Rapport du projet LU2IN013

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

Encadrante : Mme Valérie Ménissier-Morain

Helder Brito, O'nel Hounnouvi

Février 2025 – Juin 2025

Table des matières

1 Introduction

La cryptographie constitue depuis toujours un pilier dans la sécurisation des communications. Historiquement, les cryptosystèmes classiques – chiffrements par substitution, par transposition, de Vigenère, de Playfair, etc. – ont permis de chiffrer autant des messages privés que des secrets militaires et civils. Le déchiffrage de ces messages reposait sur des méthodes manuelles, dont le degré de rigueur et de succès variait selon les époques. Avec l'avènement de l'informatique et l'amélioration des capacités de calcul, le domaine a évolué : ce qui était autrefois l'apanage d'experts peut désormais s'appuyer sur des algorithmes modernes d'optimisation. Que ce soit avec le *hill climbing, le recuit simulé* ou encore *les algorithmes génétiques*, il est désormais possible d'automatiser et d'améliorer ce processus de cryptanalyse.

L'objectif de ce projet est de formaliser et comparer ces approches d'automatisation.

2 Substitution monoalphabétique

2.1 Définition

La substitution monoalphabétique est l'une des plus anciennes méthodes de chiffrement. Elle consiste à remplacer dans le message clair une lettre donnée de l'alphabet par une autre lettre. 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

Le message *SUBSTITUTION* devient *PRYPQFQRQFLK*.

L'alphabet latin comporte 26 lettres. Cela permet donc de construire $26! = 4 \times 10^{26}$ permutations, soit de l'ordre de 2^{88}. Sachant qu'environ 2^{58} secondes se sont écoulées depuis la création de l'univers, il serait impossible d'explorer toutes les permutations. Ce chiffre donne une impression de sûreté qui est toutefois trompeuse...

2.2 Cryptanalyse

2.2.1 Un peu de vocabulaire

Le **déchiffrage** consiste à convertir un message chiffré en son contenu original (le texte clair) lorsqu'on est en possession de la clé.

La **cryptanalyse** quant à elle est l'ensemble des techniques mises en œuvre pour tenter de casser le message codé sans avoir la clé de chiffrement.

Le **cryptogramme** est le message chiffré, c'est-à-dire le texte codé 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 ($n = 2$) incluent *CR*, *RY*, *YP*, ..., tandis que les trigrammes ($n = 3$) sont *CRY*, *RYP*, *YPT*, ...

2.2.2 Attaque

La substitution monoalphabétique possède une grosse faiblesse structurelle. Chaque occurrence d'une lettre du texte clair est systématiquement remplacée par la même lettre chiffrée. Ainsi, les caractéristiques statistiques de la langue française sont conservés en partie dans le message chiffré. C'est cette faille que nous allons exploiter. Notre analyse ici sera basée sur l'analyse des fréquences d'apparition des n-grammes dans le message chiffré.

En utilisant un dictionnaire de référence contenant les fréquences relatives des n-grammes dans un large corpus de textes en français, il est possible d'estimer la probabilité qu'un texte donné soit écrit dans cette langue.

2.2.3 Fitness function

Pour évaluer la qualité des solutions potentielles et identifier celle qui convient le mieux, nous attribuons un score à chaque solution avec une *fitness function*. Pour être efficace, une telle fonction doit être suffisamment discriminante c'est à dire ressortir les différences entre un texte encore très chiffré (score mauvais) et un texte proche du clair (score bon). Elle doit être aussi rapide à calculer, car elle sera appelée de nombreuses fois. La fitness function utilisée dans ce projet est basée sur la somme des probabilités logarithmiques des n-grammes du texte chiffré. C'est la fonction de log-vraisemblance. Elle est définie comme suit :

$$score = -\sum \log(frequence(c_1 \dots c_n))$$

L'objectif ici est de minimiser ce score. La solution ayant le plus petit score est celle qui sera «le plus» français.

2.3 Métaheuristiques

Une métaheuristique est un algorithme d'optimisation itératif et stochastique[1] destiné à résoudre des problèmes pour lesquels aucune méthode classique plus efficace n'existe. Ici elle servira à nous rapprocher le plus possible du minimum glabal de la fonction score.

2.3.1 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 $C1$.

1. Un processus stochastique est un processus dont l'évolution est déterminée par des phénomènes aléatoires.

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`

2.3.2 Hill Climbing 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.

2.3.3 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 2.3.1)
   (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.

2.3.4 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 2.3.1).
   (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.

3 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.

3.1 Hill Climbing

— Figure 1 : 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.

3.2 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.

3.3 Recuit simulé

Annexes

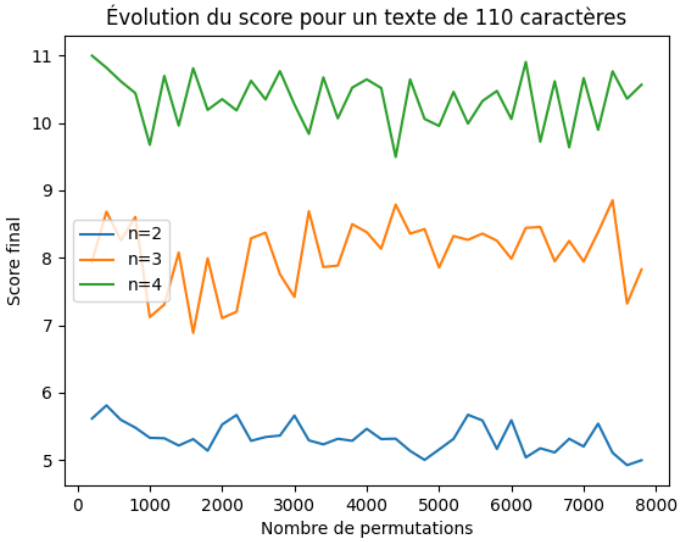

FIGURE 1 – Hill Climbing sur un texte de 110 caractères.

Références

— https://www.bibmath.net/crypto/

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