Rapport du projet LU2IN013

Automatisation de la cryptanalyse des cryptosystèmes classiques à l'aide d'algorithme modernes

Encadrante : Mme Valérie Ménissier-Morain

Helder Brito (21304177) et O'nel Hounnouvi (21315612)

Février 2025 – Juin 2025

Table des matières

1 Introduction

La cryptographie constitue depuis longtemps un fondement essentiel dans la protection des communications sensibles. Les cryptosystèmes dits classiques, tels que les chiffrements par substitution monoalphabétique, par transposition, ou encore les méthodes de Vigenère et de Playfair, ont historiquement joué un rôle central dans la préservation de la confidentialité, aussi bien dans les sphères civiles que militaires. Leur vulnérabilité résidait toutefois dans le fait que le déchiffrement reposait sur des procédés manuels, dont l'efficacité variait selon le contexte historique et les connaissances disponibles.

La cryptanalyse, discipline complémentaire, vise précisément à étudier et à mettre à l'épreuve ces mécanismes de chiffrement, dans le but d'en évaluer la solidité face à des tentatives d'attaque. L'émergence de l'informatique et l'essor des capacités de calcul ont profondément renouvelé les approches dans ce domaine : les méthodes empiriques d'autrefois peuvent désormais être automatisées à l'aide d'algorithmes d'optimisation heuristique. Des techniques telles que le hill climbing, le recuit simulé ou la recherche tabou permettent ainsi d'explorer de manière efficace l'espace des clés possibles, en s'appuyant sur des propriétés statistiques de la langue pour guider les attaques.

Ce projet a pour but de développer des techniques de cryptanalyse automatisée appliquées aux chiffrements classiques, en particulier le chiffrement par substitution monoalphabétique. Il poursuit un double objectif : d'une part, mettre en œuvre différentes méthodes heuristiques d'attaque ; d'autre part, analyser et comparer leurs performances afin d'évaluer leur efficacité.

2 Substitution monoalphabétique et cryptanalyse

2.1 Définition

La **substitution monoalphabétique** est l'une des plus anciennes méthodes de chiffrement. Elle consiste à remplacer, dans le message clair, chaque lettre de l'alphabet par une autre selon une permutation fixe. Voici un exemple :

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

Par exemple, le message *SUBSTITUTION* devient *PRYPQFQRQFLK*.

L'alphabet latin comportant 26 lettres, on peut définir $26! \approx 4 \times 10^{26}$ permutations possibles, soit environ 2^{88}. À titre de comparaison, environ 2^{58} secondes se sont écoulées depuis le début de l'univers, ce qui rend une attaque par force brute totalement irréaliste.

Cependant, cette impression de sécurité est **trompeuse**, car la substitution monoalphabétique conserve en grande partie les **caractéristiques statistiques** de la langue d'origine (fréquences de lettres, combinaisons courantes, etc.).

2.2 Cryptanalyse

2.2.1 Quelques définitions

- Le **déchiffrage** consiste à retrouver le texte clair à partir du cryptogramme, en utilisant la clé.
- La **cryptanalyse** désigne l'ensemble des techniques visant à casser un message chiffré *sans* connaître la clé.
- Le **cryptogramme** est le message chiffré, que l'on cherche à déchiffrer.
- Un **n-gramme** est une séquence de n lettres consécutives dans un texte. Par exemple, dans le mot *CRYPTANALYSE*, les bigrammes ($n = 2$) sont : CR, RY, YP, etc., et les trigrammes ($n = 3$) sont : CRY, RYP, YPT, etc.

2.2.2 Attaque par analyse fréquentielle

La substitution monoalphabétique présente une **faiblesse structurelle majeure** : chaque lettre du texte clair est systématiquement remplacée par la même lettre chiffrée. Ainsi, la structure statistique de la langue (fréquences des lettres et des séquences de lettres) est **partiellement préservée** dans le cryptogramme.

Cette propriété permet de mettre en œuvre une **attaque par analyse fréquentielle**, qui repose sur la comparaison des fréquences d'apparition des n-grammes dans le message chiffré avec celles issues d'un corpus de référence en français.

En s'appuyant sur un dictionnaire de n-grammes (bigrammes, trigrammes, etc.) issu d'un grand ensemble de textes en français, il est possible d'estimer la plausibilité linguistique d'un texte. Cette estimation permet de guider la cryptanalyse vers des textes proches du message original.

2.2.3 Fonction de score (fitness function)

Afin d'évaluer la qualité des solutions proposées (c'est-à-dire des hypothèses de clé), on utilise une **fonction de score**, ou *fitness function*. Celle-ci doit remplir deux critères essentiels :

— **Discriminante** : elle doit bien différencier un texte encore très chiffré (score mauvais) d'un texte proche du clair (score bon).

— **Efficace** : elle doit être rapide à calculer, car elle sera invoquée un très grand nombre de fois.

Dans ce projet, la fonction utilisée est fondée sur la **log-vraisemblance** des n-grammes du texte déchiffré. Elle est définie comme suit :

$$\text{score} = -\sum \log\left(\text{fréquence}(c_1 \dots c_n)\right)$$

où $(c_1 \dots c_n)$ désigne un n-gramme du texte analysé.
L'objectif est de **minimiser ce score** : plus le texte est linguistiquement probable en français, plus les n-grammes qu'il contient sont fréquents, et plus le score est faible.

3 Premiers essais : Hill Climbing

3.1 Principe de l'algorithme Hill Climbing

L'idée générale est la suivante :

1. **Initialisation :**
   (a) Partir d'une clé aléatoire $C1$ et l'utiliser pour déchiffrer le cryptogramme
   (b) Calculer le score du texte obtenu
2. **Boucle principale :**
   (a) Générer une solution voisine $C2$ en faisant une légère modification et calculer le nouveau score
   (b) Si ce score est meilleur que le score précédent, adopter cette nouvelle clef comme clef courante : $C1 \leftarrow C2$
   Sinon, conserver l'ancienne clé $C1$.
3. **Critère d'arrêt :** Terminer l'algorithme après un nombre prédéfini d'itérations, ou lorsqu'on a atteint un nombre prédéfini d'itérations sans amélioration du score (stagnation).

Il arrive fréquemment que l'algorithme se retrouve bloqué dans un minimum local : il n'arrive plus à améliorer la solution actuelle, bien que meilleure solution globale n'ait pas encore été trouvée. C'est pour éviter qu'il reste longtemps dans cette impasse que nous introduisons une condition d'arrêt basée sur la variable max-stagnation.

Modification de la clef

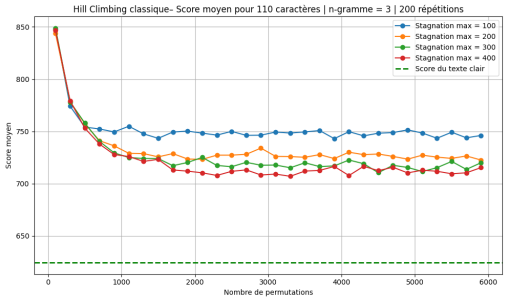
Il nous faut maintenant définir comment générer une clé voisine à la clé courante (étape 2-(a) de l'algorithme). Pour cela, on effectue une permutation aléatoire de deux lettres dans la clef. En guise d'exemple :

`QWERTZUIOPASDFGHJKLXCVBNM` → `QGERTZUIOPASDFWHJKLXCVBNM`

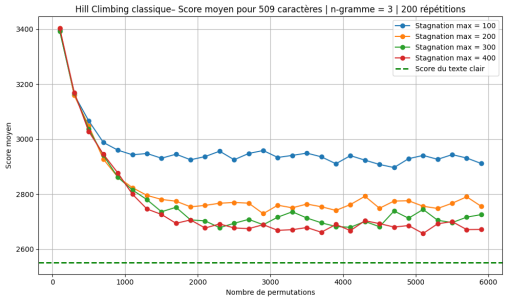
3.2 Résultats initiaux et observations

3.2.1 Influence de la taille du texte et du nombre maximum de stagnations

Nous commençons par étudier l'effet de la longueur du texte chiffré sur les performances de l'algorithme de *hill climbing*. La figure 1 compare l'évolution du score moyen pour deux tailles de texte : 110 et 509 caractères, en utilisant des trigrammes ($n = 3$). Chaque courbe est issue d'une moyenne sur 200 essais indépendants, pour différentes valeurs du paramètre de stagnation.



(a) Texte court : 110 caractères



(b) Texte plus long : 509 caractères

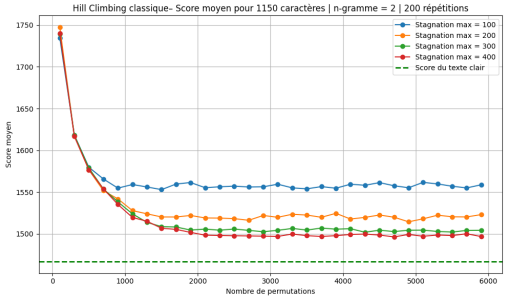
FIGURE 1 – Évolution du score moyen avec $n = 3$ selon la taille du texte.

On observe que les scores moyens sont significativement meilleurs et plus stables avec un texte de 509 caractères, car le score moyen se trouve plus proche du score du texte clair. Les statistiques de trigrammes sont en effet plus fiables avec un plus grand corpus, ce qui rend la fonction de score plus informative. En revanche, pour les textes courts (110 caractères), la rareté des trigrammes entraîne un bruit important et nuit à la stabilité des résultats. Cela montre que la taille du texte joue un rôle déterminant dans la qualité de la cryptanalyse fondée sur les n-grammes.

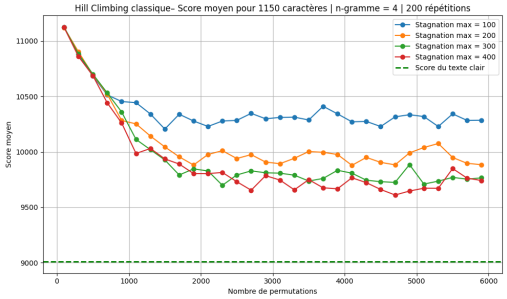
Enfin, le paramètre de stagnation, qui fixe le nombre maximal d'itérations sans amélioration avant arrêt de l'algorithme, joue également un rôle clé. Si cette valeur est trop faible, l'algorithme risque de s'arrêter prématurément avant d'avoir atteint un minimum local de qualité. À l'inverse, une valeur trop élevée entraîne un temps d'exécution plus long sans nécessairement améliorer les performances finales. Il est donc essentiel de trouver un compromis entre efficacité temporelle et qualité des résultats. Nos courbes montrent généralement une amélioration du score moyen jusqu'à un certain seuil de stagnation, au-delà duquel les gains deviennent marginaux.

3.2.2 Influence du choix de n dans les n-grammes

Nous analysons à présent l'impact de la taille des n-grammes sur les performances, en nous fixant un texte court d'environ 110 caractères. La figure 2 compare les scores moyens obtenus avec des bigrammes ($n = 2$) et des quadgrammes ($n = 4$), toujours sur une moyenne de 200 essais.



(a) $n = 2$ (bigrammes)



(b) $n = 4$ (quadgrammes)

FIGURE 2 – Comparaison du score moyen pour $n = 2$ et $n = 4$ (texte de 1150 caractères).

Les résultats mettent en évidence un contraste marqué :

— Les bigrammes ($n = 2$) fournissent de meilleurs scores et relativement stables.

— Les quadgrammes ($n = 4$) conduisent à des performances très variables et souvent médiocres. La rareté des séquences de 4 lettres dans un petit corpus les rend peu informatives, ce qui provoque une optimisation instable et chaotique.

Ces observations suggèrent que, pour des textes courts, il est préférable d'utiliser des n-grammes de petite taille. À l'inverse, des n-grammes plus longs nécessitent un corpus nettement plus important pour être exploitables. Comme l'illustre la section précédente, les trigrammes ($n = 3$) constituent un bon compromis à partir d'environ 400 à 500 caractères.

3.3 Limites identifiées

Malgré sa simplicité et sa rapidité, l'algorithme de *Hill Climbing* présente plusieurs limitations qui réduisent son efficacité sur certains cryptogrammes :

— **Blocage dans des minima locaux**
L'algorithme n'accepte que les modifications qui améliorent le score. Ainsi, s'il atteint une solution pour laquelle aucune permutation simple n'apporte d'amélioration, il se retrouve bloqué, même si une meilleure solution globale existe ailleurs dans l'espace des clés.

— **Dépendance à l'initialisation**
Le point de départ (clé aléatoire) a un impact fort sur la solution finale. Une exécution peut aboutir à une solution lisible, tandis qu'une autre, à partir d'une autre clé de départ, peut rester dans un état très chiffré.

— **Résultats instables sur les textes courts**
Lorsque le texte à déchiffrer est court, l'analyse fréquentielle devient moins fiable, et le Hill Climbing tend à converger vers des textes partiellement déchiffrés mais peu compréhensibles.

— **Absence de mémoire ou de stratégie d'évitement**
Hill Climbing ne conserve aucune trace des solutions précédemment explorées. Il peut donc revisiter inutilement les mêmes configurations et ne dispose d'aucun mécanisme pour forcer la sortie de zones stagnantes.

3.4 Optimisation de Hill Climbing

3.4.1 Principe de la version optimisé

Comme son nom l'indique, cette méthode est une version améliorée du Hill Climbing classique. Elle apporte une solution au blocage dans un minimum local. Au lieu d'abandonner après une stagnation excessive, elle effectue une réinitialisation de la recherche. Une nouvelle clef aléatoire est générée, et l'algorithme recommence depuis l'étape 1. Cette stratégie permet d'explorer plusieurs régions de l'espace des solutions, augmentant ainsi significativement les chances d'atteindre le minimum global — autrement dit, la solution correcte.

Notons qu'on enregistre la meilleure solution trouvée jusqu'à présent, afin de ne pas perdre les progrès réalisés.

4 Recherche d'alternatives plus robustes

4.1 Motivation

Bien que notre version optimisée du *Hill Climbing* ait permis d'améliorer significativement les résultats initiaux, certaines limites structurelles persistent. En particulier, la stratégie reste majoritairement basée sur l'exploitation locale, ce qui peut la rendre sensible aux pièges du paysage de solution, notamment dans les cas de cryptogrammes courts ou bruités.

Afin d'évaluer des approches potentiellement plus robustes, nous avons décidé d'explorer d'autres métaheuristiques capables de mieux équilibrer l'exploration globale et l'exploitation locale. Le *recuit simulé* et la *recherche tabou* offrent des mécanismes spécifiques qui leur permettent d'éviter plus efficacement les minima locaux : acceptation temporaire de solutions moins bonnes dans le premier cas, et mémoire des états précédemment visités dans le second. Ces caractéristiques en font des candidats intéressants pour améliorer la résilience de l'algorithme, diversifier les zones explorées et, à terme, renforcer la qualité de la cryptanalyse automatique.

4.2 Recuit simulé

Le recuit simulé est une méthode d'optimisation inspirée du processus de recuit en métallurgie, où un matériau est chauffé puis refroidi lentement pour atteindre un état de faible énergie. Notre algorithme est présenté de la façon suivante :

1. **Initialisation :**
   (a) Partir d'une clé aléatoire $C1$ et d'une température initiale T_0
   (b) Utiliser la clé pour dechiffrer le cryptogramme
   (c) Calculer le score du texte obtenu
2. **Boucle principale :**
   (a) Générer une solution voisine $C2$ en faisant une légère modification (voir 3.1)
   (b) Calculer le changement de coût, défini par

$$\Delta = score(C2) - score(C1).$$

   (c) Si $\Delta \leq 0$, accepter $C2$ (la solution s'améliore ou reste équivalente)

(d) Sinon, accepter $C2$ avec une probabilité donnée par

$$P_{\text{accept}} = \exp\left(-\frac{\Delta}{T}\right).$$

(e) Mettre à jour la température avec le coefficient de refroidissement α après un nombre d'itérations :

$$T \leftarrow \alpha T, \quad \text{avec } 0 < \alpha < 1.$$

3. **Critère d'arrêt :** Terminer l'algorithme après un nombre prédéfini d'itérations, puis retourner la solution finale s.

Le recuit simulé échappe aux minima locaux en introduisant une étape d'acceptation probabiliste des solutions moins bonnes. Concrètement, au lieu d'accepter uniquement les modifications qui améliorent le score, l'algorithme accepte une solution voisine avec la probabilité P_{accept}. Au début, la température T est élevée, ce qui rend l'expression $\exp\left(-\frac{\Delta}{T}\right)$ relativement grande. Cela permet donc à l'algorithme d'accepter des solutions moins bonnes et d'explorer plus librement l'espace de recherche, en sautant potentiellement hors d'un minimum local.

Au fur et à mesure que l'algorithme progresse, la température est progressivement abaissée (refroidissement), ce qui diminue la probabilité d'accepter des solutions moins performantes. Ainsi, en phase finale, le recuit simulé affine la solution dans un voisinage qui se rapproche d'un minimum global.

4.3 Recherche tabou

4.3.1 Principe de la recherche tabou

La recherche tabou utilise une mémoire pour éviter de revisiter des solutions déjà explorées. L'idée générale est la suivante :

1. **Initialisation :**
   (a) Partir d'une clé aléatoire $C1$ et l'utiliser pour déchiffrer le cryptogramme.
   (b) Calculer le score du texte obtenu.
   (c) Initialiser une liste tabou vide, qui servira à mémoriser les solutions récemment explorées.
2. **Boucle principale :**
   (a) Explorer un echantillon de clés voisines $C2$ en faisant de légères modifications (voir 3.1).
   (b) Vérifier si $C2$ est dans la liste tabou. Si oui, rejeter la solution
   (c) Sinon, calculer le score de $C2$. Si ce score est meilleur que le score précédent, adopter cette nouvelle clé comme clé courante : $C1 \leftarrow C2$.
   (d) Ajouter $C2$ à la liste tabou et, si nécessaire, retirer les éléments les plus anciens pour maintenir la taille maximale de la liste.
3. **Critère d'arrêt :** Terminer l'algorithme après un nombre prédéfini d'itérations.

En interdisant temporairement certaines solutions, la recherche tabou favorise l'exploration de nouvelles régions de l'espace de recherche et donc d'échapper aux minima locaux.

5 Résultats expérimentaux

Pour évaluer l'efficacité des différentes métaheuristiques, nous avons réalisé trois jeux de tests pour chaque algorithme. Dans chacun de ces tests, nous avons utilisé des textes chiffrés de longueurs différentes : 110, 509 et 1150 caractères.

L'objectif était d'observer l'évolution du score en fonction des différents paramètres clés. Cela permet d'analyser la rapidité de convergence, la stabilité, ainsi que la capacité de chaque méthode à s'approcher du minimum global.

Les résultats obtenus sont présentés en annexe sous forme de graphiques pour une comparaison visuelle claire entre les méthodes.

5.1 Hill Climbing

— Figure **??** : Hill Climbing sur un texte de 110 caractères,

Impact de la longueur du texte : Plus le texte est long, plus les scores tendent à baisser en moyenne, notamment pour les n-grammes de taille 3 et 4. Cela s'explique par le fait qu'un texte plus long fournit plus de contexte statistique, facilitant ainsi la détection de motifs valides en français.

Limites du hill climbing : L'algorithme ne parvient pas toujours à améliorer le score significativement, même en augmentant le nombre de permutations. Cela illustre bien la faiblesse principale de la méthode : son incapacité à sortir facilement d'un minimum local, surtout pour des critères exigeants (comme les quadrigrammes). **Remarque** : Il est important ici de noter que le nombre de permutations effectuées n'équivaut pas forcément au nombre de permutations maximales autorisées. En effet, on peut très bien imaginer un cas où il reste 2000 permutations disponibles mais notre algorithme s'arrête car on a stagné durant 200 itérations.

5.2 Hill Climbing optimisé

Lorsque l'on applique la version optimisée du hill climbing, on observe une amélioration notable de la qualité des résultats, notamment pour les trigrammes ($n = 3$) et quadrigrammes ($n = 4$). Grâce au mécanisme de relance (génération d'une nouvelle clef aléatoire après 200 itérations sans amélioration), l'algorithme parvient à explorer davantage l'espace des solutions.

Observations principales :

— **Amélioration des scores moyens** : Pour un même nombre de permutations, les scores finaux sont généralement supérieurs à ceux obtenus par le hill climbing standard, en particulier pour les grands n-grammes.

— **Stabilité accrue** : Les résultats sont plus réguliers, avec une variance réduite par rapport au hill climbing simple. (Voir figure **??**) Cela signifie que l'algorithme est plus fiable, et ne dépend pas autant du hasard de la clef initiale.

— **Gain d'efficacité sur les grands textes** : Le bénéfice de la relance devient particulièrement visible avec les textes plus longs. (Voir figure **??**) La diversité des séquences statistiques donne plus de chances à une clef correcte d'être distinguée par le score.

Remarque : Ces observations ne semblent pas tout à fait correspondre dans le cas où les textes sont courts : il n'y à presque aucun changement avec le hill climbing classique.

5.3 Recuit simulé

Annexes

Références

— https://www.bibmath.net/crypto/

— https://www.apprendre-en-ligne.net/crypto/index.php.