



Rapport du projet LU2IN013

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

Encadrante : Mme Valérie Ménissier-Morain

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

Table des matières

1	Introduction	2
2	Substitution monoalphabétique et cryptanalyse	2
2.1	Définition	2
2.2	Cryptanalyse	2
2.2.1	Quelques définitions	2
2.2.2	Attaque par analyse fréquentielle	2
2.2.3	Fonction de score (fitness function)	3
3	Premiers essais : Hill Climbing	3
3.1	Principe de l'algorithme Hill Climbing	3
3.2	Résultats initiaux et observations	3
3.2.1	Influence de la taille du texte et du nombre maximum de stagnations	3
3.2.2	Influence du choix de n dans les n -grammes	4
3.2.3	Taux de réussite selon la taille du texte	5
3.3	Limites identifiées	5
4	Recherche d'alternatives plus robustes	5
4.1	Motivations	5
4.2	Amélioration du Hill Climbing	6
4.3	Recuit simulé	7
4.3.1	Principe du recuit simulé	7
4.3.2	Résultats	7
4.4	Recherche tabou	9
5	Comparaisons des différentes métaheuristiques	9
6	Limites de l'attaque par analyse fréquentielle	9
	Annexes	11

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**...

2.2 Cryptanalyse

2.2.1 Quelques définitions

- Le **cryptogramme** est un message chiffré à l'aide d'une clé.
- 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 cryptogramme *sans connaître* la clé.
- 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.
- Une **métaheuristique** est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficiles pour lesquels on ne connaît pas de méthode classique plus efficace. C'est généralement un algorithme stochastique itératif, qui progresse vers un optimum global, en passant d'une solution à une solution voisine (si possible meilleure).

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 :

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

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

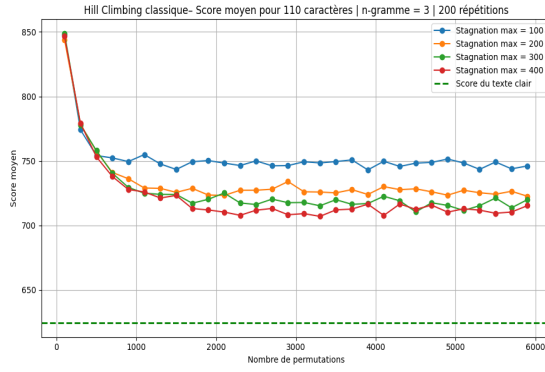
Il nous faut maintenant définir comment générer une clé voisine à la clé courante (étape 2-(a) de l'algorithme). Pour cela, on effectue une permutation aléatoire de deux lettres dans la clef. En guise d'exemple :

QWERTZUIOPASDFGHJKLXCVBNM → QGERTZUIOPASDFWHJKLXCVBNM

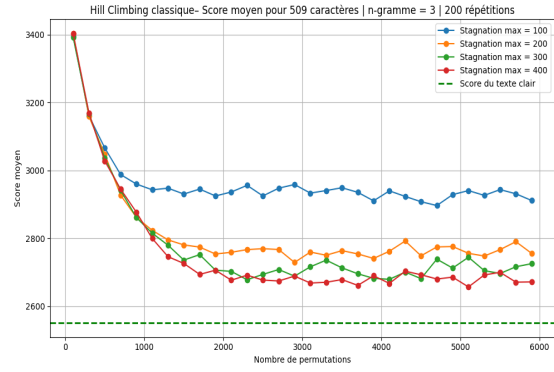
3.2 Résultats initiaux et observations

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

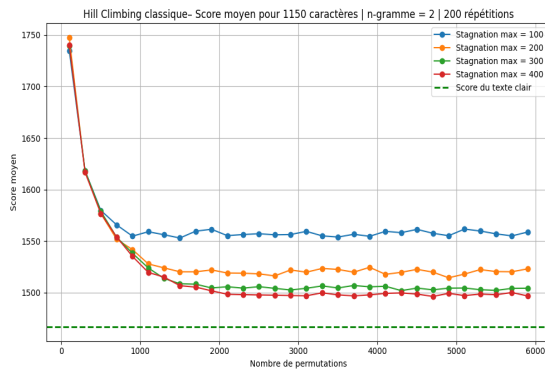
FIGURE 1 – Évolution du score moyen avec $n = 3$ selon la taille du texte.

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.

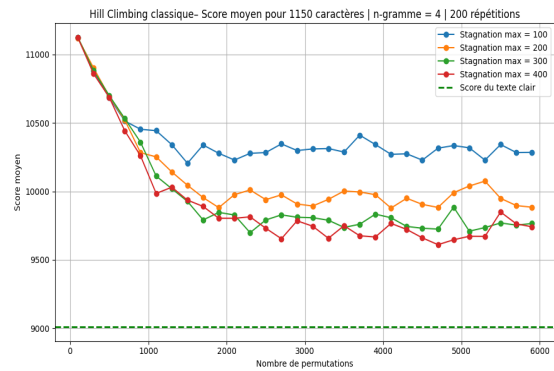
Enfin, le paramètre de stagnation, qui fixe le nombre maximal d'itérations sans amélioration avant arrêt de l'algorithme, joue également un rôle clé. Si cette valeur est trop faible, l'algorithme risque de s'arrêter prématurément avant d'avoir atteint un minimum local de qualité. À l'inverse, une valeur 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.

3.2.3 Taux de réussite selon la taille du texte

Pour compléter l'analyse, nous présentons sur la [figure 3](#) le taux de réussite moyen de l'algorithme de *hill climbing* en fonction de la longueur du texte à déchiffrer. Ce taux est défini comme la proportion d'essais (sur 200) où la solution obtenue présente une similarité caractère à caractère supérieure avec un seuil de 0.9

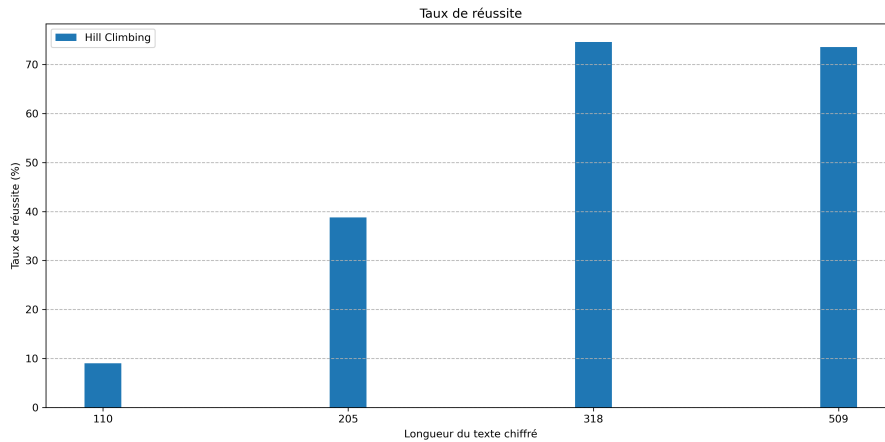


FIGURE 3 – Taux de réussite du *hill climbing* en fonction de la longueur du texte.

Les résultats confirment une tendance claire : lorsque le texte chiffré contient moins de 300 caractères, le taux de réussite reste faible et très instable. Cela s'explique par l'insuffisance des statistiques n -grammiques sur un corpus trop court, ce qui rend la fonction de score peu fiable.

En revanche, à partir de 300 à 400 caractères, les performances s'améliorent nettement, avec une stabilisation du taux de réussite d'environ 70 %. Le *hill climbing* devient alors une méthode relativement fiable pour la cryptanalyse monoalphabétique. Ce constat rejoint les observations précédentes sur la nécessité d'un corpus suffisamment long pour que la fonction de score basée sur les n -grammes soit efficace.

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.

4 Recherche d'alternatives plus robustes

4.1 Motivations

L'algorithme de *hill climbing* adopte une stratégie purement gloutonne : il se contente d'exploiter localement les solutions voisines qui améliorent immédiatement le score, sans jamais envisager de déplacements temporaires vers des solutions moins favorables. Cette approche le rend particulièrement vulnérable aux minima locaux : une fois piégé, il ne dispose d'aucun mécanisme pour en sortir.

De plus, ses performances dépendent fortement du point de départ (clé initiale), et deux exécutions peuvent produire des résultats très différents selon l'initialisation, en particulier pour les textes courts.

Pour surmonter ces limitations, nous explorons des alternatives plus robustes et plus flexibles, capables :

- de mieux équilibrer *exploration* et *exploitation*
- de s'échapper des minima locaux
- et d'améliorer les performances globales, en particulier la stabilité des résultats.

4.2 Amélioration du Hill Climbing

Pour pallier les limitations du *hill climbing* classique, nous introduisons une version optimisée : au lieu d'abandonner après une stagnation excessive, elle effectue une réinitialisation de la recherche. Une nouvelle clé aléatoire est générée, et l'algorithme recommence depuis l'étape 1. Cette stratégie permet d'échapper plus facilement aux pièges des minima locaux tout en conservant la rapidité de l'approche gloutonne.

La [figure 4](#) compare l'évolution du score moyen (avec $n = 3$ et une taille de texte de 110 caractères) entre l'algorithme classique (déjà présenté précédemment) et sa version optimisée.

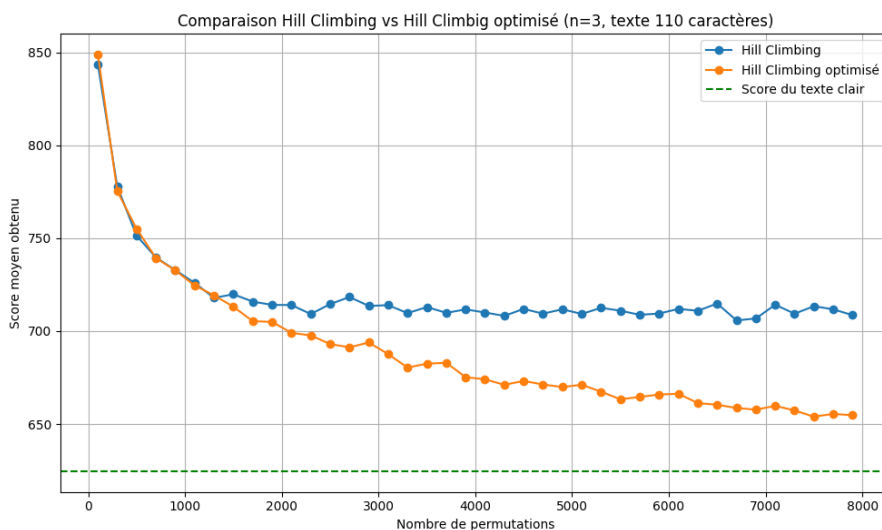


FIGURE 4 – Comparaison de l'évolution du score moyen pour le *hill climbing* classique et optimisé ($n = 3$, texte de 110 caractères).

On observe que le score moyen obtenu par l'algorithme optimisé est bien meilleur que celui du *hill climbing* classique. Cela indique une convergence vers des solutions de meilleure qualité, malgré la faible longueur du texte.

Cette amélioration se reflète également dans le taux de réussite global : comme le montre la [figure 8](#), le *hill climbing* optimisé obtient un taux de réussite nettement supérieur au classique, en particulier pour les textes court (100 à 200 caractères), où l'approche classique reste souvent piégée dans des configurations sous-optimales.

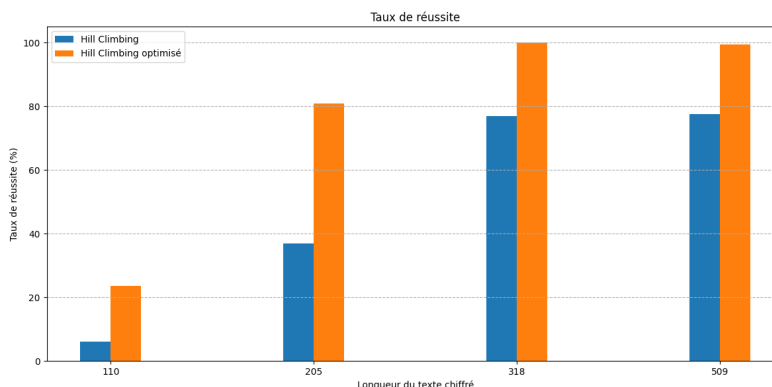


FIGURE 5 – Taux de réussite comparé entre le *hill climbing* classique et optimisé selon la longueur du texte.

En somme, cette version optimisée du hill climbing parvient non seulement à améliorer la qualité des solutions trouvées, mais elle renforce également la robustesse de l'algorithme face aux difficultés posées par les textes courts. Cette amélioration a toutefois un coût en temps de calcul, un aspect que nous analyserons plus en détail dans la [section 5](#).

4.3 Recuit simulé

4.3.1 Principe du 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 déchiffrer le cryptogramme
- (c) Calculer le score du texte obtenu

2. Boucle principale :

- (a) Générer une solution voisine $C2$ en faisant une légère modification (voir ??)
- (b) Calculer le changement de coût, défini par

$$\Delta = \text{score}(C2) - \text{score}(C1).$$

- (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.

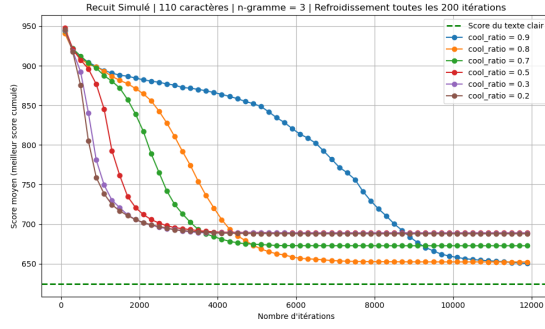
Au fur et à mesure que l'algorithme progresse, la température est progressivement abaissée (refroidissement), ce qui diminue la probabilité d'accepter des solutions moins performantes. Ainsi, en phase finale, le recuit simulé affine la solution dans un voisinage qui se rapproche d'un minimum global.

4.3.2 Résultats

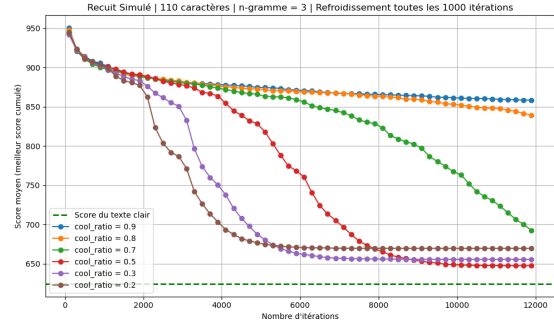
Dans cette section, nous avons trois paramètres importants à considérer : la température initiale T_0 , le coefficient de refroidissement **cool-ratio** (indiquant la fraction de la température actuelle qui est conservée) et le nombre d'itérations au bout duquel on applique le refroidissement **cool-time**.

La température initiale T_0 est calculée en évaluant la variation moyenne des scores obtenus sur un échantillon de 100 clés générées aléatoirement. Cette moyenne est ensuite multipliée par un facteur empirique, permettant d'obtenir une température adaptée, ni trop faible ni trop élevée. Ainsi, la température s'ajuste automatiquement à la taille du message chiffré et au n-gramme choisi, assurant une exploration efficace de l'espace de recherche.

La figure 6 présente l'évolution du score moyen pour un texte court de 110 caractères, en utilisant des trigrammes ($n = 3$) et en faisant varier le **cool-ratio** entre 0.2 et 0.9 et selon le cas où le refroidissement est fréquent (cool-time = 200) ou moins fréquent (cool-time = 1000).



(a) Cool-time = 200 itérations



(b) Cool-time = 1000 itérations

FIGURE 6 – Texte court (110 caractères)

- Pour un refroidissement fréquent (cool-time = 200), on observe qu'un cool-ratio bas (par exemple, 0.2 ou 0.3) refroidit rapidement le système. Cela conduit à une convergence précoce vers des scores inférieurs à celui du texte clair, probablement en piégeant l'algorithme dans des minima locaux. En revanche, avec un cool-ratio plus élevé (0.8 ou 0.9), la température décroît lentement, permettant à l'algorithme de continuer à accepter des solutions moins bonnes pendant un plus grand nombre d'itérations, avant de se stabiliser sur une solution assez proche du texte clair. On voit clairement que la valeur **cool-ratio** = **0.8** donne de meilleurs résultats, à partir de 7000 itérations.
- Pour un refroidissement moins fréquent (cool-time = 1000), on observe que la décroissance est plus lente, moins immédiate mais finit pas converger. Un cool-ratio élevé (0.8 ou 0.9) retarde la diminution de la température sur le long terme, ce qui donne une impression de stagnation du score moyen, sans aucune amélioration significative. Par contre un cool-ratio plus bas (0.2 ou 0.3), permet une décroissance rapide mais contrôlée (à cause de la fréquence réduite) de la température et donc du score moyen. Cela permet à l'algorithme de continuer à explorer l'espace de recherche pendant un plus grand nombre d'itérations et de mener à des scores finaux meilleurs. On remarque qu'un cool-ratio de 0.3 apparaît comme un bon compromis en convergeant vers un score proche du score du texte clair dès environ 7000 itérations, tout en restant comparable aux performances obtenues avec un cool-ratio de 0.5 à 10000 itérations.

Dans la suite des comparaisons, nous allons considérer un refroidissement fréquent (cool-time = 200) car il donne de meilleurs résultats en terme de stabilité.

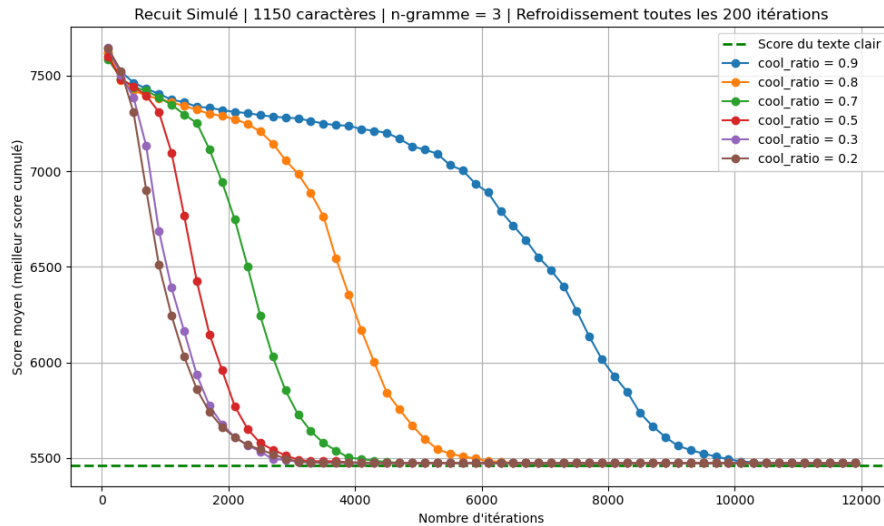


FIGURE 7 – Texte long (1150 caractères)

On remarque que les graphes ont la même allure, que ce soit pour un texte court ou long. La différence est que la courbe des scores moyens (peu importe la valeur de cool-ratio) se rapproche beaucoup plus du score du texte clair pour le texte long. On en déduit que le recuit simulé est plus efficace sur des textes longs, tout comme avec les deux algorithmes précédents.

En ce qui concerne l'influence du choix des n-grammes, on note une légère amélioration de la convergence du score moyen les bigrammes ($n = 2$). Mais les tétragrammes ($n = 4$) ne semblent pas apporter d'amélioration significative (voire même dégradent les performances pour les textes

courts). Les figures 9 et 10 en annexes le montrent clairement. Les trigrammes restent une fois de plus le bon compromis pour tous les textes.

Généralement, à partir de 200-300 caractères, on obtient d'excellents avec le recuit simulé.

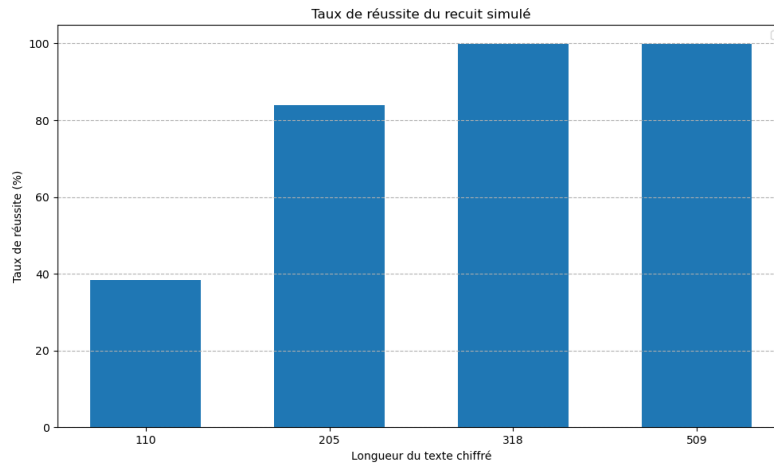


FIGURE 8 – Taux de réussite du recuit simulé selon la longueur du texte.

4.4 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 explorées.

2. **Boucle principale :**

- (a) Explorer un échantillon de clés voisines $C2$ en faisant de légères modifications (voir ??).
- (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

3. **Critère d'arrêt :** Terminer l'algorithme après un nombre prédéfini d'itérations.

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

5 Comparaisons des différentes métaheuristiques

6 Limites de l'attaque par analyse fréquentielle

Bien qu'efficace sur le chiffrement par substitution, l'attaque par analyse fréquentielle montre rapidement ses limites face à certains cas particuliers. En voici quelques exemples notables :

- **Longueur insuffisante du texte :** Comme on a pu le souligner plus haut, les textes courts ne fournissent pas assez de données statistiques pour que l'analyse fréquentielle soit fiable.
- **Structures linguistiques atypiques :** Certains textes littéraires exploitent des contraintes formelles qui brouillent les statistiques usuelles.

Premier exemple :

La Disparition de Georges Perec est un roman **lipogramme** de 300 pages qui exclut entièrement la lettre « e », pourtant la plus fréquente en français. En voici un extrait :

Anton Voyl n'arrivait pas à dormir. Il alluma. Son Jaz marquait minuit vingt. Il poussa un profond soupir, s'assit dans son lit, s'appuyant sur son polochon. Il prit un roman, il l'ouvrit, il lut ; mais il n'y saisissait qu'un imbroglio confus, il butait à tout instant sur un mot dont il ignorait la signification...

L'analyse des fréquences fait disparaître le « e » et rend donc impossible toute cryptanalyse.

Deuxième exemple :

De Zanzibar à la Zambie et au Zaïre, des zones d'ozone font courir les zèbres en zigzags zinzens.

Ici, c'est la surabondance de la lettre « Z » qui pose problème.

Annexes

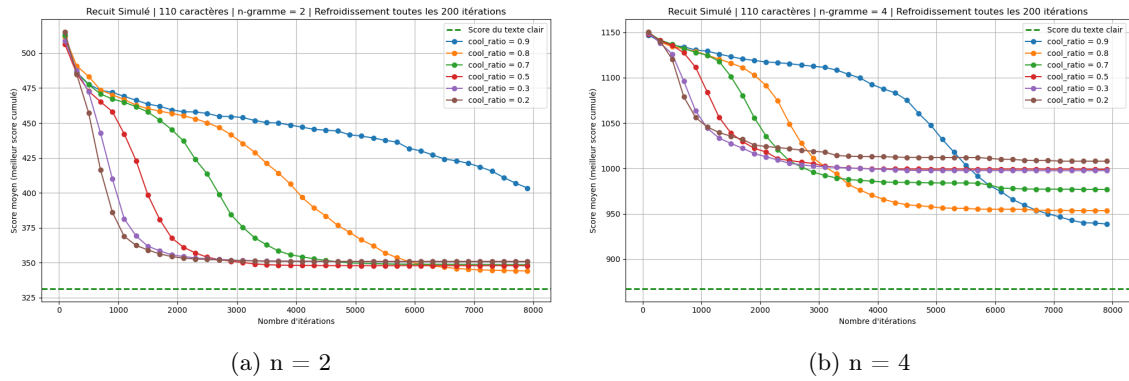


FIGURE 9 – Influence des n-grammes. Texte court (110 caractères) avec recuit simulé.

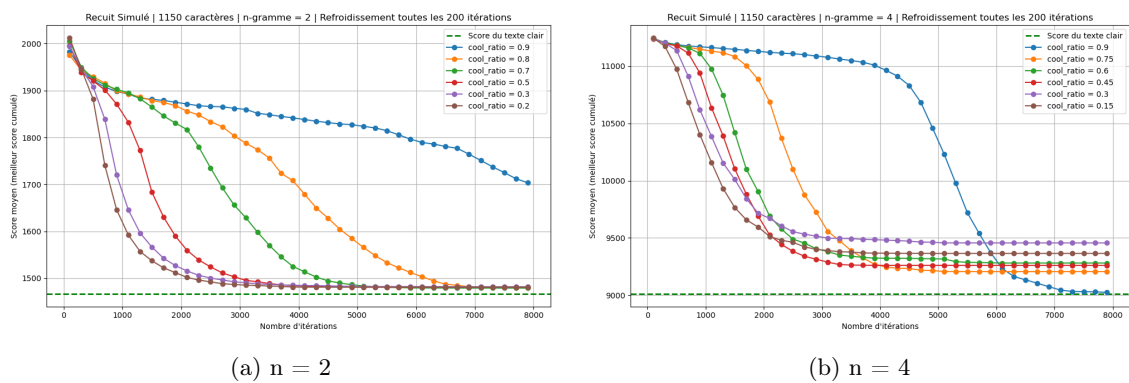


FIGURE 10 – Influence des n-grammes. Texte court (110 caractères) avec recuit simulé.

Références

- [1] Page web Bibmath, <https://www.bibmath.net/crypto/>
- [2] Didier Müller, *Les métaheuristiques en cryptanalyse*, Bulletin no 143 de la SSPMP, mai 2020, <https://www.apprendre-en-ligne.net/auteur/articles/metaheuristiques-cryptanalyse.pdf>
- [3] *Automated cryptanalysis of substitution cipher using hill climbing*, <https://www.montis.pmf.ac.me/vol44/11.pdf>
- [4] Helder Brito, O'nel Hounnouvi, *Dépôt GitHub du projet*, <https://github.com/onelhounnouvi/LU2IN013-Projet>