Licence Informatique

La Rochelle Université

GALACTIC Core
algorithmes de treillis de concepts en Rust

Julien Linares

2026

Table des matières

Liste des figures

Abstract

Résumé en français

Ce rapport présente mon stage au laboratoire **L3i** de La Rochelle Université, au sein de l'équipe **Modèle et connaissances**, dans le cadre du projet **GALACTIC**. Le cadriciel GALACTIC propose une analyse de données transparente et interactive basée sur l'Analyse Formelle des Concepts. L'objectif du stage est d'améliorer les performances des algorithmes de treillis de concepts en développant un noyau en `Rust`, tout en préparant l'exposition des fonctionnalités vers d'autres environnements.

Le travail s'organise autour d'une structure de données optimisée (`Context`) reposant sur des bitmaps compressés et des opérations efficaces sur les relations. Plusieurs algorithmes ont été implémentés en version itérative (CloseByOne, InClose4, InClose5, NextClosure) afin de mieux contrôler la mémoire et d'éviter les limites de récursion. Une pipeline CI a été mise en place (linting, tests, build, benchmarks) et une campagne d'évaluation a permis d'analyser l'impact de la densité et de la dispersion des données sur les performances.

Les résultats montrent des gains significatifs sur les algorithmes de treillis, en particulier avec InClose5, et une amélioration globale de la robustesse. Le projet pose les bases d'une exposition vers Python via `Maturin` et d'une API RESTful adaptée aux traitements volumineux. Les perspectives incluent une version 64 bits, des benchmarks mémoire, la finalisation de la documentation et la publication sur crates.io.

Abstract in English

This report presents my internship at the **L3i** laboratory of La Rochelle University, within the **Model and Knowledge** team, in the context of the **GALACTIC** project. GALACTIC is a transparent and interactive data analysis framework based on Formal Concept Analysis. The internship aims to improve the performance of concept lattice algorithms by developing a `Rust` core while preparing their exposure to other environments.

The work is built on an optimized data structure (`Context`) using compressed bitmaps and efficient relation operations. Several algorithms were implemented in iterative form (CloseByOne, InClose4, InClose5, NextClosure) to improve memory control and avoid recursion limits. A CI pipeline was set up (linting, tests, build, benchmarks), and an evaluation campaign analyzed how data density and dispersion affect performance.

Results show significant gains for lattice algorithms, especially with InClose5, and improved overall robustness. The project lays the groundwork for Python bindings via `Maturin` and a RESTful API suited to large result sets. Future work includes a 64-bit version, memory benchmarks, documentation finalization, and publication on crates.io.

1 Introduction

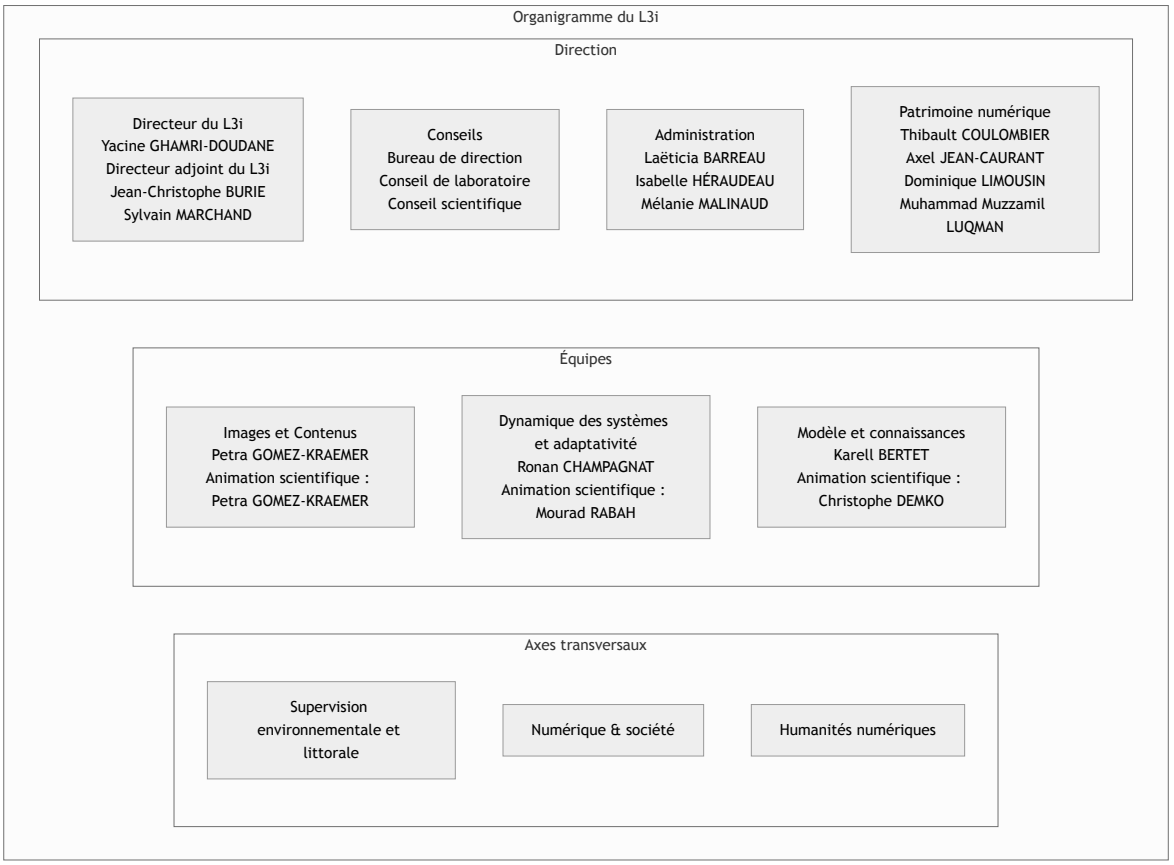
1.1 Le Laboratoire Informatique, Image et Interaction

Le **L**aboratoire **I**nformatique, **I**mage et **I**nteraction (**L3i**) est un centre de recherche de **La Rochelle Université**. Ses recherches se distinguent par l'accent mis sur les interactions humaines. (« L3i – Laboratoire Informatique, Image et Interaction » 2025)

- **1993** : création du « Laboratoire Informatique et Imagerie Industrielle »;
- **1997** : labellisation en Équipe d'Accueil (Ministère de la Recherche);
- **2003** : changement de nom en « Laboratoire Informatique, Image et Interaction »;
- **2007** : labellisation en Équipe de Recherche en Technologie « Interactivité Numérique »;
- **2008** : intégration dans le programme régional **PRIDES**;
- **2011** : obtention de la note A à la suite de l'évaluation de l'**A**gence d'**E**valuation de la **R**echerche et de l'**E**nseignement **S**upérieur (**AERES**).

1.1.1 Organigramme

Durant mon stage au **L3i**, je suis affecté à l'équipe **Modèle et connaissances**. Le présent organigramme est daté de septembre 2025. Le **L3i** est composé d'environ **110 membres**.



1.1.2 Équipes de recherche

Le **L3i** mène ses travaux autour de trois axes principaux organisés en équipes :

- **M**odèles et **C**onnaissances (**MC**) : travaux sur les modèles formels, la représentation des connaissances, les données spatio-temporelles et les systèmes de raisonnement. L'équipe participe à des projets comme I-Trace et **GALACTIC**.
- **I**mages et **C**ontenus (**IC**) : recherche sur le traitement, l'analyse et la structuration de contenus multimédias faiblement structurés (images, vidéos, textes). Elle s'intéresse à l'extraction, l'indexation et la recherche d'informations.
- Dynamique des systèmes et adaptativité (**eAdapt**) : étude des systèmes logiciels adaptatifs, notamment ceux qui réagissent au comportement de l'utilisateur. L'équipe travaille sur des architectures logicielles dynamiques et le pilotage du traitement en fonction du contexte.

1.1.3 Partenariats du L3i

Depuis sa création, le **L3i** collabore avec de nombreux partenaires **académiques**, **industriels** et **institutionnels**.

Il est à l'origine du **consortium** *Valconum* pour la valorisation des recherches numériques.

Le laboratoire entretient des partenariats **internationaux** avec des institutions comme le *Computer Vision Center* (Barcelone), l'*Université de Hanoï* et l'*Université de Sfax*, ainsi qu'avec des laboratoires **nationaux** tels que le *GIPSA-lab* et l'*INRAE*.

Il travaille également avec plusieurs **entreprises**, comme *AM Créations*, *Arkhênum* et *Shift Technology*, ainsi qu'avec des **institutions publiques** telles que la *Région Poitou-Charentes* et la *Communauté d'Agglomération de La Rochelle*.

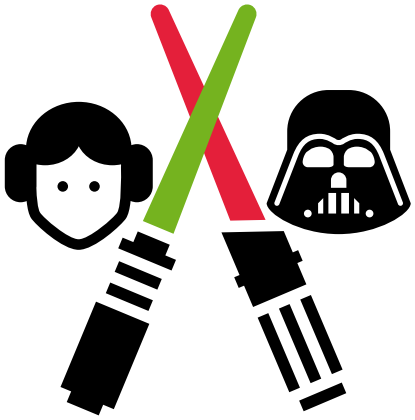
1.2 Le cadriciel GALACTIC

GAlois **LA**ttices, **C**oncept **T**heory, **I**mplicational systems and **C**losures (**GALACTIC**) Demko et al. (2024) est un projet de recherche débuté en janvier 2004. Les référents du projet sont **Karell BERTET** (directrice de recherche) et **Christophe DEMKO** (architecte logiciel). (« GAlois LAttices, Concept Theory, Implicational Systems and Closures » 2026)

GALACTIC se distingue par son approche « boîte blanche », c'est-à-dire qu'elle reste **transparente** sur les procédés appliqués aux données. Elle permet ainsi aux analystes de données d'explorer de manière interactive et progressive les données.

L'**interactivité** permet à l'analyste de cibler les stratégies adaptées à son contexte, de limiter les traitements sans intérêt et coûteux en ressources, et de favoriser le **numérique responsable**.

Étymologie
Le projet **GALACTIC** s'appuie sur les travaux du mathématicien français Évariste GALOIS, décédé en 1832 lors d'un duel. Son nom a inspiré celui du projet.



1.2.1 Architecture

Le noyau de **GALACTIC** est structuré en plusieurs couches sur lesquelles peuvent venir se greffer des extensions :

- **les caractéristiques**, qui permettent d'extraire tout type de valeur à partir des données;
- **les descriptions**, qui définissent un ensemble de valeurs (et donc de données) à travers les frontières d'une enveloppe convexe généralisée;
- **les stratégies**, qui explorent un ensemble de données en se concentrant sur des groupes répondant à certains prédicats monadiques;
- **les mesures**, qui alimentent des méta-stratégies en informations quantitatives;
- **les lecteurs de données**, qui permettent d'importer des fichiers de tout format pour constituer des ensembles de données.

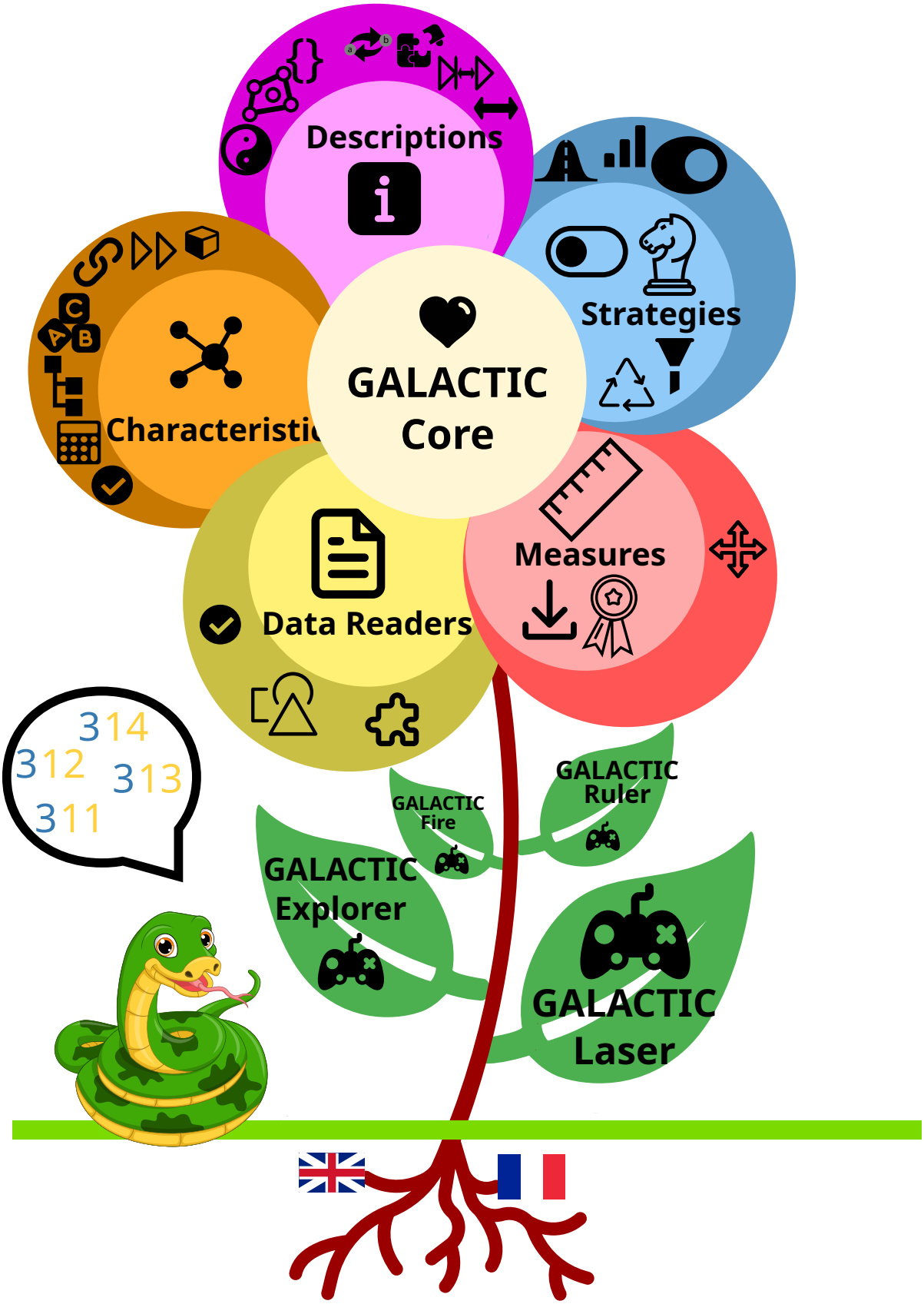


FIGURE 1 – Architecture de **GALACTIC**

1.2.2 Fondements scientifiques

GALACTIC est basé sur l'**A**nalyse **F**ormelle des **C**oncepts (**AFC**) parue en 1982. Elle organise les données en sous-groupes à partir de descriptions communes et exploite les propriétés algébriques des **treillis** et des **opérateurs de fermeture**.

En 2020, deux verrous scientifiques ont été levés : l'**analyse de données complexes et hétérogènes** et le **déluge des motifs descriptifs de sous-groupes**.

Exemple
Dans le cadre du **Dispositif d'Analyse des Traces numériques pour la valorisation des Territoires Touristiques** (**DA3T**), des touristes à La Rochelle ont été tracés (avec leur accord, via une application smartphone) pour étudier leurs mouvements. Les localisations spatiales et temporelles sont des données hétérogènes et complexes. **GALACTIC** permet de traiter ces données pour en fournir une **lecture plus explicite**.

1.2.3 Partenariats du projet GALACTIC

GALACTIC est financé par le *réseau SATT* ainsi que le **L3i**. Il compte de nombreux partenaires, par exemple l'*OIP Atlantique* [1]

1. Introduction reprise et adaptee du rapport de stage de Cyprien Robinaud (2025), avec mises a jour.

1.3 Le noyau GALACTIC

Le projet consiste à implémenter et exposer des algorithmes de treillis de concepts en `Rust`. Certains de ces algorithmes, comme NextPriorityConcept, étaient implémentés en `Python` et formaient un goulot d'étranglement pour le projet **GALACTIC**, notamment son coeur. L'objectif est de fournir une librairie performante et facile à utiliser pour les chercheurs et les praticiens travaillant avec des treillis de concepts.

1.3.1 Comprendre un concept

Avant toute chose, il est important de comprendre ce qu'est un concept dans l'Analyse Formelle des Concepts (AFC). Un **concept** se décrit toujours par **deux faces inséparables** :

- l'**extension** : l'ensemble des objets qui partagent certains attributs ;
- l'**intension** : l'ensemble des attributs communs à ces objets.

Ces deux faces se « verrouillent » l'une l'autre : si l'on choisit un ensemble d'objets, on récupère les attributs communs (intension). Si l'on choisit un ensemble d'attributs, on récupère les objets qui les possèdent tous (extension). Un concept, c'est donc un **groupe maximal d'objets** avec **un paquet d'attributs communs**.

On considère un petit contexte formel avec trois objets (animaux) et cinq attributs simples.

Objet	bruit	courir	voler	poil	plume
chien	✓	✓		✓	
chat	✓	✓		✓	
oiseau	✓		✓		✓

1) **Recherche des objets possédant un ensemble d'attributs donné (extension)**

- Attributs {poil} → objets qui ont du poil = **{chien, chat}**
- Attributs {voler} → objets qui volent = **{oiseau}**
- Attributs {bruit} → objets qui font du bruit = **{chien, chat, oiseau}**

2) **Recherche des attributs communs à un ensemble d'objets donné(intension)**

- Objets {chien, chat} → attributs communs = **{bruit, courir, poil}**
- Objets {oiseau} → attributs communs = **{bruit, voler, plume}**
- Objets {chien, chat, oiseau} → attributs communs = **{bruit}**

Un concept formel est une paire (A, B) où **A est l'extension** (ensemble d'objets) et **B est l'intension** (ensemble d'attributs), vérifiant la propriété fondamentale : l'extension de A égale B et l'intension de B égale A. Mathématiquement : α(A) = B et β(B) = A.

- **({chien, chat}, {bruit, courir, poil})**
  - Si on prend {chien, chat}, on obtient {bruit, courir, poil}.

  - Si on prend {bruit, courir, poil}, on retombe sur {chien, chat}.
- **({oiseau}, {bruit, voler, plume})**
  - {oiseau} donne {bruit, voler, plume} et inversement.
- **({chien, chat, oiseau}, {bruit})**
  - Tous font du bruit, et l'attribut {bruit} renvoie bien aux trois.

Un concept en AFC n'est pas une simple catégorie « intuitive » : c'est un **couplage exact** entre un ensemble d'objets et les attributs qu'ils partagent, et **aucun des deux ne peut être élargi sans perdre l'autre**. C'est cette précision qui permet ensuite de construire un treillis de concepts.

1.3.2 Objectifs

1. Implémenter l'algorithme CloseByOne en Rust
   - Utilisation de structures de données optimisées (bitmaps compressés).
   - Intégration de la librairie *roaring*.
   - Validation de la correction algorithmique et évaluation des performances.
2. Généraliser l'algorithme
   - Permettre différents parcours d'un treillis de concepts.
   - Adapter l'implémentation à la complétion d'un opérateur de fermeture.
   - Concevoir une architecture modulaire facilitant l'extension.
3. Créer une interface Python
   - Réaliser un *mapping* des fonctionnalités Rust vers Python.
   - Fournir une *API* simple pour l'expérimentation et le prototypage.
   - Documenter les usages.
4. Développer une *API RESTful* en Rust
   - Exposer les fonctionnalités via un serveur.
   - Support natif de la notion d'itérateur pour le traitement de résultats volumineux.
   - Conception orientée performance et robustesse.

2 Mise en place du projet (méthodologie)

Il était nécessaire de structurer le projet et le stage. J'ai donc commencé par me renseigner sur l'architecture et les outils que j'allais utiliser, afin d'en faire une roadmap et d'anticiper les étapes à suivre pour mener à bien le projet.

Je n'avais jamais travaillé avec `Rust` auparavant, ni avec les algorithmes de treillis de concepts. J'ai donc commencé par me documenter sur ces sujets afin d'avoir une bonne base pour le projet.

Je me suis d'abord renseigné sur le fonctionnement de Rust, les systèmes d'ownership et de borrowing, et j'ai réalisé, durant les deux premiers jours, des projets de test et d'apprentissage du langage. Ensuite, j'ai étudié les algorithmes de treillis de concepts, notamment CloseByOne (Andrews 2018), puis l'environnement de développement `Rust` et les outils associés comme `cargo`, `rustfmt` et `clippy`, la librairie *`roaring`*, les méthodes d'exposition de fonctionnalités Rust vers Python avec `Maturin`, et enfin les *API RESTful* et la manière de les implémenter en `Rust`.

Suite à ces recherches, j'ai organisé une réunion avec mon maître de stage, M. Demko, pour lui présenter mes résultats et définir avec lui les besoins du projet et les structures à implémenter.

Cela m'a permis d'établir une roadmap pour le projet et de définir les étapes à suivre. Même si je présente la structure du projet en top-down, j'ai adopté une approche bottom-up pour la mise en place du projet : j'ai commencé par implémenter les structures de données et les algorithmes de base avant de passer à l'implémentation de l'*API* et à l'exposition vers Python.

Roadmap

1. Librairie de treillis de concepts en Rust
   - ☒ Mise en place du projet Rust (cargo)
   - ☒ Mise en place d'une *pipeline* (*linting*, tests CI, *build*)
   - ☒ Création de la structure de données en utilisant *roaring* et les bitmaps compressés (`Context`)
   - ☒ Implémentation des méthodes liées à la structure de données (création, suppression, modification : domain, codomain et relation)
   - ☒ Ajout de méthodes plus sûres avec gestion des erreurs, mais moins performantes
   - ☒ Ajout à la *pipeline* d'une v1 de *benchmark* pour évaluer les performances de la structure de données (Criterion)
   - ☒ Implémentation de `intent` et `extent` pour la structure de données
   - ☒ Implémentation d'opérateurs sur les relations (intersection, union, différence et différence symétrique sur `predecessors` et `successors`)
   - ☒ Refactor de la structure du projet afin de le rendre plus modulaire et facile à maintenir
   - ☒ Rework des *datasets* de *benchmark* avec une loi de Bernoulli pour la génération de données aléatoires
   - ☒ Optimisation des méthodes d'`intent` et d'`extent` grâce à l'intersection par lot min-k
   - ☒ Documentation de la librairie Rust et ajout d'exemples d'utilisation
   - ☒ Implémentation de l'algorithme CloseByOne original en version itérative (Kuznetsov 1993)

- ☒ Rework des *datasets* de *benchmark* pour évaluer les performances des algorithmes CloseByOne
- ☒ Implémentation des algorithmes InClose4 et InClose5 en itératif (Andrews 2018)
- ☒ Implémentation de NextClosure en itératif
- ☒ *Benchmark* et comparaison de l'autre seule librairie de FCA en Rust (Dominik 2026)
- ☐ Implémentation des algorithmes CloseByOne, NextClosure, InClose4 et InClose5 en version récursive pour comparaison
- ☐ Création d'un itérateur pour les algorithmes de treillis de concepts afin de traiter les résultats volumineux de manière efficace en mémoire, ce qui est essentiel pour l'exposition de ces algorithmes via une *API RESTful*
- ☐ Ajout d'un *benchmark* mémoire au *benchmark* existant pour évaluer les performances en mémoire de la librairie Rust
- ☐ Création d'une version 64 bits de la librairie avec *Roaring Treemap* au lieu de *Roaring Bitmap* pour permettre de traiter des treillis de concepts plus volumineux
- ☐ Ajout de la documentation finale et d'exemples d'utilisation pour la librairie Rust
- ☐ Publication de la librairie sur crates.io
- ☐ Ajout de la publication sur crates.io à la *pipeline* CI/CD

2. Outils et interfaces : Maturin & *API RESTful* en Rust
   - ☐ Mise en place du projet Python avec Maturin
   - ☐ *mapping* des fonctionnalités Rust vers Python
   - ☐ Création d'une *API* simple pour l'expérimentation et le prototypage en Python
   - ☐ Documentation des usages de l'*API* Python avec Sphinx
   - ☐ Implémentation d'une *API RESTful* en Rust pour exposer les fonctionnalités de la librairie de treillis de concepts
   - ☐ Support natif de la notion d'itérateur pour le traitement de résultats volumineux dans l'*API RESTful*
   - ☐ Documentation de l'*API RESTful* et exemples d'utilisation
   - ☐ Mise en production de l'*API RESTful* et du package Python
3. Bilan, perspectives et suite du projet
   - ☒ Rédaction d'un rapport de stage détaillé présentant les travaux réalisés, les résultats obtenus et les perspectives d'évolution du projet
   - ☐ Présentation des résultats du projet lors de la soutenance de stage
   - ☐ Rédaction d'un article scientifique présentant les travaux réalisés, les performances obtenues et les perspectives d'évolution du projet
   - ☐ Maintien et évolution de la librairie Rust, de l'*API* Python et de l'*API RESTful* en fonction des retours des utilisateurs et des besoins du projet **GALACTIC**.

3 Conception et développement

3.1 Librairie de treillis de concepts en Rust

3.1.1 Environnement de développement

Suite à mes projets d'apprentissage de Rust, j'ai choisi d'utiliser `JetBrains RustRover` comme environnement de développement pour ce projet. En local, j'utilise les outils de mon IDE pour le *linting* et les tests, et pour la CI j'ai mis en place une *pipeline* GitLab définie dans `.gitlab-ci.yml` et lancée sur un *runner* **GALACTIC** dédié.

C'était la première fois que je réalisais une *pipeline* CI/CD. J'ai donc dû me renseigner sur les bonnes pratiques pour mettre en place une *pipeline* efficace et adaptée à ce projet. J'ai choisi d'utiliser d'abord trois étapes : le *linting*, les tests et le *build*, puis j'ai ajouté une étape de *benchmark* pour évaluer les performances de la librairie Rust.

Pour le *linting*, j'ai utilisé `rustfmt` pour le formatage du code et `clippy` pour l'analyse statique. Pour les tests et le *build*, j'ai utilisé `cargo`, l'outil de *build* de Rust. Pour le *benchmark*, j'ai utilisé la librairie `Criterion`, un *framework* de *benchmark* pour Rust.

3.1.2 Structure de données

La structure de données principale de la librairie est le `Context`, qui représente un contexte formel, c'est-à-dire un ensemble d'objets, d'attributs et de relations entre eux. Le `Context` est défini dans le dossier `context`, dans le fichier `model.rs`, et est implémenté en utilisant des bitmaps compressés pour les objets, les attributs et les relations. Cela permet d'optimiser les performances de la librairie.

Ces relations entre objets et attributs sont représentées par deux `HashMap` : `successors` et `predecessors`. `successors` mappe un objet à l'ensemble de ses attributs, tandis que `predecessors` mappe un attribut à l'ensemble de ses objets. Ces deux structures permettent d'implémenter efficacement les méthodes d'`intension` et d'`extension`, ainsi que les opérateurs sur les relations.

Les trois dernières variables sont simplement des compteurs pour les objets et attributs, ainsi qu'un numéro de version modifié à chaque changement du contexte pour vérifier la validité des itérateurs sur les treillis de concepts.

3.1.3 Implémentation de base

Les méthodes de base de la librairie sont les méthodes de création, suppression et modification du `Context`, c'est-à-dire l'ajout et la suppression d'objets, d'attributs et de relations entre eux. Ces méthodes sont implémentées dans le fichier `base.rs`. À cause de notre structure de données, qui privilégie les performances des algorithmes de treillis de concepts, ces méthodes, notamment celles de suppression d'objets et d'attributs, réduisent quelque peu les performances de la librairie car elles nécessitent de mettre à jour les relations entre les objets et les attributs.

J'ai ensuite implémenté les opérateurs sur les relations, c'est-à-dire l'intersection, l'union, la différence et la différence symétrique sur les `successors` et `predecessors`. Ces méthodes sont implémentées dans le fichier `ops.rs`.

3.1.4 Algorithmes de treillis de concepts

Les algorithmes de treillis de concepts sont implémentés dans le dossier `algorithms`. J'ai commencé par implémenter l'algorithme CloseByOne original en version itérative (Kuznetsov 1993) dans le fichier `cbo_original.rs`, puis j'ai implémenté les algorithmes InClose4 et InClose5 en version itérative (Andrews 2018) dans les fichiers `in_close4.rs` et `in_close5.rs`, et enfin l'algorithme NextClosure en version itérative dans le fichier `nextclosure.rs`.

Pour les implémenter en version itérative, j'ai suivi les algorithmes originaux et j'ai utilisé des piles pour simuler la récursion.

J'ai choisi d'implémenter ces algorithmes en version itérative car cela permet de mieux gérer la mémoire et d'éviter les problèmes de *stack overflow* qui peuvent survenir avec des algorithmes récursifs sur des treillis de concepts volumineux.

Ces algorithmes étaient le goulot d'étranglement de l'ancienne version de **GALACTIC** en Python, et les implémenter en Rust permet de bénéficier de meilleures performances.

3.1.5 Arborescence actuelle de la librairie

La librairie est actuellement structurée de la manière suivante :

```
|-- .gitignore
|-- .gitlab-ci.yml
|-- Cargo.lock
|-- Cargo.toml
|-- README.md
|
|-- benches
|   |-- benchmark.rs
|   |
|   `-- datasets
|       `-- mod.rs
|
`-- src
   |-- lib.rs
   |
   |-- algorithms
   |   |-- cbo_original.rs
   |   |-- formal_concept.rs
```

```
|   |-- in_close4.rs
|   |-- in_close5.rs
|   |-- mod.rs
|   `-- nextclosure.rs
|
`-- context
    |-- base.rs
    |-- intent_extent.rs
    |-- mod.rs
    |-- model.rs
    |-- ops.rs
    `-- safe.rs
```

Cette structure permet de séparer les différentes parties de la librairie et de faciliter la maintenance et l'évolution du projet. Le dossier `src` contient le code source de la librairie, le dossier `benches` contient les *benchmarks* pour évaluer les performances, et les fichiers à la racine contiennent les configurations et la documentation du projet.

Cette implémentation, qui sépare algorithmes et structures de données, rend le projet plus modulaire : un utilisateur peut utiliser la structure de données sans forcément utiliser les algorithmes de treillis de concepts.

3.2 Outils et interfaces : Maturin & *API RESTful* en Rust

Exposition de la librairie Rust vers Python avec Maturin

Cette étape n'est pas encore réalisée, mais j'ai déjà commencé à me renseigner sur les différentes méthodes d'exposition de fonctionnalités Rust vers Python. J'ai choisi d'utiliser `Maturin`, un outil de *build* pour créer des packages Python à partir de code Rust. Cet outil est simple à utiliser et permet de créer des packages Python facilement installables via pip.

C'est une étape essentielle pour connecter ma librairie au coeur de **GALACTIC**, qui est principalement développé en Python, et permettre aux chercheurs travaillant avec des treillis de concepts de bénéficier des performances de la librairie Rust tout en continuant à utiliser Python pour leurs expérimentations et prototypages.

3.3 Qualité du code, tests et documentation

Comme indiqué précédemment, la librairie est implémentée de façon modulaire avec séparation en modules pour les algorithmes et les méthodes liées à la structure de données. Chaque algorithme ou type de méthode a son propre fichier, ce qui facilite la maintenance et l'évolution du projet. J'ai également suivi les bonnes pratiques de Rust pour l'implémentation du code, notamment en utilisant des structures de données optimisées pour les performances et en appliquant correctement les règles de Rust.

En plus de la *pipeline* CI pour le linting avec `rustfmt` et `clippy`, les tests unitaires permettent de vérifier la correction du code et de s'assurer que les modifications n'introduisent pas de régressions. J'ai également mis en place des *benchmarks* pour évaluer les performances de la librairie, identifier les goulots d'étranglement et mesurer l'impact des modifications. Tous les tests essaient de couvrir un maximum de cas d'utilisation afin de garantir une librairie robuste et fiable.

La documentation est réalisée au fur et à mesure de l'implémentation des différentes fonctionnalités. Elle est écrite en anglais pour être accessible au plus grand nombre et est générée automatiquement à partir des commentaires dans le code grâce à `rustdoc`. J'ai également ajouté des exemples d'utilisation pour illustrer les différentes fonctionnalités de la librairie et faciliter la prise en main.

4 *Benchmark* et évaluation des performances

Les *benchmarks* ont été réalisés avec quatre versions de *datasets* et des méthodes d'évaluation différentes où :

- $|G|$ représente le nombre d'objets
- $|M|$ représente le nombre d'attributs
- $|I|$ représente la taille de la relation
- δ représente la densité de la relation
- σ représente la dispersion entre attributs

version	nom du *dataset*	$\lvert G\rvert$	$\lvert M\rvert$	$\lvert I\rvert$	δ	σ	groupe
v1	small	5000	5000	50_000	-	-	1
v1	medium	20_000	20_000	200_000	-	-	1
v1	large	50_000	50_000	500_000	-	-	1
v3 / v4	small-sfixed-delta-low	5000	5000	-	0.0002	12.0	2
v3 / v4	small-sfixed-delta-mid	5_000	5_000	-	0.002	12.0	2
v3 / v4	small-sfixed-delta-high	5_000	5_000	-	0.01	12.0	2
v3 / v4	small-dfixed-s-low	5_000	5_000	-	0.002	5.0	2
v3 / v4	small-dfixed-s-mid	5_000	5_000	-	0.002	12.0	2
v3 / v4	small-dfixed-s-high	5_000	5_000	-	0.002	30.0	2
v3 / v4	medium-sfixed-delta-low	20_000	20_000	-	0.0002	12.0	2
v3 / v4	medium-sfixed-delta-mid	20_000	20_000	-	0.002	12.0	2
v3 / v4	medium-sfixed-delta-high	20_000	20_000	-	0.01	12.0	2
v3 / v4	medium-dfixed-s-low	20_000	20_000	-	0.002	5.0	2
v3 / v4	medium-dfixed-s-mid	20_000	20_000	-	0.002	12.0	2
v3 / v4	medium-dfixed-s-high	20_000	20_000	-	0.002	30.0	2

version	nom du *dataset*	$\|G\|$	$\|M\|$	$\|I\|$	δ	σ	groupe
v4	slow-500-sfixed-delta-low	500	500	-	0.0002	12.0	3
v4	slow-500-sfixed-delta-mid	500	500	-	0.002	12.0	3
v4	slow-500-sfixed-delta-high	500	500	-	0.01	12.0	3
v4	slow-500-dfixed-s-low	500	500	-	0.002	5.0	3
v4	slow-500-dfixed-s-mid	500	500	-	0.002	12.0	3
v4	slow-500-dfixed-s-high	500	500	-	0.002	30.0	3

Information sur les versions

- v1 Création des relations avec nombre de relations prédéfini.
- v2 Création des relations avec une loi de Bernoulli, densité et dispersion pour des *datasets* plus réalistes. (Non inclus dans le tableau car c'est une version de transition entre v1 et v3, elle a été réalisée pour tester la création de relations avec une loi de Bernoulli mais les résultats n'ont pas été conservés car ils étaient trop volumineux et difficiles à analyser).
- v3 Même version que v2, mais suppression des anciens résultats en cache devenus obsolètes, moins de méthodes *benchmarksées* (surtout celles qui font doublon), sans les *datasets* larges trop volumineux, et meilleure organisation des résultats pour faciliter lecture et analyse.
- v4 Créée pour les algorithmes de treillis de concepts plus coûteux en temps d'exécution, avec une sélection plus restreinte de méthodes *benchmarksées* et des *datasets* plus petits.

Information sur les groupes de *benchmark*

- Groupe 1: Groupe de méthodes de base, d'opérateurs sur les relations, d'intention et d'extension, qui sont relativement rapides à exécuter même sur de grands jeux de données, et qui permettent de tester les performances de la structure de données.
- Groupe 2: Suppression des doublons de méthodes du groupe 1, avec par exemple le test de suppresion d'un objet et celui de suppression d'un attribut rassemblés en un seul test de suppression d'un objet ou d'un attribut.
- Groupe 3: Groupe réalisé pour les algorithmes de treillis de concepts, qui sont plus coûteux en temps d'exécution, et qui nécessitent des jeux de données plus petits pour obtenir des résultats en un temps raisonnable.

Configuration des *benchmarks*

Tous les *benchmarks* ont été réalisés sur le *runner* **GALACTIC** grâce à la librairie `Criterion` avec une taille d'échantillon de 12, un temps de warm-up de 2 secondes et un temps de mesure de 15 secondes pour obtenir des résultats fiables tout en limitant le temps d'exécution.

Les différentes versions de *benchmark* ne sont pas comparables entre elles, mais les présenter permet de suivre l'évolution des performances de la librairie au fur et à mesure de son développement, et d'identifier les méthodes les plus coûteuses en temps d'exécution afin de pouvoir les optimiser.

4.1 *Benchmark* v1

La v1 du *benchmark* a été réalisée au début du projet, après l'implémentation des méthodes de base et des opérateurs sur les relations. Elle a permis d'identifier les méthodes les plus coûteuses en temps d'exécution. Elle a notamment permis d'optimiser les méthodes d'`intension` et d'`extension` grâce à l'intersection par lot min-k qui consiste à intersecter les ensembles en commençant par les k plus petits. Cette version a permis de mettre en place une base de *benchmark* pour les futures versions.

4.1.1 Méthodes de base et opérateurs sur les relations

Avec la v1 du *benchmark*, j'ai pu tester les performances des méthodes de base et des opérateurs sur les relations avec un *dataset* large, et toutes les méthodes testées étaient autour de 30 millisecondes d'exécution.

4.1.2 Évolution d'intent et d'extent

L'implémentation initiale des méthodes d'`intension` et d'`extension` était relativement simple, mais coûteuse en temps d'exécution. Suite à une discussion avec mon maître de stage, M. Demko, j'ai implémenté une optimisation pour ces méthodes en utilisant l'intersection par lot min-k, ce qui a permis de réduire considérablement le temps d'exécution sur de petits jeux de données.

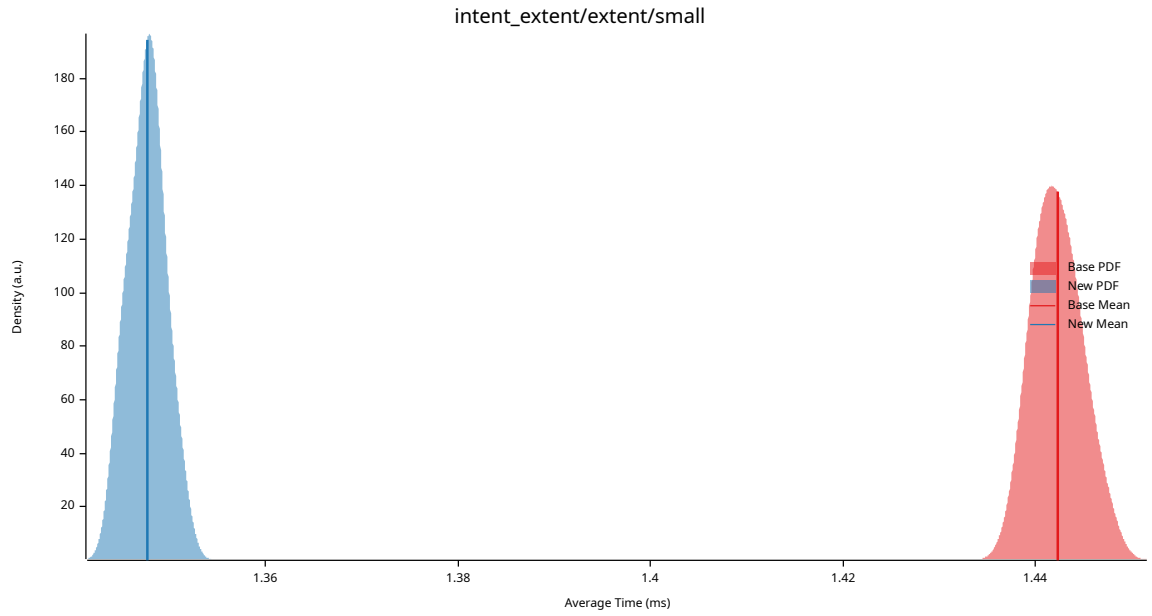


FIGURE 2 – Graphique de l'évolution de l'extension et de l'intention sur un petit *dataset* après optimisation

Ici, on peut constater une amélioration de 6.5%, ce qui paraît relativement faible, mais ces méthodes

sont utilisées dans les algorithmes de treillis de concepts, et une amélioration de 6.5% sur ces méthodes peut se traduire par un gain significatif sur les algorithmes.

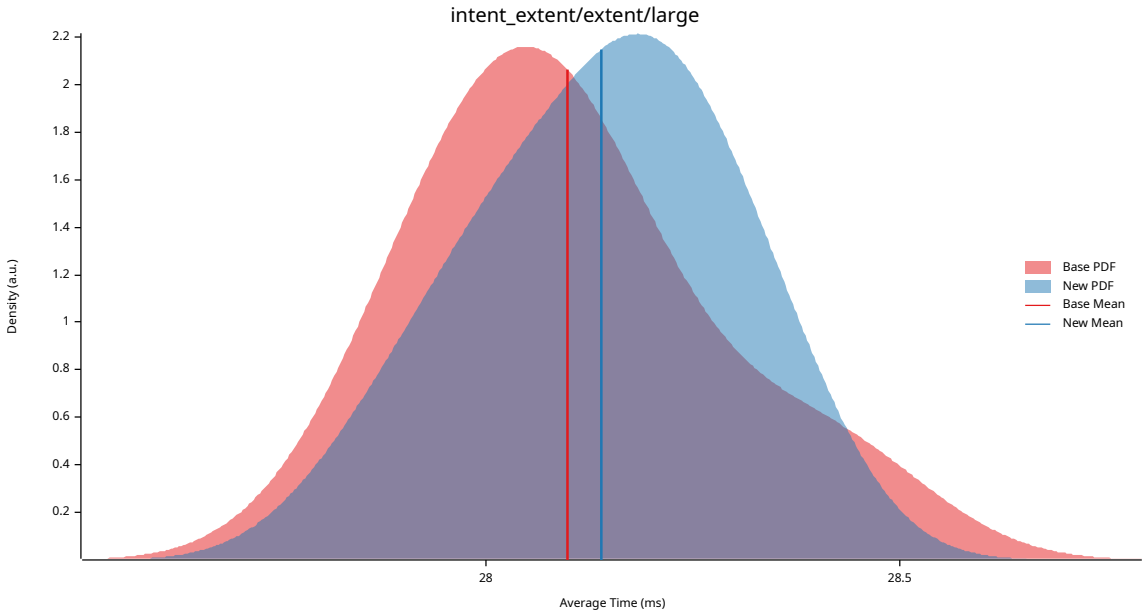


FIGURE 3 – Graphique de l'évolution de l'extension et de l'intention sur un grand *dataset* après optimisation

Cependant, pour les grands jeux de données, cette optimisation n'a pas permis d'améliorer les performances.

4.2 *Benchmark* v3

La v2 puis la v3 du *benchmark* ont été réalisées avec une nouvelle version des *datasets* plus réalistes générés avec une loi de Bernoulli, ce qui a permis de fixer soit la densité des relations (δ), soit la dispersion (σ). Cela a permis de générer des jeux de données plus réalistes et plus variés, et de comprendre les conditions dans lesquelles les différentes méthodes de la librairie sont les plus performantes.

4.2.1 Dispersion fixée, densité variable

Avec les *datasets* à dispersion fixée et densité variable, on observe une relation claire entre la densité du contexte et le temps d'exécution : multiplier δ par 50 (de 0.0002 à 0.01) multiplie le temps par un facteur de l'ordre de ×2.9 sur les petits contextes et ×3.6 sur les moyens, reflétant la croissance du nombre de relations à traiter. La taille du contexte amplifie cet effet : à densité égale, passer de small à medium (×4 sur chaque dimension, soit ×16 en nombre de relations potentielles) coûte ×6.5 à faible densité et ×8.2 à haute densité, signe d'une complexité super-linéaire. On note également que la qualité statistique des mesures se dégrade pour les cas medium-sfixed-delta-high : le R^2 s'effondre (0.024 pour extent, ~0.000007 pour predecessors_union_update) et la courbe d'itération ne présente plus de tendance

linéaire exploitable. Ce phénomène est lié au faible nombre d'échantillons configuré (sample_size = 12) combiné à une variance absolue plus élevée sur les grands contextes. Les estimations de moyenne et médiane restent fiables, mais la régression de Criterion n'a pas assez de points pour être significative.

4.2.2 Densité fixée, dispersion variable

Pour les *benchmarks* où la densité δ est fixée et la dispersion σ varie, l'effet sur le temps d'exécution est bien plus modeste : passer de s-low à s-high (soit $\sigma = 5$ à $\sigma = 30$) ne multiplie le temps que par ×1.21 sur les petits contextes (753 µs → 913 µs) et ×1.27 sur les moyens (4.45 ms → 5.63 ms). La distribution β est utilisée pour générer aléatoirement les relations entre objets et attributs, où le paramètre σ contrôle sa concentration autour de la densité cible δ : une valeur élevée de σ produit des colonnes homogènes (peu de variance), tandis qu'une valeur faible introduit plus d'hétérogénéité entre les colonnes. L'impact sur le temps d'exécution est donc limité : la densité moyenne restant identique, le nombre total de relations traitées est similaire, et seule leur répartition change. À la différence des *benchmarks* à σ fixe, la qualité statistique reste ici excellente sur tous les cas ($R^2 > 0.9997$), y compris pour les contextes medium, ce qui confirme que c'est bien la combinaison haute densité + grande taille qui dégrade les mesures, et non la taille seule.

4.3 *Benchmark* v4

La v4 du *benchmark* a été réalisée avec une sélection plus restreinte de méthodes *benchmarksées*, des *datasets* plus petits et une configuration adaptée pour les algorithmes de treillis de concepts plus coûteux en temps d'exécution. C'est aussi avec la v4 que j'ai *benchmarké* les algorithmes de la *crate* Odis (Dominik 2026) pour comparer les performances de ma librairie avec l'autre seule librairie de treillis de concepts en Rust.

4.3.1 Algorithmes de treillis de concepts

Pour ces *benchmarks*, les algorithmes de treillis de concepts sont encore à vérifier. J'ai d'abord implémenté `cbo` en suivant l'algorithme original de Kuznetsov (Kuznetsov 1993) trouvé sur internet, puis j'ai implémenté l'algorithme InClose4. Je me suis rendu compte que mes implémentations des deux étaient trop similaires, ce qui explique que leurs performances se rapprochent, tandis que InClose5 est implémenté différemment et se révèle plus performant. Je dois encore vérifier que mes implémentations sont correctes, mais voici les résultats obtenus pour le moment :

On peut encore constater que δ a un impact significatif sur les performances des algorithmes :

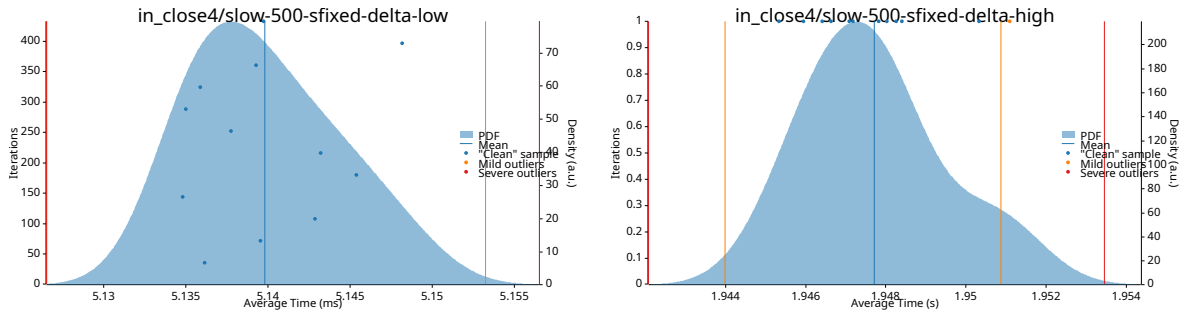


FIGURE 4 – Graphique de l'impact de δ sur les performances de INCLOSE4

Pour les algorithmes INCLOSE, l'effet de la dispersion σ sur le temps d'exécution est sensiblement plus prononcé que pour les opérations élémentaires : à densité fixée, passer de s-low à s-high multiplie le temps par ×1.66 (54.25 ms → 90.19 ms), contre ×1.21–1.27 pour les opérations de mise à jour. Ce contraste s'explique par la nature même de l'algorithme : là où `predecessors_union_update` opère localement sur une colonne dont la taille dépend principalement de δ, INCLOSE explore des intersections de colonnes pour construire des concepts formels. Une dispersion élevée concentre la distribution Beta autour de δ, produisant des colonnes plus homogènes entre elles, ce qui enrichit les intersections et augmente le nombre de concepts à explorer, gonflant ainsi la profondeur de récursion. L'algorithme INCLOSE est donc sensible non seulement au volume de relations, mais à leur structure, que σ modifie directement.

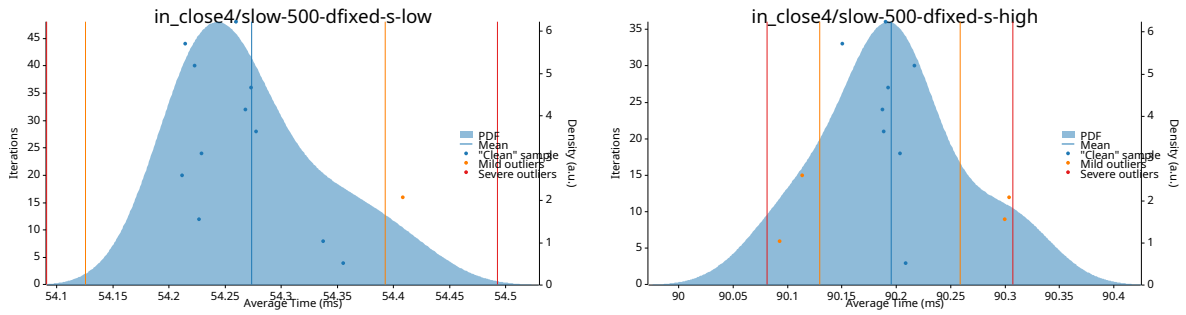


FIGURE 5 – Graphique de l'impact de σ sur les performances de INCLOSE4

L'algorithme INCLOSE5 est le moins coûteux en temps d'exécution ; c'est celui à privilégier :

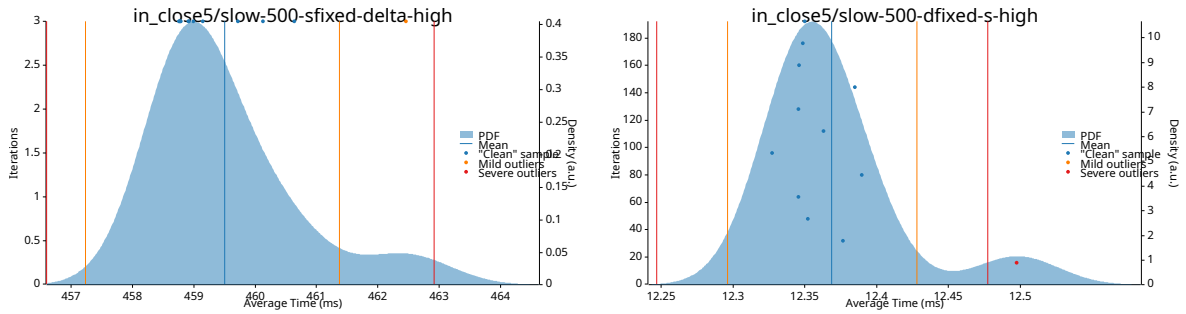


FIGURE 6 – Graphique des performances de INCLOSE5 avec une haute densité et une haute dispersion

4.3.2 Comparaison avec la *crate* Odis

Odis est la seule autre librairie permettant de faire du FCA en Rust à ce jour. Elle est implémentée par Dominik Dürrschnabel, professeur à la Baden-Wuerttemberg Cooperative State University Mosbach (DHBW Mosbach) en Allemagne, et disponible sur crates.io (Dominik 2026).

Voici les deux algorithmes de treillis de concepts les plus performants de la librairie Odis :

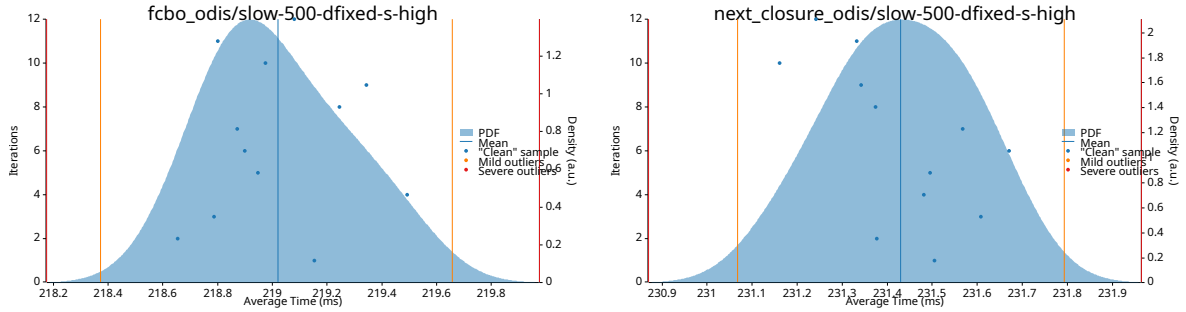


FIGURE 7 – Graphique des performances de la *crate* Odis - FASTCLOSEBYONE & NEXTCLOSURE

Et voici l'algorithme INCLOSE5 et NextClosure de ma librairie :

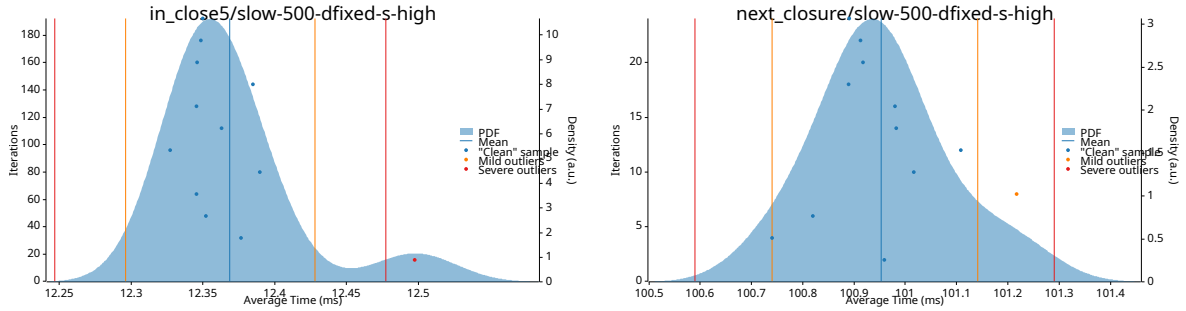


FIGURE 8 – Graphique des performances de INCLOSE5 & NEXTCLOSURE

On peut constater que ma librairie est, au minimum, deux fois plus performante que celle d'Odis pour NEXTCLOSURE, et qu'elle est environ 18 fois plus performante que FASTCLOSEBYONE d'Odis pour

InCLose5, à *dataset* et résultats identiques. Quand la librairie sera publiée sur crates.io, ce sera donc la librairie la plus performante pour faire du FCA en Rust.

J'ai réalisé la même comparaison pour les méthodes de base, mais c'est moins pertinent car nos structures de données sont implémentées différemment. Voici tout de même les résultats pour la méthode `remove_codomain` / `remove_attribute` :

Information sur la comparaison avec Odis

Je compare `remove_domain` avec `remove_attribute` car `remove_domain` et `remove_codomain` sont similaires dans ma librairie et que le *benchmark* est carré, donc cela n'a pas d'influence sur les résultats.

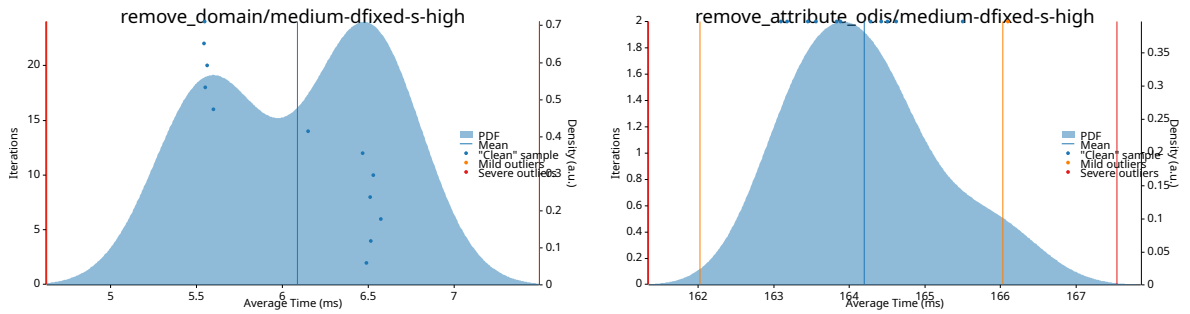


FIGURE 9 – Graphique comparant les performances de `remove_domain` de ma librairie et `remove_attribute` d'Odis

Pour la méthode `remove_domain` / `remove_attribute`, ma librairie est environ 25 fois plus performante que celle d'Odis, mais ces résultats sont à prendre avec précaution car nos structures de données sont implémentées de manière très différente.

5 Suite du projet et perspectives d'évolution

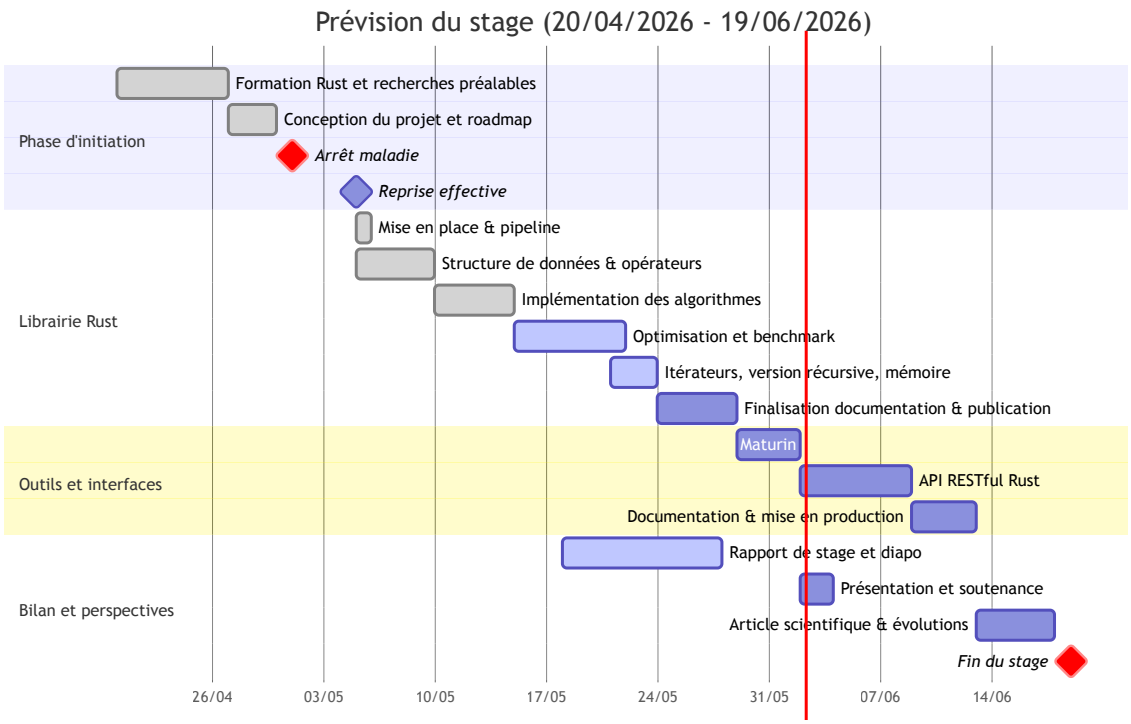
5.1 Résultats obtenus

À ce stade du projet, la librairie de FCA est presque prête pour être publiée sur crates.io. Elle est performante et robuste, avec une bonne couverture de tests et une documentation presque complète. Les derniers éléments en cours sont des tests de comparaison avec les méthodes récursives, la mise en place d'un *benchmark* mémoire, la création de l'itérateur pour la future *API*, une duplication des méthodes existantes en 64 bits avec *`roaring treemap`* et la finalisation de la documentation.

Deux de ces éléments sont pour le futur document scientifique, un est pour l'*API*, et les deux derniers sont nécessaires à la publication sur crates.io.

5.2 Perspectives d'évolution

Voici le diagramme de Gantt prévisionnel pour la suite du projet, avec les différentes étapes à venir :



Je n'ai pas suivi de planning prévisionnel rigoureux jusqu'à ce stade du projet. J'ai une approche bottom-up et, comme je me suis principalement concentré sur l'implémentation de la librairie Rust, je n'avais pas besoin de planifier les étapes suivantes. Pour la suite du projet, je vais suivre ce planning afin de respecter les délais et de mener à bien toutes les étapes.

Je précise également que je maintiendrai tout ce que j'aurai réalisé et publié pendant le stage.

6 Conclusion

Ce stage au sein du L3i a été une expérience très enrichissante : j'ai pu apprendre Rust, me familiariser avec les algorithmes de treillis de concepts, et réaliser ma première *pipeline* CI/CD. Les réunions sur l'avancement du projet **GALACTIC** et sur sa suite, ainsi que les discussions avec M. Demko sur les manières d'implémenter ou d'optimiser certaines méthodes, ont été très enrichissantes et m'ont permis de mieux comprendre les enjeux du projet et de trouver des solutions adaptées.

Même la rédaction de ce rapport m'apporte, car elle me permet de travailler avec Markdown et d'apprendre à utiliser des outils comme Mermaid et Pandoc.

C'est aussi un honneur d'avoir pu contribuer à **GALACTIC**. Je suis impatient et fier de voir ma contribution être utilisée. Il est exceptionnel de ressortir d'un stage avec une publication sur crates.io, un package Python, une *API RESTful* opérationnelle et un article scientifique en préparation, et je suis très reconnaissant d'avoir l'opportunité de réaliser tout cela pendant ce stage.

Ce stage m'a apporté des compétences en Rust, en analyse de données, en travail en équipe et en gestion de projet. Il m'apporte également un bagage présent chez peu de profils aujourd'hui, ce qui est une aubaine pour rejoindre le monde professionnel.

Références

Andrews, Simon. 2018. « A New Method for Inheriting Canonicity Test Failures in Close-by-One Type Algorithms ». In *Proceedings of CLA 2018*, édité par Dmitry I. Ignatov et Lhouari Nourine. Department of Computer Science, Palacký University Olomouc.

Demko, Christophe, Karell Bertet, Cyril Faucher, Jean-François Viaud, et Sergei O. Kuznetsov. 2020. « NextPriorityConcept: A New and Generic Algorithm Computing Concepts from Complex and Heterogeneous Data ». *Theoretical Computer Science* 845 (décembre): 1-20.

Demko, Christophe, Karell Bertet, Jean-François Viaud, Cyril Faucher, et Damien Mondou. 2024. « Description Lattices of Generalised Convex Hulls ». *International Journal of Approximate Reasoning*, août, 109269.

Dominik, Dürrschnabel. 2026. *odis: Formal Concept Analysis in Rust*. Released. https://crates.io/crates/odis.

« GAlois LAttices, Concept Theory, Implicational Systems and Closures ». 2026. The Galactic Organization. https://www.thegalactic.org/.

Kuznetsov, S. O. 1993. « A Fast Algorithm for Computing All Intersections of Objects ». *Automated Documentation and Mathematical Linguistics* 27 (5): 42-54.

« L3i – Laboratoire Informatique, Image et Interaction ». 2025. L3i – Laboratoire Informatique, Image et Interaction. https://l3i.univ-larochelle.fr/.