Alexandre Cormier
(111 101 150)

Alexandre Picard-Lemieux
(111 103 625)

Patrick Côté
(111 103 743)

Vincent Beaudoin
(111 103 778)

Optimisation combinatoire
IFT-4001

Projet d'exploration

Travail présenté à
Claude-Guy Quimper

Informatique et génie logiciel
Université Laval
H2016

1 Introduction

Utilisée depuis l'antiquité, la cryptographie sert à protéger des messages. Celle-ci se fait souvent à l'aide de secrets ou de clés. Cette cryptographie a principalement été utilisée à des fins militaires, commerciales ou simplement pour protéger la vie privée. L'opération de déchiffrer un message peut donc avoir des conséquences assez importantes sur la vie des gens, car ces informations peuvent mener à des actions destructives ou créatives.

L'optimisation combinatoire est une branche de l'informatique et des mathématiques appliquées. C'est la recherche d'une solution au coût minimal d'un problème dont l'espace des solutions est discret. Nous ferons une étude de la viabilité de la programmation par contraintes pour attaquer la cryptographie classique par substitution.

2 Description du problème

Le chiffrement par substitution est une façon de cacher un message en remplaçant chaque lettre par une autre. On parle de substutition monoalphabétique lorsqu'une lettre est toujours remplacée par la même autre lettre. Autrement, on parle de substitution polyalphabétique.

Un exemple de chiffrement par substitution monoalphabétique est le chiffre de César. Pour ce chiffre, on associe à chaque lettre sa position dans l'alphabet, commençant par 0. La clé est une lettre et chaque lettre du message est décalée d'un nombre de positions correspondant à la clé. Par exemple, le message ABC chiffré avec la clé B devient BCD. Les calculs de décalage se font en modulo 26, tel que XYZ chiffré avec la même clé devient YZA.

Le chiffre de vigenère est un exemple de chiffrement par substitution polyalphabétique. La clé est composée d'une ou plusieurs lettres et le message est divisé en partie de la même longueur que la clé. Si les différentes parties du message sont placées l'une en dessous de l'autre, il suffit d'appliquer le chiffre de César à chaque colonne i avec la $i^{ème}$ lettre de la clé. Par exemple, le message ABCD chiffré avec la clé BC devient BDDF.

La substitution est toujours une composante à la base de la cryptographie moderne, mais ces méthodes de chiffrement simples ne sont plus utilisées comme systèmes cryptographiques à part entière puisqu'elles sont vulnérables à différentes attaques. Il est possible, notamment, d'analyser la fréquence relative des lettres dans le message chiffré et comparer avec la fréquence relative des lettres dans la langue du message d'origine pour trouver les clés les plus probables.

Dans le cadre de ce projet, nous étudierons une méthode alternative, utilisant la programmation par contrainte, pour retrouver la clé de tels chiffrements à partir du message chiffré. Il est évidemment impossible pour un programme automatisé de trouver la clé de façon certaine, car il n'y a pas nécessairement moyen de reconnaître le message d'origine lorsque la bonne clé est trouvée. Par contre, nous pourrons ordonner les clés les plus probables selon, notamment, la fréquence relative de chaque lettre dans le message déchiffré.

L'objectif ici est d'étudier la viabilité d'une telle approche, par contrainte, pour l'attaque du chiffrement par substitution. Nous étudierons l'efficacité de l'approche selon la longueur du message et, dans le cas du

chiffre de Vigenère, de la longueur de la clé.

Par exemple, en prenant le message chiffré LXFOPVEFRNHR, le solveur doit trouver que la clé la plus probable est LEMON et que le texte original est ATTACKATDAWN. L'implémentation utilisée utilisera les fréquences des lettres trouvées d'une langue donnée. [1] Cette implémentation sera donc plus efficace si la chaîne est longue et si elle utilise des vrais mots de la langue choisie.

3 Approche proposée

La façon la plus préconisée pour attaquer le chiffrement de Vigenère est l'analyse de fréquences. C'est aussi l'approche que nous proposons, seulement elle sera implémentée à l'aide de la programmation par contraintes. Il s'agit de calculer la fréquence de chaque lettre dans le message déchiffré et de comparer avec la fréquence de chaque lettre dans la langue ciblée.

3.1 Modèle de base

Le modèle présenté ici est le premier modèle conçu, purement théorique, avant toute tentative de l'implémenter dans le solveur de contraintes.

3.1.1 Modélisation du chiffrement de Vigenère

Soit :
— n la longueur de la chaîne à déchiffrer ;
— k la longueur de la clé ;
— l le nombre de lettres dans l'alphabet de la langue ciblée ;
— $\mathbb{N}_n$ l'ensemble des nombres naturels modulo n ;
— $\mathbb{Z}_n$ l'ensemble des nombres naturels modulo n ainsi que leur opposé.

On a donc, $(\forall i : 1 \leq i \leq n)$:
— $C_i \in \mathbb{N}_l$ la $i^{ème}$ lettre de la chaîne chiffrée ;
— $P_i \in \mathbb{N}_l$ la $i^{ème}$ lettre de la chaîne déchiffrée.

Et aussi, $(\forall i : 1 \leq i \leq k)$:
— $K_i \in \mathbb{N}_l$ la $i^{ème}$ lettre de la clé.

On a la contrainte suivante :

$$P_i + K_{i \bmod k} \equiv C_i \mod l \qquad \forall i : 1 \leq i \leq n \tag{1}$$

3.1.2 Modélisation de l'analyse de fréquences

Soit, $(\forall i : 1 \leq i \leq l)$:
— f_i la fréquence de la $i^{ème}$ lettre de l'alphabet dans la langue ciblée ;
— $Z_i \in \mathbb{N}_{n+1}$ le nombre de fois qu'apparaît la $i^{ème}$ lettre de l'alphabet dans le message déchiffré.

1. Wikipedia, [En ligne]. `https://en.wikipedia.org/wiki/Letter_frequency` (Page consultée le 3 avril 2016)

Et soit la contrainte suivante :

$$Z_i = Count(i, P) \qquad \forall i : 1 \leq i \leq l \tag{2}$$

Nous devons avoir une mesure d'évaluation des solutions trouvées. La distance euclidienne entre la fréquence des lettres dans le message déchiffré et celle dans la langue ciblée est un choix intuitif. La fonction objectif est donc la suivante :

$$\min \sqrt{\sum_{i=1}^{n} (\frac{Z_i}{n} - f_i)^2} \tag{3}$$

ou de façon équivalente :

$$\min \sum_{i=1}^{n} (\frac{Z_i}{n} - f_i)^2 \tag{4}$$

3.2 Modèle implémenté

Le modèle de base est simple, mais il présente quelques défis à l'implémentation [2], notamment en lien avec les nombres réels pour les fréquences. On doit aussi ajouter quelques variables et contraintes intermédiaires.

3.2.1 Variables et contraintes intermédiaires

Pour la modélisation du (dé)chiffrement, il faut ajouter des variables supplémentaires, ($\forall i : 1 \leq i \leq n$). Soit donc :

— $I_i \in \mathbb{N}_{2l-1}$ la $i^{ème}$ lettre du texte intermédiaire.

La contrainte 1 est alors séparée en deux contraintes distinctes :

$$P_i + K_{i \bmod k} = I_i \qquad \forall i : 1 \leq i \leq n \tag{5}$$

$$I_i \bmod l = C_i \qquad \forall i : 1 \leq i \leq n \tag{6}$$

De plus, il faut séparer la fonction objectif en plusieurs variables et contraintes. On introduit donc les variables intermédiaires suivantes, ($\forall i : 1 \leq i \leq l$) :

— $F_i \in [0, 1]$ la fréquence de la $i^{ème}$ lettre de l'alphabet dans le message déchiffré ;

— $D_i \in [-1, 1]$ la différence entre la fréquence de la $i^{ème}$ lettre dans le message déchiffré et celle dans la langue ciblée ;

— $A_i \in [0, 1]$ cette différence élevée à la puissance 2.

Il faut aussi ajouter une variable à minimiser :

— $S \in [0, l]$ la somme des différences de fréquences au carré.

Pour lier ces variables ensemble, il faut de nouvelles contraintes :

$$F_i = \frac{Z_i}{n} \qquad \forall i : 1 \leq i \leq l \tag{7}$$

2. Nous avons fait l'implémentation avec le solveur Choco.

$$D_i = F_i - f_i \qquad \forall i : 1 \leq i \leq l \tag{8}$$

$$A_i = D_i^2 \qquad \forall i : 1 \leq i \leq l \tag{9}$$

$$S = \sum_{i=1}^{n} A_i \tag{10}$$

3.2.2 Gestion des nombres réels

Pour simplifier l'implémentation avec le solveur, nous avons adopté une stratégie pour les transformer en entier. Nous avons choisi le niveau de précision désiré, selon la précision des fréquences dont nous disposons [3] pour les langues cibles.

Soit donc :

— p la précision choisie en nombre de décimales.

On applique donc la modification suivante aux fréquences relatives des lettres de la langue ciblée, $(\forall i : 1 \leq i \leq l)$:

$$f_i = f_i * 10^p \qquad \forall i : 1 \leq i \leq l \tag{11}$$

On doit aussi multiplier le compte des lettres, alors on introduit des variables intermédiaires, $(\forall i : 1 \leq i \leq l)$:

— $M_i \in \mathbb{N}_{10^p(n+1)}$ le nombre de fois qu'apparaît la $i^{ème}$ lettre de l'alphabet dans le message déchiffré, multiplié par 10^p.

On ajoute donc la contrainte suivante :

$$M_i = Z_i * 10^p \qquad \forall i : 1 \leq i \leq l \tag{12}$$

Et la contrainte (7) devient plutôt :

$$F_i = \frac{M_i}{n} \qquad \forall i : 1 \leq i \leq l \tag{13}$$

Puisqu'on se retrouve ainsi avec de grands nombres, il faut se méfier du dépassement d'entiers (*integer overflow*). On peut donc simplifier la fonction objectif en remplaçant le carré par la valeur absolue :

$$\min \sum_{i=1}^{n} |\frac{Z_i}{n} - f_i| \tag{14}$$

Cette nouvelle fonction objectif n'est pas équivalente à la première, mais elle permet d'éviter des problèmes et nos résultats sont même meilleurs avec celle-ci que nos résultats préliminaires avec la distance euclidienne.

On doit aussi modifier le domaine des variables reliées à la fréquence. Soit, $(\forall i : 1 \leq i \leq l)$:

— $F_i \in \mathbb{N}_{10^p+1}$ la fréquence de la $i^{ème}$ lettre de l'alphabet dans le message déchiffré ;

— $D_i \in \mathbb{Z}_{10^p+1}$ la différence entre la fréquence de la $i^{ème}$ lettre dans le message déchiffré et celle dans la langue ciblée ;

— $A_i \in \mathbb{N}_{10^p+1}$ la valeur absolue de cette différence.

Et de même pour la variable à minimiser :

3. Wikipedia, [En ligne]. `https://en.wikipedia.org/wiki/Letter_frequency` (Page consultée le 24 avril 2016)

— $S \in \mathbb{N}_{l(10^p+1)}$ la somme des valeurs absolues des différences de fréquences.

Finalement, la division de la contrainte 7 doit maintenant être une division entière.

Au final, nous avons des listes de n, k et l variables. Le nombre de variables est donc borné par $\theta(n+k+l)$.

Chaque variable dans les listes de n ont l ou $2l-1$ valeurs possibles.
Chaque variable dans les listes de k ont l valeurs possibles.
Chaque variable dans les listes de l ont $n+1$, $10^p(n+1)$ ou $2(10^p(n+1))$ valeurs possibles.
Le nombre de valeurs est donc borné par $\theta(nl+kl+l(10^p n))$. p peut toutefois être considéré comme une constante (nous utilisons 5) et donc on obtient $\theta(l(n+k))$.

Nous avons n contraintes de type (5) et (6), l contraintes de type (2), (8), (9), (12) et (13) ainsi qu'une contrainte de type (10). Le nombre de contraintes est donc borné par $\theta(n+l)$.

3.3 Amélioration des heuristiques

Pour tenter d'améliorer les performances du solveur, nous avons créé une nouvelle heuristique adaptée au modèle. Cette nouvelle heuristique comprend le choix de la variable ainsi que le choix de la valeur. Elle fait tous ses calculs lors de son initialisation et elle retourne les résultats de ses calculs lors d'une requête par le solveur.

3.3.1 Stratégie pour choisir la variable

L'heuristique requiert le message chiffré C et la longueur de la clé à trouver k. Ensuite, elle place tous les indices des caractères $c \in C$ dans une liste, dans l'ordre de celui qui a le plus d'occurrences pour l'une des k sous-chaînes. Ces sous-chaînes commencent aux différents indices $[0...k-1]$ dans C et elles contiennent tous les caractères à des bonds de k. Nous séparons en sous-chaînes, puisqu'on sait que deux caractères identiques dans la chaîne chiffrée resteront identiques s'ils sont à une distance égale à k. Lors de la requête d'une variable à choisir par le solveur, l'heuristique ne fait que retourner les indices de sa liste dans l'ordre.

Voyons deux exemples avec $C = ABABBCACBC$

A	**B**	**A**	**B**	**B**	**C**	**A**	**C**	**B**	**C**
0	1	2	3	4	5	6	7	8	9

Exemple avec $k = 1$

Dans ce cas, on a une seule sous-chaîne S qui contient $ABABBCACBC$. On trouve alors que le caractère le plus présent est B avec 4 occurrences. On place finalement le premier indice de B dans la sous-chaîne S, soit 1, dans la liste. Les caractères restants A et C ont la même fréquence dans S, alors on place leur premier indice dans la liste. La liste finale est donc : [1,0,5] qui représente : $[B,A,C]$

Exemple avec $k = 2$

Maintenant, on a deux sous-chaînes à considérer.
$S_1 = AABAB$ les caractères aux indices pairs.
$S_2 = BBCCC$ les caractères aux indices impairs.
Pour S_1, le caractère le plus fréquent est A avec 3 occurrences. Pour S_2, il s'agit plutôt de C avec 3

occurrences. On compare maintenant les caractères retournés par les sous-chaînes et on prend celui qui était le plus fréquent dans sa sous-chaîne. En cas, d'égalité, on choisi le premier caractère trouvé, soit A dans cet exemple. On ajoute à la liste le premier indice de ce caractère dans la chaîne de départ. Après toutes les itérations on obtient [0,5,4,1] qui représente : [A, C, B de S_1, B de S_2]

On remarque ici que pour une même chaîne chiffrée de départ, on retourne un indice différent selon la longueur de la clé.

3.3.2 Stratégie pour choisir la valeur

Nous avons déterminé les lettres les plus utilisées pour une langue. Par exemple, pour l'anglais, on a dans l'ordre les lettres suivantes :

e,t,a,o,i,n,s,h,r,d,l,c,u,m,w,f,g,y,p,b,v,k,j,x,q,z

La stratégie est donc de choisir la première lettre de cette liste, dans cet ordre, qui est toujours dans le domaine de la variable. Il s'agit alors d'une heuristique statique puisqu'elle ne fait que retourner les indices dans un ordre précis. On commence donc par retourner 4 pour la lettre e, suivit de 19 pour t et ainsi de suite.

3.4 Recherche des m meilleures solutions

Dans la majorité des cas, un problème d'optimisation combinatoire demande une solution valide, la meilleure solution ou toutes les solutions. Dans notre cas, seulement la meilleure solution n'est pas suffisante, puisqu'il n'est pas assuré que la solution la plus probable du point de vue de l'analyse de fréquence sera réellement la solution qui mène au bon message en clair. En effet, certains messages peuvent ne pas respecter les fréquences relatives typiques des lettres de la langue dans laquelle il est écrit. Trouver toutes les solutions n'est pas non plus envisageable, puisqu'il serait beaucoup trop long de toutes les calculer et qu'au final, ce serait même moins efficace qu'une attaque par force brute traditionnelle.

Ce que l'on veut donc, c'est obtenir une liste des m solutions les plus probables, pour qu'elle soit ensuite analysée par un humain ou un autre programme. Le choix de m est important : s'il est trop petit, la bonne solution ne sera pas trouvée ; s'il est trop grand, la recherche sera beaucoup trop longue. Pour être efficace, m doit varier selon la longueur de la clé et la longueur de l'alphabet de la langue cible, puisque ce sont les paramètres qui font varier le nombre total de solutions. Le nombre total de solutions est l^k. Le nombre de solutions à retourner que nous avons choisi est $(\frac{l}{4})^k$. Par contre, pour trouver toutes ces solutions, plus de $(\frac{l}{4})^k$ solutions doivent être explorées.

4 Protocole d'expérimentation

Nous avons choisi de faire des tests sur des instances de différentes longueurs de message et de clé, avec des messages qui respectent bien la distribution de lettres anglaises et d'autres qui ne la respectent pas. Les métriques que nous allons observer durant les expérimentations sont le temps de réponse, le nombre de retours arrière, ainsi que le nombre de solutions trouver.

La première instance est considérée comme étant un message long qui a une distribution correcte de la fréquence des lettres par rapport à la distribution typique en anglais.
Voici le texte en clair : FRIEN DSROM ANSCO UNTRY MENLE NDMEY OUREA RSICO METOB URYCA ESARN OTTOP RAISE HIMTH EEVIL THATM ENDOL IVESA FTERT HEMTH EGOOD ISOFT INTER REDWI THTHE IRBON ESSOL ETITB EWITH CAESA RTHEN OBLEB RUTUS HATHT OLDYO UCAES ARWAS AMBIT IOUSI FITWE RESOI TWASA GRIEV OUSFA ULTAN DGRIE VOUSL YHATH CAESA RANSW ERDIT HEREU NDERL EAVEO FBRUT USAND THERE STFOR BRUTU SISAN HONOU RABLE MANSO ARETH EYALL ALLHO NOURA BLEME NCOME ITOSP EAKIN CAESA RSFUN ERAL

La deuxième instance est considérée comme étant un message d'une courte longueur et qui a une distribution de la fréquence des lettres moyenne. Par contre, les lettres les plus courantes en anglais apparaissent particulièrement souvent dans ce message. Voici le texte en clair : GENIUS WITHOUT EDUCATION IS LIKE SILVER IN THE MINE

La troisième instance est une instance qui est considérée comme un long message ayant une très bonne distribution de la fréquence des lettres.
Voici le texte en clair : HEREUPON LEGRAND AROSE WITH A GRAVE AND STATELY AIR AND BROUGHT ME THE BEETLE FROM A GLASS CASE IN WHICH IT WAS ENCLOSED IT WAS A BEAUTIFUL SCARABAEUS AND AT THAT TIME UNKNOWN TO NATURALISTS OF COURSE A GREAT PRIZE IN A SCIENTIFIC POINT OF VIEW THERE WERE TWO ROUND BLACK SPOTS NEAR ONE EXTREMITY OF THE BACK AND A LONG ONE NEAR THE OTHER THE SCALES WERE EXCEEDINGLY HARD AND GLOSSY WITH ALL THE APPEARANCE OF BURNISHED GOLD THE WEIGHT OF THE INSECT WAS VERY REMARKABLE AND TAKING ALL THINGS INTO CONSIDERATION I COULD HARDLY BLAME JUPITER FOR HIS OPINION RESPECTING IT

La quatrième instance est une instance ayant une longueur courte et qui ne respect pas bien la distribution des lettres de la langue anglaise.
Voici le texte en clair : ZJXQKB

Tous les tests ont été effectués avec des clé de longueur 1 à 4, et aussi de longueur 5 dans le cas de la deuxième instance. Ils ont aussi été faits avec l'heuristique par défaut du solveur (domOverWDeg) ainsi que notre propre heuristique.

5 Résultats [4]

Voici les résultats sans l'heuristique que nous avons créée :

Instances	**Résultats**			
	Longueur clé	**Retour arrière**	**Temps de réponse**	**Solutions (trouvées/retournées)**
1	1	51	0.038 seconde	15/6
	2	1 361	0.432 seconde	163/36
	3	32,259	7.134 secondes	1 139/216
	4	648,563	105.211 secondes	10 316/1 296
2	1	55	0.007 seconde	19/6
	2	1369	0.306 seconde	172/36
	3	68 345	1.944 seconde	1 576/216
	4	6 607 263	114.837 secondes	11 646/1 296
	5	1 611 566 903	6h 49m 47s	292/10
3	1	51	0.047 seconde	15/6
	2	1 345	0.535 seconde	155/36
	3	32 091	8.419 secondes	1 372/216
	4	648 031	134.406 secondes	10 309/1 296
4	1	63	0.034 seconde	15/6
	2	4 297	0.251 seconde	166/36
	3	263 683	3.301 secondes	1 497/216
	4	3 097 089	25.853 secondes	13 062/1 296

Voici les résultats avec l'heuristique que nous avons créée :

Instances	**Résultats**			
	Longueur clé	**Retour arrière**	**Temps de réponse**	**Solutions (trouvées/retournées)**
1	1	51	0.041 seconde	10/6
	2	1 351	0.502 seconde	69/36
	3	35 151	9.579 secondes	542/216
	4	913 951	187.582 secondes	3 947/1 296
2	1	51	0.029 seconde	8/6
	2	1 351	0.073 seconde	98/36
	3	35 151	1.266 seconde	663/216
	4	913 951	26.016 secondes	5 920/1 296
	5	23 575 363	548,744 secondes	44 471/7 776
3	1	51	0.173 seconde	12/6
	2	1 351	0.585 seconde	67/36
	3	35 151	11.192 secondes	499/216
	4	913 951	209.476 secondes	5 470/1 296
4	1	51	0.042 seconde	10/6
	2	1 351	0.021 seconde	102/36
	3	35 151	0.437 seconde	774/216
	4	240 687	2.853 secondes	7 105/1 296

4. Nous avons effectué nos tests sur une machine équippée d'un processeur i7-4770K et de 16 Go de mémoire.

Il est important de noter que le test de l'instance 2 avec une clé de 5 lettres a été modifié pour ne retourner que les 10 meilleures solutions lorsque nous l'avons exécuté sans notre heuristique, en raison du temps d'exécution très élevé. C'est indiqué en rouge dans les tableaux ci-dessus et en pointillés dans la figure 1 ci-dessous.

Il est aussi important de noter que, pour tous ces tests, la fréquence relative des lettres est assez près de la fréquence relative typique de l'anglais pour que le solveur trouve la bonne solution parmi celles qui sont retournées, à l'exception des tests de l'instance 4 et du test mentionné précédemment pour lequel qu'on a spécifié manuellement de ne retourner que les 10 meilleures solutions.

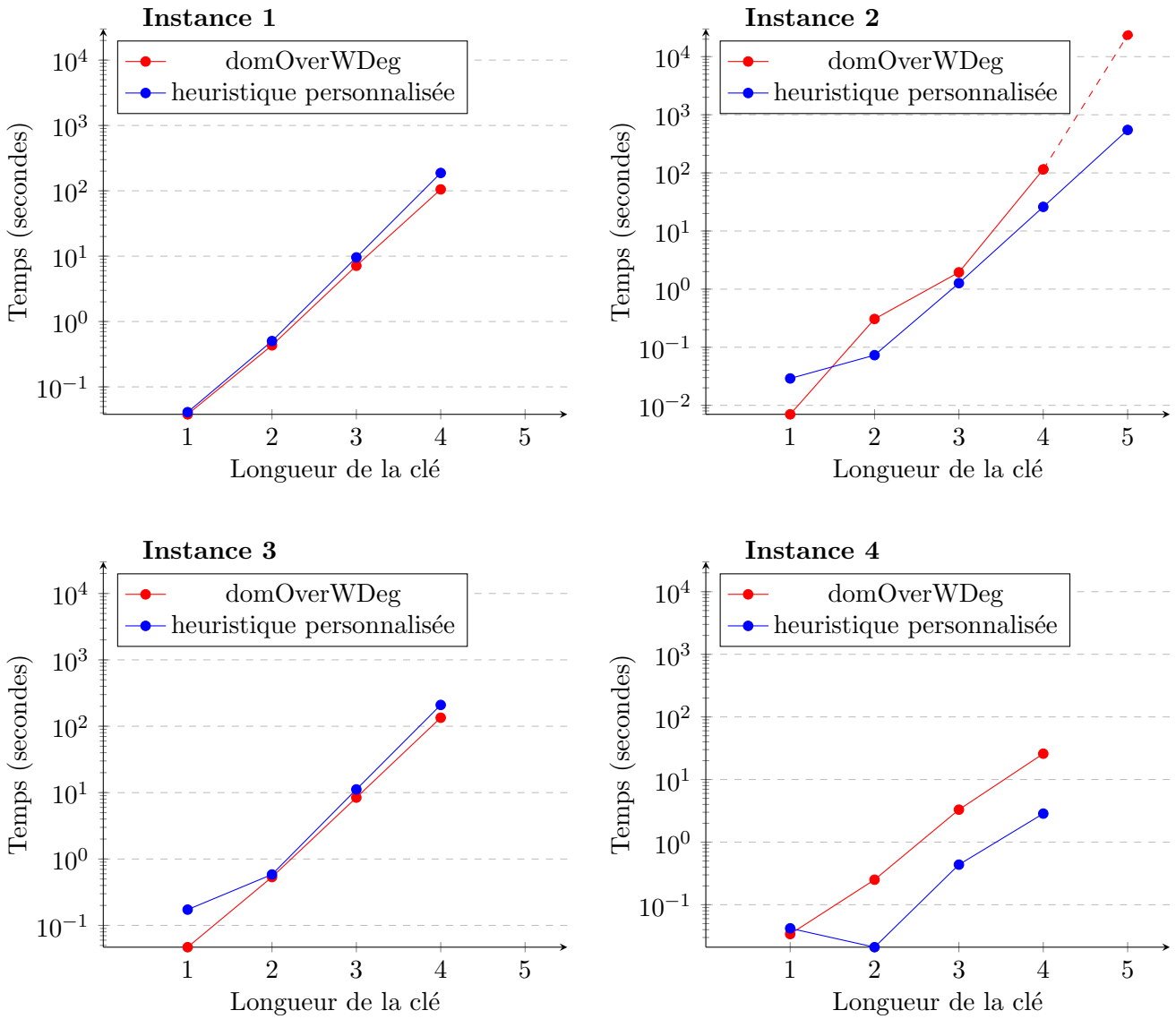

FIGURE 1 – Différence entre l'heuristique par défaut (domOverWDeg) et notre heuristique personnalisée

6 Discussion

En observant les figures ci-dessus, on peut voir que notre modèle fonctionne bien pour des textes chiffrés avec une clé inférieure ou égale à 5. Par contre, le temps de résolution augmente de façon exponentielle selon la longueur de la clé, ce qui fait que c'est moins pratique pour des clés plus longues.

On peut aussi voir que la même heuristique n'est pas la meilleure pour toutes les instances.

L'heuristique que nous avons créée nous a apporté un impressionnant gain en temps par rapport à l'heuristique par défaut sur notre deuxième instance de test, passant de près de 7 heures pour trouver les 10 meilleures solutions à seulement 9 minutes pour trouver les 1296 meilleures. Ça s'explique par le fait que les lettres les plus fréquentes en anglais se répètent souvent dans cette instance, ce qui est idéal pour notre heuristique qui essaie les lettres les plus courantes d'abord.

Dans d'autre cas, l'heuristique par défaut est plus rapide, bien que la différence soit moins grande que dans le cas précédent.

On peut toutefois remarquer que dans tous les cas, notre heuristique trouve moins de solutions au total, c'est-à-dire qu'elle trouve les meilleures plus rapidement. Ça laisse croire qu'il y aurait moyen de l'optimiser, ce qu'il serait intéressant d'explorer en guise de travail futur. Il serait aussi intéressant d'essayer plusieurs heuristiques en parallèle, pour pouvoir bénéficier de l'heuristique la mieux adaptée à chaque problème.

De plus, cela nous a permis de trouver des limitations telles que lorsque le message est plus court. En effet, puisque notre modèle choisit la bonne clé en fonction de la fréquence, il est plus facile d'avoir un bon résultat avec une longue chaîne. On peut aussi voir une limitation si le message chiffré n'est pas dans un langage formel. Par exemple, si ce sont des dérivations de mots tels que *coooooool* plutôt que *cool*.

7 Conclusion

Dans le cadre de ce travail, nous avons attaqué la cryptographie classique par substitution à l'aide de la programmation par contrainte. Pour faire cela, nous avons fait un modèle que nous avons implémenté à l'aide de *Choco*. L'expérimentation a été un succès et nous avons résolu des problèmes simples de cryptographie. Par contre, nous avons vite réalisé que la programmation par contraintes ne serait pas bien adaptée pour régler des problèmes plus complexes telle que la cryptographie moderne.

Il aurait été intéressant de tester différents types de cryptographie pour pouvoir comparer l'efficacité de ceux-ci.