algorithme itère sur les historiques de L et P et pour chaque itération de cette double boucle, itère sur V. Pour chaque sommet, on itère sur V' à la recherche des extrémités de chemin. Pour chaque extrémité de chemin trouvée, on applique successivement les opérations **Calcul de la longueur d'un chemin depuis l'une de ses extrémités**, **Breadth First Search pour calculer tous les chemins de longueur** n **partant d'un sommet**, et pour chaque correspondance entre un chemin appartenant à la solution de ce BFS avec un chemin de P'_t, on exécute **Construction d'une bijection à partir de deux chemins de même longueur** pour une complexité en $O(longueur(c) + n \times m_T^{n'} + p \times$

Data: Forêt temporelle $L : (V, E, T)$, forêt temporelle linéaire $P(V', E', T')$, le tableau Tab
Result: Liste des paires $\{f, t\}$ solution du problème de l'isomorphisme de sous-forêts temporelles où le motif est une forêt temporelle linéaire

Je pose *Solution* une liste de paires $\{f, t\}$ initialement vide
Je pose la structure de données $Iso[T][T']$
for *tout instant temporel* $t \in T$ **do**
  for *tout instant temporel* $t' \in T'$ **do**
    if $t' = Tab[t] - 1$ **then**
      $Iso[t][t'] \leftarrow Iso[Tab[t] - 1]][Tab[t] - 1]$
    end
    else
      for *tout sommet* $v \in V$ **do**
        Je pose $Temp[v]$ une liste vide de paires $\{f, t\}$.
        for *tout sommet* $v' \in V'$ **do**
          if $\|Voisinage(v')\| \leqslant 1$ **then**
            Je pose $ListChemin = BFS(L_t, v, longueur(P_{t'}, v'), null)$
            for *tout chemin* $C \in ListChemin$ **do**
              Je pose $f : V' \rightarrow V = createIsomorphism(chemin(v'), C)$.
              J'ajoute $\{f, t\}$ à $Temp[v]$.
            end
          end
        end
      end
      if $Iso[t][t']$ *est vide* **then**
        $Iso[t][t'] \leftarrow Temp[v]$
      end
      else
        $Iso[t][t'] \leftarrow isomorphismProlongation(L, G, Iso[t][t'], Temp[v])$
      end
    end
    if $t' \neq 0$ **then**
      $Iso[t][t'] \leftarrow isomorphismIntersection(L, G, Iso[t][t'], Iso[t - 1][t' - 1])$
    end
    if $t' = max(T')$ **then**
      Ajouter $Iso[t][t']$ à *Solution*
    end
  end
end
return *Solution*.

Algorithm 15: Algorithme d'énumération des sous-forêts temporelles isomorphes à des forêts temporelles linéaires

$longueur(c)$) où $longueur(c)$ est la taille du chemin concerné, p la taille de la solution du BFS dans l'arbre concerné, $m_T = max(\|E_t\|)_{t\in T}$, n le nombre de sommets de L et n' le nombre de sommet de P. La taille du chemin recherché dans un arbre par le BFS est bornée par n'. Le nombre de solutions du BFS est donc borné par $\binom{n}{n'}$ puisqu'au pire cas, celui du graphe complet, qui n'est pas un arbre mais permet de borner la complexité au pire cas, tout sous-ensemble de sommets de taille n' appartient à la solution. Donc au pire cas, le calcul réalisé pour chaque extrémité de chemin $v' \in V'$ est de complexité $O(n \times m_T^{n'} + n' \times \binom{n}{n'})$. Pour chaque sommet de V, l'opération de construction de tous les isomorphismes pour lesquels il pourrait être image d'un sommet de V' est donc réalisé en $O(n' \times (n \times m_T^{n'} + n' \times \binom{n}{n'}))$ où n est le nombre de sommets de L, n' le nombre de sommets de P et m_T le nombre maximal d'arêtes instantanées de L. Une fois ces opérations utilisées, les isomorphismes découlant du sommet de V concerné font l'objet d'un prolongement deux à deux avec les isomorphismes déjà calculés pour les autres sommets de V, opération réalisée avec une complexité en $O(\binom{n}{n'}^2 \times n')$. Pour chaque sommet de V la complexité totale du traitement est donc en $O(n' \times (n \times m_T^{n'} + n' \times \binom{n}{n'}) + n' \times \binom{n}{n'}^2))$. Une fois cette opération réalisée pour chaque sommet de V, on procède à une intersection du résultat obtenu avec celui obtenu lors de la précédente itération sur T. L'intersection a une complexité en $O(n' \times \binom{n}{n'}^2)$. Pour chaque itération de la double boucle, la complexité totale du traitement est donc en $O(n \times (n' \times (n \times m_T^{n'} + n' \times \binom{n}{n'}) + n' \times \binom{n}{n'}^2))$ soit en $O(n \times n' \times (n \times m_T^{n'} + n' \times \binom{n}{n'}^2))$. La complexité totale de cet algorithme est donc au pire cas en $O(\tau \times \tau' \times n \times n' \times (n \times m_T^{n'} + n' \times \binom{n}{n'}^2))$ avec τ la taille de l'historique de L, τ' la taille de l'historique de P, n le nombre de sommets de L, n' le nombre de sommets de P, m_t le nombre maximal d'arêtes instantanées de L. □

Néanmoins, cette complexité au pire cas n'est que rarement atteinte. En effet, le nombre d'isomorphisme appartenant à chaque liste, est ici borné par $\binom{n}{n'}^2$, qui correspondrait au cas où à chaque instant $t \in T$ et chaque instant $t' \in T'$, tous les sommets de L pourraient être image par un isomorphisme de tous les sommets de P (cas où $\|E\| = \|E'\| = 0$ par exemple).

Il n'est de même pas tenu compte dans ce pire cas du pré-traitement défini plus haut, ce qui, dans le cas des graphes qui m'intéressent dans cette thèse, qui sont peu denses, est peu probable, la majorité des graphes instantanés présentant peu ou pas d'arêtes instantanées.

Il est également possible d'améliorer la complexité en ne réalisant l'ensemble du traitement que pour $t' = 0$. $\forall t' \in T'\backslash\{0\}$, il suffira de vérifier que les isomorphismes calculés pour $t' = 0$ restent cohérents. Ceci est réalisé par l'opération définie dans l'algorithme 16 de vérification de la cohérence d'une liste d'isomorphisme calculée à l'instant précédent.

La complexité de cette opération est en $O(i^2 \times n'^2)$ où i est le nombre d'isomorphismes à vérifier, au pire cas en $O(\binom{n}{n'})$. La complexité au pire cas de cet algorithme devient alors en $O(\tau \times (n^2 \times n' \times (m_T^{n'} + \binom{n}{n'}^2) + \tau' \times n'^2 \times \binom{n}{n'}^2))$ soit en $O(\tau \times \binom{n}{n'}^2 \times n' \times (n^2 \times m_T^{n'} + \tau' \times n'))$. La complexité spatiale de cet algorithme est quant à elle bornée par la structure de données de tableau à double entrée listant toutes les bijections possibles entre L_t et $P_{t'}$ pour tout $\{t, t'\} \in T \times T'$. Pour chaque $\{t, t'\} \in T \times T'$, le nombre de bijections possibles est de l'ordre de $\binom{n}{n'}$. J'obtiens donc une complexité spatiale au pire cas en $O(\tau \times \tau' \times \binom{n}{n'})$.

J'ai ainsi défini un algorithme d'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire. Cet algorithme présente une complexité temporelle en $O(\tau \times \binom{n}{n'}^2 \times n' \times (n^2 \times m_T^{n'} + \tau' \times n'))$ et une complexité spatiale en $O(\tau \times \tau' \times \binom{n}{n'})$, où τ est la taille de l'historique de L, n' le nombre de sommets de P, n le nombre de sommets de L, m_T le nombre d'arêtes instantanées maximal de L_t pour $t \in T$ et τ' la taille de l'historique de P. Si ces complexités satisfont l'exigence de dépendance linéaire en τ (ou plutôt bilinéaire en τ et τ'), elles sont également proportionnelle à la factorielle de n, ce qui n'est assurément pas très encourageant. Dans le cadre des jeux de données traités dans cette thèse, ces complexités risquent de m'empêcher d'aboutir à des solutions expérimentales faute de mémoire ou à tout le moins d'handicaper la possibilité d'énumérer les sous-forêts temporelles isomorphes à une forêt temporelle linéaire dans un temps raisonnable (de l'ordre de quelques dizaines de minutes). Cependant, cet algorithme peut s'avérer utile pour énumérer les sous-forêts temporelles isomorphes à une forêt temporelle linéaire dans des graphes présentant de très faibles nombres de sommets (moins d'une dizaine) mais s'étendant sur

Data: Deux graphes temporels $L(V, E, T)$ et $P(V', E', T')$, une liste *List* de paires $\{f, t\}$, un instant temporel $t \in T$ et un instant temporel $t' \in T'$
Result: Une liste de paires $\{f, t\}$ qui restent cohérentes pour les instants t et t'
Je pose *Solution* une liste de paires $\{f, t\}$ initialement vide.
for *toute paire* $\{f, t\} \in List$ **do**
  On pose $coherent \leftarrow true$.
  for *tout sommet* $u \in V'$ *ayant une image définie par* f **do**
    for *tout sommet* $v \in V'$ *ayant une image définie par* f **do**
      if *Si* u *et* v *sont liés à l'instant* t' *mais* $f(u)$ *et* $f(v)$ *ne sont pas liés à l'instant* $t + t'$ *ou si* u *et* v *ne sont pas liés à l'instant* t' *mais* $f(u)$ *et* $f(v)$ *sont liés à l'instant* $t + t'$ **then**
        $coherent \leftarrow false$.
      end
    end
  end
  if $coherent = true$ **then**
    On ajoute $\{f, t\}$ à *Solution*.
  end
end
return *Solution*.

Algorithm 16: Opération de vérification de la cohérence d'une liste d'isomorphisme calculée à l'instant précédent

de grandes tailles d'historique.

4.2 Énumération de sous-graphes temporels isomorphes

Maintenant que j'ai apporté une solution au cas particulier de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire, je me penche sur le problème général de l'énumération des sous-graphes temporels isomorphes. Pour ce faire, je m'appuie sur trois articles :

— L'article de Paranjape et al. [40] propose un algorithme pour résoudre une version modifiée du problème d'isomorphisme de sous-graphe temporel que je définis ci-dessous comme **Problème d'isomorphisme de sous-graphe temporel par suite d'arêtes**. La solution apportée à ce problème par l'article requière une résolution du problème d'énumération de sous-graphes statiques isomorphes.
— L'article de Kozen [32] propose un algorithme pour résoudre le problème d'énumération de sous-graphes statiques isomorphes en le ramenant à un problème d'énumération de cliques de taille n' (où n' est le nombre de sommets dans le graphe statique motif) dans un graphe intermédiaire baptisé et défini dans la suite de cette thèse sous le nom de **graphe de conformité**.
— L'article de Carraghan et Pardalos [12] propose un algorithme permettant de calculer la clique de taille maximale d'un graphe statique.

Les algorithmes de chacun de ces articles vont pouvoir être combinés et modifiés pour me permettre de proposer une solution au problème d'énumération de sous-graphes temporels isomorphes. Si, initialement, le problème résolu par la combinaison de ces algorithmes est le problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes, je montrerai dans la dernière sous-section de cette section comment l'algorithme peut être modifié pour permettre d'énumérer les sous-graphes temporels isomorphes.

La définition du problème d'énumération des sous-graphes temporels isomorphes par suite d'arêtes nécessite l'introduction de la notion de graphe temporel compressé à l'infini. Je commence donc par définir la notion de graphe temporel compressé :

Définition 4.2.1 (Graphe temporel compressé)**.** Soit $L(V, E, T)$ un graphe temporel et $r \in \mathbb{N}$. Le graphe temporel $L_r(T_r, V, E_r)$ est appelé **graphe temporel compressé de** L **au ratio** r si :

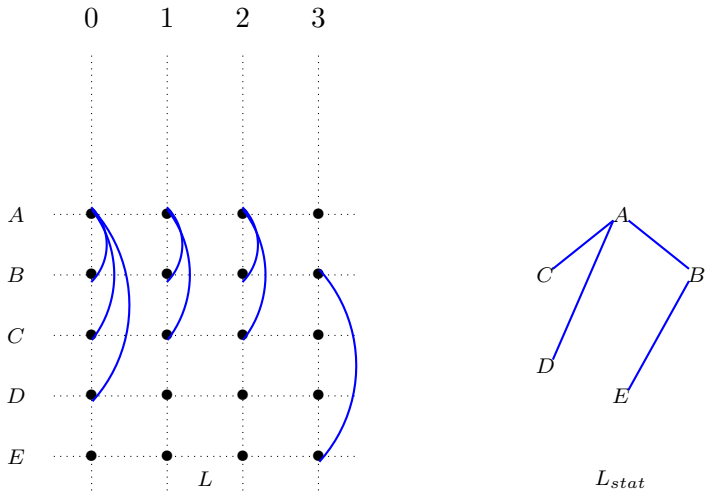

FIGURE 4.1 – Exemple de graphe temporel compressé à l'infini

— $\|T_r\| = \lfloor \frac{\|T\|}{r} \rfloor$.
— Pour tout $t \in T_r$, deux sommets u et v sont liés par une arête à l'instant t dans L_r si il existe $t_0 \in [\![t \times r, min((t+1) \times r, \|T\| - 1)]\!]$, tel que u et v sont liés par une arête à l'instant t_0 dans L.

Je peux ainsi introduire la notion de graphe temporel compressé à l'infini, en appliquant un ratio $r = \infty$ afin d'obtenir un graphe statique.

Définition 4.2.2 (Graphe temporel compressé à l'infini)**.** Soit $L(V, E, T)$ un graphe temporel. Le graphe statique $G(V, \sum_{t \in T}(E_t))$ est appelé **graphe temporel compressé à l'infini de** L.

Je donne un exemple de graphe temporel compressé à l'infini à la figure 4.1.

Maintenant que ces notions ont été définies, définissons le problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes auquel la combinaison des trois articles cités plus haut me permet d'apporter une solution.

Définition 4.2.3 (Problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes)**.** Soit $L(V, E, T)$ et $P(V', E', T')$ deux graphes temporels tels que P n'est vide à aucun instant de T' et E' est classé par instant temporel d'existence de l'arête croissant, et soit $\Delta \in \mathbb{N}$. Le problème d'énumération des sous-graphes temporels isomorphes par suite d'arêtes est le problème d'énumération des paires $\{f, (e_k)_{k \in [\![1, \|E'\|]\!]}\}$ telles que $\exists A \subseteq V, f : V' \to A$ tel que f est un isomorphisme de sous-graphe de P_{stat} vers L_∞, avec L_∞ le graphe temporel compressé à l'infini de L et P_∞ le graphe temporel compressé à l'infini de P, et $(e_k)_{k \in [\![0, \|E'\|]\!]}$ est une suite d'arêtes telle que
— Les arêtes e_k sont des arêtes de E liant deux sommets de A
— Une arête e_k existe à un instant temporel t supérieur ou égal aux instants temporels auxquels existent toutes les arêtes précédant e_k dans la suite $(e_k)_{k \in [\![0, \|E'\|]\!]}$
— Deux arêtes de $(e_k)_{k \in [\![0, \|E'\|]\!]}$ existent à des instants temporels distants au maximum de Δ instants temporels.
— La k-ième arête de la suite $(e_k)_{k \in [\![0, \|E'\|]\!]}$ est l'image de la k-ième arête de E'
— Deux arêtes successives de $(e_k)_{k \in [\![0, \|E'\|]\!]}$ existent au même instant temporel dans L si et seulement si les arêtes de P dont elles sont l'images existent au même instant.

4.2.1 Énumération de cliques de rang k

Je commence donc par apporter une réponse au problème d'énumération de cliques de taille k en utilisant l'algorithme de l'article de Carraghan et Pardalos [12] qui calcule la clique de taille maximale et en le modifiant pour énumérer les cliques de rang k.

Soit $G(V_G, E_G)$ un graphe statique. Je trie V_G par degré décroissant. Ce tri peut être réalisé en temps linéaire $O(n_G)$ où n_G est le nombre de sommets de G. J'itère ensuite sur V_G et pour chaque sommet $v \in V_G$, j'applique récursivement l'algorithme d'énumération de cliques au voisinage de v, afin d'y calculer toutes les cliques de taille $k-1$. Lorsque $k = 1$, je retourne comme solution une liste des singletons du sous-ensemble de sommets traité dans cette itération. Sinon, après avoir récupéré la liste *Solution* de cliques de taille $k-1$ dans le voisinage de v, j'ajoute v à chacune de ces cliques et j'ajoute chacune à la solution. J'interromps le parcours lorsque $\|Voisinage(v)\| - i - k < 0$ où i est l'indice du sommet u traité dans le $Voisinage(v)$ trié car il ne restera pas assez de sommets pour former une clique de taille k qui n'est pas encore listée par le parcours de V.

Propriété 4.2.1. *Soit $G(V_G, E_G)$ un graphe statique avec V_G rangé par degré décroissant et deux entiers $\{i, k\} \in [\![0, \|V_G\|]\!]^2$. Si $i > \|V_G\| - k$, Toute clique de taille k contenant le i-ème sommet de V_G contient obligatoirement au moins un sommet d'indice dans V_G inférieur à i.*

Démonstration. Soit $G(V_G, E_G)$ un graphe statique avec V_G rangé par degré décroissant, $k \in [\![0, \|V_G\|]\!]$ et un entier i strictement supérieur à $\|V_G\| - k$.

Raisonnons par l'absurde et supposons une clique A de taille k dans G, qui contient le i-ème sommet de V_G mais ne contient aucun sommet d'indice dans V_G inférieur à i.

Par conséquent $A \subseteq B$ avec B l'ensemble des sommets d'indice dans V_G supérieur ou égal à i. $\|B\| = \|V_G\| - i$, or puisque $i > \|V_G\| - k$, cela signifie que $\|B\| < k$. Donc $\|A\| < k$, ce qui contredit notre hypothèse de raisonnement par l'absurde. □

Je définis ainsi un algorithme d'énumération des cliques de taille k dans un graphe statique, tel que décrit dans l'algorithme 17.

```
Data: Un graphe statique G(V_G, E_G) et un entier k ∈ ℕ
Result: La liste de toutes les cliques de taille k dans G
On pose Solution la liste de cliques, initialement vide.
if k = 1 then
    for tout sommet u ∈ V_G do
        On ajoute {u} à Solution.
    end
end
else
    On trie V_G par degré décroissant.
    On pose i = 0
    while ‖V_G‖ − i − k ⩾ 0 do
        if Le voisinage de V_G[i], le i-ème sommet de V_G, n'est pas vide then
            On applique cet algorithme à Voisinage(V_G[i]) et k − 1) et on enregistre le
             résultat dans List
            for toute clique c ∈ List do
                J'ajoute V_G[i] à c.
                J'ajoute c à Solution.
            end
        end
        J'incrémente i.
    end
end
return Solution.
```

Algorithm 17: Algorithme d'énumération des cliques de taille k dans un graphe statique $G(V_G, E_G)$

Propriété 4.2.2. *$A \subseteq V_G$ est une clique de taille k de $G(V_G, E_G)$ si et seulement si A appartient à la liste construite par l'algorithme 17 appliqué à G et k. Les complexités spatiale et temporelle de cet algorithme sont indépendantes de la taille de l'historique.*

Démonstration. Soit $G(V_G, E_G)$ un graphe statique et $k \in \mathbb{N}$.

— $\Rightarrow$

Supposons $A \subseteq V_G$ une clique de taille k et prouvons que cette clique est énumérée par l'algorithme 17 d'énumération des cliques de taille k dans un graphe statique $G(V_G, E_G)$.

Si $k = 1$, l'algorithme ajoute tous les sommets du graphe. A est donc bien ajoutée à la solution de l'algorithme si $k = 1$.

Si $k \neq 1$, l'algorithme va sélectionner un premier sommet $v \in A$ puis être appliqué récursivement sur $Voisinage(A)$ pour y chercher les cliques de taille $k - 1$. Il me suffit donc de prouver qu'un sommet $v \in A$ est sélectionné par l'algorithme.

Je raisonne par l'absurde et je suppose qu'aucun $v \in A$ n'est sélectionné par l'algorithme. Cela signifie qu'aucun sommet $V \in A$ n'a un indice inférieur à $\|V_G\| - k$ dans V_G. De par la propriété 4.2.1, A ne peut alors être une clique de taille k, ce qui contredit mes hypothèses.

— $\Leftarrow$

La démonstration de cette réciproque est triviale puisqu'on peut facilement noter que l'algorithme s'exécute récursivement k fois, ajoutant à chaque application un sommet, construisant donc des ensembles de k sommets, et qu'à chaque application, on itère récursivement sur le voisinage des sommets sélectionnés, si bien que tous les sommets ajoutés à A appartiennent au voisinage de tous les autres sommets de A, faisant de A une clique.

Donc $A \subseteq V_G$ est une clique de taille k de $G(V_G, E_G)$ si et seulement si A appartient à la liste construite par l'algorithme 17 d'énumération des cliques de taille k dans un graphe statique $G(V_G, E_G)$ appliqué à G et k.

On établit maintenant la complexité temporelle de cet algorithme. Cet algorithme itère, à chaque appel de la récursion, à $\|V_p\| - k$ itérations où $\|V_p\|$ est la taille du voisinage du sommet sélectionné à la p-ième itération. A chaque itération, on procède à une récursion pour $Voisinage(u)$ et $k - 1$ avec $u \in V$. Une fois cette récursion accomplie, on itère sur la solution obtenue et on réalise une opération en temps constant. Par conséquent, la complexité temporelle totale est en $O((\|V_G\| - k) \times ((dmax - k - 1) \times ..(dmax + s) + s) + s)$ où k est la taille de la clique recherchée, $dmax$ le degré maximal des sommets de V_G et s la taille de la solution. □

Dans le cas qui m'intéresse ici, à savoir l'application de l'énumération de cliques de taille k à la résolution du problème d'isomorphisme de sous-graphes, k correspondra au nombre de sommets de P alors que $\|V_G\|$ correspondra au produit du nombre de sommets de P et du nombre de sommets de L (voir sous-section suivante et définition du graphe de consolidation). On peut donc approximer cette complexité en considérant que $k << \|V_G\|$. Quant au degré maximal au sein du graphe, il est nécessairement supérieur à k, faute de quoi, aucune clique de taille k ne pourra exister dans G. On peut ainsi simplifier cette complexité en $O(\|V_G\| \times (dmax + s)^k)$ avec $\|V_G\|$ le nombre de sommets de G, $dmax$ le degré maximal des sommets de G, k la taille des cliques recherchées et s le nombre de cliques de taille k dans G. La complexité spatiale de cet algorithme n'est due qu'à la structure de données utilisée pour stocker les résultats. De sorte que la complexité spatiale de cet algorithme est proportionnelle à $O(k \times s)$.

J'ai donc établi un algorithme énumérant les cliques de rang n dans un graphe statique. Cet algorithme présente une complexité temporelle en $O(\|V_G\| \times (dmax + s)^k)$ et de complexité spatiale en $O(k \times s)$, où $\|V_G\|$ le nombre de sommets de G, $dmax$ le degré maximal des sommets de G, k la taille des cliques recherchées et s le nombre de cliques de taille k dans G.

4.2.2 Énumération de sous-graphes statiques isomorphes

L'article de Kozen [32] présente un algorithme pour énumérer les sous-graphes statiques isomorphes en se rapportant à un problème d'énumération de cliques de taille de k dans un graphe statique intermédiaire que je baptise graphe de conformité.

J'introduis le **graphe de conformité** :

Définition 4.2.4. Soit $G(V_G, E_G)$ et $P(V_P, E_P)$ deux graphe statiques. Je définis $M(V_G \times V_P, E_m)$ le **graphe de conformité de G et P** tel qu'il existe une bijection $g : V_G \times V_P \rightarrow V_m$ et tel que :

— Pour tous sommets u et v de V_G et u' et v' de V_P, une arête lie deux sommets $g(u, u')$ et $g(v, v')$ dans M si u et v sont liés dans G et u' et v' sont liés dans P, ou si u et v ne sont pas liés dans G et u' et v' ne sont pas liés dans P.

Ce graphe de conformité $M(V_G \times V_P, E_m)$ de deux graphes statiques $G(V_G, E_G)$ et $P(V_P, E_P)$ a donc la complexité spatiale au pire cas suivante :
— $n_G \times n_P$ sommets.
— $m_G \times m_P + (n_G^2 - m_G) \times (n_P^2 - m_P)$ arêtes où m_G est le nombre d'arêtes de G, n_G le nombre de sommets de G, n_P le nombre de sommets de P et m_P le nombre d'arêtes de P.
Un exemple de graphe de conformité est donné en figure 4.2.
Afin de construire ce graphe de conformité, j'utilise l'algorithme 18 de construction du graphe de conformité entre deux graphes statiques.

Data: $G(V_G, E_G)$ et $P(V_P, E_P)$ deux graphes statiques
Result: $M(V_m, E_m)$ le graphe de conformité de G et P
J'initialise M vide.
for *tout sommet* $u \in V_G$ **do**
  for *tout sommet* $u' \in V_P$ **do**
    Je définis $v = g(u, u')$.
    J'ajoute v à V_m.
  end
end
for *tout sommet* $u \in V_G$ **do**
  for *tout sommet* $v \in V_G$ **do**
    for *tout sommet* $u' \in V_P$ **do**
      for *tout sommet* $v' \in V_P$ **do**
        if $(u, v) \in E_G$ **then**
          if $(u', v') \in E_P$ **then**
            J'ajoute $(g(u, u'), g(v, v'))$ à E_m.
            J'ajoute $(g(u, v'), g(v, u'))$ à E_m.
          end
        end
        else
          if $(u', v') \notin E_P$ **then**
            J'ajoute $(g(u, u'), g(v, v'))$ à E_m.
            J'ajoute $(g(u, v'), g(v, u'))$ à E_m.
          end
        end
      end
    end
  end
end
return M.

Algorithm 18: Algorithme de construction du graphe de conformité de deux graphes statiques

Propriété 4.2.3. *L'algorithme de construction du graphe de conformité M de $G(V_G, E_G)$ et $P(V_P, E_P)$ présente des complexités temporelle et spatiale au pire cas indépendant de la taille de l'historique.*

Démonstration. Cette opération itère donc sur les paires de sommets de V_G et sur les paires de sommets de V_P et une fois ces quatre sommets sélectionnés, vérifie la présence d'une arête dans E_G et d'une autre dans E_P. La complexité de la construction de M est au pire cas en

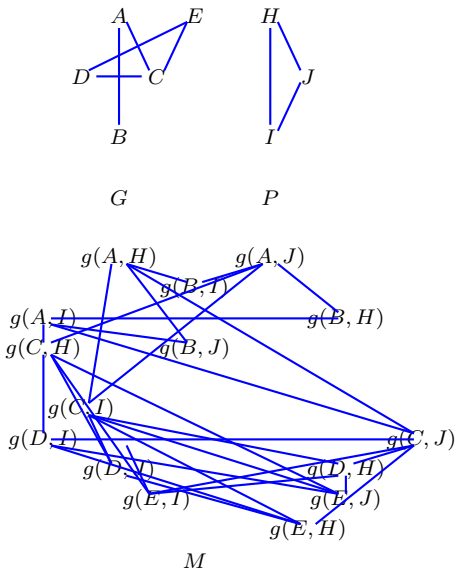

FIGURE 4.2 – Exemple de graphe de conformité de deux graphes statiques

$O(n_G^2 \times n_P^2 \times (m_G + m_P))$ où n_G est le nombre de sommets de G, n_P le nombre de sommets de P, m_G le nombre d'arêtes de G et m_P le nombre d'arêtes de P.

□

Une fois ce graphe de conformité construit, j'utilise la propriété suivante :

Propriété 4.2.4. *Une fonction bijective $f : V_P \to A \subseteq V_G$ est un isomorphisme de sous-graphes entre $G(V_G, E_G)$ et $P(V_P, E_P)$ si et seulement si l'ensemble de sommets de M représentant $g(f(u), u)$, pour tout $u \in V_P$, est une clique de taille $\|V_P\|$ dans M.*

Démonstration. Soit deux graphes statiques $G(V_G, E_G)$ et $P(V_P, E_P)$.

— $\Rightarrow$

Soit une fonction bijective $f : V_P \to A \subseteq V_G$ isomorphisme de sous-graphe entre $G(V_G, E_G)$ et $P(V_P, E_P)$. Donc pour toute paire de sommets $\{u, v\}$ de V_P, u et v sont liés par une arête dans P si et seulement si $f(u)$ et $f(v)$ sont liés dans G. Donc $g(u, f(u))$ et $g(v, f(v))$ sont liés dans M. Donc les sommets $g(u, f(u))$ pour tous sommets $u \in V_P$ forment une clique de taille $\|V_P\|$ dans M.

— $\Leftarrow$

Soit $B \subseteq V_G \times V_P$ une clique de taille $\|V_P\|$ dans M. Chaque sommet de B correspond à $g(u, u')$ pour $u \in V_G$ et $u' \in V_P$. Je pose une fonction $f : V_P \to A \subseteq V_G$, tel que pour chaque sommet $g(u, u')$ de B, $u' = f(u)$. Par construction de M, pour chaque sommet $g(u, f(u))$ de B, il n'existe aucun autre sommet $g(v, f(v))$ de B tel que $v = u$ ou $f(v) = f(u)$. Donc f est une bijection.
Par construction de M, pour toute paire de sommets $(g(u, f(u)), g(v, f(v)))$ de B, u et v sont liés par une arête dans G si et seulement si $f(u)$ et $f(v)$ sont liés par une arête dans P. Donc f est un isomorphisme de sous-graphes entre G et P.

□

Pour déterminer f à partir d'une clique A de taille $\|V_P\|$, on parcourt les sommets $u_m \in A$ et on obtient la paire $(u, u') \in E_G \times E_P$ telle que $u_m = g(u, u')$ afin de définir $f(u') = u$. Cette opération est réalisé en $O(n_P)$ où n_P est le nombre de sommets de P.

Ainsi, en construisant le graphe de conformité M de $G(V_G, E_G)$ et $P(V_P, E_P)$ et en lui appliquant l'algorithme d'énumération de cliques de tailles $\|V_P\|$, on peut énumérer les sous-graphes de G isomorphes à P. L'algorithme de recherche de cliques a dans M une complexité en $O(\|V_G \times V_P\| \times (dmax + s)^{\|V_P\|})$ où $dmax$ est le degré maximal dans M et s le nombre de cliques de taille $\|V_P\|$ dans M. On peut borner dmax par le nombre de sommets de M. Soit une complexité d'énumération

de cliques en $O(n_G \times n_P \times (n_G \times n_P + s)^{n_P})$. Enfin, la construction des solutions est réalisé avec une complexité temporelle $O(s \times n_P)$. On obtient donc une complexité temporelle totale en $O(n_G^2 \times n_P^2 \times (m_G + m_P) + n_G \times n_P \times (n_G \times n_P + s)^{n_P} + s \times n_P)$ que l'on peut simplifier en $O((n_G \times n_P + s)^{n_P})$.

4.2.3 Énumération des sous-graphes temporels isomorphes par suite d'arêtes

Je m'appuie ensuite sur l'article de Paranjape et al. [40] pour apporter une solution au problème d'énumération des sous-graphes temporels isomorphes par suite d'arêtes.

En effet, dans cet article, une solution au problème d'énumération des sous-graphes temporels de L isomorphes par suite d'arêtes à P est proposée. Cette solution commence par énumérer les sous-graphes statiques de L_∞ isomorphes à P_∞, où L_∞ est la compression à l'infini de L et P_∞ est la compression à l'infini de P. Puis pour chacun de ces isomorphismes statiques, à énumérer les listes d'arêtes temporelles de L satisfaisant la définition du problème d'énumération de sous-graphes isomorphes par suite d'arêtes.

Grâce à l'algorithme 18 de construction du graphe de conformité de deux graphes statiques et à l'algorithme 17 d'énumération des cliques de taille k dans un graphe statique, je peux énumérer les isomorphismes de sous-graphes de P_∞ vers L_∞. Pour chacun de ces isomorphismes $f : V' \to A \subseteq V$ ainsi obtenu, je n'ai plus qu'à énumérer les suites d'arêtes $(u_k)_{k\in[\![1,\|E'\|]\!]}$ remplissant les autres conditions de la définition d'isomorphisme par suite d'arêtes.

Pour chaque isomorphisme f, je vais donc itérer sur l'ensemble des arêtes restreintes à E_A de L. Par cette itération, j'assure la condition 1 de la définition d'isomorphisme par suite d'arêtes. J'ordonnerai E_A de sorte d'assurer la condition 2.

Puis je parcours E_A et dès que je trouve une arête $e \in E_A$ telle qu'il existe une arête e' de E'_0 dont e est l'image par f, je créé une nouvelle suite $(u_k)_{k\in[\![1,\|E'\|]\!]}$ avec $u_1 = e$ et j'associe à cette suite un entier $ordre \in \mathbb{N}$ que j'initialise à $ordre = 0$ et un ensemble d'arêtes B qui va lister toutes les arêtes de E'_0 qui doivent encore trouver leur image à l'instant temporel de L auquel existe e. Si une arête de e' a pour image e par f, pour toutes les suites $(u_k)_{k\in[\![1,\|E'\|]\!]}$ créée par cet algorithme, si l'$ordre$ associé à cette suite est égal à l'instant temporel t' auquel existe e', que $e' \in B$ et que e existe au même instant temporel que la dernière arête introduite dans cette suite, j'ajoute e à la suite et je supprime e' de B. En revanche, si e existe à un instant temporel suivant celui auquel existe la dernière arête introduite, la condition 4 de la définition de l'isomorphisme par suite d'arêtes ne peut plus être satisfaite et la suite est supprimée. Si e existe à un instant temporel qui précède celui auquel existe la dernière arête introduite, je crée une nouvelle suite, qui sera identique à celle que l'on traite, sauf qu'on ne copiera pas les arêtes qui sont images d'arêtes de P existant à l'instant temporel t'. Cette nouvelle suite se voit associer l'$ordre = t'$ et $B = E'_{t'} \backslash e'$. Si B est vide et que l'instant temporel auquel existe e' est strictement supérieure à l'instant temporel auquel existe la dernière arête introduite dans la suite, on incrémente $ordre$ et on rend B égal à E'_{ordre}, avant d'insérer e dans la suite, et de retirer e' de B. Si e existe à un instant temporel distant d'au moins Δ instants temporels de la première arête de la suite, on supprime la suite de la liste des candidats. Par contre, si la suite contient $\|E'\|$ arêtes, j'ai construit une suite solution, que j'ajoute donc à ma liste de solution et que je retire de celle des candidats.

Pour alléger la lecture de l'algorithme, je commence par définir les opérations suivantes :

— L'opération de création de E'_{ordre} est décrite dans l'algorithme 19. Elle est réalisée par itération sur E' et pour chaque arête, par réalisation d'une opération en temps constant, pour une complexité temporelle totale en $O(m')$ où m' est le nombre d'arêtes de P.

— L'opération d'initialisation d'un tuple du calcul d'isomorphisme par suite d'arêtes est décrit dans l'algorithme 20 et est réalisée en temps constant à l'exception de l'utilisation de l'opération $ConstructE'_{ordre}(0, E')$ qui est réalisée avec une complexité en $O(m')$. Cette opération a donc une complexité en $O(m')$.

— L'opération d'ajout d'arête temporelle à un tuple est décrit dans l'algorithme 21 et est réalisée en temps constant.

— L'opération de construction de E_A est donnée dans l'algorithme 22. Elle itère sur E et pour chaque arête temporelle, elle réalise une opération en temps constant. Le complexité du tri de E_A est linéaire de la taille de E_A, donc au pire cas bornée par le nombre d'arêtes de L. La complexité de cette opération est donc en $O(m)$ où m est le nombre d'arêtes de L.
— L'opération de copie tronquée d'une suite d'arêtes est donnée en algorithme 23. Cette opération itère sur $(u'_k)_{k\in[\![1,\|E'\|]\!]}$ jusqu'à ce que $E'[c].t = t'$, ce qui arrive avant la fin de la suite puisque je n'exécute cette opération que si $t' < ordre$ donc si $\exists k \in [\![1, \|E'\|]\!], E'[k].t \geqslant t'$, et que par définition de P, à chaque instant temporel t' de T', $P_{t'}$ n'est pas vide. Donc, au pire cas, j'explore tout $(u'_k)_{k\in[\![1,\|E'\|]\!]}$, qui est au pire cas de taille égale à $\|E'\|$. Pour chaque itération, je réalise une opération en temps constant. Suite à cette itération, je réalise des opérations en temps constant puis une opération $ConstructE'_{ordre}(ordre', E')$ de complexité $O(m')$. J'obtiens donc une complexité totale de cette opération en $O(m')$.

Data: Un entier $ordre \in \mathbb{N}$ et un ensemble d'arête E'
Result: L'ensemble d'arêtes E'_{ordre}
Je pose B un ensemble d'arêtes initialement vide.
for *Toute arête temporelle* $(u', v', t') \in E'$ **do**
  if $t' = ordre$ **then**
    J'ajoute (u', v', t') à B.
  end
end
return B.

Algorithm 19: Opération de création de E'_{ordre}

Data: Une arête temporelle (u', v', t') et un ensemble d'arêtes E'
Result: Un tuple $((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B)$ initialisé pour le calcul d'isomorphisme par suite d'arêtes
Je pose $(u_k)_{k\in[\![1,\|E'\|]\!]}$ une suite d'arêtes initialement vide.
$u_1 \leftarrow (u', v', t')$.
Je pose $B \leftarrow ConstructE'_{ordre}(0, E')$.
return $((u_k)_{k\in[\![1,\|E'\|]\!]}, 0, B)$

Algorithm 20: Opération d'initialisation d'un tuple du calcul d'isomorphisme par suite d'arêtes

Data: Un tuple $((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B)$, une arête temporelle e' et un booléen $progress$
Result: Le tuple $((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B)$ mis à jour
if $progress = true \wedge t' > u_{\|(u_k)_{k\in[\![1,\|E'\|]\!]}\|}.t$ **then**
  J'ajoute e' à $(u_k)_{k\in[\![1,\|E'\|]\!]}$.
  Je retire e' de B.
end
if $progress = false \wedge t' = u_{\|(u_k)_{k\in[\![1,\|E'\|]\!]}\|}.t$ **then**
  J'ajoute e' à $(u_k)_{k\in[\![1,\|E'\|]\!]}$.
  Je retire e' de B.
end
return $((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B)$.

Algorithm 21: Opération d'ajout d'arête temporelle à un tuple $((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B)$ dans le cadre du calcul d'isomorphisme par suite d'arêtes

Je peux ainsi définir l'algorithme énumérant toutes les suites d'arêtes $(u_k)_{k\in[\![1,\|E'\|]\!]}$ pour une bijection f donnée tel que $(f, (u_k)_{k\in[\![1,\|E'\|]\!]})$ soit solution du problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes entre L et P. Je détaille cet algorithme dans l'algorithme 24.

Cet algorithme étant défini, je calcule théoriquement sa complexité temporelle. Cet algorithme commence par construire E_A pour une complexité en $O(m)$. Puis il itère sur E_A qui est au pire

Data: Un ensemble d'arête E et un ensemble de sommets $A \subseteq V$
Result: L'ensemble E_A.
Je pose E_A un ensemble d'arêtes initialement vide.
for *toute arête temporelle* $(u, v, t) \in E$ **do**
  if $u \in A \wedge v \in A$ **then**
    J'ajoute (u, v, t) à E_A.
  end
end
Je trie E_A par t croissant.
return E_A.

Algorithm 22: Opération de création de E_A

Data: Un tuple $((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B)$, une arête (u, v, t), une arête (u', v', t'), une bijection $f : V' \to A \subseteq V$ et un ensemble d'arêtes E'
Result: Un tuple $((u'_k)_{k\in[\![1,\|E'\|]\!]}, ordre', B')$
Je pose $(u'_k)_{k\in[\![1,\|E'\|]\!]}$ une suite d'arêtes initialement vide.
Je pose $ordre' = 0$.
Je pose $c = 0$.
while $ordre' \leqslant t'$ **do**
  $u'_c \leftarrow u_c$.
  $ordre' \leftarrow E'[c].t$.
  $c++$.
end
$u'_c \leftarrow (u, v, t)$.
$ordre' \leftarrow t'$.
Je pose $B' \leftarrow ConstructE'_{ordre}(ordre', E')$.
J'enlève (u', v', t') de B'.
return $((u'_k)_{k\in[\![1,\|E'\|]\!]}, ordre', B')$.

Algorithm 23: Opération de création de séquence copiée sur une autre séquence et tronquée au niveau d'une arête

Data: Deux graphes statiques $L(V, E, T)$ et $P(V', E', T')$, une bijection $f : V' \to A \subseteq V$ et un entier $\Delta \in \mathbb{N}$
Result: Une liste de paires $(f, (u_k)_{k\in[\![1,\|E'\|]\!]})$ constituant une solution au problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes de L vers P
Soit *Solution* une liste de paires $(f, (u_k)_{k\in[\![1,\|E'\|]\!]})$ initialement vide.
Soit $Temp$ une liste de tuples $((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B)$, initialement vide.
Je pose $E_A \leftarrow ConstructEA(E, A)$.
for *toute arête temporelle* $(u, v, t) \in E_A$ **do**
  Je pose $i = 0$.
  for *toute arête temporelle* $(u', v', t') \in E'$ **do**
    if $(f(u') = u \wedge f(v') = v) \vee (f(u') = v \wedge f(v') = u)$ **then**
      if $t' = 0$ **then**
        J'ajoute $InitialiseSequence((u', v', t'), E')$ à $Temp$.
      end
      for *Tout tuple* $((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B) \in Temp$ **do**
        Je note $progress \leftarrow false$.
        if $(t - u_1.t > \Delta) \vee (B \neq \varnothing \wedge t > u_{\|(u_k)_{k\in[\![1,\|E'\|]\!]}\|}.t)$ **then**
          Je retire $((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B)$ de $Temp$.
        end
        else
          if B *vide* **then**
            if $ordre = \|T'\| - 1$ **then**
              On ajoute $\{f, (u_k)_{k\in[\![1,\|E'\|]\!]}\}$ à *Solution*.
              On retire $((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B)$ de $Temp$.
            end
            else
              $ordre + +$.
              $B \leftarrow ConstructE'_{ordre}(ordre, E')$.
              $progress \leftarrow true$.
            end
          end
        end
        if $t' = ordre$ **then**
          if $(u', v', t') \in B$ **then**
            $AddToSequence(((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B), (u', v', t'), progress)$.
          end
        end
        else
          if $t' < ordre$ **then**
            Ajouter
            $BuildNewSequence((u, v, t), (u', v', t'), f, ((u_k)_{k\in[\![1,\|E'\|]\!]}, ordre, B), E')$
            à $Temp$.
          end
        end
      end
    end
  end
end
return *Solution*

Algorithm 24: Algorithme permettant de calculer toutes les suites d'arêtes répondant au problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes entre deux graphes temporels pour un isomorphisme de sous-graphes statiques f donné

cas de la même taille que E. Pour chaque arête temporelle sélectionnée par cette première boucle, j'itère sur E'. Pour chaque arête temporelle $(u', v', t') \in E'$, si $t' = 0$, je réalise une opération en $O(m')$ Puis j'itère sur les suites construites par les précédentes itérations. Je peux considérer que le nombre de suites construites par les précédentes itérations sont de l'ordre de $O(S)$ où S est la taille de la solution de l'algorithme. Plusieurs cas se présentent alors :

— Si t' est distant d'au moins Δ instants temporels de l'instant pour lequel existe l'arête u_1 ou si B n'est pas vide et t' est supérieur à l'instant temporel auquel existe la dernière arête introduite dans la suite $(u_k)_{k\in[\![1,\|E'\|]\!]}$, je réalise une opération en temps constant.
— Sinon, plusieurs cas se présentent :
  — Si B est vide, je réalise un certain nombre d'opérations en temps constant et une opération $ConstructE'_{ordre}(ordre, E')$ de complexité $O(m')$.
  — Si $t' = ordre$, je réalise une opération en temps constant.
  — Si $t' < ordre$, je réalise une opération $BuildNewSequence$ de complexité $O(m')$

Donc au pire cas, dans le deuxième cas de cette alternative, la complexité de l'opération est en $O(m')$. Si bien qu'au pire cas, cette alternative a une complexité en $O(m')$. J'obtiens donc une complexité totale de cet algorithme en $O(m + m \times m'^2 \times S)$ soit $O(m \times m'^2 \times S)$ où m est le nombre d'arêtes de L, m' le nombre d'arêtes de P et S le nombre de solution à cet algorithme.

Néanmoins, la condition 4 de l'isomorphisme de sous-graphes par suite d'arêtes n'est remplie que si la suite d'arêtes temporelles de L considérée est telle que la i-ème arête de cette suite est l'image par l'isomorphisme statique f considéré de la i-ème arête de P. Or les solutions de l'algorithme énumérant pour chaque isomorphisme statique toutes les listes d'arêtes répondant au problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes ne garantissent pas cet ordre. Le seul ordre garanti pour ces solutions est un ordre des arêtes temporelles par instant temporel croissant. Mais si deux arêtes temporelles interviennent au même instant temporel, aucun ordre n'est garanti entre ces deux arêtes. Afin de garantir cet ordre, j'introduis donc un algorithme 25 de tri de la solution.

Data: Une paire $\{f, (u_k)_{k\in[\![1,\|E'\|]\!]}\}$ produite par l'algorithme 24 d'énumération des suites d'arêtes temporelles répondant au problème d'énumération des sous-graphes temporels isomorphes par suite d'arêtes pour un isomorphisme de sous-graphe statique donné
Result: Une paire $\{f, (u'_k)_{k[\![1,\|E'\|]\!]}\}$ où la suite a été réorganisée pour satisfaire la condition 4
J'initialise une suite $(u'_k)_{k\in[\![1,\|E'\|]\!]}$).
for *toute arête temporelle* $(u', v', t') \in E'$ **do**
  Je pose $i = 0$.
  while $((u_i.x \neq u') \vee (u_i.y \neq v')) \wedge ((u_i.x \neq v') \vee (u_i.y \neq u'))$ **do**
    i++.
  end
  J'ajoute u_i à la suite $(u'_k)_{k\in[\![1,\|E'\|]\!]}$).
  J'enlève u_i de la suite $(u_k)_{k\in[\![1,\|E'\|]\!]}$).
end
return $\{f, (u'_k)_{k\in[\![1,\|E'\|]\!]}\}$.

Algorithm 25: Algorithme de tri de la suite d'arêtes solution du problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes

Ce tri itère sur E' et au pire cas sur $(u_k)_{k\in[\![1,\|E'\|]\!]}$ qui est par définition de même taille que E'. Ce tri est donc réalisé en temps $O(m'^2)$ où m' est le nombre d'arêtes de P. Il ne me reste plus qu'à prouver que la condition 4 est remplie par cette construction suivie du tri décrit en algorithme 25.

Propriété 4.2.5. *Soit $L(V, E, T)$ et $P(V', E', T')$ deux graphes temporels. Soit $\{f, (u_k)_{k\in[\![1,\|E'\|]\!]}\}$ une paire construite par l'algorithme 24 et $(u'_k)_{k\in[\![,\|E'\|]\!]}$ la solution de l'algorithme 25 appliqué à $(u_k)_{k\in[\![1,\|E'\|]\!]}$. Alors, pour tout $k \in [\![1, \|E'\|]\!]$, la k-ième arête de $(u'_k)_{k\in[\![,\|E'\|]\!]}$ est l'image de la k-ième arête de E'.*

Démonstration. Supposons $L(V, E, T)$ et $P(V', E', T')$ deux graphes temporels et $(u'_k)_{k\in[\![1,\|E'\|]\!]}$, la solution de l'algorithme 25 de tri de la suite d'arêtes appliqué à $\{f, (u_k)_{k\in[\![1,\|E'\|]\!]}\}$ construite par l'algorithme 24.

Il est trivial de prouver que l'algorithme 25 trie $(u'_k)_{k\in[\![1,\|E'\|]\!]}$ pour que la k-ième arête de la suite soit l'image par f de la k-ième arête de E', à condition que la suite $(u_k)_{k\in[\![1,\|E'\|]\!]}$ contienne une arête qui soit image de chaque arêtes de E'.

Il est également trivial de prouver que l'algorithme 24 crée des suites $(u_k)_{k\in[\![1,\|E'\|]\!]}$ contenant une arête qui soit image par f de chaque arêtes de E', en notant que si il existe une arête $e' \in E'$ qui n'ait pas d'image par f dans $(u_k)_{k\in[\![1,\|E'\|]\!]}$, lorsque e' appartiendra à B dans le déroulé de l'algorithme, e' ne sera pas retiré de B si aucune image par f d'e' n'a été ajoutée à la suite, et que B ne sera donc jamais vide, ne permettant pas l'ajout de la suite à la solution de l'algorithme. □

Ainsi, en appliquant l'algorithme 24 d'énumération des suites d'arêtes temporelles répondant au problème d'énumération des sous-graphes temporels isomorphes par suite d'arêtes pour un isomorphisme de sous-graphe statique donné suivi de l'algorithme 25 de tri de la suite d'arêtes solution du problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes pour un $f : V' \to A \subseteq V$, tel que f est un isomorphisme de sous-graphes statiques de P_∞ vers L_∞, j'énumère bien des solutions $\{f, (u_k)_{k\in[\![1,\|E'\|]\!]}\}$.

Etant donné un isomorphisme de sous-graphes statiques $f : V' \to A \subseteq V$ de P_∞ vers L_∞, je réalise ce calcul avec une complexité en $O(m'^2 + m \times m'^2 \times S)$ donc en $O(m \times m'^2 \times S)$ où m est le nombre d'arêtes de L, m' le nombre d'arêtes de P et S la taille de la solution.

Propriété 4.2.6. *Soit $L(V, E, T)$ et $P(V', E', T')$ deux graphes statiques et une bijection $f : V' \to A \subseteq V$ représentant un isomorphisme de sous-graphes statiques de P_{infty} vers L_{infty}. Soit $(u_k)_{k\in[\![1,\|E'\|]\!]}$ une suite d'arêtes de E. Si la paire $\{f, (u_k)_{k\in[\![1,\|E'\|]\!]}\}$ est une solution du problème d'énumération des sous-graphes temporels isomorphes par suite d'arêtes pour P et L, alors $\{f, (u_k)_{k\in[\![1,\|E'\|]\!]}\}$ appartient à la solution de l'algorithme 24 d'énumération des suites d'arêtes temporelles répondant au problème d'énumération des sous-graphes temporels isomorphes par suite d'arêtes pour un isomorphisme de sous-graphe statique donné appliqué à L, P et f et trié par l'algorithme 25 de tri de la suite d'arêtes solution du problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes.*

Démonstration. Soit $L(V, E, T)$ et $P(V', E', T')$ deux graphes statiques, $f : V' \to A \subseteq V$ un isomorphisme de sous-graphes statiques de P_∞ vers L_∞ et $(u_k)_{k\in[\![1,\|E'\|]\!]}$ telle que $\{f, (u_k)_{k\in[\![1,\|E'\|]\!]}\}$ est une solution du problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes de P et L.

Par définition du problème d'isomorphisme de sous-graphes temporels par suite d'arêtes, pour tout $k \in [\![1, \|E'\|]\!]$, $u_k \in E_A$, donc u_k sera parcourue par l'algorithme 24, qui trie également E_A par instant temporel croissant. Par définition également, u_k est l'image par f de la k-ième arête de E', et la progression des instants temporels des arêtes de la suite est respectée.

Pour tout $i \in [\![0, \|T'\|]\!]$, j'appelle U_i l'ensemble des arêtes de $(u_k)_{k\in[\![1,\|E'\|]\!]}$ qui sont images par f des arêtes de E'_i. Lorsque l'algorithme va itérer sur une arête de U_0 pour la première fois, une suite $(a_k)_{k\in[\![1,\|E'\|]\!]}$ est initialisée avec cette arête. Pour tout $i > 0$, l'algorithme va itérer sur une arête e_i de U_i pour la première fois, et comme, par définition du problème d'isomorphisme de sous-graphes temporels par suite d'arêtes, la suite $(u_k)_{k\in[\![1,\|E'\|]\!]}$ est triée par instant temporel croissant, toutes les arêtes de tous les U_j pour $j < i$ ont déjà été parcourues et ajoutées à la suite $(a_k)_{k\in[\![1,\|E'\|]\!]}$, l'arête e_i va être ajoutée à cette suite si aucune arête de cette suite n'est l'image par f de $f^{-1}(e_i)$. S'il existe déjà une telle arête, l'algorithme va copier dans une nouvelle suite $(b_k)_{k\in[\![1,\|E'\|]\!]}$ dans laquelle il va copier toutes les arêtes de tous les U_j pour $j < i$ et l'arête e_i.

Une fois la première arête d'un U_i ajoutée à une suite $(a_k)_{k\in[\![1,\|E'\|]\!]}$, lorsque le parcours de E_A va itérer sur chacune des autres arêtes de U_i, elles seront ajoutées à la suite $(a_k)_{k\in[\![1,\|E'\|]\!]}$.

Donc l'algorithme produit une suite $(a_k)_{k\in[\![1,\|E'\|]\!]}$ contenant toutes les arêtes de $(u_k)_{k\in[\![1,\|E'\|]\!]}$, mais pas forcément ordonnée. Cela signifie que si on applique l'algorithme 25 à $(a_k)_{k\in[\![1,\|E'\|]\!]}$, on obtient bien la suite $(u_k)_{k\in[\![1,\|E'\|]\!]}$. □

Maintenant que je suis capable d'énumérer, pour un isomorphisme statique f de P_∞ vers L_∞ toutes les suites d'arêtes temporelles $(u_k)_{k\in[\![1,\|E'\|]\!]}$ telles que $\{f,(u_k)_{k\in[\![1,\|E'\|]\!]}\}$ est une solution du problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes, je peux l'appliquer à la suite de l'algorithme énumérant les isomorphismes de sous-graphes statiques de P_∞ vers L_∞ défini dans la sous-section précédente. Je suis désormais capable, à partir de P et L deux graphes temporels, d'énumérer toutes les solutions au problème d'énumération sous-graphes temporels isomorphes par suite d'arêtes de P vers L.

La complexité de la recherche des isomorphismes de sous-graphes statiques de P_∞ vers L_∞ est réalisé en $O(n^2 \times n'^2 \times (m+m') + n \times n' \times (n \times n' + s)^{n'} + s \times n')$. J'ai donc une complexité totale en $O(n^2 \times n'^2 \times (m_\infty + m'_\infty) + n \times n' \times (n \times n' + s)^{n'} + s \times n' + F \times m \times m'^2 \times S)$ où n est le nombre de sommets de L, n' le nombre de sommets de P, m le nombre d'arêtes de L, m' le nombre d'arêtes de P, m_∞ le nombre d'arêtes de L_∞, m'_∞ le nombre d'arêtes de P_∞, F le nombre d'isomorphismes de sous-graphes statiques de P_∞ vers L_∞, s le nombre de cliques de taille n' dans M et S le nombre de suites d'arêtes maximal pour former avec un isomorphisme donné une solution au problème. J'ai déjà établi qu'une clique de taille n' dans M correspondait à un isomorphisme statique de P_∞ vers L_∞ donc $s = F$. Il va de soi que chaque arête statique de L_∞ existant du fait d'une arête temporelle de L, m_∞ est bornée par m et m'_∞ est bornée par m'. Je peux également établir que $m' < m$ sinon, aucun isomorphisme ne serait possible entre P et L. Les solutions de cet algorithme sont des énumérations de toutes les suites solutions pour tous les isomorphismes statiques entre P_∞ et L_∞ donc $F \times S$ forme la taille de la solution. J'obtiens donc une complexité au pire cas en $O(n \times n' \times (n \times n' + F)^{n'} + Iso \times m \times m'^2)$ où n est le nombre de sommets dans L, n' le nombre de sommets dans P, m le nombre d'arêtes dans L, m' le nombre d'arêtes dans P, F le nombre d'isomorphismes statiques entre P_∞ et L_∞ et Iso est la taille de la solution du problème. La complexité spatiale de cet algorithme consiste en la complexité spatiale de l'algorithme énumérant les sous-graphes statiques isomorphes entre L_∞ et P_∞ à laquelle on doit ajouter la complexité spatiale introduite par cette nouvelle étape, à savoir la taille de la solution, puisque pour chaque isomorphisme de sous-graphes statiques entre L_∞ et P_∞, on va énumérer toutes les listes d'arêtes temporelles de L répondant aux conditions posées par la définition du problème d'énumération des sous-graphes temporels isomorphes par suite d'arêtes. La complexité spatiale totale de cet algorithme est donc en $O(n \times n' + m \times m' + (n^2 - m) \times (n'^2 - m') + n' \times F + Iso)$.

J'ai ainsi mis au point un algorithme énumérant les sous-graphes temporels isomorphes par suite d'arêtes entre deux graphes temporels P et L. Cet algorithme a une complexité temporelle en $O(n \times n' \times (n \times n' + F)^{n'} + Iso \times m \times m'^2)$ et une complexité spatiale en $O(n \times n' + m \times m' + (n^2 - m) \times (n'^2 - m') + n' \times F + Iso)$, où n est le nombre de sommets dans L, n' le nombre de sommets dans P, m le nombre d'arêtes dans L, m' le nombre d'arêtes dans P, F le nombre d'isomorphismes statiques entre P_∞ et L_∞ et Iso est la taille de la solution du problème. Ces complexités sont donc indépendantes des tailles d'historique τ et τ' des deux graphes temporels considérés, ce qui correspond au but que je visais. Il est néanmoins à noter que ces complexités sont polynomialement proportionnelles au nombre d'arêtes temporelles de L et de P mais également exponentiellement proportionnelle au nombre de sommets de P, ce qui signifie que dans la recherche de sous-graphes temporels isomorphes par suite d'arêtes à un motif contenant un grand nombre de sommets, le temps de calcul est susceptible de quitter les limites du raisonnable que je me suis fixées précédemment. Néanmoins, si le motif considéré est de petite taille (par rapport au graphe L), cet algorithme devrait être en mesure d'énumérer les sous-graphes temporels isomorphes par suite d'arêtes en temps raisonnable. Dans le cadre de cette thèse, les jeux de données utilisés présentant un faible nombre de sommets mais une taille d'historique importante, l'indépendance des complexités en τ est très encourageante, malgré les influences importantes des autres paramètres du problème.

4.2.4 Énumération de sous-graphes temporels isomorphes

Maintenant que j'ai apporté une solution au problème d'énumération sous-graphes temporels isomorphes par suite d'arêtes, je vais expliquer comment je peux modifier l'algorithme obtenu pour

énumérer les sous-graphes temporels isomorphes.

Je rappelle que par définition, je cherche $f : V' \to A \subseteq V$ bijective et $t_0 \in T$, tels que toute paire de sommets $\{u', v'\} \in V'^2$ est liée par une arête à l'instant $t' \in T'$ dans P si et seulement si $f(u')$ et $f(v')$ sont liés par une arête à l'instant $t' + t_0$ dans L.

Malheureusement, là où l'isomorphisme par suite d'arêtes de sous-graphes temporels pouvait bénéficier du sous-traitement consistant à énumérer les isomorphismes de sous-graphes statiques, ce n'est plus le cas ici car si $f : V' \to A \subseteq V$ bijective est telle que $\exists t_0 \in T', \{f, t_0\}$ forme un isomorphisme de sous-graphes temporels, f n'est pas nécessairement un isomorphisme de sous-graphes statiques entre P_∞ et L_∞, où P_∞ est la compression à l'infini de P et L_∞ la compression à l'infini de L. Prenons un contre-exemple pour le prouver. Dans la figure 4.3, si je pose $f : V' \to \{A, B, C, D\} \subseteq V$ telle que $f(A') = A$, $f(B') = B$, $f(C') = C$, $f(D') = D$ et $t_0 = 1$, $\{f, t_0\}$ forme bien un isomorphisme de sous-graphes temporels entre L et P et pourtant, f n'est pas un isomorphisme de sous-graphes statiques entre L_∞ et P_∞ car $(A', D') \notin E'$ mais $(f(A'), f(D')) = (A, D) \in E$.

Je ne peux donc pas utiliser le même pré-traitement que pour le problème d'énumération des sous-graphes temporels isomorphes, qui consistait à énumérer les isomorphismes de sous-graphes statiques. Dans le but d'établir un pré-traitement qui permettra néanmoins de diminuer le temps de calcul de tout algorithme énumérant les sous-graphes temporels isomorphes, je m'inspire de ce pré-traitement et cherche une propriété qui permettrait de disqualifier certaines bijections possibles entre les sommets de L et les sommets de P. Par exemple, une paire de sommets $\{u, v\} \in V^2$ ne peut être image par un isomorphisme de sous-graphes temporels d'une paire $\{u', v'\} \in V'^2$, s'il existe un instant temporel $t' \in T'$ tel que u' et v' sont liés par une arête à l'instant t' dans P, mais qu'il n'existe aucun instant temporel de L tel que u et v sont liés par une arête temporelle dans L.

Je définis un graphe de conformité altéré de P_∞ et L_∞ de la manière suivante :

Définition 4.2.5. Soit $G(V_G, E_G)$ et $P(V_P, E_P)$ deux graphes statiques. Je définis le **graphe de conformité de G et P altéré pour le problème d'énumération des sous-graphes temporels** $M(V_G \times V_P, E_m)$ tel qu'il existe une bijection $g : V_G \times V_P \to V_m$ telle qu'il existe dans M une arête entre deux sommets $g(u, u')$ et $g(v, v')$, pour $\{u, v\} \in V_G^2$ et $\{u', v'\} \in V_P^2$, si et seulement si u et v sont liés par une arête dans G ou u' et v' ne sont pas liés par une arête dans P.

Ce graphe de conformité altéré pour le problème d'énumération de sous-graphes temporels $M(V_G \times V_P, E_m)$ de deux graphes statiques $G(V_G, E_G)$ et $P(V_P, E_P)$ est donc une version du graphe de conformité de G et P contenant des arêtes supplémentaires (celles correspondant aux compatibilités entre les sommets $g(u, u')$ et $g(v, v')$ composant les arêtes $(u, v) \in E_G$ et les non-arêtes $(u', v) \notin E_P$). Ceci altère la complexité spatiale de M, et si le nombre de sommets est identique, la complexité spatiale de ce graphe est au pire cas :

— $n_G \times n_P$ sommets, où n_G est le nombre de sommets de G et n_P le nombre de sommets de P.
— $n_G^2 \times (n_P^2 - m_P) + m_G \times m_P$ où m_G est le nombre d'arêtes de G et m_P le nombre d'arêtes de P.

Afin de construire ce graphe de conformité altéré pour le problème d'énumération de sous-graphes temporels isomorphes, j'utilise l'algorithme 26.

Cette opération ne diffère de la construction du graphe de conformité de G et P que par le retrait d'une opération de contrôle. Cette modification n'a pas d'impact sur la complexité de l'opération. Cette opération présente donc une complexité au pire cas en $O(n_G^2 \times n_P^2 \times (m_G + m_P))$ où n_G est le nombre de sommets de G, n_P le nombre de sommets de P, m_G le nombre d'arêtes de G et m_P le nombre d'arêtes de P.

J'applique ensuite sur ce graphe MV_m, E_m de conformité altéré pour le problème d'énumération des sous-graphes temporels de L_∞ et P_∞ l'algorithme de recherche de cliques de tailles n', où n' est le nombre de sommets de P, défini dans une sous-section précédente. La modification de la dimension du graphe de conformité va impacter la complexité de l'algorithme d'énumération des cliques de taille n' puisque le nombre d'arêtes va augmenter et que, le nombre de sommets étant constant, le graphe M va gagner en densité, ce qui va augmenter le nombre de cliques et le degré maximal dans M, dont la complexité dépend. La complexité de l'énumération de cliques de taille n' dans le graphe

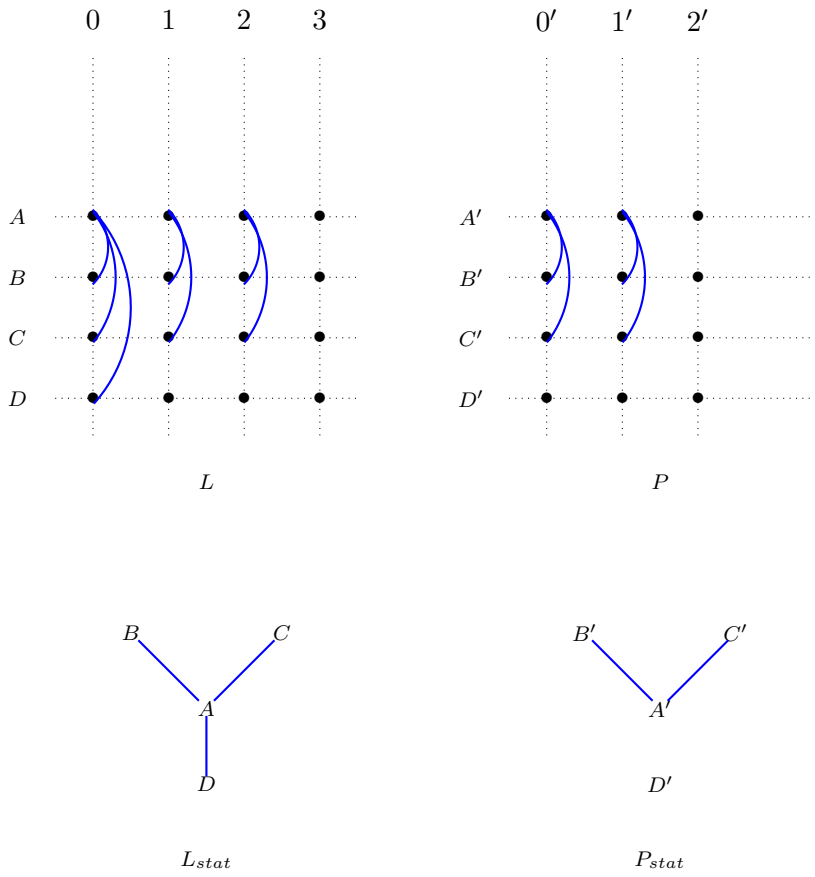

FIGURE 4.3 – Contre-exemple prouvant l'absence de lien entre isomorphisme statique et isomorphisme temporel

Data: $G(V_G, E_G)$ et $P(V_P, E_P)$ deux graphes statiques
Result: $M(V_M, E_m)$ le graphe de conformité altéré pour le problème d'isomorphisme de sous-graphe temporel de G et P
J'initialise M vide.
for *tout sommet* $u \in V_G$ **do**
  for *tout sommet* $u' \in V_P$ **do**
    Je définis $v = g(u, u')$.
    J'ajoute v à V_m.
  end
end
for *tout sommet* $u \in V_G$ **do**
  for *tout sommet* $v \in V_G$ **do**
    for *tout sommet* $u' \in V_P$ **do**
      for *tout sommet* $v' \in V_P$ **do**
        if $(u, v) \in E_G$ **then**
          J'ajoute $(g(u, u'), g(v, v'))$ à E_m.
          J'ajoute $(g(u, v'), g(v, u'))$ à E_m.
        end
        else
          if $(u', v') \in E_P$ **then**
            Je retire $(g(u, u'), g(v, v'))$ d'E_m.
            Je retire $(g(u, v'), g(v, u'))$ d'E_m.
          end
          else
            J'ajoute $(g(u, u'), g(v, v'))$ à E_m.
            J'ajoute $(g(u, v'), g(v, u'))$ à E_m.
          end
        end
      end
    end
  end
end
return M.

Algorithm 26: Algorithme de construction du graphe de conformité altéré pour le problème d'énumération de sous-graphes temporels isomorphes de deux graphes statiques

de conformité altéré pour le problème d'isomorphisme de sous-graphe temporel M de G et P est donc en $O(\|V\|times\|V'\| \times (dmax + s)^{\|V'\|})$ où $dmax$ est le degré maximal dans M et s la taille de la solution. J'ai donc une complexité totale en $O(n^2 \times n'^2 \times (m+m') + n \times n' \times (n \times n' + s)^{n'} + s \times n')$ pour le calcul de toutes les bijections $f : V' \to A \subseteq V$ telles que pour toute paire de sommets $\{u, v\} \in A^2$, si u et v ne sont liés à aucun instant dans L, alors $f^{-1}(u)$ et $f^{-1}(v)$ ne sont liés à aucun instant dans P. La complexité spatiale de la construction du graphe de conformité altéré suivi de l'algorithme de recherche de cliques de taille n' est donc modifiée par la modification de taille de M par rapport à la sous-section précédente en $O(n^2 \times (n'^2 - m') - m \times m')$.

Pour chacune des solutions restantes, il convient ensuite de modifier l'algorithme de traitement défini pour le problème d'énumération de sous-graphes temporels isomorphes par suite d'arêtes pour prendre en compte les différences entre les deux problèmes.

Pour chaque bijection $f : V' \to A \subseteq V$, je considère E_A. J'itère ensuite sur T jusqu'à trouver un $t_0 \in T$ tel que $E_{At_0} = f(E'_0)$. Si un tel t_0 est trouvé, $\{f, t_0\}$ devient candidat pour appartenir à la solution de l'énumération de sous-graphes temporels isomorphes entre L et P. Je continue alors à itérer sur T, et pour chaque $t \in T$, je vérifie pour chaque candidat $\{f, t_0\}$ si $E_{At} = f(E'_{t-t_0})$. Si c'est le cas, $\{f, t_0\}$ reste candidat. Sinon, $\{f, t_0\}$ n'est plus candidat à appartenir à la solution. Si $\{f, t_0\}$ reste candidat jusqu'à $t = t_0 + \|T'\|$, $\{f, t_0\}$ est définitivement ajouté à la solution. J'obtiens ainsi l'algorithme 27.

Je vais m'employer à démontrer théoriquement la correction et la complétude de cet algorithme comme solution au problème d'énumération des sous-graphes temporels isomorphes.

Propriété 4.2.7. *Soit deux graphes temporels $L(V, E, T)$ et $P(V', E', T')$. Soit une bijection $f : V' \to A \subseteq V$ et $t_0 \in T$. $\{f, t_0\}$ constitue un isomorphisme de sous-graphes temporels entre L et $P \iff$ (f correspond à une clique de taille n' du graphe de conformité altéré pour le problème d'énumération de sous-graphes temporels isomorphes de P et L, et t_0 appartient à la solution de l'algorithme 27 d'énumération des sous-graphes temporels isomorphes pour une bijection donnée appliqué à P, L et f). Cet algorithme présente une complexité temporelle linéaire en la taille de l'historique et une complexité spatiale en la taille de l'historique.*

Démonstration. Soit $L(V, E, T)$ et $P(V', E', T')$ deux graphes temporels, $f : V' \to A \subseteq V$ une bijection et $t_0 \in T$.

— $\Leftarrow$

Supposons que $\{f, t_0\}$ constitue un isomorphisme de sous-graphes temporels entre L et P.
Il est trivial de prouver que f correspond à une clique de taille n' du graphe de conformité altéré pour le problème d'énumération de sous-graphes temporels isomorphes de P_∞ et L_∞, puisque par définition du problème d'isomorphisme de sous-graphes temporels, pour tout $\{u, v\} \in V'$, s'il existe un instant temporel de P auquel u et v sont liés par une arête temporelle dans P, il existe un instant temporel de L auquel $f(u)$ et $f(v)$ sont liés par une arête temporelle.
L'algorithme 27 appliqué à L, P et à f, va itérer sur T. Pour t_0, puisque $\{f, t_0\}$ est un isomorphisme de sous-graphes temporels, $E_t = f(E'_0)$, et $\{f, t_0\}$ va être ajouté à la liste de candidats. $\{f, t_0\}$ ne sera retiré de cette liste que si il existe un instant $t' \in T'$ tels que $f(E'_{t'}) \neq E_{t+t_0}$, ce qui contredit la définition de l'isomorphisme de sous-graphes temporels. Donc $\{f, t_0\}$ appartient à la solution de l'algorithme 27 appliqué à P, L et f.

— $\Rightarrow$

Il est trivial de prouver la réciproque puisque pour toute bijection $h : V' \to B \subseteq V$, un instant temporel t_1 est fourni dans la solution de l'algorithme 27 appliqué à L, P et à f, si pour tout instant temporel $t' \in T'$, $h(E'_{t'}) = E_{t_1+t'}$, ce qui me permet trivialement de prouver que tout $\{h, t_1\}$ et donc parmi eux $\{f, t_0\}$ est un isomorphisme de sous-graphes temporels entre P et L.

On établit théoriquement la complexité temporelle de cey algorithme. Cet algorithme itère sur toutes les bijections $f : V' \to A \subseteq V$ obtenues par les opérations précédentes d'énumération de cliques de taille n' dans le graphe de conformité altéré pour le problème d'énumération de sous-graphes temporels isomorphes M entre P_∞ et L_∞, qui sont donc de même nombre que le nombre de cliques de taille n' dans M. Pour chacune de ces bijections, on construit E_A, de taille au pire cas m,

Data: Deux graphes temporels $L(V, E, T)$ et $P(V', E', T')$ et une liste de bijections $f : V' \to A \subseteq V$
Result: Une liste de $\{f, t_0\}$ solutions du problème d'énumération de sous-graphes temporels isomorphes

```
Je pose Solution une liste de {f, t_0} initialement vide.
for toute bijection f : V' → A ⊆ V donnée en argument do
    Je pose E_A = ConstructEA(E, A)
    Je pose Temp une liste de {f, t_0} initialement vide.
    for tout instant temporel t ∈ T do
        E_t = ConstructE'_ordre(t, E_A)
        Je pose E_temp une liste d'arêtes initialement vide.
        for toute arête temporelle e' ∈ E' do
            if e'.t = 0 then
                J'ajoute (f(e'.x), f(e'.y), t) à E_temp.
            end
        end
        if E_temp = E_t then
            J'ajoute {f, t} à Temp.
        end
        for tout candidat {f, t_0} ∈ Temp do
            if t − t_0 > ‖T'‖ then
                Ajouter {f, t_0} à Solution.
                Retirer {f, t_0} de Temp.
            end
            else
                Je pose E_temp une liste d'arêtes initialement vide.
                for toute arête temporelle e' ∈ E' do
                    if e'.t = t − t_0 then
                        J'ajoute (f(e'.x), f(e'.y), t) à E_temp.
                    end
                end
                if E_temp ≠ E_t then
                    Je retire {f, t_0} de Temp.
                end
            end
        end
    end
end
return Solution.
```

Algorithm 27: Algorithme d'énumération des sous-graphes temporels isomorphes pour une bijection f donnée

en $O(m)$ puis on itère sur T. Dans cette seconde boucle, on construit E_{A_t} en $O(m)$, puis on itère sur E' pour construire E'_0 avant de procéder à une vérification d'égalité entre E_{A_t} et $f(E'_0)$, réalisée en temps linéaire de m'_T où m'_T est la taille maximale sur T de E'_t. Puis on procède à une itération sur $Temp$ qui sera de taille au pire cas le nombre de $t_0 \in T$ tels que $E_{A_{t_0}} = E'_0$, donc qui sera de l'ordre du nombre de solution $0\{g, t_0\}$ au problème telles que $g = f$, que l'on notera iso_f. Pour chacun de ces candidats $\{f, t_0\}$ on procède au pire cas à une itération sur E' pour construire E'_{t-t_0} suivi d'une vérification d'égalité entre E_{A_t} et E'_{t-t_0} réalisée en temps linéaire de m'_t. On obtient donc une complexité au pire cas pour cet algorithme en $O(s \times (m + \tau \times (m + m' + m'_T + iso_f \times (m' + m'_T))))$ où s est le nombre de cliques de taille n' dans M. Cette complexité peut donc être simplifiée en $O(F \times \tau \times iso_f \times m')$ soit en $O(\tau \times m' \times S)$ où $\tau = \|T\|$ est la taille de l'historique de L, m' est le nombre d'arêtes de P et S est la taille de la solution. On obtient donc une complexité totale en $O(n \times n' \times (n \times n' + s)^{n'} + \tau \times m' \times Iso)$ pour énumérer tous les sous-graphes de L isomorphes à P, où n est le nombre de sommets de L, n' le nombre de sommets de P, $\tau = \|T\|$ la taille de l'historique de L, m' le nombre d'arêtes de E', s le nombre de cliques de taille n' dans M et Iso le nombre de sous-graphes temporels isomorphes.

La complexité spatiale de cet algorithme, quant à elle, est proportionnelle à $O(n^2 \times (n'^2 - m') + m \times m' + Iso)$. □

Cette complexité rend l'utilisation de cet algorithme sur des jeux de données issus de données d'exploitation réelles compliquée. Malgré tout, l'algorithme peut fournir des résultats sur des graphes de taille plus modeste.

J'ai ainsi présenté et justifié théoriquement un algorithme énumérant les sous-graphes temporels isomorphes entre deux graphes temporels P et L. Cet algorithme présente une complexité temporelle en $O(n \times n' \times (n \times n' + s)^{n'} + \tau \times m' \times Iso)$ et une complexité spatiale en $O(n^2 \times (n'^2 - m') + m \times m' + Iso)$ pour énumérer tous les sous-graphes de L isomorphes à P, où n est le nombre de sommets de L, n' le nombre de sommets de P, $\tau = \|T\|$ la taille de l'historique de L, m' le nombre d'arêtes de E', s le nombre de cliques de taille n' dans M et Iso le nombre de sous-graphes temporels isomorphes entre P et L. Encore une fois, ces complexités sont exponentiellement dépendantes du nombre de sommets de L et du nombre de sommets de P. La complexité spatiale est également quadratiquement dépendante du nombre de sommets de P. Néanmoins, puisque cette thèse se concentre sur l'influence de τ sur ces complexités, les complexités de cet algorithme restent intéressantes, puisque la complexité temporelle est linéaire en τ lorsque la complexité spatiale est indépendante de τ. Sur les jeux de données traités dans le cadre cette thèse, le nombre de sommets et d'arêtes risquent de poser des problèmes tant en terme de mémoire que de temps de calcul mais l'algorithme présenté dans cette thèse devrait être capable d'énumérer les sous-graphes isomorphes dans des graphes temporels à la taille d'historique élevé si le nombre de sommets est raisonnable. Cela signifie également que le nombre de sommets du motif P doit être faible.

Ce chapitre présente donc trois algorithmes, l'un permettant d'énumérer les sous-forêts temporelles isomorphes à une forêt temporelle linéaire, un second énumérant les sous-graphes temporels isomorphes par suite d'arêtes et un dernier énumérant les sous-graphes temporels isomorphes. Le premier de ces algorithmes ne m'intéresse que pour servir de point de comparaison, puisqu'il est bâti sur une utilisation récursive d'un algorithme calculant une solution au problème statique de l'énumération de sous-forêts isomorphes à une forêt linéaire à chaque instant temporel. L'intérêt de cet algorithme est donc de valider qu'une approche structurelle du problème, telle que celle utilisée pour concevoir les deux autres algorithmes, est préférable. Quant aux deux autres, leurs complexités spatiales et temporelles sont théoriquement linéaires en τ. Dans le chapitre 5, je présente également une étude expérimentale des implémentations de ces algorithmes sur le jeu de données `TimeProgression`, similaire à celle menée sur les implémentations des algorithmes d'énumération de modules éternels et présentée en chapitre 3, afin de confirmer ces influences. Cette linéarité en τ présente un intérêt certain et est conforme au but que je me suis fixé dans le cadre de cette thèse. Cependant, les autres facteurs, tels que le nombre de sommets ou le nombre d'arêtes temporelles dans le graphe temporel considéré, et l'influence de ces facteurs sur les complexités ne sont pas à

négliger, et dans le chapitre 5, j'étudie également le passage à l'échelle de ces algorithmes, afin de déterminer si la linéarité en τ est suffisante pour garantir que ces algorithmes sont capables de calculer une solution en temps raisonnable (moins d'une heure) sur des jeux de données d'exploitation réelle.

Chapitre 5

Expérimentations

Afin de vérifier la cohérence des résultats théoriques, je procède à une étude expérimentale sur des jeux de données issus de données d'exploitation réelle pour vérifier que la théorie présentée dans les chapitres précédents est en accord avec les observations expérimentales.

De plus, outre la validation expérimentale des résultats établis par l'étude théorique présentée dans les chapitres 3 et 4, et notamment la validation de l'influence de τ sur les temps de calcul et les utilisations mémoires des divers algorithmes, je m'intéresse également à la capacité de ces algorithmes à passer à l'échelle et à traiter des jeux de données issus de données de terrain. En effet, si j'ai principalement cherché à diminuer autant que possible l'influence de τ sur les complexité des algorithmes présentés dans cette thèse, d'autres facteurs comme le nombre de sommets dans le graphe temporel considéré ou le nombre d'arêtes temporelles de ce graphe, ont quant à eux une influence importante sur ces complexités. Je cherche donc à établir si, malgré cette forte influence, les algorithmes présentés dans cette thèse parviennent à énumérer les ensembles recherchés dans les jeux de données de terrain considérés en un temps de calcul raisonnable, que je définis dans cette thèse comme inférieur ou de l'ordre de l'heure.

Il est néanmoins à noter que certains des algorithmes présentés dans cette thèse présentent une complexité trop importante pour parvenir à fournir des résultats en temps raisonnable sur les jeux de données qui m'intéressent. Tous les algorithmes présentés dans cette thèse ont été implémentés en Java (version 1.8 Standard Edition). Les expériences ont été réalisées sur un ordinateur portable équipé d'un processeur 4 cores cadencé à 2.7 Ghz et disposant de 32 giga-octets de mémoire vive.

Après avoir présenté les jeux de données considérés dans cette étude 5.1 et avoir précisé quelques détails d'implémentation 5.2, je présenterai une validation expérimentale de l'influence de la taille de l'historique τ du graphe temporel considéré sur les complexités spatiale et temporelle des algorithmes d'énumération des sous-graphes isomorphes 5.3, par le biais d'une étude des résultats de ces algorithmes sur le jeu de données `Timeprogression`.

Dans le cadre de cette section de validation expérimentale de l'influence théoriquement établie de τ sur les complexités des algorithmes d'énumération de sous-graphes temporels isomorphes, je m'intéresse à la validation des propriétés suivantes :

— *Propriété 9* : Le temps de calcul pour l'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire est linéaire en τ.
— *Propriété 10* : L'utilisation mémoire pour l'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire est linéaire en τ.
— *Propriété 11* : Le temps de calcul pour l'énumération des sous-graphes temporels isomorphes par suite d'arêtes est indépendant de τ.
— *Propriété 12* : L'utilisation mémoire pour l'énumération des sous-graphes temporels isomorphes par suite d'arêtes est indépendante de τ.
— *Propriété 13* : Le temps de calcul pour l'énumération des sous-graphes temporels isomorphes est linéaire en τ.
— *Propriété 14* : L'utilisation mémoire pour l'énumération de sous-graphes temporels isomorphes est indépendante en τ.

Si je me suis principalement concentré dans cette thèse sur l'influence de la taille de l'historique τ des graphes temporels sur les complexités spatiales et temporelles des algorithmes, c'est au détriment d'une forte dépendance de ces complexités aux autres paramètres, notamment le nombre de sommets dans les graphes et le nombre d'arêtes temporelles dans ces graphes.

Je m'attaquerai donc ensuite au cœur de cette étude expérimentale en étudiant le passage à l'échelle, en procédant à des expérimentations sur les jeux de données présentés ci-dessous 5.4.

Dans le cadre de la section d'analyse du passage à l'échelle des algorithmes présentés dans cette thèse sur les jeux de données réels, je vais chercher à valider expérimentalement les propriétés suivantes :

— *Propriété 15* : Les Δ-jumeaux peuvent être énumérés en temps raisonnable (moins d'une heure).
— *Propriété 16* : L'algorithme MLEI est capable d'énumérer les Δ-jumeaux d'un graphe temporel pour les graphes temporels qui provoquent des problèmes de manque de mémoire pour l'algorithme MEI.
— *Propriété 17* : Les modules éternels peuvent être énumérés en temps raisonnable (moins d'une heure).
— *Propriété 18* : L'algorithme MLEI est capable d'énumérer les modules éternels dans des graphes temporels sur lesquels l'algorithme MEI rencontre des problèmes de manque de mémoire.
— *Propriété 19* : L'algorithme d'énumération de sous-graphes temporels isomorphes est capable de fournir un résultat plus rapidement que l'algorithme d'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire.
— *Propriété 20* : L'algorithme d'énumération de sous-graphes temporels isomorphes par suite d'arêtes est capable de fournir une solution en temps raisonnable (moins d'une heure).
— *Propriété 21* : L'algorithme d'énumération de sous-graphes temporels isomorphes est capable de fournir une solution en temps raisonnable (moins d'une heure).

5.1 Description des jeux de données utilisés

Le jeu de données `Rollernet` introduit par Tournoux et al. [44] a été collecté à partir de données issues de la navigation de rollerbladers à travers Paris. Les sommets représentent chacun un rollerblader et une arête temporelle lie deux sommets à un instant t si les deux rollerbladers concernés sont suffisamment proches à l'instant concerné. Il s'agit du graphe temporel le plus dense que je traite dans cette thèse, avec une taille d'historique plus petite que celle des autres jeux de données, ce qui me permet de tester les limites des algorithmes, qui ont été pensés pour être appliqués à des jeux de donnés peu denses s'étendant sur de grandes tailles d'historique. Le graphe contient 62 sommets, liés par $4\dot{1}0^6$ arêtes temporelles pendant 10^4 instants temporels. Ce graphe temporel géométrique est également plus susceptible de contenir des Δ-jumeaux et des Δ-modules que les deux autres jeux de données, puisque ces modules temporels représenteront des groupes de rollerbladers décidant de se promener ensemble pendant une période de temps de plus de Δ instants temporels, ces groupes ayant donc pendant cette période des voisinages quasiment identiques puisque ces voisinages dépendent de la position géographique des rollerbladers. Les sous-graphes temporels isomorphes permettront quant à eux d'identifier des motifs comportementaux des rollerbladers, permettant par exemple d'identifier des groupes d'amis qui naviguent ensemble avant de se séparer pour se retrouver de nouveau. De ce jeu de données ont été extraits treize bancs de tests (batches), chaque batch contenant 100 sous-graphes temporels de `Rollernet`. Les quatre premiers batches concernés contiennent des sous-graphes temporels induits par une restriction du graphe temporel initial à $n_1 = 40$ sommets et $n_2 = 50$ sommets sélectionnés arbitrairement. Spécifiquement pour l'étude des algorithmes d'énumération des sous-graphes temporels isomorphes, dont les complexités sont plus importantes, j'ai également réalisé deux autres batches pour lesquels $n_3 = 30$ et $n_4 = 20$. Cinq autres batches contiennent des sous-graphes temporels pour lesquels respectivement $m_1 = 10^5$, $m_2 = 2 \cdot 10^5$ et $m_3 = 3 \cdot 10^5$ arêtes temporelles du graphe temporel initial ont été sélectionnées arbitrairement. Spécifiquement pour l'étude des algorithmes d'énumération des sous-graphes temporels isomorphes,

dont les complexités sont plus importantes, j'ai également réalisé deux autres batches pour lesquels $m_4 = 2,5\dot{1}0^5$ et $m_5 = 1,5\dot{1}0^5$. Les deux derniers batches contiennent des sous-graphes temporels restreints à respectivement $\tau_1 = 5\dot{1}0^3$ et $\tau_2 = 8\dot{1}0^3$ instants temporels consécutifs, l'instant temporel initial étant sélectionné uniformément entre 0 et $\tau - \tau_i - 1$. Spécifiquement pour l'étude des algorithmes d'énumération des sous-graphes temporels isomorphes, dont les complexités sont plus importantes, j'ai également réalisé deux autres batches pour lesquels $\tau_3 = 3\dot{1}0^3$ et $\tau_4 = 2\dot{1}0^3$.

Le jeu de données `Enron` publié par Klimt et Yang [31] est une représentation de l'historique d'e-mails échangés entre les employés de l'entreprise Enron sur une période de 3 ans. Il est constitué de 150 sommets liés par $2,5\dot{1}0^5$ arêtes temporelles pendant 10^9 instants temporels. Des Δ-modules représenteront dans ce graphe des membres d'un même service, ceux-ci recevant les mêmes e-mails sur certaines périodes et ayant une réponse uniforme envers l'extérieur sur cette même période, que celle-ci soit non-existante ou concertée en interne. Les sous-graphes temporels isomorphes, quant à eux, permettront d'identifier des relations hiérarchiques entre personnels, une utilisation assez commune de l'isomorphisme de sous-graphes, y compris dans le cas statique. Ce graphe temporel est extrêmement peu dense et représente le plus grand des jeux de données en terme de nombre d'arêtes temporelles et en de taille d'historique, ce qui me permettra de tester les limites des graphes pouvant être traités par les algorithmes présentés dans cette thèse. J'applique la même méthode d'extraction de batches que pour `Rollernet` pour générer sept batches de 100 graphes temporels chacun, avec $n_1 = 50$, $n_2 = 100$, $m_1 = 5\dot{1}0^3$, $m_2 = 10^4$, $m_3 = 2\dot{1}0^4$, $\tau_1 = 10^7$, et $\tau_2 = 5{\cdot}10^7$.

Le jeu de données `LesFurets` est une représentation de l'historique du tracking des actions d'un sous-ensemble d'utilisateurs à travers le funnel de conversion du site d'e-commerce lesfurets.com, un funnel de conversion étant l'ensemble des actions qu'un utilisateur doit entreprendre, à partir de son arrivée sur le site, pour atteindre une opération qui apporte de la rentabilité au site internet. Certains sommets représentant les divers utilisateurs anonymisés et les autres sommets représentant des événements trackés par le site sur son funnel de conversion. Le graphe comporte 10^3 sommets, liés par 10^4 arêtes temporelles pendant $1,5\dot{1}0^5$ arêtes temporelles. Le site d'e-commerce lesfurets.com ne permettant pas de communication entre les divers utilisateurs, ce graphe temporel est donc biparti, les utilisateurs ne pouvant interagir qu'avec des événements du funnel de conversion tandis que ces événements n'interagissent pas entre eux. L'historique est ici traité de sorte à translater les comportements de chaque utilisateur afin que ceux-ci soient considérés comme entrant dans le funnel de conversion à l'instant temporel 0, si bien que les arêtes temporelles sont enregistrées de manière asynchrone, un utilisateur 1 entrant dans le funnel deux jours après un utilisateur 2 enregistrant des arêtes temporelles pour l'instant 0 bien après que l'utilisateur 2 enregistre des arêtes temporelles pour l'instant 3600. Le graphe temporel n'est donc par ordonné par instant temporel. Cette caractéristique me permet de tester la robustesse des algorithmes à ce non-ordonnancement, afin de m'assurer qu'aucune erreur n'est induite par l'asynchronicité du graphe. Ce jeu de données est également le plus grand des jeux de données en terme de nombre de sommets dans le graphe temporel. Dans ce graphe temporel, les Δ-modules représenteront des groupes d'utilisateurs ayant une activité similaire pendant une certaine période de temps, présentant des temps de réaction similaires et réalisant les opérations dans le même ordre. J'espère donc qu'avec les bons paramètres, ces Δ-modules me permettront de regrouper des groupes d'utilisateurs aux comportements similaires, ce qui permettrait au système d'identifier des populations qui ne parviendront pas à la fin du funnel de conversion, de sorte à entreprendre une action pour modifier cette tendance. Les sous-graphes temporels isomorphes permettront d'identifier des comportements, notamment des réactions spécifiques à une série de questions du questionnaire du funnel de conversion qui présentent un certain rapport, de sorte que les utilisateurs parviennent à rapidement enchaîner plusieurs questions dont les réponses sont très liées. Je réalise à nouveau un extrait de sept batches avec les paramètres suivant : $n_1 = 300$, $n_2 = 600$, $m_1 = 3\dot{1}0^3$, $m_2 = 6\dot{1}0^3$, $m_3 = 8\dot{1}0^3$, $\tau_1 = 10^4$, et $\tau_2 = 1,3\dot{1}0^4$.

Jeu de données	$\|V\|$	$\|E\|$	τ
Rollernet	20 à 62	$15 \cdot 10^3$ à $4 \cdot 10^5$	2000 à 10^4
Enron	50 à 150	5000 à 25000	10^7 à 10^8
LesFurets	300 à 1000	3000 à 10000	10^5 à $1,5 \cdot 10^5$

TABLE 5.1 – Les jeux de données utilisés dans cette thèse et leurs caractéristiques

5.2 Description des implémentations

Tous les algorithmes présentés dans cette thèse ont été implémentés en langage Java. Le choix de ce langage a été motivé par l'environnement technique de la société Courtanet, basé sur l'environnement Java, de sorte que les implémentations de ces algorithmes puissent être facilement intégrés au code source de l'entreprise, sans ajout de nouvelle technologie qui devrait s'interfacer avec le code déjà présent, tout en permettant une maintenance de cette implémentation par l'équipe IT de la société, formée à cette technologie.

J'ai eu recours aux librairies standards du langage Java, utilisant principalement des HashSet pour implémenter les ensembles, voire des ArrayList si l'ordonnancement de l'ensemble doit être pris en compte, et des HashMap pour permettre un enregistrement en mémoire optimisé autant en terme d'utilisation mémoire qu'en terme de complexité temporelle des recherches dans la structure de données. Par exemple, dans le cadre de l'implémentation de l'algorithme Edge Iteration, j'utilise une HashMap pour enregistrer la liste de toutes les arêtes temporelles existant à un instant temporel $t \in T$ en l'associant à l'indice t. Cette implémentation me permet de réorganiser E pour faciliter la recherche des arêtes existant à l'instant t, me permettant de garantir la complexité temporelle d'Edge Iteration présentée dans la partie théorique de cette thèse en n'itérant que sur E_t lors de la recherche de l'existence d'une arête temporelle.

Pour procéder à la mesure du temps de calcul des implémentations des divers algorithmes, j'utilise l'horloge du système, calculant avec une précision de l'ordre de la milliseconde la différence entre l'instant de fin du calcul et l'instant de début du calcul.

Pour procéder à la mesure de l'utilisation mémoire des implémentations des divers algorithmes, j'utilise l'outil Java VisualVM permettant le monitoring de la JVM en cours d'exécution, ce qui me permet de mesurer l'utilisation mémoire du processus avec une précision de l'ordre de l'octet. Cependant, je me contenterai dans cette étude d'une précision de l'ordre du méga-octet, qui sera bien suffisante pour établir les complexités spatiales expérimentales.

Au total, l'implémentation des algorithmes d'énumération de sous-graphes temporels isomorphes représente 1972 lignes de code, et celle des algorithmes d'énumérations de modules et jumeaux temporels 1895 lignes de code, pour un total de 30 classes.

La taille maximale du tas mémoire alloué à la JVM lors de ces expérimentations est fixé à 8 giga-octets.

Du fait de la nature de prototype des implémentations, je n'ai pas eu recours à un certain nombre d'optimisations qui pourraient améliorer les performances du programme. Par exemple, il est théoriquement possible de paralléliser une partie des algorithmes présentés dans cette thèse, ce que je n'ai pas fait pour mes expérimentations, puisque je cherchais à confirmer l'ordre de grandeur des performances au pire cas des algorithmes et que la parallélisation n'aurait fait que diminuer le temps de calcul d'un facteur constant (le nombre de threads du processeur).

Par exemple, dans l'énumération des sous-graphes temporels isomorphes, l'énumération des cliques de taille n' dans le graphe de conformité altéré pour l'énumération des sous-graphes temporels isomorphes peut être parallélisée. De même, une fois la liste des bijections f établies, la recherche des instants t_0 tels que $\{f, t_0\}$ est un isomorphisme de sous-graphes temporels peut être menée en parallèle pour chaque f et pour chaque $t_0 \in T$.

Dans l'énumération des Δ-modules, une fois la liste des splitters instantannées établie pour chaque paire de sommets par le remplissage des arbres de temps discontinu, la création de tous les modules instantannés minimaux contenant chaque paire de sommets est réalisable en parallèle pour chaque paire de sommets.

5.3 Influence de τ sur les complexités des solutions aux problèmes d'isomorphisme de sous-graphes temporels

Pour réaliser les expériences sur l'énumération de sous-graphes temporels isomorphes, pour chaque expérience, je génère un graphe temporel pour qu'il serve de motif. La procédure pour créer ce motif est de réaliser une extraction du graphe temporel considéré afin de s'assurer qu'au moins un sous-graphe du graphe considéré sera isomorphe à ce motif. Pour chaque problème, la procédure d'extraction du motif sera différente.

Sauf mention contraire, dans le cadre de cette étude, je fixe le nombre de sommets du motif temporel à 2. Le nombre d'arêtes temporelles variera entre 7 et 23 arêtes temporelles.

Dans le cadre de cette thèse, je m'intéresse principalement à l'influence de la taille de l'historique τ sur les complexités temporelle et spatiale de mes algorithmes. De sorte que, comme dans le chapitre 3 sur les modules temporels, j'ai pu procéder à des expérimentations sur le jeu de données `Timeprogression` afin de valider l'influence théoriquement établie de τ sur les complexités spatiale et temporelle des algorithmes que je présente dans cette thèse, je vais procéder de même dans cette section sur les algorithmes d'énumération des sous-graphes temporels isomorphes.

Procédé pour extraire le motif temporel pour l'étude de l'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire.

Pour procéder à l'extraction du motif temporel P pour l'étude de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire, je fixe un nombre de sommets n' et une taille d'historique maximal τ'. Je choisis ensuite aléatoirement et uniformément un instant temporel de début t_0 entre 0 et $\tau - \tau'$ appartenant à l'historique du graphe temporel G considéré. Puis je tire uniformément n' sommets dans l'ensemble de sommets de G pour les ajouter à V', l'ensemble de sommets de P. Puis j'ajoute à l'ensemble des arêtes temporelles de P toutes les arêtes temporelles de G existant entre t_0 et $t_0 + \tau'$ entre les sommets de G qui ont été copiés dans V'. Bien sûr, pour chaque arête temporelle ajoutée à P, je translaterai l'instant temporel en y soustrayant t_0. Cette procédure est celle que j'utilise pour extraire le motif temporel P pour l'étude de l'énumération des sous-graphes temporels isomorphes. Ce procédé ne permet ni de garantir que G est une forêt temporelle ni que P est une forêt temporelle linéaire. Cependant, de par la nature des graphes considérés, avec un faible nombre de sommets, un nombre d'arêtes temporelles moyen et une grande taille d'historique, résultant en une densité d'arêtes temporelles faible, la probabilité que le graphe G présente instantanément des cycles est très faible. En outre, le motif P considéré disposant de 2 sommets dans le cadre de cette étude, P est nécessairement une forêt temporelle linéaire. Afin de m'assurer que G est une forêt temporelle, je procède au préalable à un parcours en profondeur de G à chaque instant temporel afin de détecter d'éventuels cycles. Si des cycles sont détectés, je ne réalise pas d'expérimentation sur le graphe concerné.

Propriété 9 (Validation de l'influence de τ sur la complexité temporelle de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire telle qu'établie théoriquement). A propos de la propriété 9, je m'intéresse aux résultats présentés en figure 5.1 qui présente l'évolution du temps de calcul pour l'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. La complexité temporelle théoriquement établie pour cette énumération est linéaire en τ. Or, sur la figure 5.1, j'observe effectivement une tendance linéaire en τ dans l'évolution du temps de calcul.

La propriété 9 de linéarité en τ de la complexité temporelle de l'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire est donc validée par l'expérience.

Propriété 10 (Validation de l'influence de τ sur la complexité spatiale de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire telle qu'établie théoriquement). A propos de la propriété 10 d'influence de τ sur la complexité spatiale de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire, je m'intéresse aux résultats présentés en figure 5.2 qui présente l'évolution de l'utilisation mémoire lors de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire dans les graphes temporels qui composent le jeu de données `Timeprogression` en fonction du nombre d'instants temporels dans le graphe temporel. La complexité spatiale théoriquement établie pour cette énumération est linéaire en τ. Or sur la figure 5.2, la tendance de l'utilisation mémoire semble globalement constante. Il est néanmoins à noter que plusieurs jeux de données présentent des utilisations de mémoire bien plus importantes. Ceci peut être expliqué par l'existence de nombreuses occurrences du motif recherché dans le graphe temporel étudié, engendrant une augmentation de la taille de la solution dont la complexité spatiale dépend, de par la structure de données utilisée pour garder en mémoire cette solution (et les résultats partiels qui permettent de la construire). Ces variations de temps de calcul semblent quant à eux engendrer un écart-type qui augmente lorsque τ augmente. Ce que j'en conclus, c'est que si la taille de la solution est faible, l'utilisation mémoire de l'algorithme d'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire peut être approximée comme indépendante de τ, mais que si la taille de la solution devient plus importante, on peut noter une tendance à l'augmentation de cette utilisation mémoire correlée à l'augmentation de la taille de l'historique. La figure 5.2 ne présente cependant pas de tendance clairement identifiable pour permettre de trancher quant à une tendance linéaire ou quadratique en τ de l'utilisation mémoire de cet algorithme. Une plus ample étude expérimentale est donc à prévoir, en utilisant de nouveaux jeux de données générés spécifiquement dans le but d'obtenir une réponse non-équivoque à cette question de l'influence de τ sur l'utilisation mémoire lors de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire, peut-être en générant des jeux de données avec moins de sommets et d'arêtes temporelles mais s'étendant sur un historique plus grand. Il est également à noter que pour nombre de graphes temporels du jeu de données `Timeprogression`, cette énumération rencontre des problèmes de manque de mémoire, dus à une trop grande taille de la structure de données permettant de stocker les résultats partiels qui servent à construire la solution. Si je ne les ai pas fait figurer sur cette courbe, afin de ne pas fausser l'échelle, le nombre d'occurrences de ces problèmes de mémoire a tendance à augmenter avec τ.

La propriété 10 de linéarité en τ de la complexité spatiale de l'algorithme d'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire reste donc une question ouverte, l'utilisation mémoire semblant quasi-indépendante de τ lorsque la taille de la solution est faible mais s'avérant dépendre de τ lorsque cette solution devient plus grande. Avec les résultats obtenus, impossible de statuer sur la tendance de cette augmentation.

Procédé pour extraire le motif temporel pour l'étude de l'énumération des sous-graphes temporels isomorphes par suite d'arêtes.

Pour procéder à l'extraction du motif temporel P pour l'étude de l'énumération de sous-graphes temporels isomorphes par suite d'arêtes, je fixe un nombre de sommets n' et un nombre m' d'arêtes temporelles. Je tire ensuite uniformément un instant temporel de début t_0 de l'historique du graphe temporel G considéré. Puis je tire uniformément n' sommets dans l'ensemble de sommets de G pour les ajouter à V', l'ensemble de sommets de P. Puis, en commençant à $t' = t_0$, j'ajoute à l'ensemble des arêtes temporelles de P toutes les arêtes temporelles de G existant à t' entre les sommets de G qui ont été copiés dans V'. Initialement, j'attribue à ces arêtes temporelles copiées un instant temporel égal à 0. Une fois que j'ai ajouté toutes les arêtes temporelles de G existant à t' entre les sommets de G qui ont été copiés dans V', j'incrémente t', et j'attribue aux copies des arêtes temporelles ajoutées à P un instant temporel également incrémenté. Je réitère jusqu'à avoir ajouté m' arêtes temporelles à P.

Propriété 11 (Validation de l'influence de τ sur la complexité temporelle de l'énumération de sous-graphes temporels isomorphes par suite d'arêtes telle qu'établie théoriquement). A propos de la propriété 11, je m'intéresse aux résultats présentés en figure 5.3 qui présente l'évolution

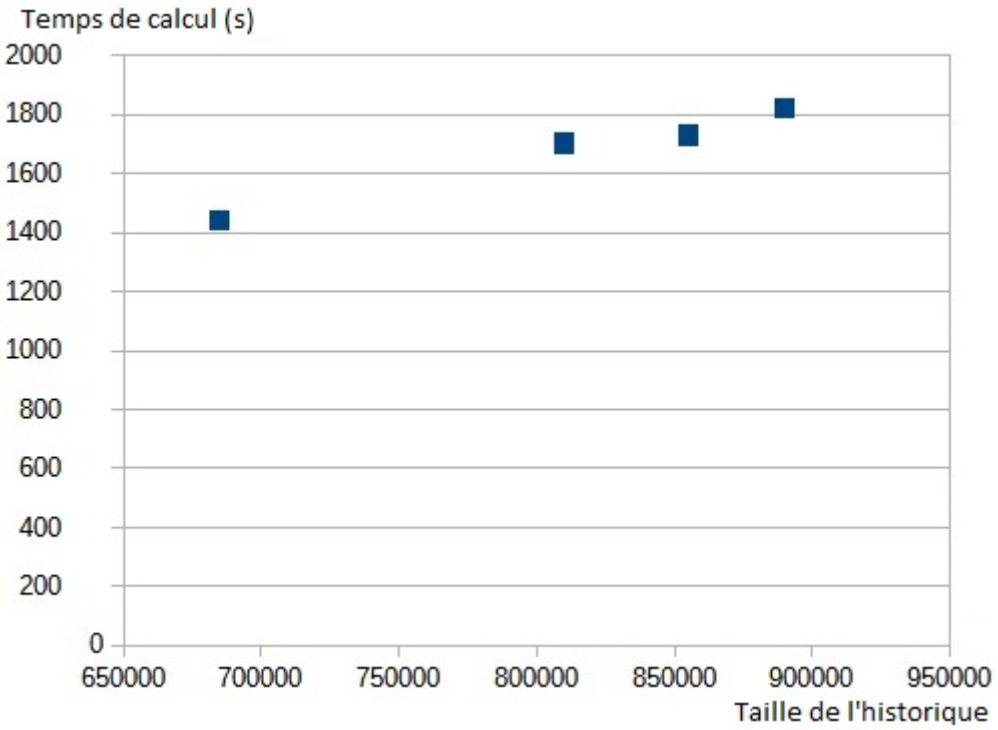

FIGURE 5.1 – Temps de calcul de l'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. Le graphe contient 50 sommets et le motif compte une dizaine d'arêtes

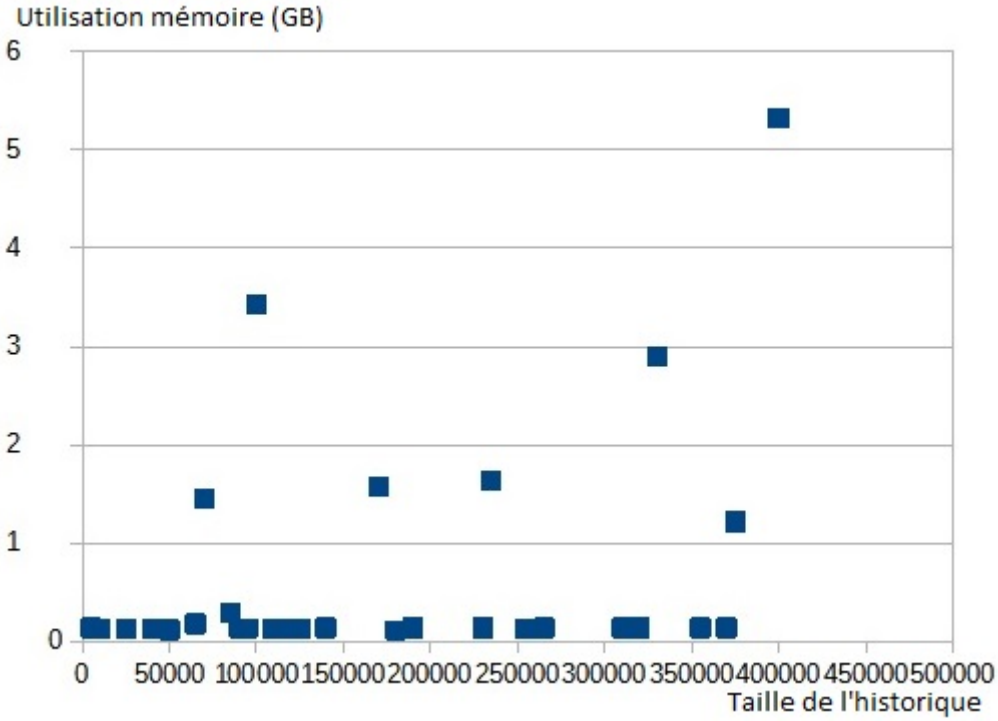

FIGURE 5.2 – Utilisation mémoire lors de l'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. Le graphe contient 50 sommets et le motif compte une dizaine d'arêtes

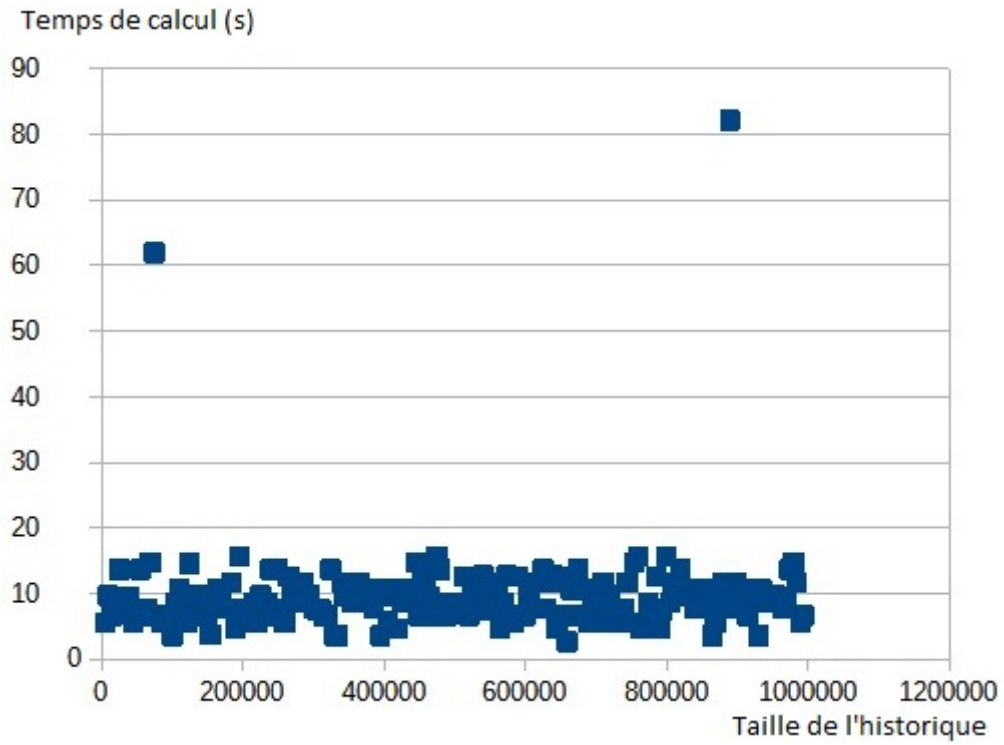

FIGURE 5.3 – Temps de calcul de l'énumération des sous-graphes temporels isomorphes par suite d'arêtes en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. Le graphe contient 50 sommets et le motif compte 15 arêtes

du temps de calcul pour l'énumération des sous-graphes temporels isomorphes par suite d'arêtes en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. La complexité temporelle théoriquement établie pour cette énumération est indépendante de τ. Or, sur la figure 5.3, j'observe effectivement que malgré l'augmentation de τ, le temps de calcul pour l'énumération des sous-graphes temporels isomorphes par suite d'arêtes semble osciller autour de 10 secondes sans évolution significative lors de l'augmentation de τ, malgré deux valeurs marginales respectivement légèrement supérieures à 60 secondes et 80 secondes. Ces deux valeurs extrêmes peuvent être expliquées par l'utilisation du processeur par d'autres applications lors du calcul et n'entâchent pas l'étude expérimentale. Je peux donc considérer que τ n'a effectivement aucune incidence sur le temps de calcul pour cette énumération.

La propriété 11 d'indépendance en τ de la complexité temporelle de l'énumération des sous-graphes temporels isomorphes par suite d'arêtes est donc validée par l'expérience sur `Timeprogression`.

Propriété 12 (Validation de l'influence de τ sur la complexité spatiale de l'énumération de sous-graphes temporels isomorphes par suite d'arêtes telle qu'établie théoriquement). A propos de la propriété 12 d'influence de τ sur la complexité spatiale de l'énumération de sous-graphes temporels isomorphes par suite d'arêtes, je m'intéresse aux résultats présentés en figure 5.4 qui présente l'évolution de l'utilisation mémoire lors de l'énumération des sous-graphes temporels isomorphes par suite d'arêtes en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. La complexité spatiale théoriquement établie pour cette énumération est indépendante de τ. Or, sur la figure 5.4, j'observe effectivement que malgré l'augmentation de τ, l'utilisation mémoire lors de l'énumération des sous-graphes temporels isomorphes par suite d'arêtes semble osciller entre 100 méga-octets et 140 méga-octets, avec quelques jeux de données qui présentent des utilisations mémoires plus faibles. Aucune tendance à la hausse de l'utilisation mémoire ne se détache de cette figure.

La propriété 12 d'indépendance en τ de la complexité spatiale de l'énumération des sous-graphes

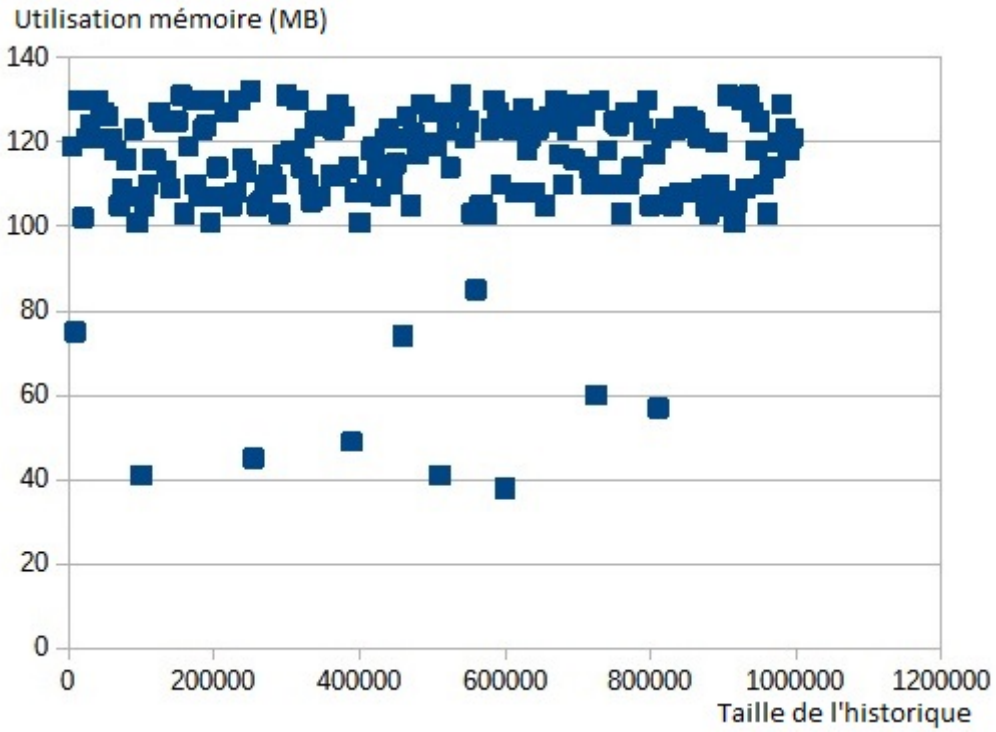

FIGURE 5.4 – Utilisation mémoire lors de l'énumération des sous-graphes temporels isomorphes par suite d'arêtes en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. Le graphe contient 50 sommets et le motif compte 15 arêtes

temporels isomorphes par suite d'arêtes est donc validée par l'expérience sur `Timeprogression`.

Procédé pour extraire le motif temporel pour l'étude de l'énumération des sous-graphes temporels isomorphes.

Pour procéder à l'extraction du motif temporel P pour l'étude de l'énumération de sous-graphes temporels isomorphes, je réalise le même procédé que dans le cas de l'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire en me dispensant de vérifier a posteriori que le graphe est une forêt temporelle.

Propriété 13 (Validation de l'influence de τ sur la complexité temporelle de l'énumération de sous-graphes temporels isomorphes telle qu'établie théoriquement). A propos de la propriété 13, je m'intéresse aux résultats présentés en figure 5.5 qui présente l'évolution du temps de calcul pour l'énumération des sous-graphes temporels isomorphes en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. La complexité temporelle théoriquement établie pour cette énumération est indépendante de τ. Or, sur la figure 5.5, j'observe effectivement que malgré l'augmentation de τ, le temps de calcul pour l'énumération des sous-graphes temporels isomorphes semble osciller autour de 820 secondes sans évolution significative lors de l'évolution de τ, malgré une forte variance, certains graphes entraînant des temps de calcul pouvant culminer à 890 secondes. Cette variance peut s'expliquer autant par l'utilisation du processeur par d'autres applications pendant le calcul que par des occurrences plus ou moins nombreuses des propriétés entraînant des points d'arrêt de boucles, qui peuvent être plus présents lorsque le graphe augmente localement en densité. Cependant, malgré cette variance, la tendance générale observable de l'évolution du temps de calcul pour l'énumération des sous-graphes temporels isomorphes est une tendance constante. Je peux donc considérer que τ n'a effectivement aucune incidence sur le temps de calcul pour cette énumération.

La propriété 13 d'indépendance en τ de la complexité temporelle de l'énumération des sous-graphes temporels isomorphes est donc validée par l'expérience sur `Timeprogression`.

Propriété 14 (Validation de l'influence de τ sur la complexité spatiale de l'énumération de

sous-graphes temporels isomorphes ainsi qu'établie théoriquement). A propos de la propriété 14, je m'intéresse aux résultats présentés en figure 5.6 qui présente l'évolution de l'utilisation mémoire lors de l'énumération des sous-graphes temporels isomorphes en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. La complexité spatiale théoriquement établie pour cette énumération est indépendante de τ. Or, sur la figure 5.6, je constate que malgré une variance bien plus grande pour les graphes temporels présentant une taille d'historique supérieure à $4\dot{1}0^5$ instants temporels, l'utilisation mémoire lors de cette énumération ne semble pas évoluer significativement avec la taille de l'historique. Je note globalement deux zones dans cette courbe. La première, pour des graphes présentant des tailles d'historique inférieures à $4\dot{1}0^5$ instants temporels, pour lesquels l'utilisation mémoire gravite aux alentours de 100 méga-octets, sans évolution lors de l'augmentation de τ. Et la deuxième, pour des graphes présentant des tailles d'historique supérieures à $4\dot{1}0^5$ instants temporels, pour lesquels l'utilisation mémoire oscille entre 100 méga-octets et 700 méga-octets, avec une très légère évolution dans l'utilisation mémoire moyenne avec l'augmentation de τ. Dans cette seconde zone, je constate néanmoins que les deux bornes de cet intervalle sont atteintes, que ce soit pour des graphes temporels d'environ $4\dot{1}0^5$ instants temporels ou des graphes temporels d'environ 10^6 instants temporels. Si une évolution légère peut donc être relevée, elle peut s'expliquer par le facteur densité du graphe plus que par l'évolution de la taille de l'historique. En effet, puisque le nombre de sommets et le nombre d'arêtes temporelles restent constants pour tous les graphes du jeu de données `Timeprogression` alors que la taille de l'historique augmente, la densité du graphe diminue avec cette augmentation de la taille de l'historique. Cette augmentation de la densité entraîne dans un premier temps une diminution des cas d'arrêt des diverses boucles constituant cet algorithme, puisque le motif temporel (dont on rappelle qu'il est, dans le cadre de cette étude expérimentale, composé de 2 sommets, est de 15 arêtes temporelles) qui initialement est composé d'un grand nombre d'instants temporels successifs lors desquels une arête existe entre les deux sommets, a de moins en moins d'occurrence dans le graphe temporel étudié. Cette augmentation de la densité conduit donc dans un premier temps à un point de rupture, aux alentours d'une taille d'historique de $4\dot{1}0^5$ instants temporels, ce qui explique les deux zones de cette courbe. Puis dans un second temps, le motif recherché va être composé d'un grand nombre d'instants temporels successifs lors desquels aucune arête n'existe entre les deux sommets. Ce motif va alors connaître une augmentation de son nombre d'occurrences dans le graphe temporel étudié lorsque la densité de celui-ci va diminuer, entraînant cette augmentation légère dans la deuxième zone du graphe.

La propriété 14 d'indépendance en τ de la complexité spatiale de l'énumération des sous-graphes temporels isomorphes peut donc être considérée comme validée par l'expérience sur `Timeprogression`, même si on notera une légère augmentation de l'utilisation mémoire moyenne en parallèle de l'augmentation de la taille de l'historique, qui pourra être expliquée comme dépendant plus de la diminution de la densité du graphe que de l'augmentation de τ.

Pour conclure quant à cette section, j'ai établi théoriquement :

— Une validation de la linéarité en τ de la complexité temporelle de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire.
— Une impossibilité de conclure en l'état quant à l'influence de τ sur l'utilisation mémoire lors de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire, qui semble constante lorsque la taille de la solution est faible et décrire une évolution dépendant de τ lorsque la taille de la solution est plus grande, sans qu'une tendance claire ne se dégage de mes expérimentations.
— Une validation de l'indépendance en τ de la complexité temporelle de l'énumération de sous-graphes temporels isomorphes par suite d'arêtes.
— Une validation de l'indépendance en τ de la complexité spatiale de l'énumération de sous-graphes temporels isomorphes par suite d'arêtes.
— Une validation de l'indépendance en τ de la complexité temporelle de l'énumération de sous-graphes temporels isomorphes.
— Une validation de l'indépendance en τ de la complexité spatiale de l'énumération de sous-graphes

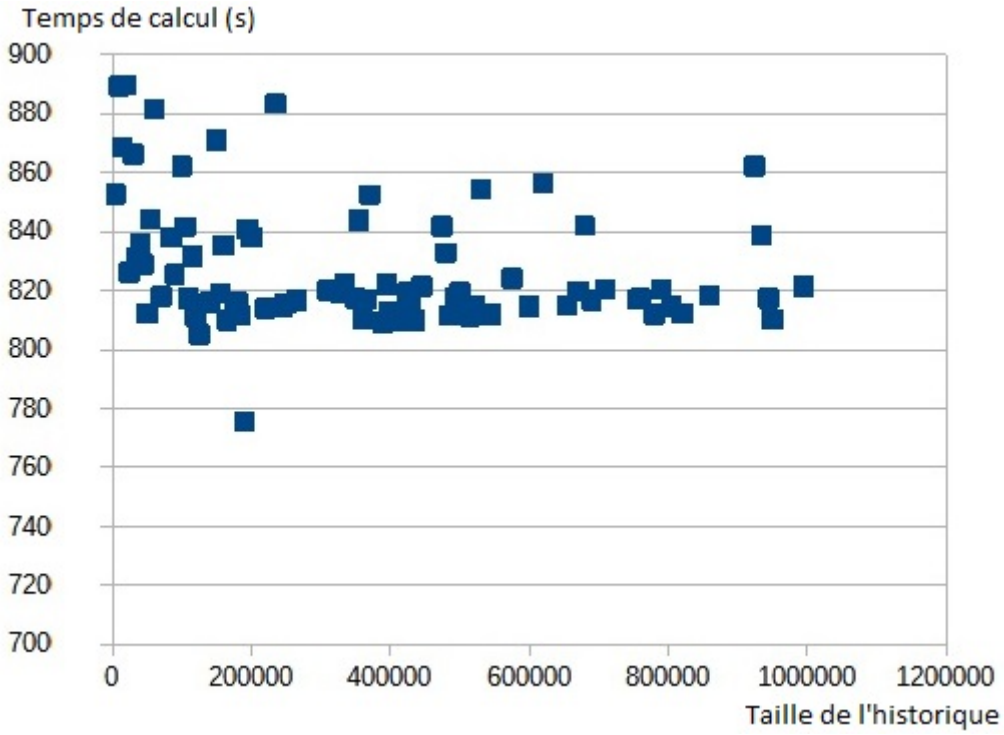

FIGURE 5.5 – Temps de calcul de l'énumération des sous-graphes temporels isomorphes en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. Le graphe contient 50 sommets et le motif compte 15 arêtes

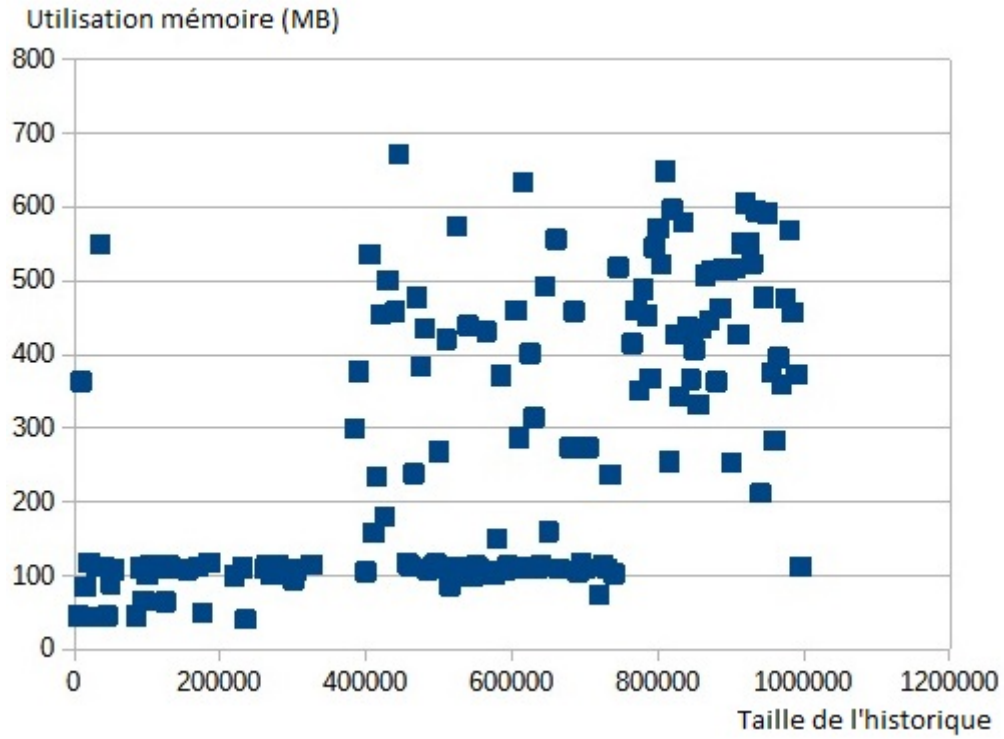

FIGURE 5.6 – Utilisation mémoire lors de l'énumération des sous-graphes temporels isomorphes en fonction de la taille de l'historique τ sur le jeu de données `Timeprogression`. Le graphe contient 50 sommets et le motif compte 15 arêtes

temporels isomorphes, bien qu'une très légère évolution de l'utilisation mémoire puisse être notée, imputable à la diminution de la densité du graphe qui entraîne une augmentation de la taille de la solution.

5.4 Expérimentations sur les jeux de données réels

J'utilise les implémentations des divers algorithmes présentés dans cette thèse sur les jeux de données sus-nommés afin d'étudier les limites du passage à l'échelle des divers algorithmes. Je vais commencer par analyser et commenter les résultats de ce passage à l'échelle sur les algorithmes d'énumération des modules temporels avant de me pencher sur les résultats de passage à l'échelle des algorithmes d'énumération des sous-graphes temporels isomorphes.

5.4.1 Étude du passage à l'échelle des algorithmes d'énumération de modules temporels

Afin de vérifier la correction des diverses implémentations, la méthode que j'utilise est celle du test unitaire. Pour les problèmes d'énumération de jumeaux éternels et de Δ-jumeaux, je compare les résultats des algorithmes MEI et MLEI à ceux d'un algorithme naïf qui itère sur toutes les paires de sommets et tous les instants temporels afin d'énumérer ces jumeaux de manière sûre, pour une complexité en $O(n^3 \times \tau)$ pour l'énumération de jumeaux éternels et en $O(n^3 \times \tau \times \Delta)$ pour l'énumération de Δ-jumeaux. Du fait de ces complexités élevées, les algorithmes naïfs n'ont pu fournir des résultats en temps raisonnable que pour les instances composant le jeu de données `Rollernet`. Je peux grâce à ces algorithmes naïfs certifier la correction des algorithmes MEI et MLEI pour énumérer jumeaux éternels et Δ-jumeaux sur $\approx 33\%$ des instances utilisées dans cette étude expérimentale. Les résultats ont été positifs pour toutes ces instances.

Pour le problème d'énumération des modules éternels, je n'ai pu me reposer sur des algorithmes naïfs pour certifier de la correction des algorithmes MEI et MLEI puisque les algorithmes naïfs d'énumération des modules éternels étaient de complexité trop importante pour être appliqués à la moindre des instances. J'ai donc choisi une autre méthode en vérifiant les résultat de MEI et MLEI énumérant les modules éternels grâce aux algorithmes MEI et MLEI énumérant les jumeaux éternels sur des graphes temporels bipartis. En effet, j'ai établi théoriquement que dans un graphe biparti, les modules éternels étaient de trois natures :

— Le module éternel est V l'ensemble des sommets du graphe.
— Le module éternel est un singleton $\{v\}$ avec $v \in V$.
— Si le module éternel A est monocolore, toutes les paires de sommets de A sont des paires de jumeaux éternels.
— Si le module éternel A est bicolore, le graphe restreint à A est isolé, c'est-à-dire que le voisinage de tout sommet de A hors de A, à tout instant temporel, est vide.

Or dans les expériences sur le jeu de données `LesFurets` qui est un graphe biparti, l'algorithme MLEI a énuméré un total de 0 jumeau éternel. La variante pour les modules éternels a quant à elle énuméré exactement $n+1$ modules éternels, où n est le nombre de sommets dans le graphe temporel, et ce pour chaque instance de `LesFurets`. Parmi ces modules, un seul, celui qui regroupe tout V, est de voisinage vide à tout instant du graphe. J'obtiens donc que MLEI énumère correctement les modules éternels sur un graphe biparti.

Dans la suite de cette analyse, j'ignorerai les questions de correction et me concentrerai exclusivement sur le temps de calcul.

Les cas d'utilisation des jumeaux éternels étant quelque peu limités, et ceux-ci n'étant qu'un cas particulier de Δ-jumeaux, je ne m'étendrai pas sur les résultats expérimentaux sur ceux-ci et passerai directement à l'énumération des Δ-jumeaux. Les calculs menés dans le cadre de cette étude expérimentale sur les jumeaux et les modules ont été réalisés avec une valeur arbitraire de $\Delta = 102$. Malheureusement, l'algorithme d'énumération des Δ-modules ne parvient à aboutir à un résultat sur aucun des jeux de données considérés dans le cadre de cette étude expérimentale. Ceci est dû à une complexité spatiale trop importante, engendré par les structures de données des

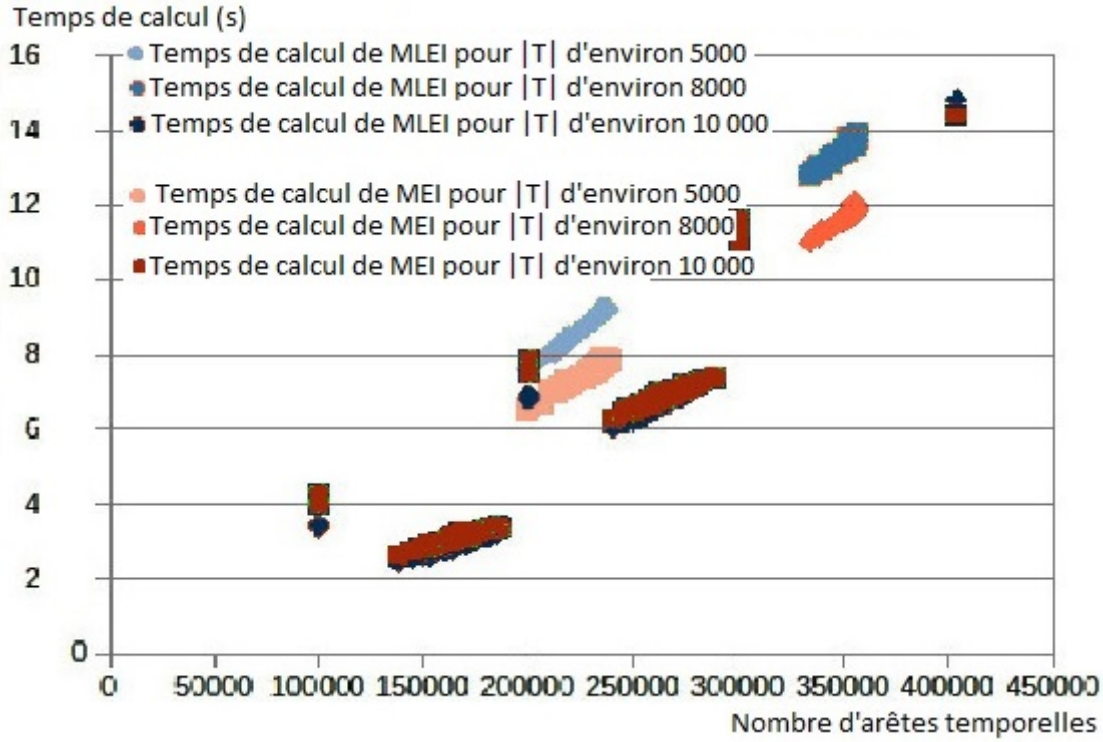

FIGURE 5.7 – Temps de calcul des algorithmes MLEI et MEI énumérant les Δ-jumeaux, en fonction du nombre d'arêtes temporelles dans les graphes temporels constituant le jeu de données `Rollernet`.

arbres de temps discontinu utilisées pour stocker les résultats partiels. Malheureusement, sur tous les jeux de données, cette complexité entraîne un manque de mémoire vive allouée au processus, ce qui m'empêche d'énumérer les Δ-modules, en vue d'une validation expérimentale des résultats théoriques.

Je commence par utiliser les algorithmes MEI et MLEI pour énumérer les Δ-jumeaux dans les graphes temporels constituant les jeux de données `Rollernet` 5.7, `Enron` 5.8 et `LesFurets` 5.9, avec des vues détaillées pour chacun des trois jeux de données. La figure 5.10 récapitule les moyennes de temps de calcul sur les trois jeux de données.

Propriété 15 (Validation du passage à l'échelle de l'énumération de Δ-jumeaux). Si je considère la heatmap présente dans la figure 5.7, qui présente l'évolution du temps de calcul de l'énumération de Δ-jumeaux, par l'algorithme MEI et l'algorithme MLEI, en fonction du nombre d'arêtes temporelles dans le graphe sur le jeu de données `Rollernet`, les points étant de couleur plus sombre avec l'augmentation de la taille de l'historique τ du graphe considéré, τ ne semble avoir qu'un impact mineur sur le temps de calcul puisque nombre de graphes temporels aux tailles d'historique importantes sont traités plus rapidement par les algorithmes MEI et MLEI que d'autres graphes temporels aux tailles d'historique plus faibles. La tendance globale d'augmentation du temps de calcul est quant à elle explicable par l'influence de l'augmentation du nombre de sommets et d'arêtes, plus que par l'augmentation de la taille d'historique.

Je me rapporte également aux figures 5.8, qui présente l'évolution du temps de calcul de l'énumération de Δ-jumeaux par l'algorithme MLEI en fonction du nombre d'arêtes temporelles sur le jeu de données `Enron`, et 5.9, qui présente l'évolution du temps de calcul de l'énumération de Δ-jumeaux par l'algorithme MLEI en fonction du nombre d'arêtes temporelles sur le jeu de données `LesFurets`, deux jeux de données pour lesquels τ et $\|V\|$ sont trop importants pour que MEI soit capable d'énumérer les Δ-jumeaux dans les graphes temporels, la complexité spatiale de cet algorithme étant fortement dépendante de ces deux facteurs. Je me concentre donc sur les résultats de l'algorithme MLEI. Dans la figure 5.8, le premier graphique, dans lequel les points sont de couleur plus sombre lorsque τ augmente, me permet de confirmer l'influence de $\|E\|$ sur la

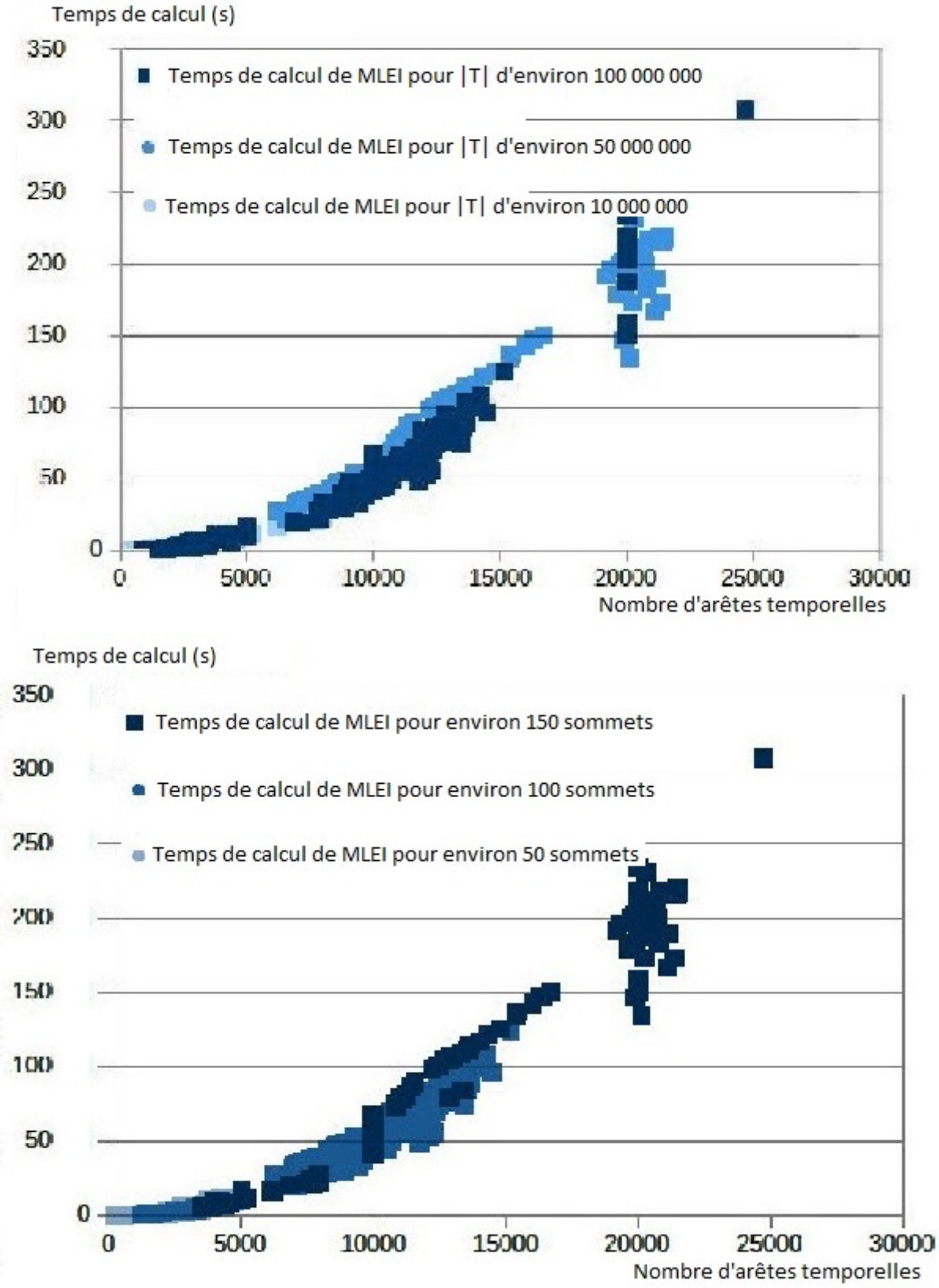

FIGURE 5.8 – Temps de calcul de l'algorithme MLEI énumérant les Δ-jumeaux, en fonction du nombre d'arêtes temporelles dans les graphes temporels constituant le jeu de données `Enron`.

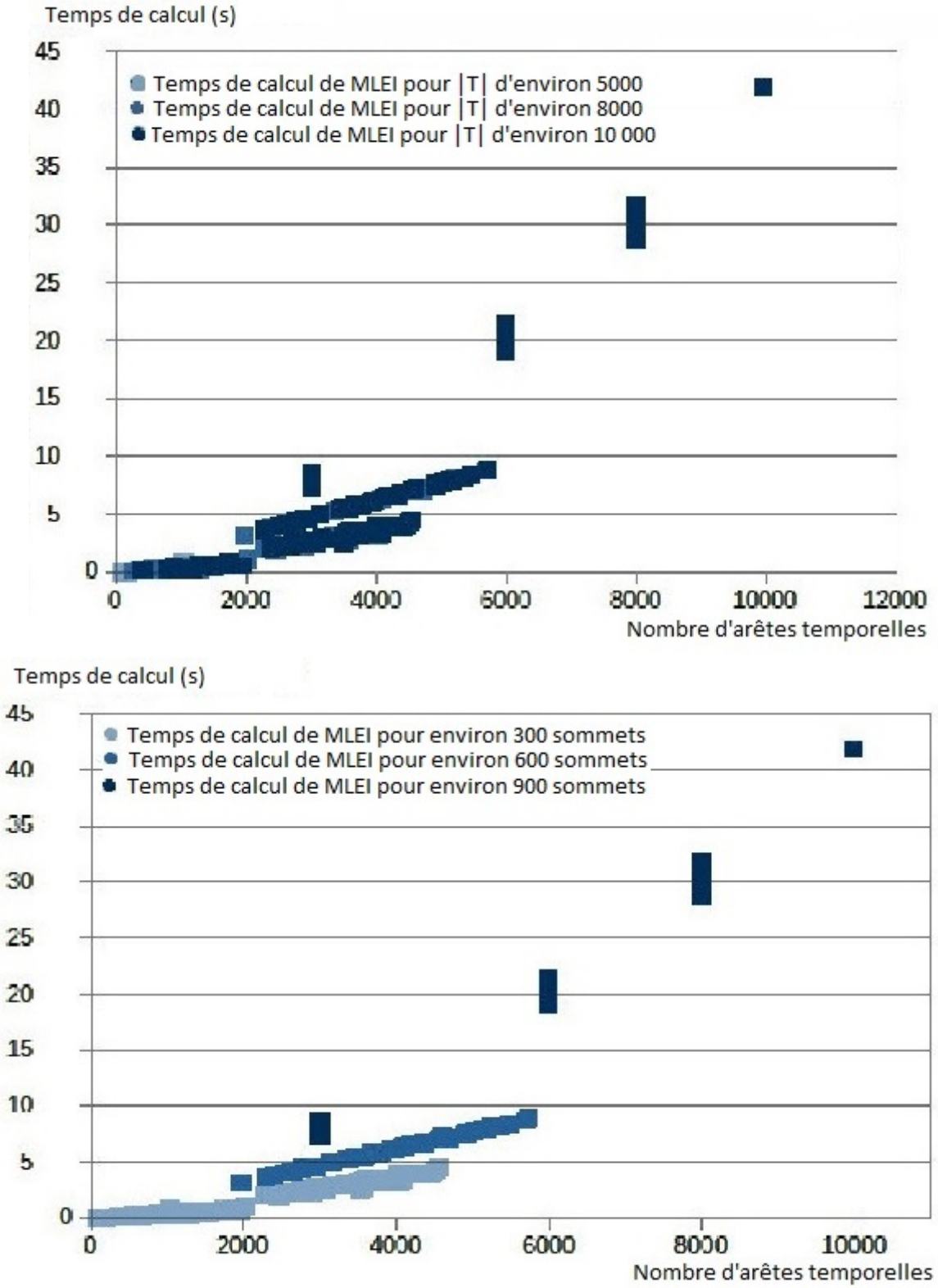

FIGURE 5.9 – Temps de calcul de l'algorithme MLEI énumérant les Δ-jumeaux, en fonction du nombre d'arêtes temporelles dans les graphes temporels constituant le jeu de données `LesFurets`.

complexité théorique établie dans la section 3 pour l'algorithme MLEI appliqué à l'énumération de Δ-jumeaux, la courbe expérimentale du temps de calcul de cet algorithme en fonction du nombre d'arêtes temporelles dans le graphe temporel semblant décrire une fonction polynomiale de degré 2. La heatmap me permet, comme c'était le cas pour le jeu de données `Rollernet` de constater une faible influence de la taille de l'historique sur ce temps de calcul, ce qui me permet de confirmer que l'influence du nombre d'arêtes temporelles est bien plus importante que l'influence de la taille de l'historique sur la complexité temporelle de l'algorithme MLEI énumérant les Δ-jumeaux. La heatmap présentée sur le deuxième graphique de la figure 5.8, pour lequel les points sont de couleur plus sombre avec l'augmentation du nombre de sommets, me permet quant à elle d'observer ce que j'espère observer quand la coloration des points dépend d'un paramètre dont l'influence sur le temps de calcul est non négligeable (ici le nombre de sommets), puisque je constate que les points les plus sombres correspondent à des temps de calcul plus grands que les points les plus clairs. Ces tendances sont confirmées en réalisant des observations de même nature sur les résultats de l'algorithme MLEI appliqué à l'énumération de Δ-jumeaux dans les graphes temporels constituant le jeu de données `LesFurets`, dont les résultats sont présentés dans la figure 5.9.

J'ai donc bien démontré que la complexité temporelle de l'algorithme MLEI appliqué à l'énumération de Δ-jumeaux dépend du nombre d'arêtes temporelles et du nombre de sommets dans le graphe temporel considéré plus que de la taille de l'historique, ce qui est le but recherché dans le cadre de cette thèse.

Si je me rapporte à la figure 5.10, je relève que sur l'ensemble des jeux de données, l'algorithme MLEI est capable d'énumérer tous les Δ-jumeaux d'un graphe temporel, sur tous les jeux de données considérés dans le cadre de cette étude expérimentale, avec un temps de calcul moyen inférieur à 80 secondes. La figure 5.8 me montre que certains graphes temporels peuvent conduire à des temps de calcul de l'ordre de 300 secondes au maximum. Ces temps de calcul restent largement dans la limite de l'acceptable telles que définie dans cette thèse (temps de calcul inférieur à une heure). Aucun problème mémoire n'est à signaler dans l'ensemble de ces expériences.

La propriété 15 de passage à l'échelle de l'énumération de Δ-jumeaux est donc validée, le temps de calcul pour l'énumération de Δ-jumeaux restant inférieur à une heure sur l'ensemble des jeux de données, et aucun problème de manque de mémoire n'intervenant lors des calculs.

Propriété 16 (Validation de la supériorité pratique de l'algorithme MLEI sur l'algorithme MEI pour l'énumération de Δ-jumeaux). Le jeu de données `Rollernet` est le seul jeu de données de cette thèse pour lequel l'algorithme MEI ne rencontre pas de problème de mémoire lors de l'énumération des Δ-jumeaux. En effet, sur les autres jeux de données, le nombre de sommets et la taille de l'historique sont trop élevés et la complexité spatiale théorique en $O(n^2 \times \tau)$, avec n le nombre de sommets et τ la taille de l'historique, pour stocker la suite de matrices d'adjacence du graphe temporel implique une utilisation mémoire trop importante. Sur ce jeu de données, si je m'en réfère à la figure 5.7, l'algorithme MEI parvient à énumérer les Δ-jumeaux plus rapidement que l'algorithme MLEI, ce qui s'explique par la différence de complexité temporelle entre les deux algorithmes.

Les résultats obtenus pour l'algorithme MLEI appliqué à l'énumération de Δ-jumeaux dans les graphes temporels constituant les jeux de données `Enron` et `LesFurets`, qui provoquent des problèmes de remplissage mémoire pour l'algorithme MEI, me permettent de valider la propriété 16 : MLEI est capable d'énumérer des Δ-jumeaux dans des graphes temporels sur lesquels MEI rencontre des problèmes de manque de mémoire.

La propriété 16 de supériorité pratique de l'algorithme MLEI sur l'algorithme MEI pour l'énumération de Δ-jumeaux est validée puisque l'algorithme MLEI est capable d'énumérer les Δ-jumeaux dans des graphes temporels sur lesquels l'algorithme MEI rencontre des problèmes de manque de mémoire.

Propriété 17 (Validation du passage à l'échelle de l'énumération de modules éternels).

Je poursuis avec l'énumération des modules éternels et je suis la même démarche expérimentale que pour les Δ-jumeaux. Malheureusement, aucun des deux algorithmes ne parvient à énumérer les modules éternels dans les graphes temporels constituant le jeu de données `Enron`, du fait de

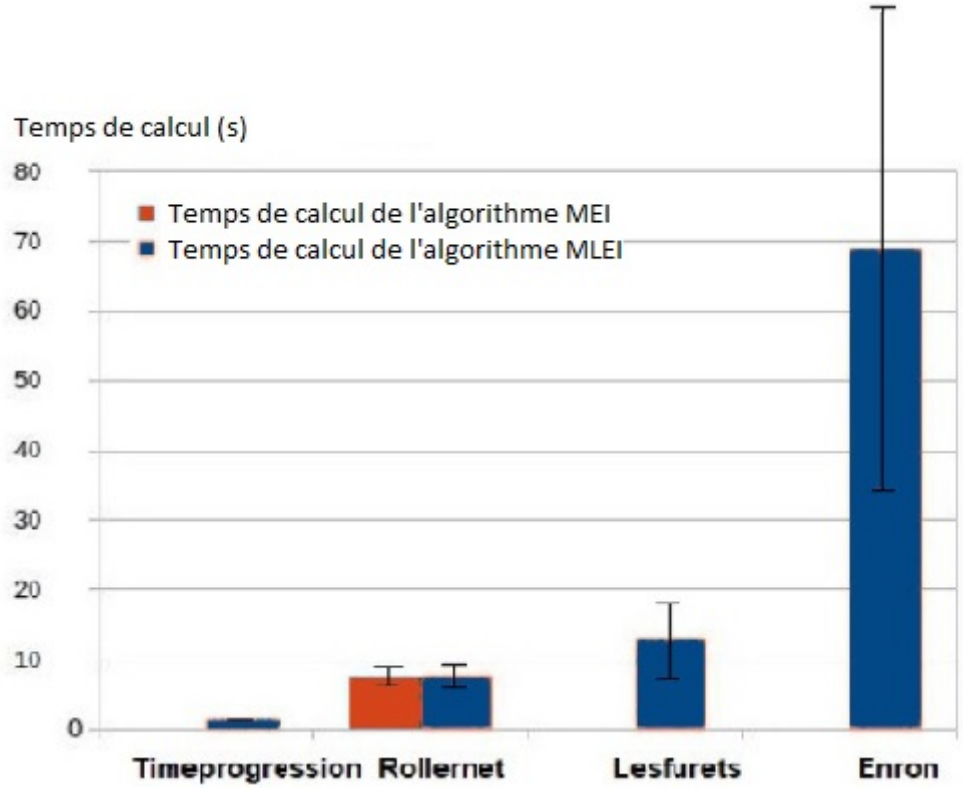

FIGURE 5.10 – Résumé des temps de calcul moyens sur toutes les expériences d'énumération des Δ-jumeaux.

problèmes de mémoire. Les algorithmes MLEI et MEI sont capables d'énumérer les modules éternels dans les graphes temporels constituant le jeu de données `Rollernet` 5.11. Seul MLEI parvient cependant à énumérer les modules éternels sur certains graphes temporels constituant le jeu de données `LesFurets` 5.12. MLEI rencontre des problèmes de mémoire sur les graphes temporels contenant un certain nombre de sommets, d'arêtes et d'instants temporels, ne permettant pas au calcul d'aboutir sur l'intégralité du jeu de données `LesFurets`, ses temps de calcul sur ce jeu de données étant présentés dans la figure 5.12.

Le premier graphique de la figure 5.12, qui présente le temps de calcul de l'algorithme MLEI énumérant les modules éternels en fonction du nombre de sommets sur le jeu de données `LesFurets`, confirme l'influence théorique du nombre de sommets sur le temps de calcul de MLEI, puisque la courbe expérimentale des temps de calcul de l'algorithme MLEI appliqué à l'énumération des modules éternels dans les graphes temporels constituant le jeu de données `LesFurets` en fonction du nombre de sommets semble décrire une fonction polynomiale de degré 3. Le deuxième graphique de la figure 5.12, qui présente le temps de calcul de l'algorithme MLEI énumérant les modules éternels en fonction du nombre d'arêtes temporelles sur le jeu de données `LesFurets`, confirme quant à lui l'influence du nombre d'arêtes temporelles sur le temps de calcul de MLEI, puisque la courbe expérimentale des temps des calcul de l'algorithme MLEI appliqué à l'énumération des modules éternels dans les graphes temporels constituant le jeu de données `LesFurets` en fonction du nombre d'arêtes temporelles semble décrire une fonction polynomiale de degré 2. Attention à ne pas comparer ces résultats sur ce graphe biparti avec la complexité temporelle indiquée dans la partie théorique pour le cas particulier de l'énumération des modules éternels monocolores des graphes bipartis, puisque j'ai ici réalisé le calcul avec l'algorithme d'énumération de modules éternels pour le cas général et non celui pour le cas spécifique des graphes bipartis qui utilise l'énumération de jumeaux éternels. Les résultats obtenus pour l'algorithme MLEI appliqué à l'énumération de modules éternels dans les graphes temporels constituant le jeu de données `LesFurets` invalident en partie la propriété 17 puisque l'influence cubique du nombre de sommets et l'influence quadratique du nombre d'arêtes

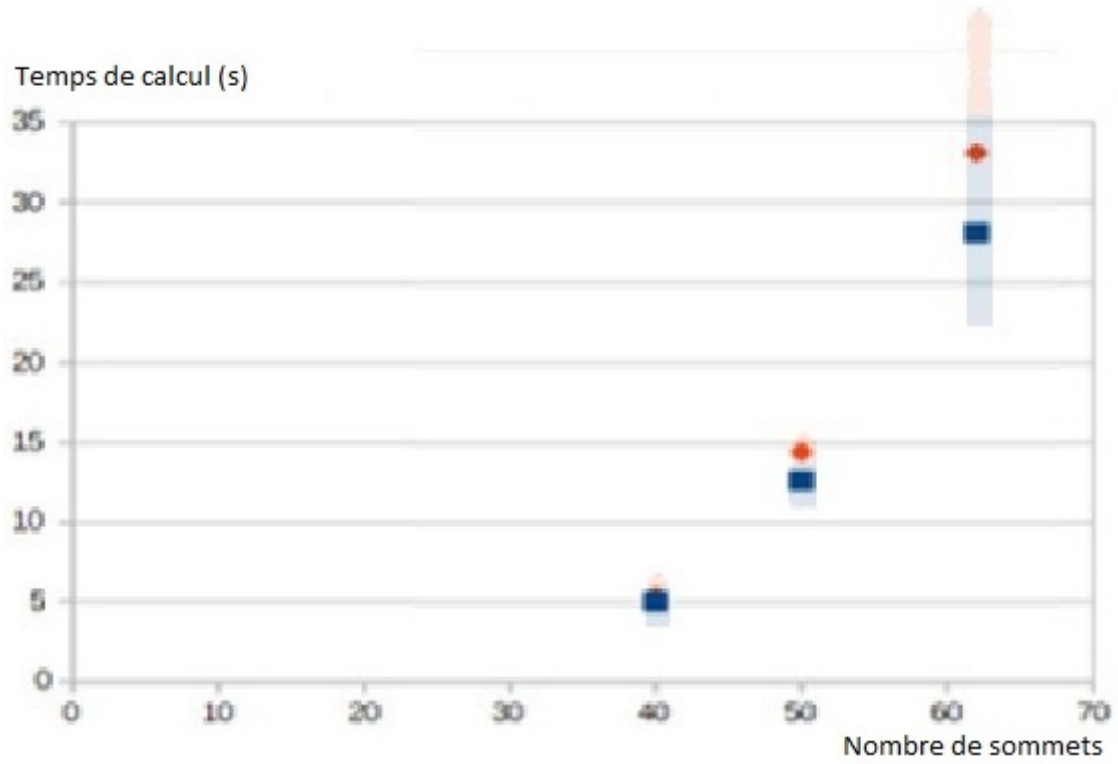

FIGURE 5.11 – Temps de calcul des algorithmes MLEI (en orange) et MEI (en bleu) énumérant les modules éternels, en fonction du nombre de sommets dans les graphes temporels constituant le jeu de données `Rollernet`.

temporelles sur le temps de calcul de l'algorithme entraînent des temps de calculs qui s'éloignent du raisonnable. De fait, avec l'augmentation du nombre de sommets, je subis une augmentation de l'utilisation mémoire pour stocker le résultat, ce qui entraîne des problèmes de manque de mémoire dus à la taille de la solution elle-même. Cette conséquence entraîne l'impossibilité de réaliser des calculs sur le jeu de données `Enron`, ce qui contredit également la propriété 17.

La propriété 17 de passage à l'échelle de l'énumération de modules éternels est donc invalidée par l'expérience sur les jeux de données puisque des problèmes mémoires sont rencontrés lorsque le nombre de sommets augmente et que l'influence trop importante du nombre de sommets et du nombre d'arêtes temporelles sur la complexité temporelle entraîne des temps de calculs qui sortent du domaine du raisonnable tel que défini dans le cadre de cette thèse.

Propriété 18 (Validation de la supériorité pratique de l'algorithme MLEI sur l'algorithme MEI pour l'énumération de modules éternels). Je m'intéresse aux temps de calcul des algorithmes MEI et MLEI appliqués à l'énumération de modules éternels dans les graphes temporels constituant le jeu de données `Rollernet`, les résultats étant présentés dans la figure 5.11.

Il apparaît que MEI est capable d'énumérer les modules éternels dans ces graphes temporels avec un temps de calcul plus faible que l'algorithme MLEI. En revanche, l'algorithme MLEI appliqué à l'énumération de modules éternels parvient à fournir des résultats dans certains graphes temporels constituant le jeu de données `LesFurets` sur lesquels MEI rencontre des problèmes de mémoire dus à la taille de la suite de matrices d'adjacence. Mais l'algorithme MLEI est bien plus limité dans son application à l'énumération de modules éternels que dans son application à l'énumération de Δ-jumeaux et rencontre des problèmes de mémoire sur tous les graphes temporels constituant le jeu de données `Enron` et une partie de ceux constituant le jeu de données `LesFurets`.

La propriété 18 de supériorité pratique de l'algorithme MLEI sur l'algorithme MEI pour l'énumération de modules éternels est validée, l'algorithme MLEI étant capables d'énumérer des modules éternels dans certains graphes temporels sur lesquels l'algorithme MEI rencontre des problèmes de manque de mémoire.

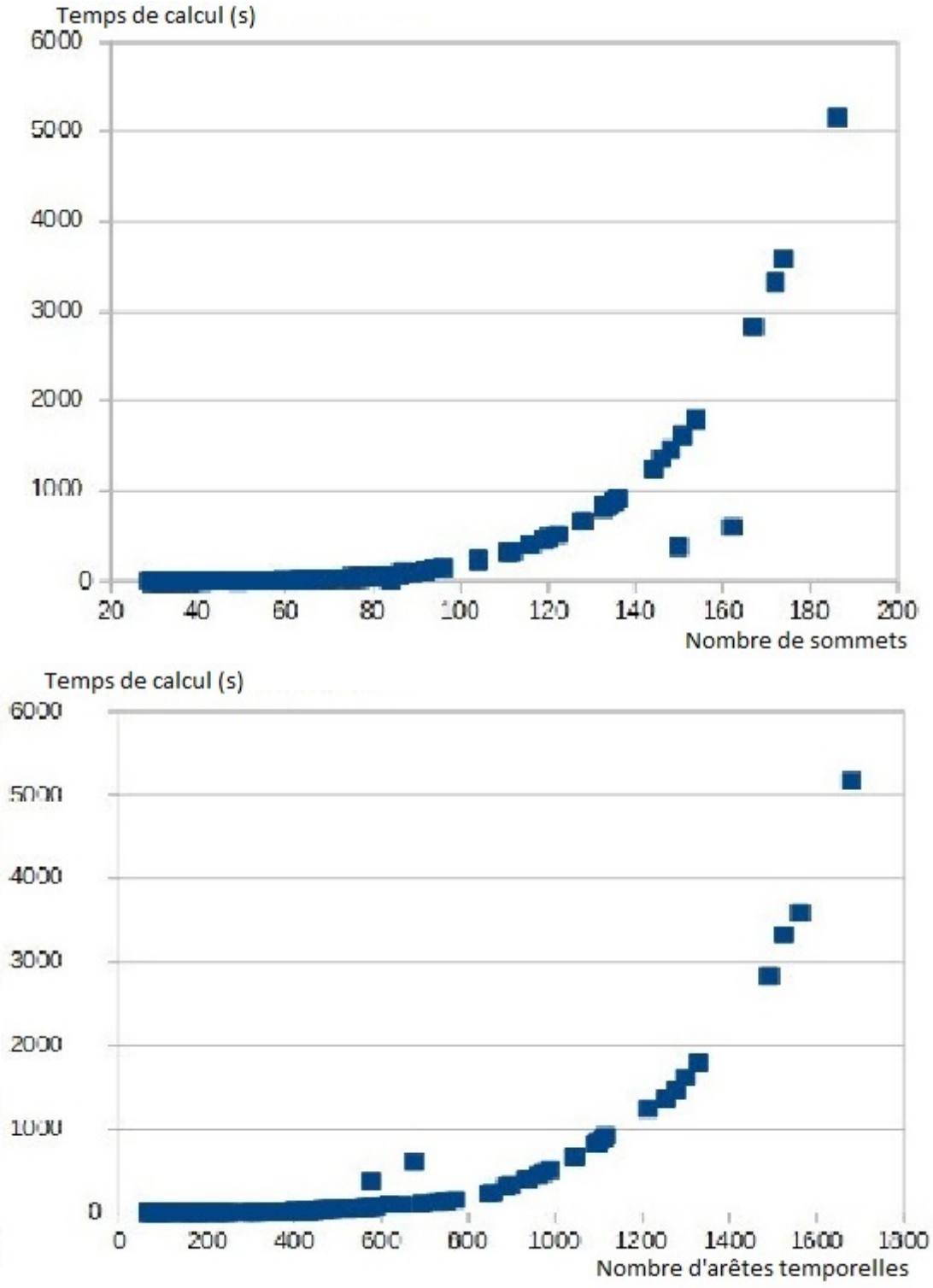

FIGURE 5.12 – Temps de calcul de l'algorithme MLEI énumérant les modules éternels, en fonction du nombre de sommets (en haut) et du nombre d'arêtes (en bas) dans les graphes temporels constituant le jeu de données `LesFurets`.

En conclusion de cette partie sur le passage à l'échelle des algorithmes d'énumération de modules temporels, nous avons établi que :

— L'algorithme MLEI passe à l'échelle pour l'énumération de Δ-jumeaux.
— L'algorithme MLEI est en pratique supérieur à l'algorithme MEI pour l'énumération de Δ-jumeaux puisqu'il est capable d'énumérer des Δ-jumeaux dans des graphes temporels sur lesquels l'algorithme MEI rencontre des problèmes de mémoire.
— L'algorithme MLEI ne passe pas à l'échelle pour l'énumération de modules éternels car sur de nombreux jeux de données, il rencontre des problèmes de manque de mémoire ou présente un temps de calcul qui sort des limites acceptables fixées dans le cadre de cette thèse.
— L'algorithme MLEI est en pratique supérieur à l'algorithme MEI pour l'énumération de modules éternels puisqu'il est capable d'énumérer des modules éternels dans des graphes temporels sur lesquels l'algorithme MEI rencontre des problèmes de mémoire.

Tout ceci considéré, bien qu'intéressants sur le plan théorique, les algorithmes d'énumération de modules éternels présentés ici ne semblent pas tout à fait applicables à des utilisations réelles. Quel que soit le jeu de données choisi parmi ceux utilisés pour cette thèse, ni l'algorithme MEI ni l'algorithme MLEI ne parviennent à énumérer les Δ-modules, du fait de complexités spatiales et temporelles trop importantes. Je dois donc en conclure que l'application de ces algorithmes au problème d'énumération de Δ-modules est théoriquement intéressante mais impossible à utiliser en pratique sur des jeux de données issus d'exploitation réelle.

5.4.2 Étude du passage à l'échelle des algorithmes d'énumération des sous-graphes temporels isomorphes

Le problème d'énumération de sous-graphes isomorphes appartenant à une classe de complexité très élevée, aucun algorithme naïf ne me permet d'énumérer les solutions à aucun des problèmes d'énumération de sous-graphes temporels isomorphes afin d'établir expérimentalement la correction des divers algorithmes. Je ne m'étendrai donc pas sur les questions de correction et me concentrerai sur les temps de calcul de ces divers algorithmes afin d'établir s'il passent à l'échelle ou non.

Propriété 19 (Validation de la supériorité pratique de l'algorithme d'énumération de sous-graphes temporels isomorphes sur l'algorithme d'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire). Afin de vérifier la propriété 19, je compare les temps de calcul de l'algorithme d'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire à ceux de l'algorithme d'énumération de sous-graphes temporels isomorphes sur le jeu de données `Timeprogression` et je présente les résultats expérimentaux en figure 5.13. Je constate que le temps de calcul de l'algorithme de sous-graphes temporels isomorphes est bien plus faible que le temps de calcul de l'algorithme de sous-forêts temporelles isomorphes à une forêt temporelle linéaire, au point d'être négligeable face à ce dernier. Je confirme ainsi expérimentalement la supériorité de l'approche structurelle sur l'approche itérative sur l'historique, qui de plus, présente une complexité spatiale linéaire en τ.

La propriété 19 de supériorité pratique de l'algorithme d'énumération de sous-graphes temporels isomorphes sur l'algorithme d'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire est donc validée par l'expérience, confirmant que l'approche itérative sur l'historique n'est pas une solution efficace pour résoudre les problèmes d'énumération de motifs temporels.

Propriété 20 (Validation du passage à l'échelle de l'énumération de sous-graphes temporels isomorphes par suite d'arêtes). A propos de la propriété 20, je réalise des expériences sur les divers jeux de données présentés ci-dessus avec l'implémentation de l'algorithme d'énumération des sous-graphes temporels isomorphes par suite d'arêtes, un résumé de ces expérimentations étant présenté en figure 5.14, dans laquelle je présente le temps de calcul moyen de cet algorithme sur chaque jeu de données considéré. Cette figure me permet de constater que sur l'ensemble des jeux de données concernés par cette étude expérimentale, cet algorithme est capable d'énumérer des

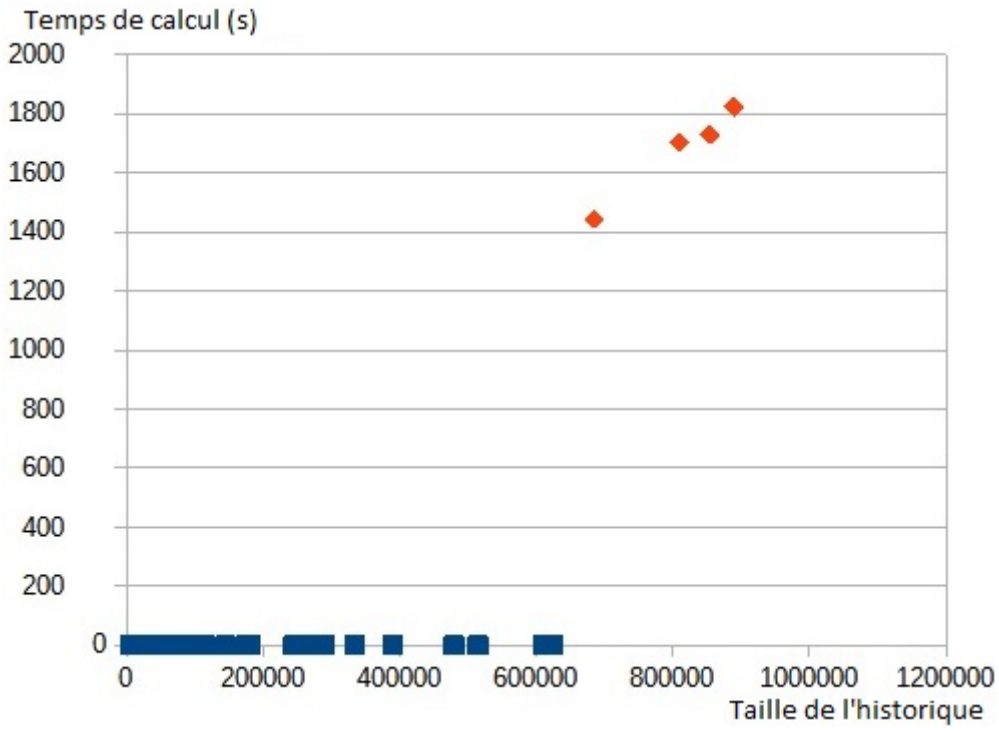

FIGURE 5.13 – Temps de calcul des algorithmes d'énumération des sous-forêts temporelles isomorphes à une forêt temporelle linéaire (en losanges oranges) et d'énumération des sous-graphes temporels isomorphes (en carrés bleus) sur le jeu de données `Timeprogression` en fonction de la taille de l'historique du graphe

sous-graphes isomorphes par suite d'arêtes à des motifs temporels de 2 sommets avec un temps de calcul moyen inférieur à 80 secondes.

Pour plus de précision, je présente également en figure 5.15 l'évolution du temps de calcul de l'algorithme d'énumération de sous-graphes temporels isomorphes par suite d'arêtes en fonction du nombre de sommets dans le graphe temporel sur le jeu de données `Lesfurets`, ce qui me permet de mettre en évidence l'influence exponentielle de ce nombre de sommets sur le temps de calcul de cet algorithme. Néanmoins, ces deux figures tendent à mettre en évidence que pour tous les jeux de données qui m'intéressent ici, cet algorithme est capable d'énumérer les sous-graphes temporels isomorphes par suite d'arêtes à un motif de 2 sommets avec un temps de calcul inférieur à 300 secondes (attention, sur la figure 5.15, les points de données correspondent à des moyennes de temps de calcul pour les instances du jeu de données possédant le même nombre de sommets, ce qui explique que les temps moyens de calcul présentés en figure 5.14 peuvent ne pas correspondre à celui visuellement identifiable sur cette figure). Il pourrait être tentant de conclure que l'algorithme d'énumération des sous-graphes temporels isomorphes par suite d'arêtes présenté dans cette thèse passe à l'échelle mais je n'ai considéré ici que des motifs de 2 sommets.

Or, le principal facteur problématique pour le passage à l'échelle de l'algorithme d'énumération de sous-graphes temporels isomorphes par suite d'arêtes est l'influence du nombre de sommets dans le motif temporel considéré sur la complexité temporelle de l'algorithme. En effet, j'ai théoriquement établi que la complexité temporelle de cet algorithme était exponentiellement dépendante de ce paramètre. Je réalise donc une expérimentation sur le jeu de données `Enron` avec plusieurs valeurs de nombre de sommets dans le motif temporel et présente les résultats expérimentaux dans la figure 5.16. Conformément à l'analyse théorique, je constate sur cette courbe que le temps de calcul moyen sur les graphes du jeu de données `Enron` augmente de manière exponentielle en fonction du nombre de sommets dans le motif temporel, avec une tendance extrêmement rapide puisque, si pour $n' = 2$, n' étant le nombre de sommets dans le motif temporel, le temps de calcul moyen est de l'ordre de la minute ou moins, pour $n' = 3$ on environne les 3 heures de temps de calcul,

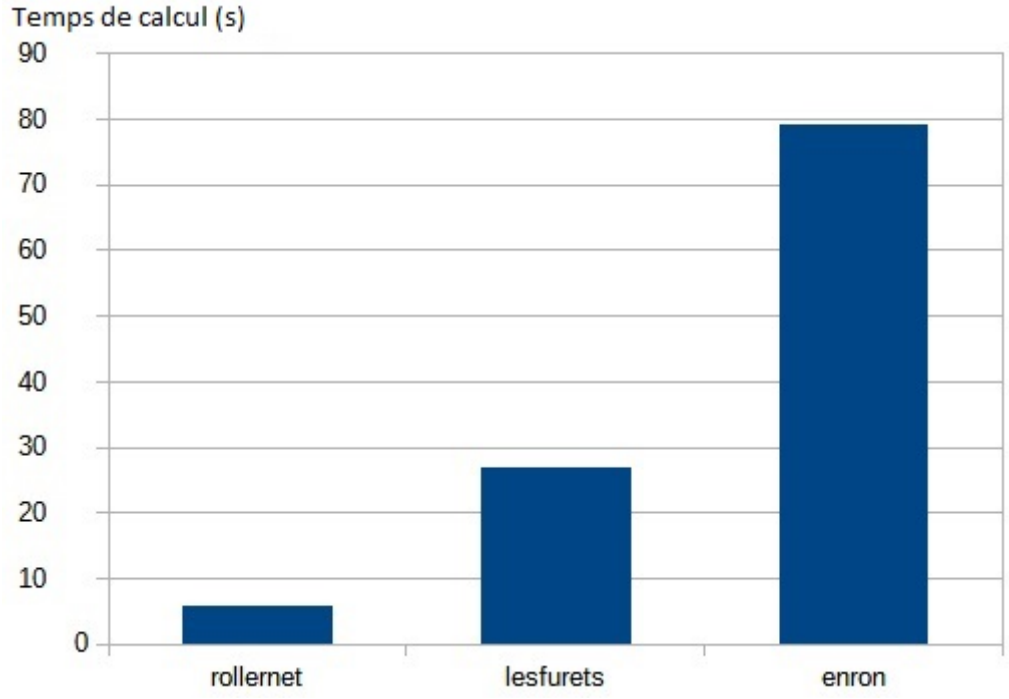

FIGURE 5.14 – Temps de calcul moyen de l'algorithme d'énumération des sous-graphes temporels isomorphes par suite d'arêtes sur les divers jeux de données de cette étude expérimentale

ce qui dépasse largement les limites que j'ai fixées comme acceptables, lorsque le temps de calcul pour $n' = 4$ dépasse les 10 heures (pour tracer cette courbe de manière parlante, j'ai fixé ce point de données à 10 heures mais en réalité, aucun calcul d'énumération de sous-graphes temporels isomorphes par suite d'arêtes n'avait abouti sur le jeu de données `Enron` pour un motif de 4 sommets en 10 heures et le temps de calcul réel est en réalité supérieur). Cette observation rend compte des limites très fortes à l'application de cet algorithme à une utilisation réelle puisqu'en l'état, des résultats ne peuvent être obtenus par l'algorithme d'énumération des sous-graphes temporels isomorphes par suite d'arêtes sur les jeux de données réelles que pour des motifs temporels de 2 sommets (et quelques motifs de 3 sommets). Cela signifie que l'algorithme d'énumération des sous-graphes temporels isomorphes par suite d'arêtes n'est capable d'énumérer que des arêtes temporelles qui existent pour au moins m' instants temporels de T, un résultat qui peut être obtenu par des algorithmes bien plus simples (par exemple, si on construit le graphe temporel en utilisant le formalisme du graphe évolutif ou du graphe variant dans le temps, il n'est même plus nécessaire de recourir à un algorithme pour obtenir cette information qui est enregistrée dans l'arête temporelle elle-même. Avec un motif de 3 sommets, l'algorithme d'énumération des sous-graphes temporels isomorphes par suite d'arêtes a au moins le mérite de pouvoir identifier des motifs un tant soit peu exploitables mais en moyenne, sur ce type de motifs, l'expérience menée sur `Enron` nous apprend que le temps de calcul moyen outrepasse les limites de temps de calcul acceptable.

La propriété 20 de passage à l'échelle de l'énumération de sous-graphes temporels isomorphes par suite d'arêtes est par conséquent invalidée, l'influence du nombre de sommets dans le motif temporel considéré étant bien trop importante, rendant l'algorithme incapable de fournir des résultats pour des motifs temporels de plus de 2 sommets en temps raisonnable. Pour un motif temporel de 2 sommets, des résultats peuvent être obtenus en temps raisonnable mais la pertinence de tels motifs est extrêmement limitée et de tels résultats pourraient être obtenus par d'autres approches en temps négligeable.

Propriété 21 (Validation du passage à l'échelle de l'énumération de sous-graphes temporels isomorphes). A propos de la propriété 21 de passage à l'échelle de l'énumération de sous-graphes temporels isomorphes, je réalise des expérimentations sur l'ensemble des jeux de données et je

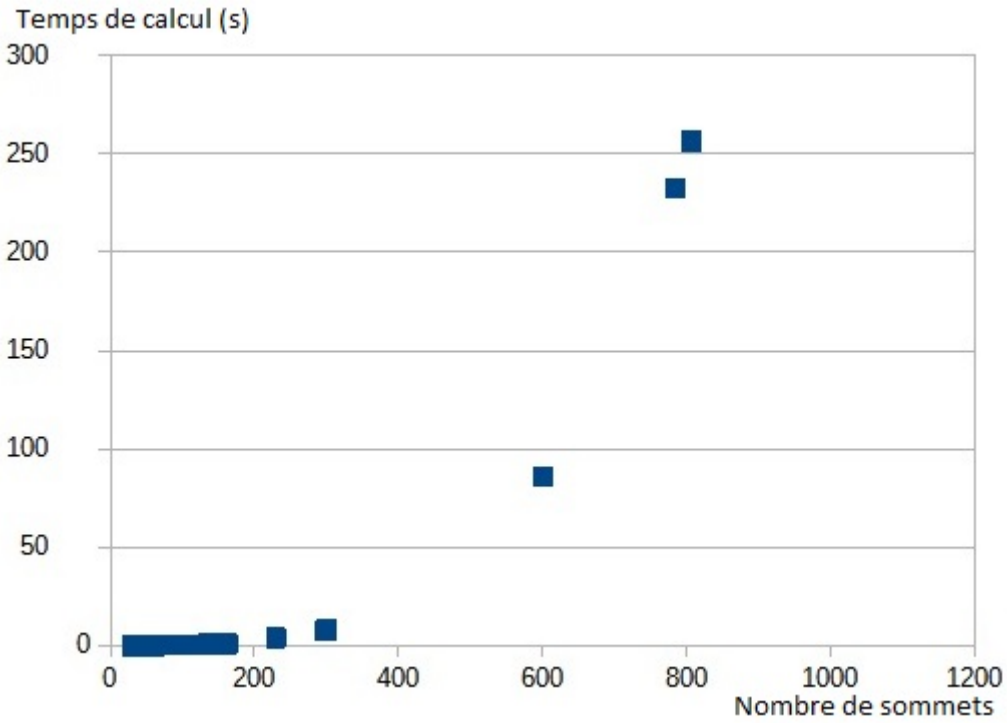

FIGURE 5.15 – Temps de calcul de l'algorithme d'énumération des sous-graphes temporels isomorphes par suite d'arêtes sur le jeu de données `Lesfurets` en fonction du nombre de sommets dans le graphe temporel

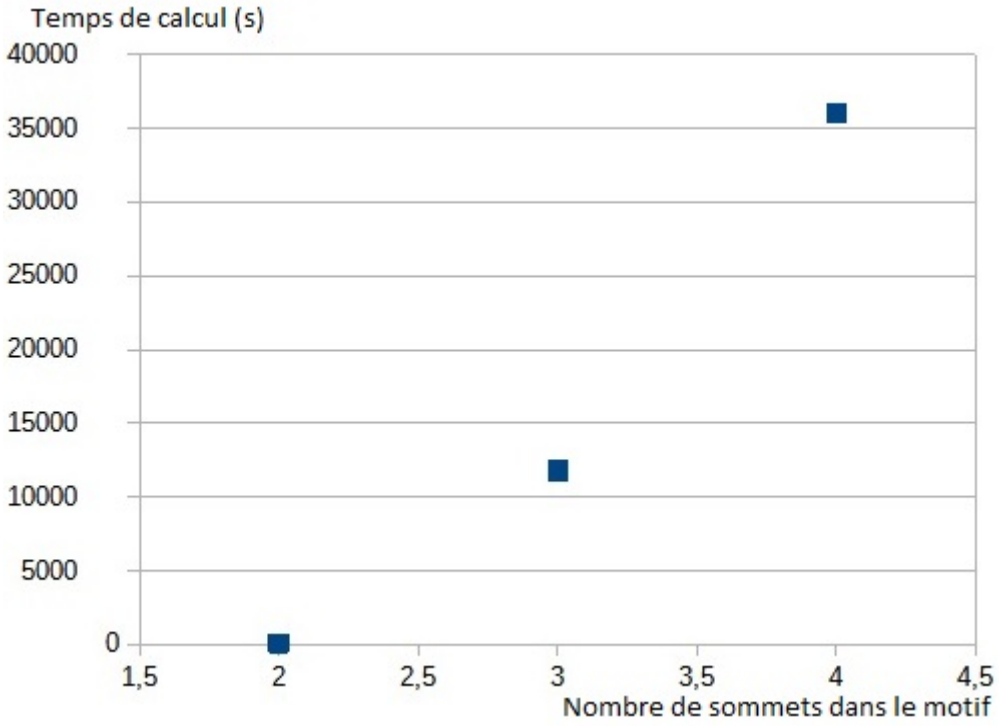

FIGURE 5.16 – Temps de calcul moyen de l'algorithme d'énumération des sous-graphes temporels isomorphes par suite d'arêtes sur le jeu de données `Enron` en fonction du nombre de sommets dans le motif temporel

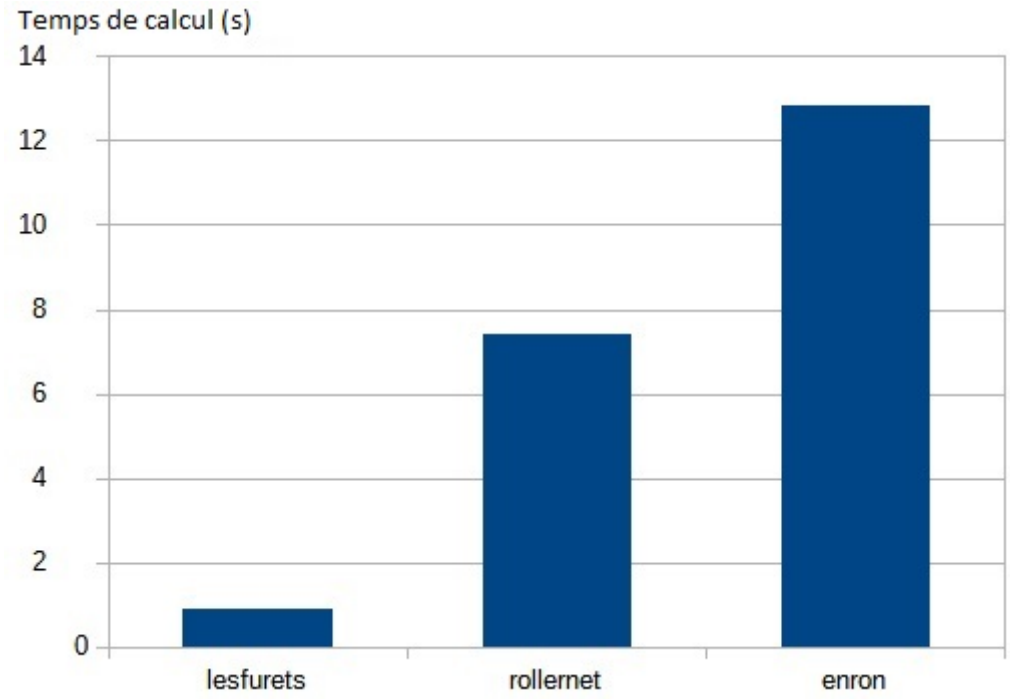

FIGURE 5.17 – Temps de calcul moyen de l'algorithme d'énumération des sous-graphes temporels isomorphes sur les divers jeux de données de cette étude expérimentale

présente une visualisation graphique globale en figure 5.17, qui me permet de constater que pour un motif à 2 sommets, cet algorithme est capable d'énumérer les sous-graphes temporels en moins de 14 secondes sur l'ensemble des jeux de données concernés par cette étude expérimentale.

Pour plus de précision, je présente également en figure 5.18 l'évolution du temps de calcul de l'algorithme d'énumération de sous-graphes temporels isomorphes en fonction du nombre de sommets dans le graphe temporel sur le jeu de données `Enron`, ce qui me permet de mettre en évidence l'influence exponentielle de ce nombre de sommets sur le temps de calcul de cet algorithme, tendance confirmée par la figure 5.19 qui présente l'évolution du temps de calcul de l'algorithme d'énumération de sous-graphes temporels isomorphes en fonction du nombre de sommets dans le graphe temporel sur le jeu de données `LesFurets`. Néanmoins, ces trois figures tendent à mettre en évidence que pour tous les jeux de données qui m'intéressent ici, cet algorithme est capable d'énumérer les sous-graphes temporels isomorphes à un motif de 2 sommets avec un temps de calcul inférieur à 30 secondes (attention, sur les figures 5.18 et 5.19, les points de données correspondent à des moyennes de temps de calcul pour les instances du jeu de données possédant le même nombre de sommets, ce qui explique que les temps moyens de calcul présentés en figure 5.17 peuvent ne pas correspondre à ceux visuellement identifiables sur ces deux figures). Il pourrait être tentant de conclure que l'algorithme d'énumération des sous-graphes temporels isomorphes passe à l'échelle mais je n'ai considéré ici que des motifs de 2 sommets.

Or, le principal facteur problématique pour le passage à l'échelle de l'algorithme d'énumération de sous-graphes temporels isomorphes est l'influence du nombre de sommets dans le motif temporel considéré sur la complexité temporelle de l'algorithme. En effet, j'ai théoriquement établi que la complexité temporelle de cet algorithme était exponentiellement dépendante de ce paramètre. Je réalise donc une expérimentation sur le jeu de données `Rollernet` avec plusieurs valeurs de nombre de sommets dans le motif temporel et présente les résultats expérimentaux dans la figure 5.20. Conformément à l'analyse théorique, je constate sur cette courbe que le temps de calcul moyen sur les graphes du jeu de données `Rollernet` augmente de manière exponentielle en fonction du nombre de sommets dans le motif temporel, avec une tendance extrêmement rapide puisque, si pour $n' = 2$, n' étant le nombre de sommets dans le motif temporel, le temps de calcul moyen est de l'ordre de la minute ou moins, pour $n' = 3$, le temps de calcul moyen est de l'ordre de 1500

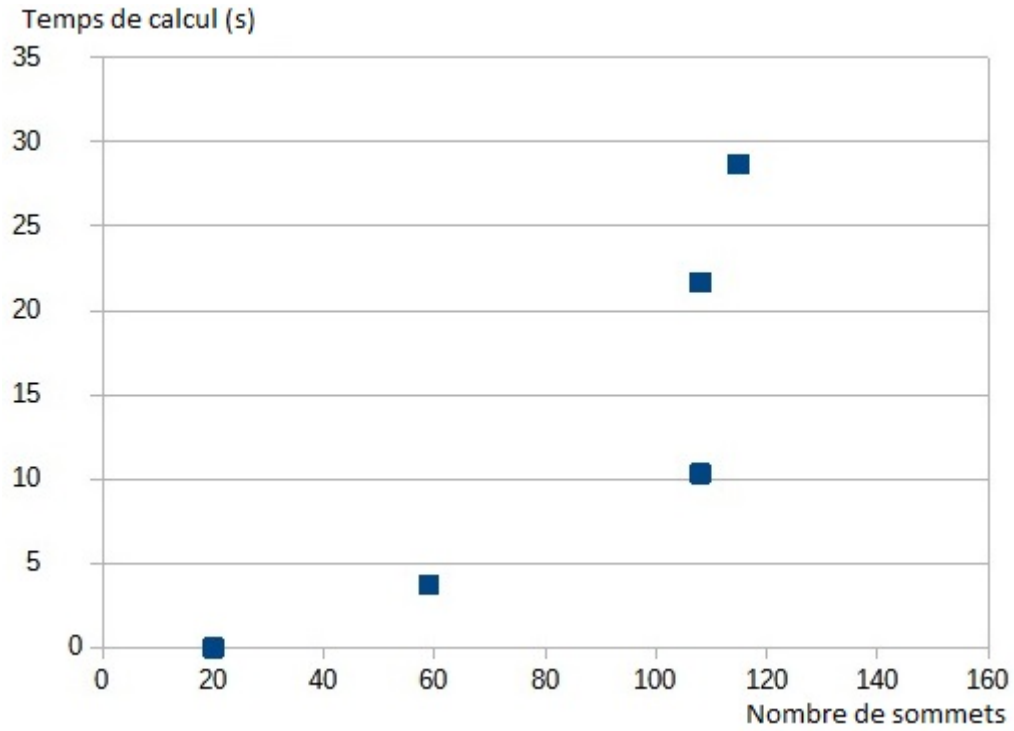

FIGURE 5.18 – Temps de calcul de l'algorithme d'énumération des sous-graphes temporels isomorphes sur le jeu de données `Enron` en fonction du nombre de sommets du graphe temporel

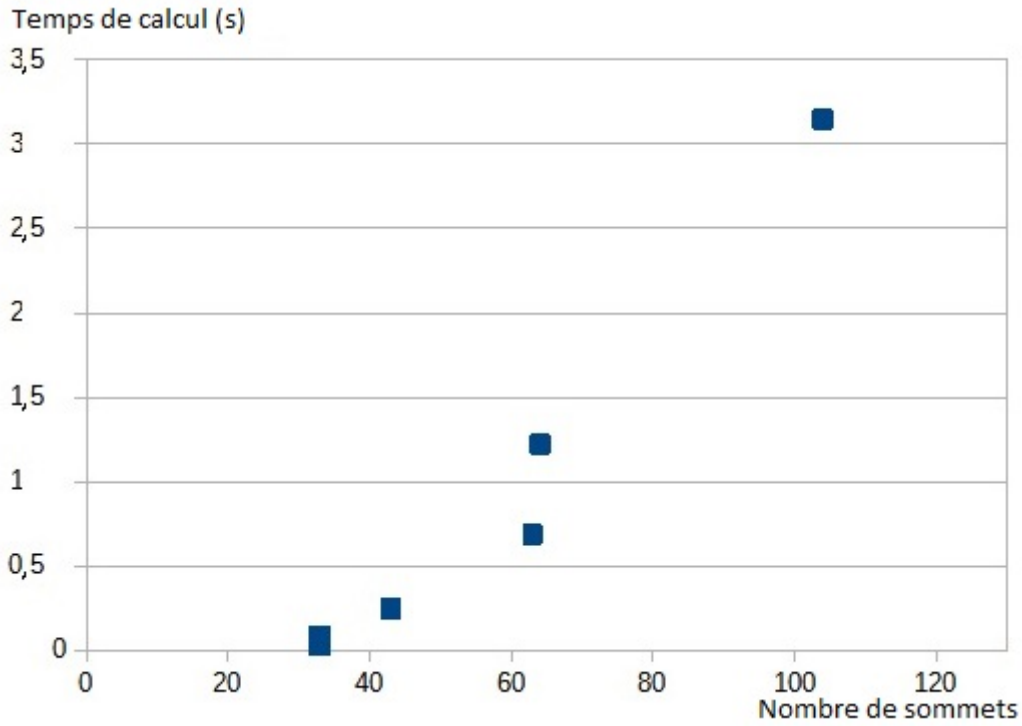

FIGURE 5.19 – Temps de calcul de l'algorithme d'énumération des sous-graphes temporels isomorphes sur le jeu de données `LesFurets` en fonction du nombre de sommets du graphe temporel

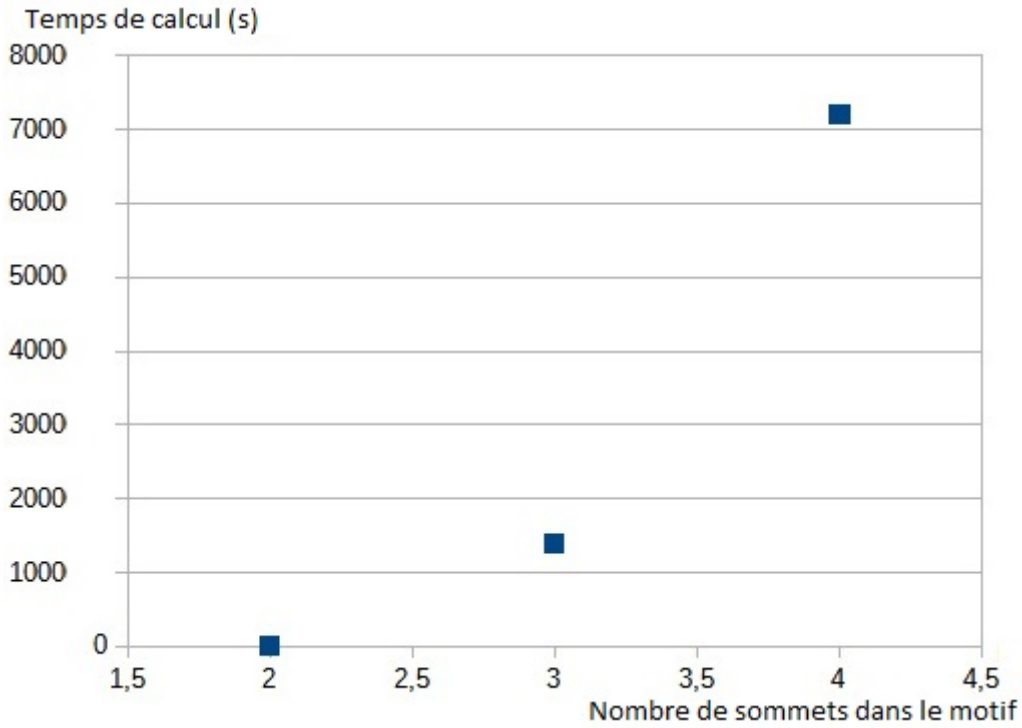

FIGURE 5.20 – Temps de calcul moyen de l'algorithme d'énumération des sous-graphes temporels isomorphes sur le jeu de données `Rollernet` en fonction du nombre de sommets dans le motif temporel

secondes de temps de calcul soit 25 minutes lorsque le temps de calcul pour $n' = 4$ dépasse les 2 heures, ce qui dépasse largement les limites que j'ai fixées comme acceptables (pour tracer cette courbe de manière parlante, j'ai fixé ce point de données à 2 heures mais en réalité, aucun calcul d'énumération de sous-graphes temporels isomorphes n'avait abouti sur le jeu de données `Rollernet` pour un motif de 4 sommets en 2 heures et le temps de calcul réel est en réalité supérieur). Cette observation rend compte des limites très fortes à l'application de cet algorithme à une utilisation réelle puisqu'en l'état, des résultats ne peuvent être obtenus par l'algorithme d'énumération des sous-graphes isomorphes sur les jeux de données réelles que pour des motifs temporels de 2 et 3 sommets. Si cette limite sur ce paramètre est plus laxe que celle qui reposait sur l'algorithme d'énumération des sous-graphes temporels isomorphes par suite d'arêtes, elle est néanmoins à noter, puisqu'au delà de 4 sommets dans le motif temporel, l'algorithme d'énumération des sous-graphes temporels isomorphes n'est pas en mesure de fournir des résultats en temps acceptable.

La propriété 21 de passage à l'échelle de l'énumération de sous-graphes temporels isomorphes est par conséquent quelque peu compromise, l'influence du nombre de sommets dans le motif temporel considéré étant bien trop importante, rendant l'algorithme incapable de fournir des résultats pour des motifs temporels de plus de 3 sommets en temps raisonnable. Pour un motif temporel de 3 sommets, des résultats peuvent être obtenus en temps raisonnable mais la pertinence de tels motifs est quelque peu limitée.

En conclusion de cette partie expérimentale, je vais récapituler les résultats obtenus quant aux propriétés que je cherchais à démonter par ces expérimentations :

— La propriété 15 de passage à l'échelle de l'énumération de Δ-jumeaux est validée, l'algorithme MLEI est capable d'énumérer les Δ-jumeaux en temps acceptable sur l'ensemble des graphes des jeux de données.

— La propriété 16 de supériorité pratique de l'algorithme MLEI sur l'algorithme MEI pour l'énumération de Δ-jumeaux est validée, car si l'algorithme MEI est capable d'aboutir à un résultat plus rapidement que MLEI, il rencontre des problèmes de mémoire sur la majorité

des graphes considérés dans cette étude alors que l'algorithme MLEI parvient à énumérer les Δ-jumeaux dans tous les jeux de données considérés.

— La propriété 17 de passage à l'échelle de l'énumération de modules éternels est invalidée par des problèmes mémoires qui apparaissent lors de l'énumération de modules éternels dans certaines instances de graphe temporel, même si dans l'ensemble, l'algorithme parvient à énumérer les modules éternels dans une forte proportion des jeux de données.

— La propriété 18 de supériorité pratique de l'algorithme MLEI sur l'algorithme MEI pour l'énumération de modules éternels est validée puisque l'algorithme MLEI est capable d'énumérer les modules éternels dans un grand nombre de graphes sur lesquels MEI rencontre des problèmes de mémoire. Il est néanmoins à noter que l'algorithme MLEI rencontre lui aussi des problèmes de manque de mémoire sur certains graphes.

— La propriété 19 de supériorité pratique de l'algorithme d'énumération de sous-graphes temporels isomorphes sur l'algorithme d'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire est validée, l'algorithme d'énumération de sous-graphes temporels isomorphes étant capable d'énumérer les sous-forêts bien plus vite que l'algorithme d'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire, au point que le temps de calcul du premier soit quasiment négligeable par rapport à celui du deuxième. Ce résultat permet de valider le pré-supposé qu'une approche basée sur des propriétés structurelles du problème temporel est préférable à une approche visant à résoudre le problème statique équivalent itérativement sur toutes les instances de graphes instantanés.

— La propriété 20 de passage à l'échelle de l'algorithme d'énumération de sous-graphes temporels isomorphes par suite d'arêtes est invalidée car si des résultats peuvent être obtenus en temps raisonnable pour un motif temporel de 2 sommets, le temps de calcul devient bien trop grand dès lors qu'on passe à un motif temporel à 3 sommets. Cet algorithme n'est par conséquent pas utilisable en pratique.

— La propriété 21 de passage à l'échelle de l'algorithme d'énumération de sous-graphes temporels isomorphes est invalidée car si des résultats peuvent être obtenus en temps raisonnable pour des motifs temporels de 2 ou 3 sommets, le temps de calcul devient bien trop grand dès lors qu'on passe à un motif temporel à 4 sommets. Si cet algorithme permet de détecter en temps raisonnable des motifs comportementaux entre trois acteurs, son application en pratique à des motifs impliquant 4 agents ou plus est fortement limitée.

Pour conclure quant à cette étude expérimentale des algorithmes présentés dans cette thèse, si les algorithmes d'énumération de Δ-jumeaux et de modules éternels présentent des résultats intéressants, notamment du fait de la faible influence de τ sur leurs complexités temporelles et spatiales, l'algorithme d'énumération des Δ-modules ne passe pas à l'échelle sur les jeux de données qui m'intéressent dans cette étude. Quant aux algorithmes d'énumération des isomorphismes, la supériorité de l'algorithme d'énumération de sous-graphes temporels isomorphes sur l'algorithme d'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire établit que l'approche structurelle est préférable à une approche agissant par itération sur l'historique d'algorithmes apportant solution au problème statique associé. Les algorithmes d'énumération de sous-graphes temporels isomorphes et de sous-graphes temporels isomorphes par suite d'arêtes se révèlent capables de passer à l'échelle sur les jeux de données à condition que la taille du motif temporel soit extrêmement limitée, notamment en terme de nombre de sommets. Au-delà de 2 ou 3 sommets cependant, et bien que les complexités spatiales et temporelles de ces algorithmes soient indépendantes de τ, la trop forte influence du nombre de sommets dans le motif temporel sur ces complexités, notamment la complexité temporelle, entraîne des temps de calcul qui sortent des limites fixées dans le cadre de cette thèse comme acceptables. L'utilisation pratique de ces algorithmes est donc limitée à des détections de motifs d'interactions entre 2 ou 3 acteurs. Si elle s'avère limitée, cette utilisation pratique permet cependant de résoudre des problèmes spécifiques comme l'identification d'une situation de blocage dans laquelle un utilisateur se retrouve dans un cycle, répétant 2 actions. Par exemple, dans notre cas d'étude des comportements utilisateurs sur

un site d'e-commerce, l'algorithme d'énumération des sous-graphes temporels isomorphes permet de détecter des utilisateurs qui se retrouvent bloqués par une des questions posées par le site et qui reviennent à la question précédente pour modifier leur réponse. Cet algorithme permet alors d'identifier les utilisateurs bloqués en fonction du nombre d'aller-retours qu'ils réalisent entre ces deux questions et en fonction du temps qui s'écoule entre leurs différentes actions. La possibilité limitée d'utilisation de cet algorithme n'empêche donc pas sa pertinence pour répondre à des problèmes spécifiques.

Chapitre 6

Conclusion

Dans cette thèse, j'ai approché le problème de détection de motifs dans des graphes temporels dans le but d'identifier des comportements d'utilisateurs sur un site internet d'e-commerce. J'ai étudié deux approches de cette détection : l'énumération de jumeaux et modules temporels d'une part et l'énumération de sous-graphes temporels isomorphes de l'autre.

Mon but principal est de garantir des complexités spatiales et temporelles des algorithmes les moins dépendantes de τ possible, c'est à dire garantir une influence de τ sur ces complexités au pire cas logarithmique. En effet, les graphes sur lesquels je souhaitais utiliser ces algorithmes se démarquaient par une grande taille d'historique et une faible densité d'arêtes temporelles.

Dans le cas des modules, je suis parvenu à concevoir des algorithmes diminuant l'influence de la taille de l'historique du graphe temporel sur les complexité temporelle et spatiale, en utilisant des structures de données d'arbres rouge-noir et des propriétés de construction par union des modules (Propriété 3.3.8). Je suis ainsi parvenu à apporter une solution aux cas particuliers de l'énumération :

— Des jumeaux éternels en temps indépendant de la taille de l'historique (en section 3.2).
— Des Δ-jumeaux en temps logarithmique de la taille de l'historique (en section 3.2).
— Des modules éternels en temps indépendant de la taille de l'historique (en section 3.3).

Les deux premiers résultats de cette liste ont pu être confirmés par une étude expérimentale sur un jeu de test généré (voir section 3.4).

En revanche, la solution proposée dans cette thèse pour le problème général d'énumération des Δ-modules ne représente qu'un intérêt purement théorique car sa complexité rend son utilisation pratique très limitée. Cet algorithme n'a en effet pas pu fournir de résultats sur les jeux de données concernés par l'étude expérimentale. Je précise à nouveau que dans le cadre d'un graphe biparti, le problème d'énumération des Δ-modules monocolores est corrélé au problème d'énumération des Δ-jumeaux et que les algorithmes indiqués dans cette thèse comme solution au problème d'énumération des Δ-jumeaux peuvent donc être utilisés pour énumérer les Δ-modules monocolores sur cette classe de graphe, pour une complexité logarithmique de la taille de l'historique. De sorte que sur les jeux de données bipartis, cette thèse apporte une réponse au cas particulier des Δ-modules monocolores en temps raisonnable, même pour des graphes de grande taille d'historique, à condition que ceux-ci restent relativement peu denses.

Dans le cas de l'isomorphisme de sous-graphes temporels, la comparaison des temps de calcul de l'algorithme d'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire (construit par itération d'un algorithme apportant une solution au problème statique de l'isomorphisme de sous-forêts isomorphes à une forêt linéaire) et de l'isomorphisme de sous-graphes temporels (construit par analyse structurelle du problème) permet de conclure quant à la supériorité de l'approche structurelle sur l'approche statique itérative.

Les algorithmes présentés dans cette thèse comme solution aux problèmes d'énumération de sous-graphes temporels isomorphes par suite d'arêtes et d'énumération de sous-graphes temporels isomorphes (qui utilisent des résultats liant ces problèmes à l'énumération de sous-graphes statiques isomorphes, et des résultats liant cette dernière énumération à l'énumération de clique de taille k)

permettent d'apporter une solution à ces deux problèmes en temps indépendant de τ. Cependant, la forte influence d'autres facteurs sur la complexité temporelle de ces deux algorithmes, notamment l'influence exponentielle du nombre de sommets dans le motif temporel, limitent fortement les utilisations pratiques de ces algorithmes. En l'état, au-delà de 3 sommets dans le motif temporel, les algorithmes d'énumération de sous-graphes temporels isomorphes ne sont pas en mesure de fournir de résultat en temps raisonnable. Par conséquent, bien qu'intéressants théoriquement, ces résultats ont une utilisation limitée en pratique sur des graphes temporels issus de données d'exploitation réelle.

Pour ce qui est des algorithmes présentés comme solution des problèmes d'énumération de jumeaux et modules temporels, l'un des principaux problèmes semble être la complexité spatiale, du fait de la structure de données des arbres rouge-noir utilisés pour stocker les réponses partielles. Une ouverture possible serait de modifier cette structure de données pour y stocker les divers modules instantanés contenant chaque paire de sommets sous une forme plus synthétique, par exemple en remplaçant ces arbres par un graphe présentant l'évolution d'une arborescence de modules minimaux et des modules pouvant être construits par leur union.

Pour le problème de l'énumération de sous-graphes isomorphes, le principal problème des algorithmes présentés dans cette thèse semble être la forte influence du nombre de sommets dans le motif temporel. Cet impact sur la complexité temporelle est du à l'accroissement du nombre d'isomorphismes candidats entre des sommets de G et les sommets de P, c'est-à-dire une augmentation de $\binom{n}{n'}$. Une piste pour diminuer cette influence in fine sur la complexité temporelle serait de mettre en place des pré-traitements permettant de discréditer le plus grand nombre possibles de candidats par une étude structurelle des deux graphes temporels (un module temporel dans P doit avoir pour image un module temporel dans $(A \subseteq V, E_A, t_0 + T)$ par exemple). Si un grand nombre de candidats peut être éliminé, l'influence du nombre de sommets dans le motif temporel sur le temps de calcul de l'énumération de sous-graphes temporels isomorphes sera diminuée, permettant d'appliquer la solution présentée dans cette thèse à des motifs de taille supérieure.

Il est également à noter que plusieurs des résultats théoriques que j'ai présentés dans le cadre de cette thèse n'ont pu être validés par les expérimentations et que plusieurs questions restent en suspens suite à l'étude du passage à l'échelle, notamment l'influence de τ sur l'utilisation mémoire lors de l'énumération de sous-forêts temporelles isomorphes à une forêt temporelle linéaire, ou sur l'utilisation mémoire et le temps de calcul de l'énumération de modules éternels. De futures expérimentations sont donc également à prévoir, sur des jeux de données différents, générés spécifiquement pour étudier plus en profondeur ces questions, afin de compléter l'analyse expérimentale de ces algorithmes, même si dans les deux cas, ce qui a pu être établi pour ces deux algorithmes, c'est qu'ils ne passaient pas à l'échelle sur les jeux de données issus d'exploitation réelle.

Pour une utilisation pratique de ces divers algorithmes, une adaptation de ceux-ci pour construire des algorithmes "au fil de l'eau" (dits online) peut également présenter une ouverture intéressante, puisque le calcul n'aura alors qu'à se soucier de la modification de la solution induite par l'introduction d'une nouvelle arête. De tels algorithmes pourront être optimisés en n'opérant que sur les modules ou isomorphismes de sous-graphes content les sommets de la nouvelle arête introduite et il est envisageable que leur complexité soit suffisament faible pour pouvoir être applicable même sur des graphes de grande taille.

La parallélisation et l'optimisation des structures de données utilisées dans les implémentations des algorithmes présentés dans cette thèse sont également deux pistes d'amélioration de ces résultats, qui pourraient permettre de diminuer fortement le temps de calcul.

Ainsi, si le travail présenté dans cette thèse présente un intérêt pratique quelque peu limité, les résultats théoriques présentés ouvrent la porte à de plus amples études en vue d'aboutir à des programmes permettant l'identification de motifs temporels en temps raisonnable sur des graphes temporels à long historique.

Bibliographie

[1] E.-C. Akrida, G.-B. Mertzios, P.-G. Spirakis, and V. Zamaraev. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 107 :108–123, 2020.

[2] D.-A. Basin. A term equality problem equivalent to graph isomorphism. *Information Processing Letters*, 51(2) :61–66, 1994.

[3] A. Bergeron, C. Chauve, F.-D. Montgolfier, and M. Raffinot. Computing common intervals of k permutations, with applications to modular decomposition of graphs. *European Symposium on Algorithms*, pages 779–790, 2005.

[4] A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. *International Computing and Combinatorics Conference*, pages 68–79, 2003.

[5] J.-A. Bondy and U.-S.-R. Murty. *Graph theory with applications*, volume 290. Macmillan, 1976.

[6] K.-S. Booth. Isomorphism testing for graphs, semigroups, and finite automata are polynomially equivalent problems. *SIAM Journal on Computing*, 7(3) :273–279, 1978.

[7] B.-M. Bui-Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2) :267–285, 2003.

[8] B.-M. Bui-Xuan, H. Hourcade, and C. Miachon. Computing temporal twins in time logarithmic in history length. In *9th International Conference on Complex Networks and their Applications*, volume 944 of *SCI*, pages 651–663, 2020.

[9] B.-M. Bui-Xuan, H. Hourcade, and C. Miachon. Computing small temporal modules in time logarithmic in history length. In *Social Network Analysis and Mining*, volume 12, 2022.

[10] M.-P. Béal, A. Bergeron, S. Corteel, and M. Raffinot. An algorithmic view of gene teams. *Theoretical Computer Science*, 320(2-3) :395–418, 2004.

[11] S. Bérard, A. Bergeron, and C. Chauve. Conservation of combinatiorial structures in evolution scenarios. *RECOMB Workshop on Comparative Genomics*, pages 1–14, 2004.

[12] R. Carraghan and Pardalos P.-M. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9 :375–382, 1990.

[13] A. Casteigts, P. Flocchini, E. Godard, N. Santoro, and M. Yamashita. Expressivity of time-varying graphs. In *19th International Symposium on Fundamentals of Computation Theory*, pages 95–106, 2013.

[14] B.-C. Dean. *Continuous-Time Dynamic Shortest Path Algorithms*. PhD thesis, Massachusetts Institute of Technology, 1999.

[15] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner. Connection scan algorithm. *ACM Journal of Experimental Algorithmics*, 23, 2018.

[16] A. Ehrenfeucht, T. Harju, and G. Rozenberg. *The Theory of 2-Structures : A Framework for Decomposition and Transformation of Graphs*. World Scientific, 1999.

[17] T. Erlebach, M. Hoffmann, and F. Kammer. On Temporal Graph Exploration. In *42nd International Colloquium on Automata, Languages, and Programming*, volume 9134 of *LNCS*, pages 444–455, 2015.

[18] J. Fiala and D. Paulusma. A complete complexity classification of the role assignment problem. *Theoretical computer science*, 349(1) :67–81, 2005.

[19] L. Foschini, J. Hershberger, and S. Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4) :1075–1097, 2014.

[20] J. Gagneur, R. Krause, T. Bouwmeester, and G. Casari. Modular decomposition of protein-protein interaction networks. *Genome Biology*, 5(8) :1–12, 2004.

[21] T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18(1-2) :25–66, 1967.

[22] M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1) :41–59, 2010.

[23] M. Habib, C. Paul, and L. Viennot. Partition refinement techniques : an interesting algorithmic tool kit. *International Journal of Foundations of Computer Science*, 10(2) :147–170, 1999.

[24] F. Harary, M. Plantholt, and R. Statman. The graph isomorphism problem is polynomially equivalent to the legitimate deck problem for regular graphs. *Caribbean Journal of Math*, 1 :15–23, 1982.

[25] S.-T. Hedetniemi. Homomorphisms of graphs and automata. *Michigan University Annual Arborescence and Communication Sciences Program*, 1966.

[26] C.-M. Hoffmann. *Group-theoretic algorithms and graph isomorphism*. Springer Berlin, 1982.

[27] P. Holme and J. Saramäki. Temporal networks. *Physics reports*, (519) :97–125, 2012.

[28] J.-E. Hopcroft. An n log n algorithm for minimizing the states in a finite automaton. *The Theory of Machines and Computations*, pages 189–196, 1971.

[29] D.-S. Johnson and M.-R. Garey. *Computers and intractability : A guide to the theory of NP-completeness*. WH Freeman, 1979.

[30] D. Kempe, J. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4) :820–842, 2002.

[31] B. Klimt and Y. Yang. Introducing the Enron Corpus. In *CEAS*, 2004.

[32] D. Kozen. A clique problem equivalent to graph isomorphism. *ACM SIGACT News*, 10 :50–52, 1978.

[33] J. Köbler, U. Schöning, and J. Toran. *The graph isomorphism problem : its structural complexity*. Springer Science and Business Media, 2012.

[34] M. Latapy, T. Viard, and C. Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining*, 8(61), 2018.

[35] A. Lubiw. Some np-complete problems similar to graph isomorphism. *SIAM Journal on Computing*, 10(1) :11–21, 1981.

[36] F. Maffray and Preissmann M. A translation of tibor gallai's paper : Transitiv orientierbare graphen. *Perfect Graphs*, pages 25–66, 2001.

[37] A. Mansfield. The relationship between the computational complexities of the legitimate deck and isomorphism problems. *Quarterly Journal of Mathematics*, 33(3) :345–347, 1982.

[38] R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3) :131–136, 1979.

[39] G.-L. Miller. Graph isomorphism, general remarks. *Proceedings of the 9th annual ACM symposium on Theory of computing*, pages 143–150, 1979.

[40] A. Paranjape, Benson A.-R., and Leskovec J. Motifs in temporal networks. *Proceedings of the Tenth ACM Internation Conference on Web Search and Data Mining*, pages 601–610, 2017.

[41] U. Redmond and P. Cunningham. Temporal subgraph isomorphism. *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2013.

[42] F.-J. Ros, P.-M. Ruiz, and I. Stojmenovic. Acknowledgment-based broadcast protocol for reliable and efficient data dissemination in vehicular ad-hoc networks. *IEEE Transactions on Mobile Computing*, 11(1) :33–46, 2012.

[43] J. Spinrad. *Efficient Graph Representations. Field Institute Monographs*, volume 19. American Mathematical Society, 2003.

[44] P.-U. Tournoux, J. Leguay, F. Benbadis, V. Conan, M.-D. De Amorim, and J. Whitbeck. The Accordion Phenomenon : Analysis, Characterization, and Impact on DTN routing. In *28th IEEE Conference on Computer Communications*, 2009.

[45] I. Tsalouchidou, R. Baeza-Yates, F. Bonchi, K. Liao, and T. Sellis. Temporal betweenness centrality in dynamic graphs. *International Journal of Data Science and Analytics*, pages 1–16, 2019.

[46] T. Viard, C. Magnien, and M. Latapy. Enumerating maximal cliques in link streams with durations. *Information Processing Letters*, 133 :44–48, 2018.

[47] I. Wegener. *Complexity theory : exploring the limits of efficient algorithms*. Springer Science and Business Media, 2005.

[48] D.-R. White and K.-P. Reitz. Graph and semigroup homomorphisms on networks of relations. *Social Networks*, 5(2) :193–234, 1981.

[49] Q. Zhao, Y. Tian, Q. He, N. Oliver, Jin R., and W.-C. Lee. Communication motifs : a tool to characterize social communications. *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1645–1648, 2010.

Table des figures

Table des matières