Sorbonne Université — Faculté des sciences
Année universitaire 2020-2021

Automatisation des cryptosystèmes classiques à l'aide d'algorithmes modernes

KAOUCH ABDELSSAMAD, **YAZID SAMY (ADEM)** et **SENIHJI WASSIM**

Encadrés par : Madame Valérie Ménissier-Morain

Sommaire

1 Description

Les cryptosystèmes classiques ont été utilisés de l'Antiquité à la seconde guerre mondiale pour protéger les affaires privées, telles des secrets militaires et civils ou des affaires d'état. Le cassage d'un texte chiffré par un tel système de chiffrement s'effectuait manuellement avec une technique plus ou moins aboutie selon les époques.
Depuis une trentaine d'années un certain nombre de techniques modernes ont été utilisées pour essayer d'automatiser la cryptanalyse de ces cryptosystèmes. Le chiffrement est le terme donné à l'action de rendre la lecture d'un texte clair compliquée si l'on ne possède pas la clef de chiffrement associée. On dit d'un texte qu'il est clair si on peut le lire et le comprendre, tandis qu'un texte chiffré est la version non lisible, celle qu'un utilisateur lambda ne peut lire sans la clef de chiffrement. Cette dernière est la clef permettant de passer du texte clair au texte chiffré. C'est ici que la cryptanalyse intervient. La cryptanalyse est le nom donné au procédé utilisé pour déduire un texte clair à partir d'un texte chiffré sans pour autant posséder la clef de chiffrement.

Le but de ce projet est de réaliser une cryptanalyse, la plus automatisée possible, sous plusieurs approches. Le point de départ sera le *hill climbing*, mais le coeur du projet est le *hill climbing* appliqué à la cryptanalyse des substitutions mono-alphabétiques. Nous avons choisi d'explorer un autre système de chiffrement, celui par transposition, pour voir comment le *hill climbing* s'y adapte.

Deux méthodes de chiffrement seront donc explorées : le chiffrement par substitution et le chiffrement par transposition. Ces deux méthodes sont dites symétriques, car elles n'utilisent qu'une seule et même clef pour chiffrer et déchiffrer. Le premier se base sur une permutation des lettres de l'alphabet avec lequel le texte clair est écrit, tandis que le second est sur un mélange méthodique des caractères composant le texte clair. Le *hill climbing* est une technique possible pour résoudre ces cryptosystèmes.

2 *Hill Climbing*

2.1 Définition

Afin de définir le *hill climbing*, on introduit d'abord la notion de *fitness function* : c'est une fonction qui note une caractéristique d'un paramètre défini par l'utilisateur.
Le *hill climbing* est une technique d'optimisation mathématique par recherche locale. Elle s'appuie sur un algorithme itératif qui débute avec une solution arbitraire à un problème. Il faut évaluer les solutions voisines, et le mieux pour se faire est d'utiliser une *fitness function*. Cette dernière choisit la meilleure solution et recommence l'opération jusqu'à arriver à une position meilleure que les précédentes, en partant de la nouvelle solution si elle est meilleure, ou de l'ancienne si il n'y a pas d'amélioration. Ces différentes positions sont choisies aléatoirement, on parle de marche aléatoire. On a alors trouvé un optimum local. Cependant, la qualité des solutions est définie manuellement, selon les critères donnés à la *fitness function*. Il se peut que la solution voisine soit estimée meilleure et qu'elle soit élue, mais la solution

voisine peut ne pas être meilleure.

2.2 Exemples

L'algorithme de *hill climbing* peut être représenté sous différentes formes. Celle que nous allons prendre pour exemple ici est la recherche d'un élément x dans un tableau avec n éléments, tel que ce x soit le plus grand. La *fitness function* utilisée ici est donc x. Soit le tableau suivant :

n	1	2	3	4	5
x	3	2	4	7	6

Si on lance la recherche à $n = 2$, on obtient comme parcours :
$2 \rightarrow 3 \rightarrow 4$
On a donc comme valeur de retour $n = 4$.

3 Analyse fréquentielle

L'analyse fréquentielle est une méthode consistant à examiner la fréquence des lettres présentes dans un message. Elle se base sur le fait que certaines lettres apparaissent plus que d'autres (exemple : le `e` est plus présent que le `w` dans la langue française). Cependant, cette méthode est d'autant plus efficace que le texte chiffré est long. Si le texte est trop court, il ne nous donnera pas assez d'informations concernant la fréquence des lettres utilisées, tandis qu'un texte long aura tendance à se rapprocher de la fréquence des lettres dans la totalité des mots de la langue utilisée. Il faut tout de même prendre en compte que la distribution des lettres d'un texte long peut cependant ne pas correspondre à celles de la langue entière. Un exemple de cas où l'analyse fréquentielle est biaisée est le lipogramme en une lettre assez fréquente, comme le e qui est la plus fréquente dans l'alphabet. Cela remettra en question le fonctionnement du chiffrement par substition. Un exemple de lipogramme en e connu est *La Disparition* de Georges Perec.

4 Chiffrement par substitution

4.1 Définition

Le chiffrement par substitution est une technique de chiffrement utilisée depuis longtemps. En effet, un des cas particuliers est le chiffrement de César, désignant un chiffrement par substitution monoalphabétique. Comme son nom l'indique, on change dans un message chaque lettre de l'alphabet par une autre. Par exemple, César changeait le `a` par le `d`, le `b` par le `e` etc. Lorsque l'alphabet reste inchangé, on définit la substitution par une permutation des lettres, correspondant à un chiffrement par permutation. Ainsi, un chiffrement par substition peut être facilement déchiffré, par exemple par analyse des fréquences des lettres, ou des polygrammes, à l'aide d'une *fitness function*. Dans notre cas, elle repose sur une analyse fréquentielle des tétragrammes qui composent le message, plus un tétragramme est présent dans la langue française, plus la valeur de retour de la fonction sera élevée.

Afin d'illustrer ces propos, un exemple sur un texte court peut être utilisé. Prenons : PROJET.
On note p la fonction qui prend un tétragramme et renvoie le nombre de fois où il est présent dans le texte. Afin d'avoir la fréquence des tétragrammes présents dans le mot PROJET, on définit une fonction f qui donne la somme des le logarithme en base 10 du nombre de fois où le tétragramme est présent dans le texte .

$$f(\text{PROJET}) = \log_{10}(p(\text{PROJ})) + \log_{10}(p(\text{ROJE})) + \log_{10}(p(\text{OJET}))$$

Soit une table implémentée en dictionnaire python fabriquée à partir d'un corpus significatif de la langue, tel qu'un livre (comme *Germinal*[13]), que l'on nommera dictionnaire théorique contenant les valeurs des fréquences de la langue utilisée dans cet exemple. Si les valeurs sont par exemple : PROJ-45, ROJE-85, OJET-36, on injecte les valeurs de chaque tétragramme dans le calcul :

$$f(\text{PROJET}) = log_{10}(45) + log_{10}(85) + log_{10}(36)$$

La qualité de la chaîne est fixée par cette somme de logarithmes, qui va être appliquée sur le possible déchiffré et ses voisins. On connaît donc le résultat de f sur la substitution actuelle et sur les voisins, si l'actuelle est meilleure on repart de celle-ci, sinon on vérifie sur les voisins.

4.2 Exemple de chiffrement par substitution

Un exemple tout simple de chiffrement par substitution. Prenons un texte quelconque. Ici, le texte sera un extrait de [13] :

```
Cette fois, il n'amenait que trois berlines.
```

Il faut l'écrire sans minuscule, ni ponctuation :

```
CETTEFOISILNAMENAITQUETROISBERLINES
```

Puis, à l'aide d'une clef de chiffrement, les lettre sont interchangées. Un texte clair dans la langue française utilise l'alphabet **ABCDEFGHIJKLMNOPQRSTUVWXYZ**, et la clef du texte chiffré correspond à une chaîne de même longueur que l'alphabet, pour laquelle chaque lettre de l'alphabet est associée à une lettre de la clef du même indice. Ici dans notre exemple, la clef de chiffrement sera **LNVPWFKBZIAEUOXDQRHSMYTCGJ**. La traduction du texte dans l'alphabet de la langue française à l'aide de la clef de chiffrement nous donne donc le texte chiffré suivant :

```
VWSSWFXZHZEOLUWOLZSQMWSRXZHNWREZOWH
```

C'est ainsi que le chiffrement par substitution fonctionne. Le procédé inverse est à réaliser

si l'on veut lire le message. Mais, lorsque l'on ne possède pas la clef de chiffrement, le *hill climbing* est une méthode plausible.

4.3 Travail effectué

Nous avons testé deux *fitness function* : La première avait pour principe de prendre la fréquence de tous les tétragrammes sur le texte en question. Nous la comparions à la fréquence des tétragrammes sur le dictionnaire théorique. Nous faisions ensuite la soustraction des fréquences, plus le résultat tendait vers 0 et plus nous nous approchions du texte clair. Nous avons aussi optimisé cette fonction, en n'important pas le dictionnaire à chaque appel.
Cette fonction n'a pas été retenue car trop peu efficace, elle ne fonctionnait pas comme nous le voulions.

Quant à la deuxième testée, elle prend chaque tétragramme de la chaîne de caractère et somme les logarithmes en base 10 du nombre d'apparition du tétragramme choisi dans le dictionnaire théorique. Dans le cas où le tétragramme n'est pas dans le dictionnaire théorique, nous rajoutons une valeur négligeable à la place, afin d'éviter un logarithme de zéro.
La première fonction était le point de départ, et l'idée de la deuxième nous est venu suite à la lecture d'articles comme [3], [4] et [5].

Les résultats du déchiffrement par substitution sont plutôt satisfaisants. Les textes que nous prenons pour tester notre algorithme sont déchiffrés dans la plus grande majorité des cas. La longueur du texte est tout de même importante, plus le texte est long, meilleur le déchiffrement est. Le seul problème rencontré lors de cette partie était l'optimisation. Notre implémentation de la première fonction était tout de même assez lente, et ne fonctionnait pas à tous les coups. La deuxième fonctionne quasi systématiquement pour des textes de longueur raisonnable. Cependant, si la longueur du texte ne dépasse pas les 500 caractères, le déchiffrement ne fonctionnera que trois fois sur quatre environ, et plus le nombre de caractère décline, moins le déchiffrement sera efficace. Il est bien connu que les textes très courts sont difficiles à déchiffrer de manière automatique.

4.4 Optimisation de plausibilité

Améliorer la *fitness function* nous parait compliqué. Elle répond déjà assez bien à nos attentes. Toutefois, il est bon de noter que le texte clair doit être d'une longueur raisonnable, le chiffrement par substition ne fonctionne pas très bien sur les textes courts, et une version adaptée à ces derniers pourrait être intéressante. Jusqu'ici, le chiffrement se base sur l'hypothèse que le texte clair est suffisement grand, et que la proportion de tétragrammes à l'intérieur puisse être représensative de la langue.

5 Chiffrement par transposition

5.1 Définition

Le chiffrement par transposition repose sur une méthode qui se sert d'anagrammes construits directements depuis le texte que l'on veut chiffrer. Ce texte clair est coupé en blocs de taille identiques, écrits ligne par ligne et rangés dans des colonnes numérotées de 1 à n, avec $n > 1$. Ces colonnes font toutes la même taille, et sont complétées (ce qu'on appelle le *padding*) si nécessaire.
Une permutation est effectuée sur ces colonnes, et le texte chiffré correspond donc au texte lu colonne par colonne dans l'ordre de la permutation. Quant à la clef de chiffrement, c'est la permutation.

Par exemple, pour chiffrer la phrase :

`Mais dis donc, on n'est quand même pas venus pour beurrer les sandwichs!`

On numérote les colonnes **123456789**. Puis, on les réordonne de façon arbitraire : **587139246**.
Il faut prendre la phrase souhaitée sans minuscule ni ponctuation, ce qui donne :

`MAISDISDONCONNESTQUANDMEMEPASVENUSPOURBEURRERLESSANDWICHS`

et l'écrire sur 9 colonnes (la longueur de la permutation), en complétant par des X si nécessaire. (c'est le *padding*).

1	2	3	4	5	6	7	8	9
M	*A*	*I*	*S*	*D*	*I*	*S*	*D*	*O*
N	*C*	*O*	*N*	*N*	*E*	*S*	*T*	*Q*
U	*A*	*N*	*D*	*M*	*E*	*M*	*E*	*P*
A	*S*	*V*	*E*	*N*	*U*	*S*	*P*	*O*
U	*R*	*B*	*E*	*U*	*R*	*R*	*E*	*R*
L	*E*	*S*	*S*	*A*	*N*	*D*	*W*	*I*
C	*H*	*S*	*X*	*X*	*X*	*X*	*X*	*X*

Les colonnes sont permutées entre elles selon la clef **587139246**, ce qui nous donne :

5	8	7	1	3	9	2	4	6
D	*D*	*S*	*M*	*I*	*O*	*A*	*S*	*I*
N	*T*	*S*	*N*	*O*	*Q*	*C*	*N*	*E*
M	*E*	*M*	*U*	*N*	*P*	*A*	*D*	*E*
N	*P*	*S*	*A*	*V*	*O*	*S*	*E*	*U*
U	*E*	*R*	*U*	*B*	*R*	*R*	*E*	*R*
A	*W*	*D*	*L*	*S*	*I*	*E*	*S*	*N*
X	*X*	*X*	*C*	*S*	*X*	*H*	*X*	*X*

Et enfin, il faut lire le texte chiffré colonne par colonne. En commençant par la colonne 5, on peut lire DNMNUAX. Le texte final chiffré correspond à celui de toutes les colonnes. Dans cet exemple, c'est :
`DNMNUAXDTEPEWXSSMSRDXMNUAULCIONVBSSOQPORIXACASREHSNDEESXIEEURNX`

5.2 Travail effectué

On se place dans le cas où le caractère de *padding*, ainsi que le nombre de colonnes (donc la taille de la clef) sont connus d'avance.
Pour le déchiffrement par transposition, nous avons testé et utilisé deux *fitness function*. Pour la première, il s'agit d'une réadaptation de la *fitness function* du déchiffrement par substition mais dans le cas de la transposition. Ainsi, son fonctionnement est similaire : nous prenons chaque tétragramme de la chaîne de caractère donnée, et nous sommons les logarithmes en base 10 du nombre d'apparition du tétragramme choisi dans le dictionnaire théorique. Dans le cas où le tétragramme n'est pas dans le dictionnaire théorique, nous rajoutons une valeur négligeable à la place.
Dans la deuxième *fitness function*, le fonctionnement est le même mais une condition est rajoutée : plus il y a de caractères de *padding* successifs à la fin, plus la valeur augmente.
Le caractère de *padding* est un caractère qui n'appartient pas au texte clair, il sert à faire en sorte que la transposition se déroule bien.
Ici, la première *fitness function* est celle du déchiffrement par substitution, nous savions que le principe fonctionnait et nous avons donc choisi de le réutiliser. Quant à la seconde, l'idée d'utiliser des caractères de *padding* nous est venue lors d'une discussion avec notre encadrante.

Pour le *hill climbing*, nous avons réadapté celui du déchiffrement par substition mais pour la transposition. La marche aléatoire ne se fait pas sur l'alphabet qui est la clé de la substition, mais bien sur la clé de la transposition qui est une suite de chiffre correspondant à l'ordre des colonnes.

5.3 Optimisation de plausibilité

Afin de le rendre meilleur, il faudrait réussir à adapter avec caractère de *padding* inconnu. Puis, caractère de *padding* connu mais nombre de colonnes inconnu. Et enfin, le cas où tout est inconnu.

6 Interface Graphique

Lors de nos nombreux tests, nous avons remarqué que l'utilisation de nos fonctions sur la console était un peu embêtante. Afin d'améliorer les conditions d'utilisation des fonctions par l'utilisateur, nous avons décidé d'implémenter une interface graphique.

Pour ce faire, nous avons rapidement appris comment utiliser la biliothèque Tkinter, permettant d'implémenter des interfaces graphiques sur python. Suite à cela nous avons implémenté notre propre interface graphique. Lors de l'exécution du programme, nous arrivons sur un menu principal demandant de choisir quel système de chiffrement l'utilisateur aimerait casser. Suite à ce choix, se faisant en cliquant sur un des deux boutons disponibles, l'utilisateur arrive sur une fenêtre sur laquelle il peut écrire son message chiffré, cliquer sur un bouton et attendre quelques instants, après quoi le message déchiffré et la clé de chiffrement seront affichés. Il y a également sur la fenêtre un autre bouton permettant de repartir sur le menu principal, permettant de changer de système de chiffrement. Il faut également préciser que sur le chiffrement par transposition, les fonctions utilisées pour casser le cryptosystème présupposent que la clé est à 9 colonnes (9 chiffres) et que le caractère de padding est «&».

7 Conclusion

Le chiffrement par substitution n'est pas assez fort, un texte chiffré de la sorte peut être déchiffré rapidement. Notre automatisation de la cryptanalyse pour cette méthode chiffrement arriver à casser des textes chiffrés de manière instantanée. Le principe reste simple : on intervertit certaines lettres avec d'autres. Aujourd'hui, la substitution est peu efficace, mais cette conclusion a déjà été faite il y a plusieurs siècles, notamment par le philosophe Al-Kindi en l'an 800. C'est la raison pour laquelle nous nous sommes penchés vers un autre type de chiffrement, celui par transposition. À notre connaissance, il n'y a pas d'outil de cryptanalyse automatique de transposition.
Son fonctionnement indique qu'il est plus complexe a déchiffrer. Et en effet, dans le cas où tout est connu (nombre de colonnes et caractère de *padding*), il est possible de déchiffrer ce texte. Mais, si l'un des paramètres est inconnu, ce qui serait le cas dans une situation réelle, la manipulation devient tout de suite beaucoup plus compliquée.
Ainsi, chaque méthode de chiffrement a ses limites, et son déchiffrement n'est qu'une question de temps.

Références Bibliographiques

[1] Bibmath. *Faiblesse de la substitution mono-alphabétique*. 2021. URL: http://www.bibmath.net/crypto/index.php?action=affiche&quoi=substi/subanalyse.

[2] CIHEAM. *Guide de présentation des normes bibliographiques*. 2010. URL: https://www.iamm.ciheam.org/download/95.

[3] Lasry D. "A Methodology for the Cryptanalysis of Classical Ciphers with Search Metaheuristic". In: *Kassel university presst* (2018).

[4] Lasry D. Greenwade. *Cryptanalysis of the Simple Substitution Cipher*. 2019. URL: http://practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-simple-substitution-cipher/.

[5] Lasry D. Greenwade. *Quadgram Statistics as a Fitness Measure*. 2017. URL: %5Curl%7Bhttp://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/%7D.

[6] David Louapre. *Markov Chain Monte Carlo*. 2021. URL: https://scienceetonnante.com/2021/04/23/mcmc-code/n.

[7] Valérie Ménissier-Morain. "La sécurité des données". In: *Laboratoire Informatique de Paris 6, UPMC* ().

[8] OrdiRoutier. *Interface graphique Tkinter*. 2020. URL: https://www.youtube.com/watch?v=wxnXNcU-YBQ&list=PLjrnnc4BZaRCR5eOXSTAgKJpBl62Y7o45.

[9] Overleaf. *Online Latex Editor*. 2012. URL: https://www.overleaf.com/project.

[10] Wikipedia. *Chiffrement par substitution*. 2019. URL: https://fr.wikipedia.org/wiki/Chiffrement_par_substitution.

[11] Wikipedia. *Chiffrement par transposition*. 2019. URL: https://fr.wikipedia.org/wiki/Chiffrement_par_transposition.

[12] Wikipedia. *Lipogramme*. 2011. URL: https://fr.wikipedia.org/wiki/Lipogramme.

[13] Emile Zola. "Germinal". In: (1885), p. 16.

Annexe

A Interface

```
#_____________________IMPORT_____________________
from hillClimbing import *
from tkinter import *
from PIL import ImageTk, Image

#_____________________INITIALISATION_____________________

window = Tk()                                       #définition de la fenêtre tkinter
window.geometry("1280x720")                         #redimensionner la taille de la fenêtre
#SCALE : 7*W , 15*H (fullscreen => W=182 H=48)
window.title("Projet PIMA - Hill Climbing")         #titre de la fenêtre
window['bg'] = 'lightgray'                          #couleur de background de la fenêtre
window.resizable(height=False, width=False)         #bloquer le redimensionnement manuel de la fenêtre
#redimensionner une image (et l'initialiser)
image = Image.open("image.png")
resized = image.resize((500, 500), Image.ANTIALIAS)
image = ImageTk.PhotoImage(resized)

image2 = Image.open("image.png")
resized2 = image2.resize((100, 100), Image.ANTIALIAS)
image2 = ImageTk.PhotoImage(resized2)

#_____________________VARIABLES_____________________

mainmenu = 0
titre = 0
description = 0
boutonSubstitution = 0
boutonTransposition = 0

strVariableS = StringVar()
substitution = 0
titreS = 0
consoleS = 0
alphabetS = 0
frameEntreeS = 0
labelEntreeS = 0
entreeS = 0
boutonValiderS = 0
boutonRetourS = 0
```

```
strVariableT = StringVar()
transposition = 0
titreT = 0
consoleT = 0
alphabetT = 0
frameEntreeT = 0
labelEntreeT = 0
entreeT = 0
boutonValiderT = 0
boutonRetourT = 0

#                          FONCTIONS

def getsize(widget):
    widget.update()
    return (widget.winfo_width(), widget.winfo_height())

def cleanWindow(window):
    for child in window.winfo_children():
        child.destroy()

def decrypteSubstitution():
    global substitution
    global strVariableS


    substitution.update()
    frameEntreeS.update()
    entreeS.update()
    message = entreeS.get()#strVariableS.get()
    cle = hill_climbing_permutation_2(message)
    messageDecrypte = translate(message, cle)
    consoleS["text"] = messageDecrypte
    alphabetS["text"] = "La clé de chiffrement est : " + cle
    substitution.update()

def decrypteTransposition():
    global transposition
    global strVariableT
```

```
    transposition.update()
    frameEntreeT.update()
    entreeT.update()
    message = entreeT.get()
    cle = hill_climbing_transposition_taillefixe(message)
    messageDecrypte = suppr_lettre(dechiffrement_transposition(message, cle), "&")
    consoleT["text"] = messageDecrypte
    alphabetT["text"] = "La clé de chiffrement est : " + cle
    transposition.update()

def updateMainMenu():
    global mainmenu
    global titre
    global description
    global boutonSubstitution
    global boutonTransposition

    cleanWindow(window)
    mainmenu = Frame(window, bg="lightgray")
    titre = Label(mainmenu, text="Projet PIMA - Hill Climbing", font="Times 40 italic bold", fg="gray", bg="lightgrey")
    titre.pack()

    description = Label(mainmenu, text="Veuillez sélectionner l'un des onglets ci-dessous :", font="Times 25 bold",
                        fg="gray", bg="lightgrey", pady=20)
    description.pack()

    boutonSubstitution = Button(mainmenu, image=image, command=updateSubstitution)
    boutonSubstitution.pack(side=LEFT, padx=55)

    boutonTransposition = Button(mainmenu, image=image, command=updateTransposition)
    boutonTransposition.pack(side=RIGHT, padx=55)
    mainmenu.pack(pady=25)
```

```
def updateSubstitution():
    global strVariableS
    global substitution
    global titreS
    global consoleS
    global alphabetS
    global frameEntreeS
    global labelEntreeS
    global entreeS
    global boutonValiderS
    global boutonRetourS

    cleanWindow(window)


    strVariableS = StringVar()

    substitution = Frame(window, bg="lightgrey", height=720)

    titreS = Label(substitution, text="Décryptage de chiffrement par substitution", font="Times 30 italic bold",
                   fg="gray", bg="lightgrey")
    titreS.pack()

    consoleS = Label(substitution, text="Le message déchiffré s'affichera ici-même",
                     font="Times 15 bold", fg="gray", bg="lightgrey", wraplength=1200, borderwidth=3, relief="groove",
                     width=100, height=20)
    consoleS.pack(pady=5)


    alphabetS = Label(substitution, text="L'alphabet/clé de chiffrement est : ABCDEFGHIJKLMNOPQRSTUVWXYZ",
                      font="Times 15 bold", fg="gray", bg="lightgrey", borderwidth=3, relief="groove", width=100,
                      height=1)
    alphabetS.pack()


    frameEntreeS = Frame(substitution, bg="lightgrey")

    labelEntreeS = Label(frameEntreeS, text="Veuillez entrer le message chiffré ici :", font="Times 18 bold underline",
                         fg="gray", bg="lightgray", width=30, height=1)
    labelEntreeS.pack(side=LEFT, anchor=NW, padx=16, pady=4)
```

```
    entreeS = Entry(frameEntreeS, textvariable=strVariableS, text="Entrer le message chiffré ici-même",
                    font="Times 15 bold", fg="gray", bg="lightgrey", borderwidth=3, relief="groove", width=90)
    entreeS.pack(padx=36, pady=5, anchor=NE)

    frameEntreeS.pack()

    boutonValiderS = Button(substitution, image=image2, command=decrypteSubstitution)
    boutonValiderS.pack(side=LEFT, padx=50)

    boutonRetourS = Button(substitution, image=image2, command=updateMainMenu)
    boutonRetourS.pack(side=RIGHT, padx=50)
    substitution.pack()

def updateTransposition():
    global strVariableT
    global transposition
    global titreT
    global consoleT
    global alphabetT
    global frameEntreeT
    global labelEntreeT
    global entreeT
    global boutonValiderT
    global boutonRetourT

    cleanWindow(window)

    strVariableT = StringVar()

    transposition = Frame(window, bg="lightgrey", height=720)

    titreT = Label(transposition, text="Décryptage de chiffrement par transposition", font="Times 30 italic bold",
                   fg="gray", bg="lightgrey")
    titreT.pack()

    consoleT = Label(transposition, text="Le message déchiffré s'affichera ici-même",
                     font="Times 15 bold", fg="gray", bg="lightgrey", wraplength=1200, borderwidth=3, relief="groove",
                     width=100, height=20)
    consoleT.pack(pady=5)
```

```
alphabetT = Label(transposition, text="La clé de chiffrement est : 123456789",
                  font="Times 15 bold", fg="gray", bg="lightgrey", borderwidth=3, relief="groove", width=100,
                  height=1)
alphabetT.pack()

frameEntreeT = Frame(transposition, bg="lightgrey")

labelEntreeT = Label(frameEntreeT, text="Veuillez entrer le message chiffré ici :", font="Times 18 bold underline",
                     fg="gray", bg="lightgray", width=30, height=1)
labelEntreeT.pack(side=LEFT, anchor=NW, padx=16, pady=4)

entreeT = Entry(frameEntreeT, textvariable=strVariableT, text="Entrer le message chiffré ici-même",
                font="Times 15 bold", fg="gray", bg="lightgrey", borderwidth=3, relief="groove", width=90)
entreeT.pack(padx=36, pady=5, anchor=NE)

frameEntreeT.pack()

boutonValiderT = Button(transposition, image=image2, command=decrypteTransposition)
boutonValiderT.pack(side=LEFT, padx=50)

boutonRetourT = Button(transposition, image=image2, command=updateMainMenu)
boutonRetourT.pack(side=RIGHT, padx=50)
transposition.pack()
```

```
#                    MAIN MENU

mainmenu = Frame(window, bg="lightgray")
titre = Label(mainmenu, text="Projet PIMA - Hill Climbing", font="Times 40 italic bold", fg="gray", bg="lightgrey")
titre.pack()

description = Label(mainmenu, text="Veuillez sélectionner l'un des onglets ci-dessous :", font="Times 25 bold", fg="gray", bg="lightgrey", pad
description.pack()

boutonSubstitution = Button(mainmenu, image=image, command=updateSubstitution)
boutonSubstitution.pack(side=LEFT, padx=55)

boutonTransposition = Button(mainmenu, image=image, command=updateTransposition)
boutonTransposition.pack(side=RIGHT, padx=55)
mainmenu.pack(pady=25)

#                    MAINLOOP

#boucle infinie
window.mainloop()
```

B Chiffrement

```python
from random import *
from math import *

#traduction
def translate(s, alphabet):
    dict_alphabet = {}
    for i in range(len(alphabet)):
        dict_alphabet[alphabet[i]] = chr(i + 65)
    s_translated = ""
    for c in s:
        s_translated += dict_alphabet[c]
    return s_translated

#dico theorique (fourni par l'encadrante):
def charge_table (fic):
    """Charge dans un dictionnaire les informations du fichier fic."""
    tab = dict()
    for ligne in fic:
        donnees = ligne.rstrip('\n\r').split(";")
        tab[donnees[0]] = int(donnees[1])
    return tab

def get_dict_theo(nom_fichier):
    return charge_table(open(nom_fichier, "r"))
```

```python
#échange de deux caractères dans un str
def swap(s, i, j):
    sf = ""
    for k in range(len(s)):
        if k != i and k != j:
            sf += s[k]
        elif k == i:
            sf += s[j]
        else:
            sf += s[i]
    return sf

def hill_climbing_substitution(s, epsilon = 0.5 ,alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"):
    dict_theo = get_dict_theo('nb_tetra_fr.csv')
    fit = fitness_substitution(s, alphabet, dict_theo)
    alpha = alphabet[:]
    freeze = 0
    while freeze < 1000:
        i, j = 0, 0
        while i == j:
            i = randint(0, len(alpha) - 1)
            j = randint(0, len(alpha) - 1)
        fit2 = fitness_substitution(s, swap(alpha, i, j), dict_theo)
        if (abs(fit - fit2) <= epsilon) or fit >= fit2:
            freeze += 1
        else:
            fit = fit2
            alpha = swap(alpha, i, j)
            freeze = 0
    return alpha
```

```python
def suppr_lettre(msg, c):
    strfinal = ""
    for e in msg:
        if e != c:
            strfinal += e
    return strfinal

#fitness function finale (transposition)
def fitness_transposition(message, cle, char_padding, dict_theo):
    somme = 0
    msg = dechiffrement_transposition(message, cle, char_padding)
    index = 1

    while (msg[-index] == char_padding):
        somme += 1000
        if (index+1 != len(msg)):
            index += 1
    msgclean = suppr_lettre(msg, char_padding)

    for i in range(len(msgclean) - 3):
        ts = ""
        for j in range(4):
            ts += msgclean[i + j]
        if ts in dict_theo:
            somme += log10(dict_theo[ts])
        else:
            somme += 0.001

    return somme
```

```python
#transposition
def transposition(message, cle, char_padding):
    #copie du str message
    msg = message[:]

    #ajouter les caractères au message pour effectuer le padding, si nécessaire
    if (len(msg) % len(cle)) != 0:
        for i in range(len(cle) - (len(msg) % len(cle))):
            msg += char_padding

    #creation de la matrice du clair
    matrice_cl = []
    for i in range(len(msg) // len(cle)):
        tmp_str = ""
        for j in range(len(cle)):
            tmp_str += msg[i*len(cle)+j]
        matrice_cl.append(tmp_str)

    #creation du message final (chiffré)
    msg_fin = ""
    for i in range(len(cle)):
        for j in range(len(msg) // len(cle)):
            msg_fin += matrice_cl[j][int(cle[i])-1]

    #retour du message final
    return msg_fin
```

```python
def dechiffrement_transposition(message, cle, char_padding="&"):
    #copie du message chiffré
    msg = message[:]

    #creation de la matrice du chiffré
    matrice_ch = []
    for i in range(len(cle)):
        tmp_str = ""
        for j in range(len(msg) // len(cle)):
            tmp_str += msg[i*(len(msg) // len(cle))+j]
        matrice_ch.append(tmp_str)

    #creation du message final (déchiffré)
    msg_fin = ""
    for i in range(len(msg) // len(cle)):
        for j in range(len(cle)):
            msg_fin += matrice_ch[cle.index(str(j+1))][i]

    #retour du message
    return msg_fin

def nb_padding_manquant(msg_dechiffre, char_padding):
    nb_char_pdg = 0
    for e in msg_dechiffre:
        if e == char_padding:
            nb_char_pdg += 1
    for i in range(1, nb_char_pdg + 1):
        if msg_dechiffre[-i] == char_padding:
            nb_char_pdg -= 1
    return nb_char_pdg
```

```python
def nb_padding_manquant(msg_dechiffre, char_padding):
    nb_char_pdg = 0
    for e in msg_dechiffre:
        if e == char_padding:
            nb_char_pdg += 1
    for i in range(1, nb_char_pdg + 1):
        if msg_dechiffre[-i] == char_padding:
            nb_char_pdg -= 1
    return nb_char_pdg
```

```python
def hill_climbing_transposition_taillefixe(message, char_padding = "&", cle = "123456789", epsilon = 0.5):
    #copie de la clé et du message
    key = cle[:]
    msg = message[:]

    #chargement du dictionnaire théorique
    dict_theo = get_dict_theo('nb_tetra_fr.csv')

    #initialisation des valeurs initiales de fit et de freeze
    msg_dechiffre = dechiffrement_transposition(msg, key, char_padding)
    nb_pdg = nb_padding_manquant(msg_dechiffre, char_padding)

    #on commence par poser la condition de rassembler les caractères de padding à la fin
    while nb_pdg != 0:
        i, j = 0, 0
        while i == j:
            i = randint(0, len(key) - 1)
            j = randint(0, len(key) - 1)
        msg_dechiffre = dechiffrement_transposition(msg, swap(key, i, j), char_padding)
        nb_pdg2 = nb_padding_manquant(msg_dechiffre, char_padding)
        if nb_pdg2 <= nb_pdg:
            nb_pdg = nb_pdg2
            key = swap(key, i, j)

    #initialisation des valeurs initiales de fit et de freeze
    fit = fitness_transposition(msg, key, char_padding, dict_theo)
    freeze = 0

    #boucle principale
    while freeze < 15000:
        i, j = 0, 0
        while i == j:
            i = randint(0, len(key) - 1)
            j = randint(0, len(key) - 1)
        fit2 = fitness_transposition(msg, swap(key, i, j), char_padding, dict_theo)
        if (abs(fit - fit2) <= epsilon) or fit >= fit2:
            freeze += 1
        else:
            fit = fit2
            key = swap(key, i, j)
            freeze = 0
    return key
```

```
#fitness function ancienne v1
"""
def fitness(alphabet, s):
    dict_theo = get_dict_theo('nb_tetra_fr.xls')
    dict_s = get_dico(alphabet, s, dict_theo)
    for key in dict_s:
        if key not in dict_theo:
            dict_theo[key] = 0
    somme = 0
    nb_tetra_t = 0
    nb_tetra_s = 0
    for tetra in dict_s:
        nb_tetra_s += dict_s[tetra]
    for tetra in dict_theo:
        nb_tetra_t += dict_theo[tetra]
    for tetra in dict_s:
        somme += abs((dict_s[tetra]/nb_tetra_s)-(dict_theo[tetra]/nb_tetra_t))
    return somme
"""
#fitness function ancienne v2
"""
def fitness(alphabet, s, dict_theo):
    dict_s = get_dico(alphabet, s, dict_theo)
    somme = 0
    nb_tetra_t = 0
    nb_tetra_s = 0
    for tetra in dict_s:
        nb_tetra_s += dict_s[tetra]
    for tetra in dict_theo:
        nb_tetra_t += dict_theo[tetra]
    for tetra in dict_s:
        if tetra in dict_theo:
            somme += abs((dict_s[tetra]/nb_tetra_s)-(dict_theo[tetra]/nb_tetra_t))
        else:
            somme += dict_s[tetra]/nb_tetra_s
    return somme
"""
```

```python
#fitness function finale (substitution)
def fitness_substitution(s, alphabet, dict_theo):
    somme = 0
    s_translated = translate(s, alphabet)
    for i in range(len(s_translated) - 3):
        ts = ""
        for j in range(4):
            ts += s_translated[i + j]
        if ts in dict_theo:
            somme += log10(dict_theo[ts])
        else:
            somme += 0.001
    return somme
```