Algorithme d'optimisation combinatoire pour la cryptanalyse de crypto systèmes classique

Flavio Sens
Dirigé par Valérie Ménissier-Morain

12 juillet 2023

Tables des matières

Introduction

1.1 Historique

L'origine et l'histoire de la cryptologie remontent à plusieurs civilisations anciennes qui ont utilisé des chiffrements de substitution mono alphabétique pour protéger des informations sensibles. Le remplacement de symboles, la forme la plus élémentaire de cryptographie, est apparu dans les écrits égyptiens antiques et mésopotamiens. On a découvert le plus ancien exemple connu dans la tombe d'un noble égyptien nommé Khnumhotep II, datant d'il y a environ 3 900 ans.

Dans les premiers cas, le remplacement de symboles dans les inscriptions de Khnumhotep avait pour but de renforcer l'aspect linguistique plutôt que de dissimuler des informations. Environ 3 500 ans auparavant, un scribe mésopotamien utilisait déjà la cryptographie pour dissimuler une formule de glaçure de poterie sur des tablettes d'argile.

Par la suite, la cryptographie a été largement utilisée pour protéger des informations militaires importantes, une pratique qui persiste encore aujourd'hui. Dans la cité grecque de Sparte, par exemple, les messages étaient chiffrés à l'aide d'un parchemin placé autour d'un cylindre d'une taille spécifique, ce qui rendait le message indéchiffrable jusqu'à ce qu'il soit enroulé autour d'un cylindre similaire par le destinataire, c'est la Scytale. On sait également que des espions de l'Inde antique utilisaient des messages codés dès le IIe siècle av. J.-C.

Le chiffrement Atbash est l'un des premiers exemples connus de chiffrement par substitution, utilisé par les Hébreux pour coder leur alphabet. Il s'agit d'un chiffrement de substitution simple qui remplace chaque lettre par sa lettre correspondante à l'extrémité opposée de l'alphabet. Par exemple, "A" est remplacé par "Z", "B" par "Y", et ainsi de suite. Un autre exemple est le chiffrement de César, également connu sous le nom de chiffrement par décalage, qui est l'un des chiffrements par substitution les plus simples. Il tire son nom de Jules César, qui l'aurait utilisé pour communiquer avec ses généraux. Le chiffrement de César fonctionne en décalant chaque lettre du texte en clair d'un nombre fixe de positions vers le bas de l'alphabet.

L'analyse de fréquence, qui consiste à étudier la fréquence de chaque lettre dans le texte chiffré pour déterminer le décalage le plus probable, peut facilement casser le chiffrement de César. Malgré ses faiblesses, il a été utilisé tout au long de l'histoire pour des tâches de chiffrement simples en raison de sa facilité de compréhension et de mise en œuvre. Il sert également de base pour les chiffrements de substitution plus complexes, qui utilisent plusieurs décalages ou d'autres méthodes de substitution pour renforcer la sécurité du chiffrement.

1.2 Objectif de la démarche

L'objectif de notre démarche est de présenter une méthode stochastique pour briser les chiffrements de substitution mono-alphabétiques en utilisant uniquement le texte chiffré et un texte représentatif de la langue Française.

Nous avons mis en œuvre cette méthode sur un ordinateur et obtenu des résultats fiables pour des textes chiffrés suffisamment longs (environ 200 caractères). Dans la suite de ce rapport, nous expliquons les concepts impliqués dans notre attaque et présentons notre implémentation. Nous présentons également quelques résultats expérimentaux sur l'efficacité de notre méthode pour différents n-gram, itérations locales et longueurs de texte chiffré.

Méthode Stochastique contre les chiffrements par substitution

La substitution mono-alphabétique est une technique de chiffrement dans laquelle chaque caractère est remplacé par exactement un autre caractère. Cela signifie qu'il existe un seul alphabet de substitution utilisé pour le chiffrement et le déchiffrement.

L'avantage de cette méthode réside dans sa simplicité de mise en œuvre. Cependant, l'inconvénient majeur est la vulnérabilité à l'analyse fréquentielle et aux attaques de déchiffrement basées sur des patterns logiques. Plus le texte chiffré est long, plus il est facile de deviner quelle lettre fréquente du texte chiffré correspond probablement à la lettre "E" la plus couramment utilisée dans la langue française par exemple.

Des tableaux de fréquences de lettres et des outils disponibles en ligne rendent cette tâche relativement simple. De plus, si le texte chiffré est structuré de manière simple, il devient encore plus facile de déchiffrer le chiffrement mono-alphabétique, même pour des personnes non expertes disposant simplement d'un stylo, d'une feuille de papier et d'un peu de temps.

Figure 1 : Exemple d'alphabet de substitution mono-alphabétique

2.1 Fonction Fitness

Dans le cadre de notre étude, nous avons mis en place différentes fonctions axées sur un paramètre spécifique que nous appellerons "n". Ce paramètre "n" représente le nombre de caractères utilisés comme référence, également appelés n-grammes, et peut varier de 1 à 5. Par exemple, l'utilisation d'une fonction fitness avec n égal à 4 permet d'évaluer un déchiffrement en se basant sur les fréquences des tétragrammes (séquences de quatre lettres successives) dans une langue.

Pour mener à bien notre recherche, nous nous sommes principalement appuyés sur des statistiques concernant la distribution des n-grammes en français. Cependant, notre programme est adaptable pour d'autres langues si nécessaire.

2.1.1 N-gramme

Dans le but d'évaluer la qualité d'un déchiffrement par rapport à un autre, il est nécessaire de disposer de certains outils pour évaluer la pertinence de la solution. À cet effet, nous définissons une fonction fitness qui attribue une valeur numérique afin de quantifier dans quelle mesure un texte donné ressemble au français. Définie de cette manière, l'adéquation aux textes français peut dépendre légèrement du corpus utilisé pour compiler les tableau des fréquences. Nous avons utilisé des textes de Germinal, Victor Hugo et Marcel Proust afin d'avoir une diversité maximum et de se rapprocher de l'indice de coïncidence officiel de la langue française.

Ici l'indice de coïncidence de notre texte de référence est de 0,0764 qui est d'un écart de 2*10^-3 de l'indice de coïncidence de référence de la langue française.

Il existe différentes approches pour définir la fonction de fitness. Dans notre cas, nous avons opté pour la somme des logarithmes des fréquences des n-grammes. Les fréquences des n-grammes sont calculées à partir d'un large corpus de textes en français. En d'autres termes, la fonction parcourt le texte potentiellement déchiffré, analyse chaque n-gramme présent, et vérifie sa présence dans un dictionnaire de référence des n-grammes de la langue française. Si le n-gramme est présent, son logarithme de fréquence dans la langue française est ajouté au score. Afin de limiter l'ampleur du score sur de longs textes, nous utilisons une base logarithmique de 2 (log2) pour additionner les différents scores des n-grammes.

$$Fitness = \sum_{i} log2\ (pi)$$

Pseudocode :

```
getfitness(key,cipher,dict_ngram,n)
          score_sum=0
          uncipher=decrypt(cipher,key)
          for i in range (0,len(cipher)-n)
               ngram=uncipher[i:i+n]
               score=dict_ngram.get(ngram)
               if el != None
                    score_sum += math.log2(el)
               else
                    score_sum += 0.0001
          return score_sum
```

2.1.1 Relation de Pearson

L'étude des corrélations de Pearson permet de modéliser la relation linéaire entre deux variables aléatoires continues, X et Y. Il s'agit de la méthode la plus connue en statistiques pour déterminer s'il existe une relation linéaire entre ces deux variables. Cette corrélation est quantifiée par le coefficient de corrélation de Pearson, noté « r », dont la valeur se situe dans l'intervalle [-1, 1]. Plus le coefficient se rapproche de 1, plus X et Y varient ensemble dans le même sens. Une corrélation parfaite entre X et Y se traduit par un « r » égale à 1.

Dans notre contexte, nous allons utiliser cette corrélation en considérant X comme la fréquence de chaque caractère dans le texte chiffré en cours de transformation, $\bar{X}$ comme la moyenne de X, Y comme la fréquence des caractères dans la langue de référence et $\bar{Y}$ comme la moyenne de Y. Nous pouvons modéliser cette corrélation de Pearson en utilisant la formule suivante :

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

Figure 2 : Formule corrélation de Pearson

2.1.3 Évaluation de la qualité de la cryptanalyse

Lorsqu'il s'agit d'évaluer la qualité d'un déchiffrement, il est courant de se concentrer sur le nombre de caractères correctement déchiffrés dans la clé. Cependant, dans notre cas, nous souhaitons évaluer la qualité globale de la cryptanalyse du texte, ce qui rend la simple évaluation du nombre de caractères corrects dans la clé limitée. Par exemple, si deux caractères sont inversés dans la clé et que ces deux caractères sont omniprésents dans le texte, la clé pourrait obtenir une note de 93%, alors que le texte peut être correctement déchiffré à moins de 93% ou peut même devenir inintelligible en raison de l'inversion des lettres et de leurs emplacements inappropriés.

Pour remédier à cette limitation, nous avons opté pour l'attribution d'une note finale à chaque cryptanalyse, basée sur le ratio des lettres incorrectement déchiffrées par rapport au nombre total de caractères du texte. Cette approche nous permet de mieux se rapprocher de la qualité réelle et la pertinence de la cryptanalyse, en prenant en compte les erreurs qui peuvent impacter la compréhension du texte, telles que des lettres inversées dans la clé omniprésentes dans le texte.

La fonction `rate_decipher(cipher,key_original,key_try)` retourne la note réelle de la cryptanalyse. Ainsi, notre évaluation tient compte du contexte global et permet d'observer la pertinence de chaque cryptanalyse de manière plus précise.

2.1.4 Itérations globales et locales

On pourrait prendre en compte dans notre démarche l'effet du nombre d'itérations globales et locales sur l'optimisation des résultats dans l'algorithme hill climbing. Dans nos observations, il est clairement visible que fixer une limite pour les itérations globales n'est pas vraiment utile, car le programme atteint généralement un état d'équilibre suite au nombre limité d'itérations locales prédéfini.

Comme le montrent les graphiques d'exécution, le score augmente de manière logarithmique par rapport au nombre d'itérations, ce qui signifie qu'après un certain point, l'amélioration apportée par chaque itération devient négligeable, donc la limite des itérations locales rentre en compte. De ce fait, dans la suite de notre expérimentation nous négligerons l'impact du nombre d'itérations globales sur l'optimisation des résultats.

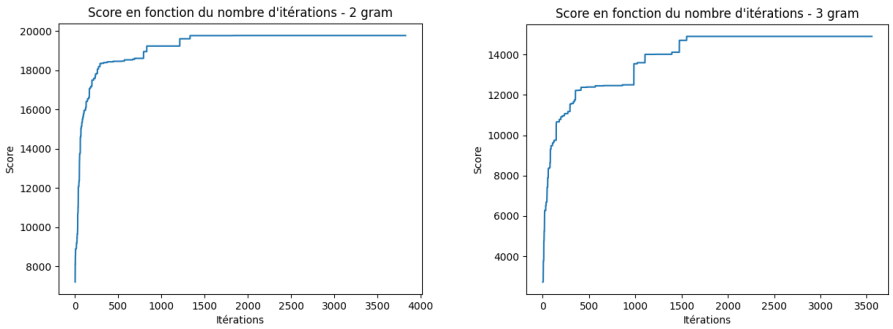

Figure 3 : Score en fonction du nombre d'itérations globale appliqué à 2-gramme et 3-gramme

2.2 Algorithme hill climbing

Hill climbing est une technique d'optimisation mathématique utilisée en analyse numérique. C'est un algorithme itératif qui vise à trouver une meilleure solution locale en apportant des modifications itératives à une solution initiale. Dans le cas d'une attaque stochastique sur les chiffrements par substitution mono-alphabétique, on utilise une clé « enfant » dérivée d'une clé « parent ». On commence avec une clé parente, puis on déchiffre le texte chiffré avec cette clé et on calcule la fitness. Ensuite, on génère une clé enfant en échangeant deux éléments choisis au hasard dans la clé parent. On déchiffre à nouveau le texte chiffré avec la nouvelle clé et on calcule la fitness. Si la fitness de la nouvelle clé est supérieure à celle de la clé parent, on remplace la clé parent par la clé enfant. Ce processus est répété jusqu'à ce que la fitness ne s'améliore plus au cours d'un nombre prédéfini d'itérations `limited_step`.

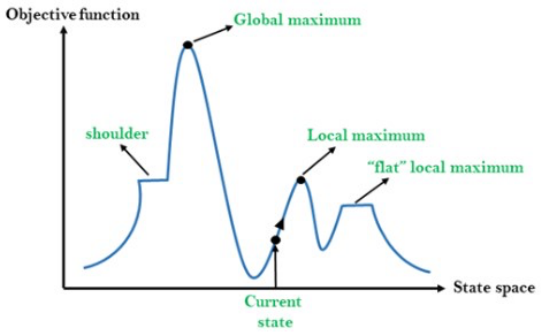

Figure 4 : Illustration des problématiques de l'algorithme de hill climbing

2.2.1 Définition de la fonction hill_climbing

La fonction `hill_climbing(cipher, limited_step, fitness, dict_ngram, n, key_originale)` prend en paramètres le texte chiffré, le nombre limité d'itérations, la fonction fitness, le dictionnaire représentatif de la langue française, le nombre de caractères dans le n-gramme, ainsi que la clé de déchiffrement d'origine. Son rôle est de tester un certain nombre de clés potentielles afin de trouver la meilleure, en fonction du nombre limité d'itérations locales spécifié. Une fois ce nombre atteint, l'algorithme s'arrête et renvoie la meilleure clé obtenue ainsi que le score associé et la note de la cryptanalyse grâce à la comparaison avec le déchiffré originale.

Pseudo code :

```
best_key=gen_key()
best_fit=getfitness(best,cipher,dict_ngram,n)
step=0

while True do
     step+=1
     new_key=gen_key(bes_keyt)
     fit_new_key=getfitness(new_key,cipher,dict_ngram,n)
     if fit_new_key>best_fit:
          best_fit=fit_new_key
          best_key=new_key
          step=0
     if step==limited_step then
          break

output best_key
```

3 Exécution

3.1 Dictionnaire de fréquences et traitement du texte

Pour faciliter l'implémentation, nous utiliserons des dictionnaires comme structure de données tout au long du processus. Ces dictionnaires représentent la distribution typique d'un alphabet de n-grammes dans la langue française. Chaque clé du dictionnaire correspond à un n-gramme, par exemple « en », et la valeur associée représente le nombre d'occurrences de ce n-gramme dans le corpus de texte ou dans le texte potentiellement déchiffré.

Nous avons développé plusieurs fonctions pour gérer différents scénarios.

La fonction `dico_ngram(txt, n)` retourne un dictionnaire de fréquences de n-gramme à partir d'un texte préalablement nettoyé et d'une valeur n représentant le nombre de caractères à utiliser comme clés associés à la valeur qui est la fréquence des n-gramme.

Les fonctions `dico_to_csv(file, n)` et `csv_to_dico(file, n)` prennent en paramètre le nom du fichier contenant un texte nettoyé et la valeur n correspondant au nombre de caractères du n-gramme. Ces fonctions permettent de lire les dictionnaires représentant la distribution de la langue française au format CSV, puis de les convertir en dictionnaire exploitable par le programme, évitant ainsi la lecture du fichier à chaque itération de la boucle.

Ensuite, la fonction `nettoyer_texte(file)` prend également le nom d'un fichier en paramètre et se charge de supprimer tous les caractères qui ne font pas partie de l'alphabet standard de 26 lettres. Elle retourne ensuite un fichier nettoyé de la forme « file_nettoye ».

3.2 Résultats

3.2.1 Textes très courts

Suite à nos expérimentations, nous avons constaté que 750 itérations étaient en moyenne optimales grâce aux score obtenus qui deviennent satisfaisant à partir de ce point-là. Nous avons donc réalisé des tests sur de courts textes allant de 30 à 100 caractères en utilisant tous les n-grammes, en fixant le nombre d'itérations à 750, conformément à notre conclusion précédente.

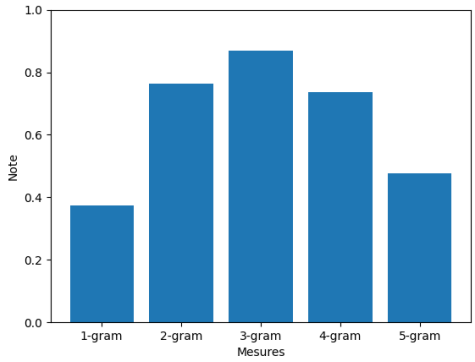

Figure 5 : Note moyenne cryptanalyse de l'algorithme hill climbing appliqué à différents n-gramme

Nous avons constaté que l'application de l'algorithme hill-climbing avec des trigrammes était la plus performante, avec une note moyenne de cryptanalyse de 90% sur 100 textes différents. Les bi-gramme et tétragramme suivent de près en termes de performances.

3.2.2 Texte Court

Dans le cas de textes courts, composés de 100 à 500 caractères, nous constatons que les trigrammes continuent de se démarquer. Cette fois-ci, `limited_step` semble optimale en étant compris entre 2250 et 3750.

3.2.3 Texte long

Lorsqu'il s'agit de textes longs, nous constatons de très bons résultats pour presque tous les types de n-grammes, à l'exception des 1-grammes. Les bi-grammes et les trigrammes obtiennent des performances presque équivalentes, bien que les trigrammes affichent une légère avance dans un seul cas. Ils sont suivis de très près par les tétragrammes. Dans ces cas-là, il n'y a pas un nombre d'itérations locale prédominante à partir de laquelle les résultats vont changer de manière significative.

3.2.4 Pearson

L'application de la fonction de corrélation de Pearson à l'algorithme hill climbing peut sembler une bonne idée, car Pearson retourne un coefficient de corrélation entre deux vecteurs, dans notre cas les listes de fréquences d'alphabet (du texte potentiellement déchiffré et du texte de référence la langue) triées par ordre alphabétique de même longueur. Cependant, d'après nos essais, cette approche n'a pas donné des résultats cohérents et se montre très variable pour des textes de moins de 1500 caractères.

Nous avons donc décidé de limiter notre analyse à des textes de plus de 1500 caractères, mais même dans ce cas, les résultats restent assez irréguliers. Les bi-grammes ont obtenu les meilleurs résultats dans ces essais, avec quelques textes correctement déchiffrés à 93%. Nous n'avons pas observé de nombre d'itérations qui change significativement les résultats. Nous avons toutefois remarqué que de bonnes performances de cryptanalyse correspondent à un coefficient de corrélation de Pearson proche de 1, mais l'inverse ne se vérifie pas systématiquement.

4 Conclusion

Au cours de cette expérimentation, nous avons pu démontrer l'efficacité redoutable de l'algorithme hill climbing, qui peut susciter, au premier abord, des doutes quant à sa fonctionnalité sur le papier.

Nous avons observé différentes performances de cryptanalyse en fonction de la fixation du nombre d'itérations locales, de la longueur du texte à déchiffrer et de la fonction fitness. Voici quelques résultats à retenir :

- Les trigrammes se sont révélés être nettement plus performants pour la cryptanalyse, indépendamment de la longueur du texte, avec un nombre d'itérations locales compris entre 2250 et 3000 pour les textes courts, et entre 500 et 750 pour les textes longs.

- Plus la longueur du texte augmente plus les tétragrammes se montrent fiables et performants

- L'utilisation de la corrélation de Pearson comme fonction fitness a donné un bon indicateur de la proximité d'un déchiffrement avec la langue française, mais cette méthode n'est pas suffisamment régulière pour être utilisée de manière fiable.

Annexes

Textes 300 caractères

1-gramme

step	score	time (s)	decipher rate
500	4956	0.21	0.25
750	4956	0.22	0.22
1000	4956	0.2	0.19
1250	4956	0.25	0.25
1500	4956	0.29	0.21
1750	4956	0.31	0.21
2000	4956	0.38	0.22
2250	4956	0.45	0.22
2500	4956	0.42	0.22
2750	4956	0.56	0.22
3000	4956	0.49	0.23
3250	4956	0.51	0.19
3500	4956	0.5	0.21
3750	4956	0.58	0.22
4000	4956	0.56	0.21
4250	4956	0.55	0.23
4500	4956	0.7	0.22
4750	4956	0.62	0.25
5000	4956	0.65	0.21

2-gramme

step	score	Time (s)	decipher rate
500	3718	0.18	0.46
750	3801	0.27	0.85
1000	3589	0.38	0.27
1250	3742	0.47	0.75
1500	3595	0.48	0.13
1750	3707	0.45	0.43
2000	3600	0.45	0.28
2250	3617	0.49	0.25
2500	3577	0.48	0.01
2750	3702	0.55	0.52
3000	3825	0.58	0.98
3250	3743	0.74	0.54
3500	3825	0.63	0.98
3750	3680	0.7	0.47
4000	3706	0.83	0.43
4250	3753	0.81	0.6
4500	3661	0.86	0.09
4750	3774	0.78	0.68
5000	3824	0.77	0.92

3-gramme

step	score	Time (s)	decipher rate
500	2374	0.28	0.58
750	2755	0.54	0.94
1000	2160	0.22	0.01
1250	2385	0.75	0.42
1500	2158	0.36	0.08
1750	2403	0.42	0.31
2000	2256	0.51	0.21
2250	2820	0.38	1.0
2500	2535	0.49	0.61
2750	2820	0.75	1.0
3000	2820	0.72	1.0
3250	2747	0.71	0.93
3500	2820	0.68	1.0
3750	2820	0.64	1.0
4000	2517	0.66	0.57
4250	2273	0.87	0.36
4500	2418	0.82	0.37
4750	2215	0.83	0.09
5000	2820	0.91	1.0

4-gramme

step	score	Time (s)	decipher rate
500	1041	0.3	0.21
750	1284	0.21	0.42
1000	993	0.26	0.13
1250	1956	0.55	1.0
1500	1233	0.3	0.31
1750	892	0.4	0.15
2000	732	0.42	0.05
2250	1146	0.64	0.19
2500	1956	0.63	1.0
2750	891	0.5	0.04
3000	1956	0.6	1.0
3250	970	0.65	0.09
3500	888	0.48	0.18
3750	908	0.62	0.37
4000	1202	0.73	0.31
4250	1956	0.73	1.0
4500	1956	0.95	1.0
4750	837	0.82	0.12
5000	817	0.72	0.09

Textes 1500 caractères

1-gramme

step	score	time (s)	decipher rate
500	25137	0.93	0.58
750	25137	1.12	0.58
1000	25137	1.27	0.58
1250	25137	1.21	0.58
1500	25137	1.54	0.58
1750	25137	1.57	0.58
2000	25137	2.11	0.58
2250	25137	2.03	0.58
2500	25137	2.32	0.58
2750	25137	2.11	0.58
3000	25137	2.35	0.58
3250	25137	2.48	0.58
3500	25137	2.77	0.58
3750	25137	3.2	0.58
4000	25137	3.1	0.58
4250	25137	3.5	0.58
4500	25137	3.21	0.58
4750	25137	3.56	0.58
5000	25137	3.84	0.58

2-gramme

step	score	time (s)	decipher rate
500	19778	0.8	1.0
750	19778	1.96	1.0
1000	19778	1.84	1.0
1250	19778	2.18	1.0
1500	19778	2.21	1.0
1750	19778	3.09	1.0
2000	18294	2.5	0.18
2250	19778	2.42	1.0
2500	19778	2.91	1.0
2750	19778	3.55	1.0
3000	19778	3.12	1.0
3250	19778	3.61	1.0
3500	19778	3.82	1.0
3750	19778	3.65	1.0
4000	19778	3.98	1.0
4250	19778	4.27	1.0
4500	19778	4.2	1.0
4750	19778	4.56	1.0
5000	19778	4.67	1.0

3-gramme

step	score	time (s)	decipher rate
500	14900	1.85	1.0
750	14900	1.38	1.0
1000	14900	2.16	1.0
1250	14900	1.84	1.0
1500	14887	1.63	1.0
1750	14900	3.05	1.0
2000	10528	2.23	0.14
2250	14900	2.67	1.0
2500	14900	2.75	1.0
2750	14900	2.66	1.0
3000	14900	3.53	1.0
3250	14900	3.8	1.0
3500	14900	3.5	1.0
3750	14900	5.27	1.0
4000	14900	3.9	1.0
4250	14900	3.85	1.0
4500	14900	5.1	1.0
4750	14900	3.97	1.0
5000	14900	4.62	1.0

4-gramme

step	score	time (s)	decipher rate
500	10511	2.11	1.0
750	10527	1.25	1.0
1000	10527	1.73	1.0
1250	10527	2.66	1.0
1500	10527	2.44	1.0
1750	10527	2.31	1.0
2000	4252	1.99	0.1
2250	10527	2.56	1.0
2500	10527	3.54	1.0
2750	10527	2.9	1.0
3000	4342	3.26	0.05
3250	10527	3.22	1.0
3500	10527	3.46	1.0
3750	10527	4.43	1.0
4000	10527	4.07	1.0
4250	4167	4.61	0.09
4500	4157	3.96	0.13
4750	10527	4.29	1.0
5000	5006	4.12	0.01

Textes 1500 Pearson

1-gramme

step	score	time (s)	decipher rate
500	1	0.91	0.46
750	1	0.99	0.46
1000	1	1.31	0.59
1250	1	1.19	0.46
1500	1	1.41	0.59
1750	1	2.37	0.5
2000	1	1.55	0.52

2-gramme

step	score	time (s)	decipher rate
500	1	2.07	0.05
750	1	3.59	0.93
1000	1	4.74	0.93
1250	1	2.32	0.24
1500	1	2.77	0.09
1750	1	3.49	0.17
2000	1	3.97	0.93

3-gramme

step	score	time (s)	decipher rate
500	1	5.9	0.01
750	1	8.71	0.0
1000	1	11.54	0.14
1250	1	17.1	0.0
1500	1	11.69	0.01
1750	1	10.01	0.01
2000	1	16.59	0.03

Bibliographie

[1] Histoire de la cryptologie : *www.apprendre-en-ligne.net, https://fr.wikipedia.org/wiki/Histoire_de_la_cryptologie, https://www.ssi.gouv.fr/particulier/bonnes-pratiques/crypto-le-webdoc/cryptologie-art-ou-science-du-secret/*

[2] Algorithme hill climbing : *https://fr.wikipedia.org/wiki/M%C3%A9thode_hill-climbing, https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/*

[3] Corrélation de Pearson : *https://www.techno-science.net/glossaire-definition/Correlation-statistiques.html, https://en.wikipedia.org/wiki/Pearson_correlation_coefficient*

[4] Indice de coïncidence : *https://fr.wikipedia.org/wiki/Indice_de_coïncidence*

[5] : Hacking : The Art of Exploitation, John Erickson