# Sujet de thèse:
# Cartographie des comportements d'algorithmes

## 1 Contexte

L'étude théorique des algorithmes est très souvent limitée à l'analyse de la complexité dans le pire des cas. Il existe pourtant de nombreuses notions de complexité qui apporte des informations essentielles à la bonne compréhension de l'efficacité des algorithmes.

- la complexité en moyenne consiste à supposer une distribution de probabilité sur les entrées (on prendra le plus souvent la distribution uniforme), et d'évaluer l'espérance du nombre de fois où une opération donnée sera effectuée.
- la complexité générique consiste à considérer la complexité dans le pire des cas sur un sous-ensemble d'entrées de l'algorithme et à démontrer qu'une entrée est presque sûrement dans cet ensemble.
- la complexité amortie est l'étude de la complexité de plusieurs itérations d'un algorithme sur une même entrée. Cette notion est très utile pour évaluer les itérateurs, fonctions permettant de parcourir un ensemble dans un ordre défini. L'idée est de démontrer que les cas où l'algorithme est coûteux sont compensés par les cas où l'algorithme a un coût très faible. La complexité globale de la séquence d'opération est alors inférieure à celle que l'on obtiendrait en multipliant le pire des cas par le nombre d'itérations.
- l'analyse lisse consiste à montrer que pour n'importe quelle instance, pour peu qu'on l'on applique une légère perturbation sur l'entrée, l'algorithme a de meilleures performances que dans le pire des cas. Ce type d'analyse est en revanche restreint aux algorithmes pour lesquels une perturbation sur l'entrée n'a pas ou peu d'influence sur le résultat de l'algorithme.

Parmi les critiques les plus récurrentes sur l'analyse théorique des algorithmes, on retrouve les suivantes:

- l'analyse dans le pire des cas n'est pas représentative de l'efficacité d'un algorithme, car bien souvent, ce pire des cas ne se produit que sur un ensemble d'entrée très restreint alors que l'algorithme semble avoir un bon comportement "en pratique".
- l'analyse en moyenne n'apporte pas d'information sur ce qui se passe "en pratique", car elle suppose souvent des distributions de probabilité qui correspondent pas "à la réalité".

La première de ces critiques a pour conséquence le développement des autres méthodes d'analyse précédemment mentionnés. Il est également possible de restreindre l'analyse sur des sous-ensembles d'entrées notables. Par exemple, au lieu de considérer un algorithme sur l'ensemble de graphes, on va démontrer que sa complexité dans le pire des cas est bien meilleure si l'on se restreint aux graphes planaires ou aux graphes séries-parallèles. La méthode est alors la suivante : on considère une sous-classe connues d'objets, la plus grande possible, et on tente de démontrer que l'algorithme s'y comporte bien mieux que dans le pire des cas.

La seconde critique est souvent infondée. En effet, l'étude de la complexité moyenne ou générique revient souvent à démontrer que les entrées les plus probables sont moins coûteuses que le pire des cas. La démonstration se fait en deux étapes :

- on identifie une ou plusieurs propriétés sur les entrées qui garantisse un comportement de l'algorithme,

- on montre que cette propriété se produit presque sûrement ou avec une probabilité suffisante, selon la distribution étudiée.

"En pratique", il s'agit donc de savoir si cette propriété qui garantit le comportement de l'algorithme est fréquemment présente ou non. Même si la distribution initiale est très éloignée de celle considérée dans l'étude théorique, il se peut que la mesure de la propriété ne varie pas ou peu.

Cette notion de "en pratique" est elle même mauvaise. Il s'agit plutôt de considérer différents contextes. La recherche de motifs dans une séquence d'ADN, une protéine, ou un texte écrit en langage naturel sont différents contextes, pour lesquels l'analyse en moyenne donne une estimation très précise de l'efficacité de l'algorithme naïf de recherche de motifs, pour ces 3 contextes pourtant bien différents de la distribution uniforme.

Si la seconde partie est dotée d'un ensemble d'outils de preuve (combinatoire énumérative, combinatoire analytique, méthodes probabilistes), la première partie se fait en général "à tâtons". Le chercheur doit manipuler l'algorithme, soit en déroulant des calculs à la main, soit en effectuant des expériences, afin d'obtenir une intuition sur les propriétés susceptibles d'avoir une influence sur l'algorithme, puis de tenter de prouver son intuition. Cette méthode, faites d'essais et d'erreurs, rend le travail du chercheur chronophage et explique en partie pourquoi peu d'algorithmes sont étudiés en moyenne.

L'utilisation de générateurs aléatoires est un premier outil qui facilite la tâche du chercheur. Ces générateurs permettent d'émettre directement des conjectures solides sur la complexité moyenne que l'on souhaite démontrer, mais également de faire des tests sur les propriétés moyenne des entrées. On retrouve par exemple cette méthode dans [1, 4, 6]. La recherche des propriétés pertinentes se fait en revanche toujours à tâtons. De plus, si la propriété importante est la présence ou l'absence d'un ou plusieurs motifs dans l'entrée, il n'existe actuellement aucun outil qui puisse extraire ces motifs automatiquement.

En résumé, l'analyse d'algorithme, permet actuellement d'identifier **un ou des** paramètres ayant une influence sur le comportement de l'algorithme. L'identification de ces paramètres est coûteuse en temps et n'est le plus souvent pas exhaustive. C'est à dire que l'on identifie pas **les** paramètres qui ont une influence. Or connaître l'ensemble des paramètres permet d'identifier au mieux l'ensemble des contextes dans lesquels l'algorithme va bien ou mal se comporter, de constituer une véritable **cartographie des comportements**.

## 2 Objectif

L'objectif de cette thèse est de faire de l'**analyse d'algorithmes selon plusieurs paramètres**. Notre but étant d'arriver à obtenir des **cartographies de comportement des algorithmes**, et d'identifier quel est l'algorithme optimal lorsque les entrées satisfont un ensemble donnée de propriétés. La Figure 3 donne un exemple d'un tel résultat en algorithmique du texte, excepté qu'il s'agit uniquement d'un résultat expérimental. Nous voudrions enrichir ce schéma en considérant un plus grand nombre de propriétés et surtout, obtenir des résultats théoriques qui valident ces expériences.

Dans ce but, nous allons dans un premier temps concevoir un outil permettant de faciliter l'analyse d'algorithmes.

Remarquons que cet outil peut également voir des vertus dans l'industrie. A l'heure du Big Data, le traitement massif des données est souvent fait via des outils comme Hadoop, un framework permettant la création d'applications massivement distribuées. Hadoop utilise des outils comme Map Reduce ou Spark pour répartir les calculs à effectuer sur plusieurs machines. S'il était possible d'identifier rapidement quel algorithme est le plus optimal pour traiter une entrée donnée, en calculant simplement quelques propriétés, la répartition pourrait alors être faite non seulement sur différentes machines, mais sur différents algorithmes, améliorant ainsi l'efficacité globale du système.

**Attention : Bien que ce sujet fasse référence à des outils de fouille de données ou de calcul distribué, il ne s'agit pas d'un sujet de recherche dans ces domaines. Le but est d'utiliser des outils préexistant afin de produire de nouveaux résultats en analyse d'algorithme.**

# 3 Projet

1. Développer l'outil d'extraction d'information sur le comportement des algorithmes et des programmes.

2. Enrichir l'outil afin de pouvoir gérer simultanément plusieurs algorithmes permettant de résoudre un même problème.

3. Utiliser cet outil sur des algorithmes dont l'analyse expérimentale et théorique est connue afin de vérifier la validité de l'outil et tenter d'apporter de nouveaux éléments lorsque c'est possible.

4. Utiliser l'outil pour conduire une analyse théorique d'algorithmes encore mal compris.

## 3.1 Construction de l'outil

La première étape du projet consiste donc à concevoir un outil capable d'analyser un algorithme ou un programme, quel que soit le langage de programmation dans lequel il a été écrit. Un point important à comprendre dans la conception de cet outil est qu'il sera architecturé en micro-services et que certains de ces services sont des programmes open-source préexistant.

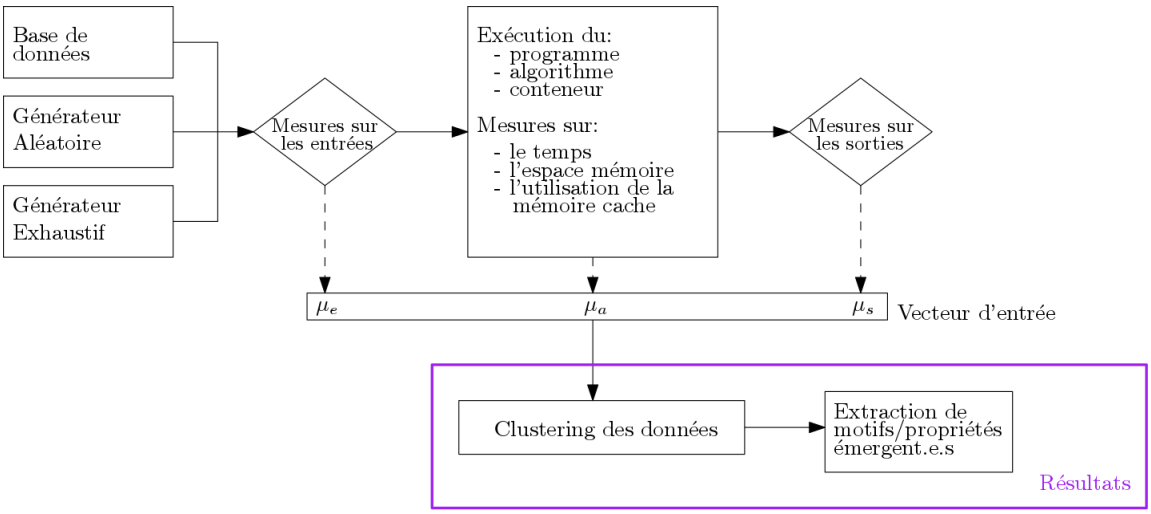


Figure 1: Conception d'un outil d'analyse automatique d'algorithmes

**Source de données** On distingue trois types de sources dans les entrées :

- les générateurs exhaustifs : prévu pour engendrer des objets de petites tailles, ces générateurs permettront de faire de l'analyse fine du comportement de l'algorithme et de détecter l'apparition de tout ses comportements pathologiques lorsque ceux-ci se produisent sur de petits objets. Cela peut être utile, notamment lors que l'utilisation d'algorithme de fouille de donnée, afin de réduire le bruit dans les résultats obtenus.

- les générateurs aléatoires : prévus principalement pour effectuer des tests sur des objets de grandes tailles, les résultats expérimentaux obtenus donneront directement une intuition sur le comportement moyen de l'algorithme pour une distribution de probabilité fixée. C'est un formidable outil pour orienter l'analyse théorique des algorithmes.

- les bases de données : nous entendons par là une collection d'entrée issues du monde réel, d'un ou plusieurs contextes particuliers, sur lesquels il est attendu que l'algorithme soit appliqué. D'un

point de vue industriel, c'est le choix le plus pertinent car il permet de faire coller les résultats de l'expérience à son cas d'utilisation. Du point de vue théorique, il permet au chercheur de comprendre ce qui peut entraîner une différence entre le comportement moyen de l'algorithme lorsqu'on l'étudie sous une distribution de probabilité fixée et un cas pratique.

Dans ce projet, nous traiteront ces trois types de sources afin d'obtenir la vision la plus complète des algorithmes que nous analyserons.

**Mesures sur les entrées** Ces mesures sont adaptées au type d'objet passé en entrée. Le but est d'extraire automatiquement un ensemble de données numériques sur une entrée qui pourraient être pertinentes pour l'analyse de l'algorithme. Certaines mesures pourront être appliquées à n'importe qu'elle type de source, comme la taille de la donnée, ou son entropie, d'autres pourront être propres à l'objet étudié. Pour les arbres on pourra considérer la largeur et la hauteur. Pour les graphes le diamètre, la treewidth, la pathwidth, la présence ou l'absence de certaines propriétés. Notons que ces mesures peuvent être faites de façon asynchrone avec le reste des calculs. Ils pourront aussi bien être fait a priori, simultanément ou a posteriori.

**Exécution du programme et mesures** L'objectif est ici de pouvoir analyser un algorithme ou un programme dans un maximum de contextes. On distingue 3 cas de figures.

- un algorithme : l'utilisateur ne fournit qu'une fonction dont il souhaite tester l'efficacité. Il s'agit du cas d'utilisation le moins générique car il nous obligera à faire des choix dans les langages de programmation utilisés. Nous fournirons alors un squelette dans différents langages qui permet d'appeler la fonction utilisateur. Cette méthode est en revanche celle qui aura notre préférence pour l'analyse d'algorithme car elle nous permettra d'avoir plus de finesse quant aux mesures produites.
- un programme : l'utilisateur fournit un programme, développé dans le langage de son choix. Notre outil servira d'interface pour fournir les entrées dont le programme a besoin, effectuer les mesures et récupérer les sorties.
- un conteneur : l'utilisateur fournit un conteneur capable de communiquer avec l'extérieur (via le réseau par exemple), les entrées-sorties seront envoyées et récupérée via ce canal de communication.

Dans tous ces cas de figures, nous tenterons d'extraire des informations sur l'exécution de l'algorithme (temps d'exécution, mémoire vive utilisée, utilisation de la mémoire cache) à l'aide d'outils préexistant (valgrind, papi [8], Qemu, ...).

**Mesures sur les sorties** Cette partie consiste à récupérer ce que le programme produit sur la sortie standard et la sortie d'erreur et à l'analyser. Qu'il s'agisse du résultat de l'algorithme, de la valeur de différents compteurs d'opérations effectuées, de messages d'erreur. Cette partie devra donc être adaptée pour chaque programme en fonction des sorties produites. De même que pour les mesures sur les entrées, on pourra vérifier des propriétés comme la taille de la sortie, etc...

**Clustering** L'ensemble des mesures récupérées lors d'une analyse forment un vecteur $(\mu_e, \mu_a, \mu_s)$. Nous effectuerons alors un clustering en utilisant des méthodes classiques ($k$-moyenne, DBSCAN, ...). Le but sera de regrouper les entrées en fonction des différents comportements qu'elles induisent sur l'algorithme, c'est à dire soit sur les mesures $\mu_a$, soit les mesures $\mu_s$, soit les deux.

Un réel avantage à effectuer ce sujet au LIPN est la présence de nombreux experts en clustering, vers lesquels nous pourrons nous tourner en cas de questions. Le sujet ne contient a priori pas de question de recherche en clustering, mais nous serions heureux de collaborer si des questions intéressantes du point du vue du clustering venait à se poser. Guénael Cabanes et Céline Rouveirol ont par exemple été approchés à ce sujet et sont ouverts à la discussion.

**Extraction de motifs émergeants** Les motifs émergeants sont des motifs dont le nombre d'occurrences varie radicalement d'un ensemble de données à un autre. Il s'agit donc d'une analyse qui, dans notre cas, ne peut s'effectuer qu'après avoir effectué une première phase de clustering. L'idée est ici de repérer des propriétés non quantitatives, mais structurelles, sur les entrées, qui induirait une modification du comportement de l'algorithme. Il s'agit d'un point essentiel pour aider les chercheurs à effectuer une analyse théorique. En effet, lorsque ces motifs existent, il peut-être difficile et chronophage de les identifier. Or, à notre connaissance, aucune étude théorique du comportement d'un algorithme n'a jamais tenté d'utiliser une méthode d'extraction automatique de motifs émergeants pour assister le travail des chercheurs.

De plus, s'il s'avère que certaines sous-classes d'entrées, possédant un motif particulier, induisent un mauvais comportement de l'algorithme, il est en revanche possible d'utiliser cette information pour développer un nouvel algorithme, moins générique, mais plus efficace pour traiter ce cas particulier.

### 3.2 Enrichir l'outil pour l'analyse simultanée de plusieurs algorithmes

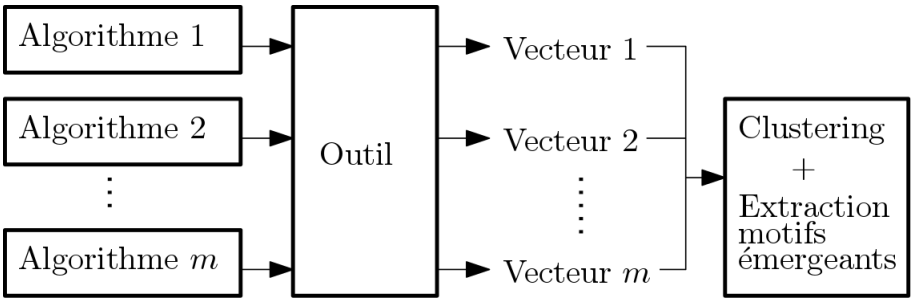


Figure 2: Utilisation de l'outil sur plusieurs algorithmes

Un autre type d'étude d'intérêt majeur est d'établir des clusters d'entrées pour lesquelles un algorithme spécifique est le plus efficace et d'identifier les propriétés et les motifs sur les entrées qui permettent de déterminer l'algorithme qui sera le plus efficace pour effectuer le calcul.

Une application industrielle à cet outil est la suivante : déterminer un ensemble de paramètres simples sur les entrées pouvant permettre de configurer des outils comme Map Reduce ou Spark pour distribuer les calculs, non seulement sur différentes machines, mais également vers différents algorithmes en choisissant celui qui s'exécutera le plus efficacement possible sur l'entrée.

Il existe néanmoins un bémol à cette méthode : le coût du test permettant de déterminer l'algorithme optimal doit être inférieur au gain de temps obtenu par le calcul effectué par l'algorithme.

### 3.3 Vérification de la validité de l'outil sur des résultats connus

**Algorithmes de recherche de séquence** Dans [5], les auteurs effectue une étude expérimentale des 84 algorithmes de recherches des occurrences d'un mot dans un texte. La complexité dans le pire des cas et en moyenne de ces algorithmes est bien connue. Ils appliquent les algorithmes sur des séquences obtenus par des générateurs aléatoires uniformes, des séquences d'ADN, ainsi que des textes en langages naturels. Ils obtiennent une carte des algorithmes les plus performant en fonction de la taille du texte et de la taille de l'alphabet considéré (voir Figure 3). C'est donc une méthode fortement similaire à celle que nous proposons.

Les auteurs proposent également un outil, SMART[1], dans lequel tous les codes sources de tous les algorithmes sont mis à disposition.

Nous proposons donc de brancher notre outil à SMART et de vérifier si nous ré-obtenons bien les mêmes résultats, mais nous tenterons également de déceler automatiquement d'autres paramètres qui permettent de déterminer pourquoi un algorithme sera plus ou moins efficace qu'un autre dans un contexte donné.

[1] https://smart-tool.github.io/smart/

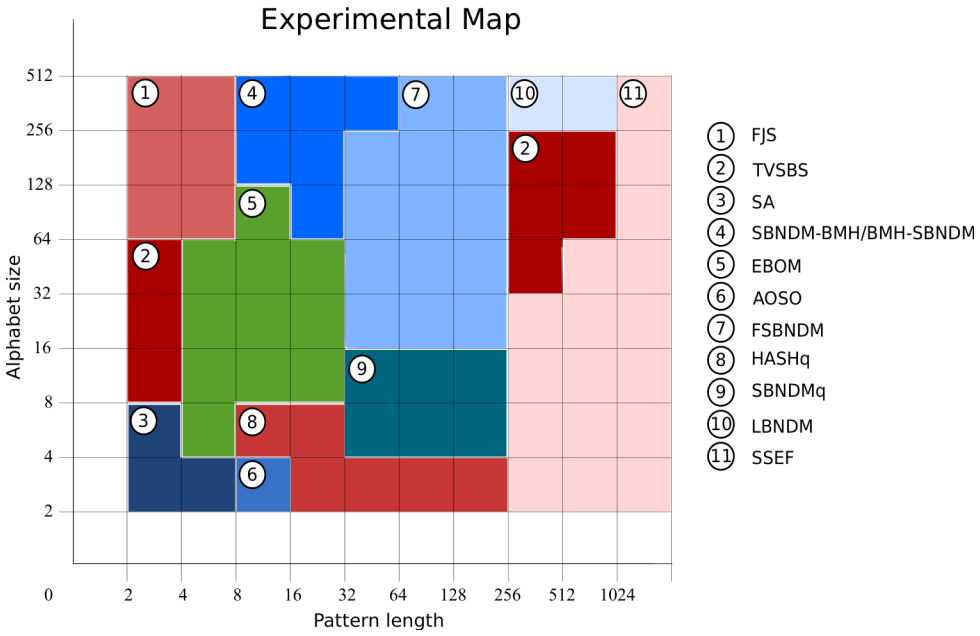


Figure 3: Carte des algorithmes optimaux de recherche de mot dans un texte, en fonction des paramètres d'entrée : la longueur du texte et la taille de l'alphabet.

**Algorithmes de tri** Il existe plusieurs sites faisant le recensement des différents algorithmes de tris : 38 sont par exemple listés sur `https://www.geeksforgeeks.org/sorting-algorithms/`, avec le code source disponibles dans différents langages. Il est toutefois à noter que certains de ces algorithmes ne s'appliquent que sur des sous-classes d'entrées, comme le tri par bac.

Nous souhaiterions effectuer une analyse similaire à celle effectuée dans [5], sur les algorithmes de tris. Nous voudrions de plus y ajouter une analyse massive des différentes propriétés connues sur les permutations passées en entrée et tenter d'effectuer le clustering sur ces différentes propriétés. Ainsi, le logiciel Sage-Combinat propose une liste de méthode permettant de calculer des propriétés sur les permutations (nombre d'inversions, de descentes, de points fixes, ...).

En combinant les deux, nous pourrions valider l'outil en vérifiant les paramètres connus pour être des indicateurs de l'efficacité des méthodes, mais aussi dresser une carte complète des cas de figures dans lesquels un algorithme sera plus efficace que les autres.

### 3.4 Analyse théorique des algorithmes

La direction que prendra la thèse à cette étape sera assez dépendante des résultats obtenus précédemment.

Un espoir serait notamment de pouvoir refaire l'analyse de certains algorithmes en ne considérant pas les compteurs d'opérateurs habituels (nombres de comparaisons, nombres d'échanges) mais en descendant à un niveau plus bas sur le système, en incluant le nombre d'erreurs de lecture de cache.

L'une des forces que nous avons pour mener à bien ce projet est la bonne connaissance des méthodes de génération aléatoire et exhaustive qui existent pour les sujets d'études que nous envisageons. Des nouveaux résultats [2] sur des générateurs aléatoires multiparamétriques nous donnerons un atout considérable lors des études expérimentales.

Nous envisageons ensuite trois familles d'algorithmes à étudier :

1. **algorithmes du texte** : nous aimerions reprendre les résultats de [5] et tenter d'y apporter une vision plus précise, permettant de comprendre pourquoi un algorithme devient meilleur que ses concurrents pour un ensemble de paramètres donnés. Cela impliquera notamment de ne pas se focaliser

sur une unique opération, supposée être le meilleure représentant de l'efficacité des algorithmes, mais d'en considérer plusieurs simultanément, ou de considérer leur somme, afin d'avoir une vision plus fine. Notons qu'une partie du travail aura déjà été effectué à l'étape précédente. Nous nous concentrerons à ce stade sur l'analyse théorique.

2. **multiplications de matrices** : il s'agit d'un problème fondamental en informatique, pour lesquels l'analyse dans le pire des cas est déjà non-triviale. Pour l'algorithme de Strassen, la meilleure borne supérieure connue à ce jour sur la complexité dans le pire des cas est $\mathcal{O}(n^{2,807})$ tandis que celle de l'algorithme de Coppersmith–Winograd [3] est de $\mathcal{O}(n^{2.3728638})$. L'algorithme de Strassen était connu pour être surtout efficace sur des matrices de grandes tailles mais de récents travaux [7] montrent qu'il est possible de développer une implantation efficace également pour des matrices de petites tailles. Nous aimerions analyser ces algorithmes en considérant différents paramètres (densité, ...) et constituer une cartographie sur ce problème, permettant d'identifier l'algorithme optimal en fonction des paramètres.

3. **plus court chemin dans un graphe** : nous voudrions utiliser l'outil que nous avons conçus pour nous indiquer un ensemble de paramètres intéressants pour l'étude et la comparaison des algorithmes de calcul du plus court chemin dans un graphe. De nombreux outils de manipulation de graphes existent, dont voici deux exemples. L'outil graph-tool[2] en Python est une boîte à outil qui nous permettra d'extraire rapidement un ensemble de mesures sur les entrées, L'outil gSpan [9] nous permettra d'extraire des motifs fréquents dans des graphes. Nous pourrons l'adapter pour l'extraction de motifs émergeants. Nous accorderons une attention particulière à l'implantation des graphes et des algorithmes. En particulier l'algorithme de Dijkstra a une complexité dans le pire des cas qui varie selon que l'on utilise une liste, un tas binaire, ou un tas de Fibonacci, ce dernier étant la meilleure structure. Mais le comportement de l'algorithme au niveau de la mémoire cache risque est meilleur avec le tas binaire qu'avec le tas de Fibonacci. Nous souhaiterions déterminer les paramètres pour lesquels l'implantation optimale varie.

## 4 Feuille de route

La feuille de route ci-dessous (voir Figure 4) est bien sûr une ébauche, amenée à évoluer au fil du temps.

## References


[1] Frédérique Bassino, Julien David, and Cyril Nicaud. On the average complexity of moore's state minimization algorithm. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 123–134. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[2] Maciej Bendkowski, Olivier Bodini, and Sergey Dovgal. Polynomial tuning of multiparametric combinatorial samplers. In Markus E. Nebel and Stephan G. Wagner, editors, *Proceedings of the Fifteenth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2018, New Orleans, LA, USA, January 8-9, 2018*, pages 92–106. SIAM, 2018.

[3] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251 – 280, 1990. Computational algebraic complexity editorial.

[4] Julien David, Loïck Lhote, Arnaud Mary, and François Rioult. An average study of hypergraphs and their minimal transversals. *Theor. Comput. Sci.*, 596:124–141, 2015.


[2] https://graph-tool.skewed.de

| Code | Tâche | Durée |
|---|---|---|
| T0 | Conception de l'outil | 9 mois |
| T1 | État de l'art | 1 an |
| T2 | Vérification de la validité de l'outil sur des résultats connus | 6 mois |
| T3 | Analyse d'algorithmes | 2 ans |
| T3a | Algorithmes du texte | 6 mois |
| T3b | Multiplications de matrices | 9 mois |
| T3c | Plus court chemin | 9 mois |
| T4 | Rédaction de la thèse | 6 mois |

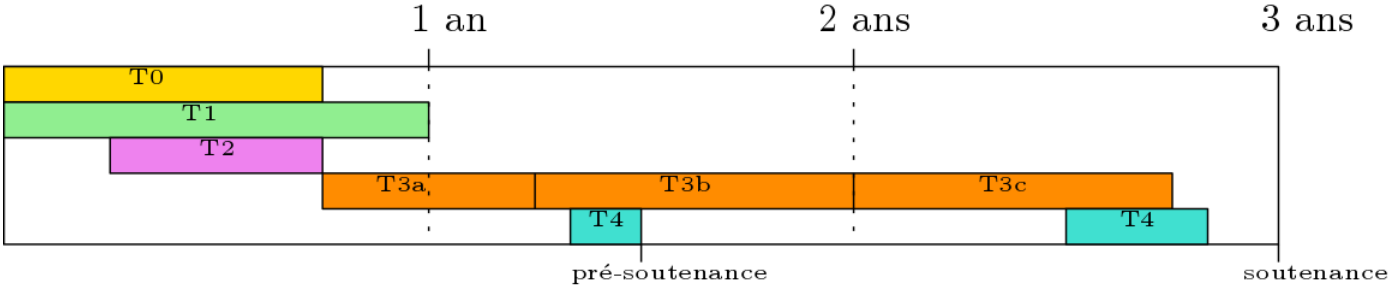


Figure 4: Feuille de route pour le sujet de thèse.


[5] Simone Faro and Thierry Lecroq. The exact string matching problem: a comprehensive experimental evaluation. *CoRR*, abs/1012.2547, 2010.

[6] Sven De Felice and Cyril Nicaud. Brzozowski algorithm is generically super-polynomial for deterministic automata. In Marie-Pierre Béal and Olivier Carton, editors, *Developments in Language Theory - 17th International Conference, DLT 2013, Marne-la-Vallée, France, June 18-21, 2013. Proceedings*, volume 7907 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2013.

[7] Jianyu Huang, Tyler M. Smith, Greg M. Henry, and Robert A. van de Geijn. Strassen's algorithm reloaded. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16. IEEE Press, 2016.

[8] Frank Winkler. Redesigning papi's high-level api. Technical Report ICL-UT-20-03, University of Tennessee, 2020-02 2020.

[9] Xifeng Yan and Jiawei Han. gspan: graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724, 2002.