N° d'ordre : 06/2004-M/IN

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie *Houari Boumédiene*
USTHB – ALGER

Faculté d'Electronique et d'Informatique
Département d'Informatique

MEMOIRE

Présenté pour l'obtention du diplôme de

Magister en Informatique

Spécialité : Intelligence Artificielle et Bases de Données Avancées

Par

M^r^ Mohamed Amine GARICI

Sujet

Adaptation de la Recherche Dispersée pour la Cryptanalyse des Chiffres Simples

Soutenu le 14-09-2004, devant le jury composé de :

Mr	A. AISSANI	Professeur	USTHB	Président
Mme	H. DRIAS	Professeur	USTHB	Directeur de thèse
Mr	H. HAMITI	Docteur	DGSCT	Co-Directeur de thèse
Mr	S. LARABI	Maître de Conférence	USTHB	Examinateur
Mme	F. YOUCEF ETTOUMI	Maître de Conférence	USTHB	Examinateur

Remerciements

- **Je tiens tout d'abord à exprimer ma grande gratitude à M^me^ Drias pour l'honneur qu'elle m'a fait en dirigeant ce travail.**
- **Mes plus vifs remerciements vont à M^r^ Hamiti pour ses conseils et aussi pour les discussions fructueuses qu'il a entretenu avec moi.**
- **Je remercie vivement M^r^ Aissani pour avoir accepté de présider le jury, et également M^r^ Larabi et M^me^ Youcef Ettoumi pour avoir fait parti de ce jury.**
- **Mes remerciements vont également à tous ceux qui m'ont aidé à réaliser ce travail.**

Sommaire

Chapitre III : ***La Recherche Dispersée***

Chapitre IV : ***Cryptanalyse des chiffres simples***

Introduction générale

Assurer la sécurité des communications dans de larges réseaux d'ordinateurs tel Internet est une des limites de l'informatique actuellement. La cryptographie tente de garantir la sécurité et la confidentialité des échanges d'informations, qui sont devenues une nécessité cruciale avec l'avènement des réseaux informatiques internationaux. A l'opposé de la cryptographie, nous trouvons la cryptanalyse, qui tend à percer les barrières de sécurité élevés par la cryptographie. La cryptanalyse est le processus de retrouver le texte en clair et/ou la clef d'un texte crypté.

Les premiers schémas de cryptographie, appelés maintenant les Chiffres Simples, étaient conçus pour exiger le moins de calculs possibles. Ces systèmes de chiffrement furent déjà utilisés des centaines d'années avant notre ère. Bien qu'ils soient totalement obsolètes et qu'ils n'apportent aucune garantie au niveau sécurité, ceci ne signifie pas qu'ils ont perdu toute importance dans le domaine de la cryptologie. L'intérêt pour de tels systèmes vient du fait que la plupart des cryptosystèmes modernes adoptent la substitution et la transposition comme base.

La tendance actuelle dans le domaine de la cryptologie est d'utiliser les techniques de recherche issues du domaine de l'intelligence artificielle et de l'optimisation combinatoire pour casser des cryptosystèmes existants, aussi bien que pour élaborer de nouveaux systèmes cryptographiques.

Notre objectif dans cette thèse est d'explorer la contribution pour la cryptanalyse des chiffres simples d'une méta-heuristique récente et souvent méconnue : la *'Recherche Dispersée'* (*Scatter Search*). Nous tentons d'illustrer la faisabilité de l'utilisation de cette méta-heuristique en tant qu'un outil de cryptanalyse. Cette approche évolutive basée population fut proposée récemment telle une méta-heuristique pour la résolution de problèmes d'optimisation combinatoire. Elle est basée sur une approche pour la

programmation discrète et utilise des combinaisons linéaires de parties de la population pour créer de nouvelles solutions. Elle fut appliquée récemment à de nombreux problèmes d'optimisation combinatoire, par exemple, des problèmes d'optimisation dans des graphes, des problèmes de routage, ou bien le problème de satisfiabilité SAT.

Dans ce travail, nous avons implémenté les méthodes de la recherche dispersée pour la cryptanalyse des chiffres simples ; ensuite, nous avons effectué des expérimentations pour fixer les paramètres des procédures, et aussi pour valider la méthode de la cryptanalyse.

Cette thèse est structurée comme suit :

- Dans le premier chapitre nous avons présenté le domaine de la cryptologie : définir la cryptographie et la cryptanalyse, et définir formellement les propriétés d'un système cryptographique, et présenter en détails les systèmes de chiffrement simple.
- D'un autre coté, dans le second chapitre, nous avons effectué un bref survol des principales notions de l'optimisation combinatoire et de la complexité, en détaillant les principaux algorithmes de résolution, et les méta-heuristiques les plus connues.
- Le troisième chapitre est une présentation détaillée de la méta-heuristique en question : la '*Recherche Dispersée*'. Ses principales méthodes sont présentées en détail et illustrées avec un exemple d'application.
- Le quatrième chapitre comporte un schéma de cryptanalyse des systèmes de chiffrement simples utilisant la Recherche Dispersée. Nous avons commencé par formaliser le problème de cryptanalyse sous la forme d'un problème d'optimisation, ensuite nous avons appliqué la méta-heuristique pour le résoudre, en détaillant les implémentations des méthodes.
- Dans le dernier chapitre sont présentées les expérimentations effectuées pour fixer les paramètres de l'algorithme, et aussi pour valider l'algorithme de cryptanalyse ; ainsi que les évaluations des performances de la recherche dispersée comparées à celles d'un algorithme génétique.

Chapitre I : *Cryptologie et Chiffres simples*

I- Introduction :

L'être humain a toujours ressenti le besoin de dissimuler des informations et à sécuriser leurs échanges. Ce besoin est devenu une nécessité cruciale avec l'avènement des réseaux informatiques internationaux ; le problème de la sécurité, lors de l'échange de données, est plus que jamais au goût du jour. La cryptographie, vieille de plus de 3000 ans, essaie d'apporter des solutions à ce problème. La cryptographie est l'art et la science de garder secret des messages. Après s'être longtemps résumée à un jeu sans fondements théoriques profonds entre d'ingénieux concepteurs de codes secrets développant des techniques de chiffrement expérimentales, elle s'est transformée à l'aube du XXI^{ème} siècle en une science à part entière. KERCHOFFS [43] posa les principes de la cryptographie moderne. L'un des principes postule que la sécurité d'un système de chiffrement ne doit résider que dans la clef et non dans le procédé de chiffrement. La cryptographie est devenue une science dynamique à l'intersection de nombreuses autres que sont les mathématiques, l'informatique et la micro-électronique.

A l'opposé de la cryptographie, la cryptanalyse tend à percer ces secrets, et à décrypter des messages chiffrés interceptés. La science qui regroupe les disciplines de la cryptographie et de la cryptanalyse est appelée cryptologie. Cette science trouve aujourd'hui des applications très nombreuses dans des domaines variés, elle doit non seulement garantir la confidentialité et l'intégrité des messages, mais aussi l'authentification dans un environnement éliminant toute présence physique (commerce électronique, paiement sécurisé, signature électronique).

II- Principes généraux :

La cryptographie consiste à transformer les caractères qui composent un message en une autre succession de caractères de telle façon à le rendre incompréhensible en utilisant une *clef de chiffrement.* Le résultat de cette modification (le message chiffré) est appelé *cryptogramme* ou *message chiffré* (*ciphertext*) par opposition au message initial, appelé *message en clair* (*plaintext*). Le déchiffrement nécessite quant à lui une *clef de déchiffrement.*

La cryptographie sert non seulement à préserver la confidentialité des données mais aussi à garantir leur intégrité et leur authenticité. Elle doit répondre essentiellement à quatre besoins :

- **La confidentialité :** consiste à rendre l'information inintelligible à d'autres personnes que les acteurs de la transaction.
- **L'intégrité :** Vérifier l'intégrité des données consiste à déterminer si les données n'ont pas été altérées durant la communication (de manière fortuite ou intentionnelle).
- **L'authentification :** L'authentification consiste à assurer l'identité d'un utilisateur, c'est-à-dire de garantir à chacun des correspondants que son partenaire est bien celui qu'il croit être. Un contrôle d'accès peut permettre (par un mot de passe) l'accès à des ressources uniquement aux personnes autorisées.
- **La non-répudiation :** La non-répudiation de l'information est la garantie qu'aucun des correspondants ne pourra nier la transaction.

III- Systèmes cryptographique :

Un algorithme cryptographique est une transformation d'une séquence (de caractères ou de bits) vers une autre séquence. Le texte en clair, le texte chiffré et la clef de chiffrement sont des vecteurs notés respectivement P, C et K :

$$E_K(P) = C$$

Nous avons une autre transformation (déchiffrement), qui retourne P à partir de C en utilisant une autre séquence K^{-1}, qui est en relation avec K :

$$D_{K^{-1}}(C) = P$$

ce couple de fonctions (E, D) doit toujours vérifier l'identité suivante pour tout texte P : $D(E(P)) = P$.

D'une façon plus formelle, un cryptosystème peut être défini comme un 5-uplet $(\mathbb{P}, \mathbb{C}, \mathbb{K}, \mathbb{E}, \mathbb{D})$ où :

- $\mathbb{P}$ est l'ensemble des textes clairs possibles.
- $\mathbb{C}$ est l'ensemble des cryptogrammes possibles.
- $\mathbb{K}$ est l'ensemble des clefs possibles.
- $\mathbb{E}$ et $\mathbb{D}$ sont respectivement l'ensemble des règles possibles de chiffrement, et l'ensemble des règles de chiffrement.
- Pour toute clef K de $\mathbb{K}$, il existe une règle de cryptage E_K $(E_K : \mathbb{P} \to \mathbb{C})$ de $\mathbb{E}$ et une règle de décryptage correspondante D_K $(D_K : \mathbb{C} \to \mathbb{P})$ de $\mathbb{D}$ telles que pour tout x de $\mathbb{P}$: $D_K(E_K(x)) = x$. La connaissance de K permet d'expliciter parfaitement les fonctions E_K et D_K.

III.1. Propriétés d'un cryptosystème :

Certaines propriétés d'un algorithme cryptographique sont très intéressantes à analyser ; mais en contre partie cette analyse est très difficile à effectuer. Parmi ces propriétés nous citons [50] :

- La propriété la plus importante d'un chiffre est qu'il est difficile de retrouver P à partir de C sans la connaissance de K. Le terme *'difficile'* ici ne signifie pas que le problème n'est pas décidable, mais qu'il est d'une très grande complexité.
- Si les clefs de chiffrement et de déchiffrement sont égales $\left(K = K^{-1}\right)$, alors le chiffre est symétrique.

- Certaines propriétés démontrent une faiblesse de l'algorithme si elles sont vérifiées, comme l'existence de clefs universelles : K est une clef universelle si on peut déchiffrer avec tout message chiffré avec une clef quelconque :

$$\forall P, \forall K' : \; P = \mathsf{D}_K\left(\mathsf{E}_{K'}(P)\right) \tag{I.1}$$

- Un autre signe de faiblesse d'un chiffre est l'existence de clefs faibles (*weak key*) : K est une clef faible si son application deux fois est équivalente à l'identité :

$$\forall P : \; P = \mathsf{E}_K\left(\mathsf{E}_K(P)\right) \tag{I.2}$$

- Un chiffre est fermé, si pour chaque texte en clair P et pour chaque deux clefs K^1 et K^2, on peut toujours trouver une troisième clef K telle que chiffrer le texte en clair avec la première clef puis avec la deuxième est parfaitement équivalent à le chiffrer avec la dernière clef.

$$\forall P, \forall K^1, \forall K^2, \exists K : \; \mathsf{E}_{K^2}\left(\mathsf{E}_{K^1}(P)\right) = \mathsf{E}_K(P)$$

  Si un chiffre est fermé, alors il est impossible d'améliorer sa sécurité en l'appliquant deux fois avec deux clefs différentes.

- Une autre combinaison intéressante de chiffrement avec différentes clefs est le schéma de triple chiffrement de TUCHMAN [67], qui consiste en un chiffrement du texte en clair avec une clef, puis le déchiffrement du résultat avec une seconde clef différente, puis de le rechiffrer encore le résultat avec une troisième clef. Si les clefs $\left(K^1, K^2, K^3\right)$ sont utilisées alors le résultat du chiffrement est :

$$\mathsf{E}_{K^3}\left(\mathsf{D}_{K^2}\left(\mathsf{E}_{K^1}(P)\right)\right)$$

  Pour que ce schéma de chiffrement permette d'améliorer la sécurité, il faut que pour chaque triplet de clefs $\left(K^1, K^2, K^3\right)$ et pour chaque texte en clair P, il n'existe aucune clef K telle que le triple chiffrement avec les clefs $\left(K^1, K^2, K^3\right)$ soit équivalent avec le chiffrement avec K :

$$\forall P, \forall K^1, \forall K^2, \forall K^3, \forall K : \; \mathsf{E}_{K^3}\left(\mathsf{D}_{K^2}\left(\mathsf{E}_{K^1}(P)\right)\right) \neq \mathsf{E}_K(P)$$

- Une dernière propriété, un chiffre est sain (*faithful*) si des clefs différentes ne peuvent générer le même texte chiffré à partir du même texte en clair : $\forall P, \forall K^1, \forall K^2 : \left(\mathsf{E}_{K^1}(P) = \mathsf{E}_{K^2}(P) \right) \Rightarrow \left(K^1 = K^2 \right)$.

III.2. Les classes de cryptosystèmes :

La première technique utilisée en cryptographie est la substitution, qui consiste à remplacer chaque lettre du texte en clair par une autre de l'alphabet (substitution monoalphabétique) ou bien un groupe de caractère dans le message par un autre groupe (substitution de polygrammes). Une autre technique utilisée est la transposition, qui consiste à réordonner géométriquement les caractères formant le texte en clair, ces systèmes, communément appelés '*chiffres classiques*', seront repris en détail ultérieurement.

Parmi les cryptosystèmes modernes, on distingue classiquement deux grands paradigmes. Le premier appelé chiffrement conventionnel, ou chiffrement symétrique, fait l'hypothèse d'une clef secrète, partagée par les correspondants.

La seconde classe comporte les systèmes asymétriques ou à clef publique. Son principe fondamental est de donner à chaque utilisateur deux clefs associées, l'une secrète et l'autre rendue publique telles qu'il est très difficile -en terme de complexité- d'en déduire la clef privée à partir de la clef publique.

III.2.1. Cryptosystèmes symétriques :

Les systèmes de chiffrement symétrique (aussi appelé *systèmes de chiffrement à clef privée* ou *à clef secrète*) consistent à utiliser la même clef k pour le chiffrement que pour le déchiffrement, c'est à dire : $\mathsf{D}_k\left(\mathsf{E}_k(P)\right) = P$

Le chiffrement consiste alors à appliquer un opérateur entre la clef privée et les données à chiffrer afin de rendre ces dernières inintelligibles. Ainsi, le

moindre algorithme (tel qu'un OU exclusif) peut rendre le système sûr. Toutefois, SHANNON [62] démontra que pour avoir un cryptosystème symétrique à confidentialité parfaite, il faut que le nombre de clefs possibles soit aussi grand que le nombre de messages possibles. En d'autres termes, la clef doit être au moins aussi longue que le message lui-même et aucune clef ne peut être réutilisée (masque jetable), ce qui dégrade sérieusement l'intérêt d'un tel système de chiffrement.

Il existe plusieurs algorithmes de chiffrement symétriques, tels *Lucifer*, *IDEA*, et le *DES*, le plus utilisé jusqu'à maintenant. Le point fort de ces systèmes, est leur rapidité et la possibilité de les implémenter sur des circuits électroniques. En contre partie, le principal inconvénient d'un cryptosystème à clefs secrètes provient de l'échange des clefs. En effet, le chiffrement symétrique repose sur la confidentialité de la clef, ce qui impose l'existence d'un canal de communication sécurisé pour l'échanger. Encore, puisque chaque clef ne peut être utilisée qu'entre un seul couple d'utilisateurs, ceci pose le problème de la distribution des clefs dont le nombre devient trop grand. En effet, Pour un groupe de n personnes utilisant un cryptosystème à clefs secrètes, il est nécessaire de distribuer $\frac{n(n-1)}{2}$ clefs.

III.2.2. Cryptosystèmes asymétriques :

Le paradigme de chiffrement à clef publique fut présenté en 1976 par DIFFIE et HELLMAN [18]. Dans un cryptosystème asymétrique, les clefs existent par paires (*bi-clefs*) :

- Une clef publique pour le chiffrement.
- Une clef privée (secrète) pour le déchiffrement.

Ainsi, dans un tel cryptosystème, les utilisateurs choisissent une clef dont ils sont seuls connaisseurs (il s'agit de la *clef privée*). A partir de cette clé, ils déduisent chacun automatiquement la clef publique. Les utilisateurs s'échangent cette clef publique à travers un canal non sécurisé. Cette paire de clef doit être telle qu'il est très difficile de déduire la partie privée de la partie publique.

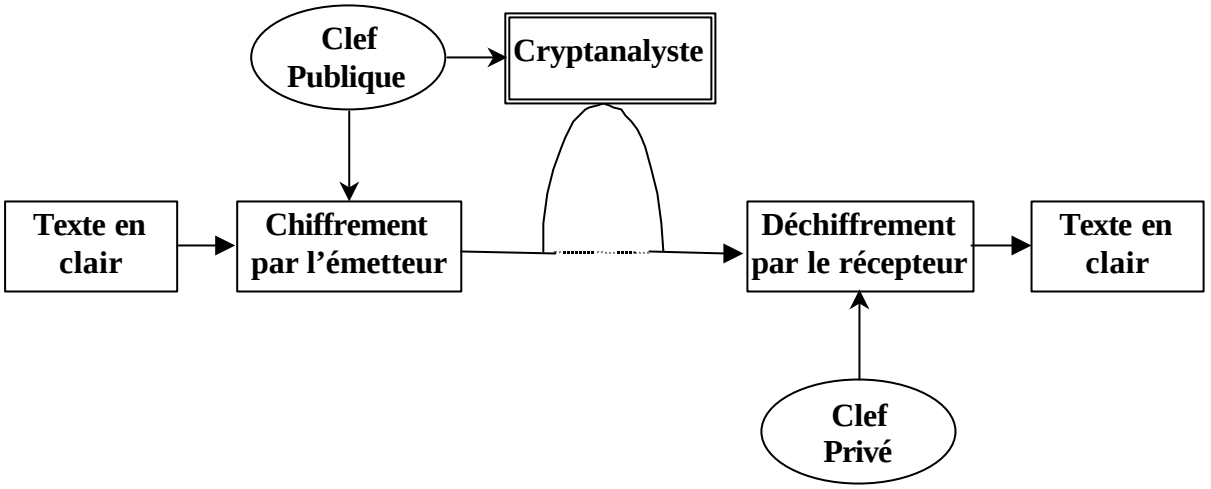

Fig I.1 : Schéma général d'un cryptosystème à clef publique

Lorsqu'un utilisateur désire envoyer un message à un autre utilisateur, il lui suffit de chiffrer le message à envoyer au moyen de la clef publique du destinataire (qu'il trouvera par exemple dans un serveur de clefs ou un annuaire). Ce dernier sera en mesure de déchiffrer le message à l'aide de sa clef privée (qu'il est seul à connaître).

Ce système est basé sur une fonction facile à calculer dans un sens appelée *fonction à trappe à sens unique* (*one-way trapdoor function*), mais qui est mathématiquement très difficile à inverser sans la clef privée (*trappe*). Le terme 'difficile' signifie que pour inverser cette fonction, aucun algorithme n'est assez rapide et économique pour valoir la peine d'être appliqué ; ceci implique que le problème du calcul de la fonction inverse sans la trappe doit avoir une complexité non polynomiale.

Cette fonction doit être inversible à gauche, c'est-à-dire, pour toute clef K_{pub}, il existe une fonction g paramétrée par la clef K_{pri}, telle que pour tout message P on a: $g_{K_{pri}}\left(f_{K_{pub}}(P)\right)=P$. Et si en plus de ça, la fonction f est inversible à droite, c'est-à-dire $f_{K_{pub}}\left(g_{K_{pri}}(P)\right)=P$ alors cette fonction permet la procédure de la signature à clef publique. Pour vérifier l'authenticité d'un message, l'expéditeur signe un message avec sa clef privé, et le récepteur applique la clef publique pour vérifier si le message est bien envoyé par l'expéditeur désiré, le seul détenteur de la clef privée.

Plusieurs cryptosystèmes à clef publique existent, tel le chiffre de MERKLE et HELLMAN [53] (basé sur le problème du Sac à dos); et le RSA [59], un des cryptosystèmes les plus utilisés actuellement. Dans de tels systèmes, le problème consistant à se communiquer la clef de déchiffrement n'existe plus, les clefs publiques pouvant être envoyées librement. Le chiffrement par clefs publiques permet donc à des personnes de s'échanger des messages cryptés sans pour autant posséder de secret en commun. Mais d'un autre coté, ce genre de système sont souvent très gourmand en matière de temps de calculs par rapport aux système symétriques. Pour cela, des systèmes hybrides combinant deux systèmes de chiffrement ont vu le jour, tel le PGP [68]. Le PGP utilise un cryptosystème à clef secrète pour la communication, à chaque session une nouvelle clef de session est crée aléatoirement et échangée à l'aide d'un algorithme à clef publique. Cette méthode associe la sécurité d'utilisation du cryptage à clef publique à la vitesse du cryptage conventionnel.

III.3. Différents types d'attaques contre un cryptosystème :

Un des axiomes de base de la cryptologie, énoncé par KERCKHOFFS [43] au XIX^{ème} siècle, est que l'ennemi (le cryptanalyste) connaît l'algorithme dans ses moindres détails et qu'il ne lui manque que la clé pour déchiffrer le message. Cette situation, même si elle semble extrême, n'est pourtant pas loin de la réalité. Les principaux types d'attaques contre cryptosystème sont élaborées en supposant que l'algorithme de cryptage est connu.

- Attaque à texte chiffré : Le cryptanalyste ne dispose que d'un ou plusieurs messages chiffrés assez longs. Ce type d'attaque, qui est le plus difficile, permet pourtant parfois de déchiffrer les messages. En particulier, les substitutions alphabétiques sont très vulnérables à ce genre d'attaque : des études statistiques du ou des messages permettent facilement de retrouver le message initial.

- Attaque à texte clair - texte chiffré : Le cryptanalyste dispose d'un ou plusieurs textes en clair avec leurs équivalents chiffrés. Certaines structures sont caractéristiques des documents et peuvent plus ou moins facilement être analysées (les en-têtes types, mot clés…).

- Attaque à texte clair choisi : Le cryptanalyste dispose du cryptosystème prêt à fonctionner mais sans connaître la clef de déchiffrement : un cas courant dans les systèmes à clef publique. Dans cette situation, le cryptanalyste a le choix du texte à chiffrer.

Selon les données disponibles au cryptanalyse, plusieurs schémas de cryptanalyse peuvent être utilisés :

- Cryptanalyse exhaustive : son principe est le plus simple : le cryptanalyste dispose du cryptosystème et d'un texte chiffré et essaye toutes les clefs possibles. Cette attaque n'est pourtant pas la plus utilisée car elle est aussi la plus longue, c'est pourquoi elle sert de base de comparaison pour estimer la sécurité d'un cryptosystème.

- Analyse fréquentielle : une technique très efficace utilisée contre les substitutions monoalphabétiques ou bien de polygrammes. En effet, certains caractères (comme la lettre '*e*') ou polygrammes ('*th*' en anglais ou bien '*le*' en français) apparaissent plus fréquemment que d'autres. Il suffit pour le cryptanalyse d'effectuer une étude statistique des cryptogrammes et analyser les fréquences d'apparition des caractères et des polygrammes pour casser le cryptosystèmes. Certaines méthodes de chiffrement peuvent devenir équivalentes à une substitution et par conséquence devenir vulnérables à une analyse fréquencielle si la taille des blocs à crypte est égale à la taille d'un caractère.

- Cryptanalyse différentielle : La notion de cryptanalyse différentielle fut introduite par BIHAM et SHAMIR en 1990 [6]. En utilisant cette méthode, ils ont pu développer une attaque à texte clair choisi contre le DES plus efficace que la recherche exhaustive et toutes les méthodes existantes. La cryptanalyse différentielle examine des paires de textes chiffrés dont les textes en clair présentent certaines différences particulières. Certaines différences dans les textes en clair, ont de fortes chances de réapparaître dans les textes chiffrés. Cette méthodes utilise ces caractéristiques pour assigner des probabilités aux clefs possibles, et déterminer avec exactitude la plupart des bits formant la clefs, et par la suite trouver la clef la plus probable. Cette technique peut être appliquée à tout les cryptosystèmes ayant un principe de fonctionnement similaire à celui du DES.

IV- Chiffres simples :

Plusieurs variantes des cryptosystèmes de substitution et de transposition existent, les systèmes présentés ici sont les formes les plus générales de ces chiffres.

IV.1. Substitution monoalphabétique :

Le chiffrement par substitution en général consiste à remplacer dans un message une ou plusieurs entités (généralement des lettres) par d'autres entités. La substitution simple (aussi appelée substitution monoalphabétique pour la différencier de la substitution polyalphabétique présentée dans la section suivante) est la forme la plus simple des systèmes de chiffrement par substitution. Chaque symbole dans le texte en clair est remplacé par un symbole (généralement différent) dans le texte chiffré. La clef d'un chiffre de substitution peut être représentée par une permutation de l'alphabet du texte en clair.

Les processus de chiffrement et de déchiffrement seront mieux décrit avec un exemple.

Exemple I.1 :

Clef :

Texte en clair :	ABCDEFGHIJKLMNOPQRSTUVWXYZ_
Texte crypté :	VYTLAKGMZXNWIEUOP_BQCRSJDHF

Texte crypté :	ABCDEFGHIJKLMNOPQRSTUVWXYZ_
Texte en clair :	ESUYN_GZMXFDHKPQTVWCOALJBIR

Chiffrement

Texte en clair :	UN_EXEMPLE_DE_SIMPLE_SUBSTITUTION
Texte crypté :	CEFAJAIOWAFLAFBZIOWAFBCYBQZQCQZUE

Le texte original peut être retrouvé en inversant la procédure de chiffrement. Un caractère du texte crypté se trouvant à la position *i* dans la clef est décrypté vers le $i^{ème}$ caractère dans l'alphabet du texte en clair.

D'une manière formelle, la transformation associée à une chiffrement par substitution simple peut être définie comme suit :

$$f : \mathbb{A}^+ \times \mathbb{K} \to \mathbb{A}^+$$
$$\left(P = (p_1, \ldots, p_m),\ K = (k_1, \ldots, k_n)\right) \longrightarrow C = (c_1, \ldots, c_m)$$
$$\text{où } c_i = k_j \setminus p_i = a_j$$

où : $\mathbb{A} = (a_1, \ldots, a_n)$ est l'alphabet du langage.

$\mathbb{K}$ est l'ensemble de clefs possibles, où chaque clef $K = (k_1, \ldots, k_n)$ est une permutation des caractères formants l'alphabet.

Pour un alphabet de 27 caractères, il existe $27! \left(\approx 1{,}1 \times 10^{28} \approx 2^{93}\right)$ clefs possibles pour une substitution simple. Ce nombre est assez grand pour rendre toute attaque basée sur la force brute quasiment inefficace. Cependant, à cause de certaines propriétés de la substitution, sa cryptanalyse est rendue relativement aisée.

La principale propriété de la substitution simple est la conservation des statistiques et des caractéristiques linguistiques du texte en clair. Les fréquences des *n*-grammes restent inchangées après le processus de chiffrement (un *n*-gramme est une suite de *n* caractères). Dans l'exemple précédant, le bigramme (2-gramme) le plus fréquent dans le message en clair **'E_'** devient **'AF'** à chaque fois qu'il est crypté (3 fois). Donc, pour chaque suite de caractères dans le texte en clair, il existe une suite correspondante unique de caractère dans le texte chiffré. Dans un texte crypté de longueur relativement grande, les *n*-grammes les plus fréquent correspondaient aux *n*-grammes les plus fréquent dans la langue de ce texte. Ce fait est utilisé comme base pour les attaques contre les substitutions simples.

L'exemple le plus courant d'une substitution simple, est le fameux chiffre de *César*, dont la clef consiste en une rotation de l'alphabet du texte en clair par *i* positions.

Une variante de la substitution simple est la Substitution des polygrammes. La substitution de polygrammes consiste à substituer un groupe de caractères (polygramme) dans le texte en clair par un autre groupe de caractères. Dans cas, l'unité de chiffrement n'est plus un seul caractère, mais plusieurs caractères. Des bigrammes ou des trigrammes (ou d'autres configurations) sont échangés par d'autres bigrammes ou trigrammes. Par exemple, remplacer 'ABA' par 'RTQ'. Généralement la clef dans ce genre de substitution ne peut être représentée directement par une permutation comme c'est le cas dans la substitution simple, parce que la clef aura une taille de n^2 où n est la taille de l'alphabet.

IV.2. Substitution polyalphabétique :

La substitution polyalphabétique est une extension de la substitution alphabétique. La différence est que le texte à crypter est divisé en plusieurs blocs de même taille B, dite *Période.* Ensuite, *B* substitutions simples sont utilisées. Chaque caractère dans un bloc (1,…,*B*) est crypté (ou décrypté) en utilisant une substitution simple, mais en changeant la clef à chaque fois. Donc, la substitution polyalphabétique est une combinaison de plusieurs substitutions simples.

D'une manière formelle, la transformation associée à une chiffrement par substitution polyalphabétique de période *B* peut être définie comme suit :

$$f : \mathbb{A}^+ \times \mathbb{K}^B \to \mathbb{A}^+$$

$$\left(P = (p_1,\ldots,p_m), \left(K^1 = \left(k_1^1,\ldots,k_n^1\right),\ldots,K^B = \left(k_1^B,\ldots,k_n^B\right)\right)\right) \longrightarrow C = (c_1,\ldots,c_m)$$

$$\text{où } c_i = k_j^{(i \bmod B)} \setminus p_i = a_j$$

Le nombre de clefs possibles pour une substitution polyalphabétique utilisant un alphabet de 27 caractères et une période *B* est $(27\,!)^B$ clefs. Ce nombre est largement grand par rapport à la substitution monoalphabétique,

spécialement pour une grande période. Le chiffrement par substitution polyalphabétique est donc plus difficile que le chiffrement par substitution simple, Mais il reste un chiffre relativement simple à casser en se basant toujours sur les statistiques des *n*-grammes dans le langage du texte en clair.

Exemple I.2 :

Clef :

Texte en clair :	ABCDEFGHIJKLMNOPQRSTUVWXYZ_
Texte crypté :	VYTLAKGMZXNWIEUOP_BQCRSJDHF
	WMKQRGFVHDYTCJ_NIBXZAELOPSU
	ZQSCEFBTHUMKO_PLIJYNGRVDWXA

Chiffrement

Position :	12312312312312312312312312312312 3
Texte en clair :	UN_EXEMPLE_DE_SIMPLE_SUBSTITUTION
Texte crypté :	CJAAOEINKAUCAUYZCLWRABAQBZHQANZ__

Dans l'exemple I.1 qui décrit la substitution monoalphabétique, le bigramme le plus fréquent 'E_' est toujours crypté vers le bigramme 'AF'. Ce n'est pas le cas pour la substitution polyalphabétique. Dans l'exemple I.2, le même bigramme est crypté deux fois vers 'AU' et une fois vers 'RA'. La valeur cryptée dépend de la valeur de chaque clef et de la position du caractère dans le bloc.

Plusieurs types de chiffrement par substitution polyalphabétique existent, comme le chiffre de BEAUFORT et le chiffre de VIGENERE qui utilise un bloc de 26 caractères.

Dans une attaque contre une substitution polyalphabétique, la détermination de la période est effectuée en premier, ensuite le schéma de cryptanalyse est le même que pour les substitutions simples.

IV.2.1. Détermination de la période :

Il existe plusieurs méthodes pour déterminer la période d'un chiffrement par substitution polyalphabétique. Deux méthodes sont citées ci-dessous.

IV.2.1.1. Calcul de distance :

Une première méthode consiste à rechercher dans le texte crypté de longues chaînes de caractères similaires. Il est très certain (mais pas sûr) que de telles chaînes proviennent des mêmes chaînes dans le texte en clair. Dans le cas où ces séquences répétées représenteraient le même texte en clair, alors la distance entre deux instances de la séquence donnera une indication sur la période du chiffre. La distance séparant chaque couple de chaînes similaires est un multiple de la période, et donc, le plus grand diviseur commun de ses distances a de grandes chances pour être la période du chiffre ou au pire des cas un multiple de cette période, et même dans ce cas, une recherche exhaustive sur les diviseurs de ce facteur aboutira rapidement à la détermination de la période.

Exemple I.3 :

CIYSBCNLNFSVYGFDDIIAIZLLWGMGJPVEJHPLEIBQIDLDAFIDIA
IYGIKPGCEYARSVSNPQIPAFSZGFLAUTPTXTNSEMHCTNZQWVEETL
GFWXJZQDUSRRGYLQGPJRWXSGUOCFBOKCJHNRRLJFSDWPFRXVEQ
TSYLAGOV**QTJURNNCA**A**JGGCPMEQ**IYRZMHNRGCMEKTLLVPSRGCCL
TBBPCFKARYCGXDZOCCLPSPFLRXBLEIYUBDAKFTLAZEQCOSWACC
IMEESEFWZWULJIBSNRPNWILGU**QTJURNNCA**VLJVAGRPGORPSUOJ
RRWXMNYSYNMXCZJTWTBNZRAKWKJRWXPRRTMVQUKKWDYSHCSPWX
WCKCSLEDUPLUMKKEMGWEETSYLBZGJYBXFRWVCQTWVKCGLBRUDC
VIYCZLIAEXEJUGZVOLQIAAIEMCWV**JGGCPMEQ**AYCPIUVESBUEEO
QRAUWUJMIAAGIYRZMKGTYHXSBFCCHMSVVBRKYCTZEJIEUVOJXN
GCRFC**NATJRAXTGEC**MEMXCZJRTRTJURG**NATJRAXTGEC**GKMNGEJN

Dans cet exemple, trois couple de chaînes similaires ont été repérés. Les distances entre chaque couple sont : 117 (QTJURNNCA), 260 (JGGCPMEQ) et 26 (NATJRAXTGEC). Le plus grand diviseur commun entre ces distances est 13, et donc ce diviseur est bien la période puisqu'il est premier.

IV.2.1.2. Le test de FRIEDMAN :

Le test de FRIEDMAN a pour premier objectif de déterminer si un texte a été chiffré avec un chiffre monoalphabétique ou polyalphabétique. Comme second bénéfice, il donne une estimation de la période du chiffre s'il est polyalphabétique. Pour réaliser ces buts, le test de FRIEDMAN s'appuie sur une métrique appelée Indice de Coï ncidence (*IC*), qui est la probabilité que deux lettres choisies aléatoirement dans un texte soient identiques [26].

En ayant un alphabet de n lettres et un texte chiffré $C = c_1 c_2 \dots c_L$ de longueur L, l'indice de coï ncidence (*IC*) se calcule comme suit :

1- Le nombre total des couples de caractères dans le texte chiffré est :

$$\binom{L}{2} = \binom{L!}{(L-2)!\,2!} = \frac{L(L-1)}{2}$$

2- Le nombre des couples contenant seulement le caractère c_i est :

$$\frac{F_i(F_i-1)}{2}$$

où F_i est le nombre d'occurrence du caractère c_i dans le texte chiffré, d'où :

$$\sum_{i=1}^{n} F_i = L$$

3- Le nombre total de couples identiques de caractères dans le texte chiffré est :

$$\sum_{i=1}^{n} \frac{F_i(F_i-1)}{2}$$

4- L'indice de coï ncidence mesure la probabilité que deux caractères choisis au hasard de la texte chiffré sont identiques :

$$IC = \frac{\text{Nombre de couples identiques}}{\text{Nombre total de couples possibles}}$$

$$IC = \frac{\sum_{i=1}^{n} \frac{F_i(F_i-1)}{2}}{\frac{L(L-1)}{2}}$$

et enfin :

$$IC = \frac{\sum_{i=1}^{n} F_i(F_i-1)}{L(L-1)}$$

Algorithme I.1 : Calcul de l'indice de coï ncidence

En calculant l'indice de coï ncidence d'un texte chiffré relativement long, on peut déterminer approximativement la période du chiffre. Pour chaque langue, une valeur de l'indice de coï ncidence caractérise le texte en clair. Aussi, pour chaque valeur de la période, l'indice de coï ncidence reste pratiquement inchangé même pour des clefs différentes.

Langue	IC
Arabe	0,085889
Français	0,074604
Anglais	0,066895
Allemand	0,076667
Danois	0,070731
Espagnol	0,076613
Finnois	0,073796
Grecque	0,069165
Italien	0,077236
Norvégien	0,069428
Portugais	0,074528
Russe	0,056074
Suédois	0,064489

Tableau I.1 : Valeurs de l'Indice de Coï ncidence pour différentes langues [64]

Période	IC (Arabe)	IC (Français)	IC (Anglais)
01	0,085889	0,074604	0,066895
02	0,060801	0,056302	0,052447
03	0,052439	0,050201	0,047631
04	0,048258	0,047151	0,045224
05	0,045749	0,045321	0,043779
06	0,044076	0,044100	0,042816
07	0,042882	0,043229	0,042128
08	0,041986	0,042575	0,041612
09	0,041289	0,042067	0,041210
10	0,040731	0,041660	0,040889
11	0,040275	0,041327	0,040627
12	0,039895	0,041050	0,040408
13	0,039573	0,040815	0,040222
14	0,039298	0,040614	0,040064
15	0,039059	0,040440	0,039926
16	0,038850	0,040287	0,039806
17	0,038665	0,040153	0,039699
18	0,038501	0,040033	0,039605
19	0,038355	0,039926	0,039521
20	0,038223	0,039830	0,039444

Tableau I.1 : Valeurs de l'Indice de Coï ncidence pour des textes chiffrés avec différentes périodes

Pour tout chiffre monoalphabétique, la distribution des fréquences est invariante, donc l'*IC* sera le même que pour le texte clair. Donc, si on applique ce test à un texte chiffré avec un chiffre monoalphabétique, on devrait trouver L'*IC* égal environ à 0,074 (en français). Si l'*IC* est beaucoup plus petit (par exemple 0,050), le chiffre est certainement polyalphabétique. On remarque que l'indice de coïncidence converge vers la valeur $0{,}03846 = \frac{1}{26}$ (sauf pour l'arabe), qui est la valeur de l'*IC* pour un texte dont la distribution des lettres est uniforme. Donc, il est très difficile de déterminer la période avec précision quand elle est grande ; pour cela, une autre méthode d'application de test de FRIEDMAN est utilisée. Pour chaque valeur de la période, des sous-chaînes sont considérées en prenant des caractères dans le texte chiffré a intervalle égal à cette période. L'indice de coïncidence des sous-chaînes correspondant à la véritable période serait très proche de l'indice de coïncidence du texte en clair.

Exemple I.4 :

En considérant le texte chiffré suivant :

```
PERTQUDCDJXESCWMPNLVMIQDIZTQFVXAKLRPICCPQSHZYDNCP
WEAJWSZGCLMQNRDEOHCGEZTQZYHELEWAUQFROICWHQMYRRUFG
BYQSEPVNEQCSEEQWEEAGDSZDCWEOHYDWQERLMFTCCQUNCPPQS
KPYFEQOIOHGPREERWIEFSDMXSYGEUELEHUSNLVGPMFVEIVXSU
SJPWHIEYSNLCDWMCRTZMICYXMNMFZQASLZQCJPYDSTTKZEPZR
ECMYWOICYGUESIUGIRCEUTYTIZTJPWHIEYIETYYHUSOFIXESC
WHOGDMZSNLVQSQPYJSCAVQSQLMQNRLPQSRLMXLCCGAMKPGQLY
```

Des chaînes seront sélectionnées en prenant des caractères à intervalle donné :

- Intervalle 1 : `PERTQUDCDJXESCWMPNL`… (le texte original).
- Intervalle 2 : `PRQDDXSWPL`…, `ETUCJECMNV`.
- Intervalle 3 : `PTDJSMLIIQ`…, `EQCXCPVQZF`…, et `RUDEWNMDTV`….
- …

Ensuite, l'indice de coï ncidence est calculé pour chacune de ces chaînes :

Intervalle	Indice de coïncidence (*IC*)	Moyenne
1	0,04561	0,04561
2	0,47695 ; 0,04430	0,26063
3	0,04944 ; 0,04424 ; 0,04491	0,04620
4	0,04658 ; 0,04252 ; 0,04658 ; 0,04491 ;	0,04515
5	0,08367 ; 0,07997 ; 0,08367 ; 0,06849 ; 0,07952;	0,07906
6	0,04071 ; 0,03715 ; 0,05129 ; 0,006612 ; 0,04315 ; 0,03858	0,03625

Du tableau ci-dessus, il est clair que quand l'intervalle est de 5, l'*IC* est très proche de l'*IC* caractéristique de la langue française, et c'est la seule ligne qui se rapproche le plus de cette valeur. Donc, la période du chiffre utilisée est certainement 5. Ceci sera vérifié en trouvant les substitutions formant la clef, et vérifié le texte obtenu.

L'inconvénient majeur du test de FRIEDMAN, est que les valeurs trouvées pour le texte chiffré se situent souvent aux milieux des intervalles formés par les valeurs du tableau, et il est très difficile de les départager ; aussi, en comptant avec les marges d'erreur, il faut vérifier tout le voisinage de la période trouvée.

Donc, il serait plus utile de combiner les deux méthodes citées ci-dessus, puisque toutes les deux fournissent des indications partielles sur la période de la substitution. L'indice de coï ncidence fournira l'intervalle où se situe la période, et la méthode des distances donnera un multiple de la période.

IV.3. Transposition :

Les méthodes de chiffrement par transposition consistent à réarranger les données à crypter de telle façon à les rendre incompréhensibles. Il s'agit généralement de réarranger géométriquement les données pour les rendre visuellement inexploitables.

Le chiffrement par transposition consiste à découper le texte en clair en blocs de taille fixe, dite *Période* ; ensuite, de permuter les caractères à l'intérieur

de chaque bloc suivant un permutation donnée Π. La clef du chiffre est la permutation Π. La principale différence avec le chiffrement par substitution est que le chiffrement par transposition a la propriété que le message crypté contient tous les caractères contenus dans le texte en clair (avec les mêmes fréquences) mais dans un ordre différent. En d'autres termes, les statistiques des unigrammes reste inchangées par le processus de chiffrement. Le déchiffrement se fait de la même manière en utilisant la permutation inverse de la permutation Π.

Exemple I.5 :

$$\Pi = \{4,3,6,1,2,5\} \qquad \Pi^{-1} = \{4,5,2,1,6,3\}$$

Le texte en clair est découpé en blocs de six (6) caractères ; ensuite, le 1er caractère de chaque bloc devient le 4ème, le 2nd devient le 5ème, le 3ème devient le 2nd, …etc.

Clef :

Texte en clair :	123456
Texte crypté :	436125

Texte crypté :	123456
Texte en clair :	452163

Chiffrement

Position :	12345612345612345612345612345612 3
Texte en clair :	UN_EXEMPLE_DE_SIMPLE_SUBSTITUTION
Texte crypté :	E_EUNXELDMP_ISPE_MS_BLEUTUISTUNIO

De cet exemple, il est clair que les statistiques des bigrammes et des trigrammes sont perdues après le chiffrement. Par exemple, le bigramme 'E_' est crypté vers 'MP', 'IS' et '_B'. Contrairement au chiffrement par substitution, il n'y a aucune correspondance directe entre les *n*-grammes du texte en clair et le texte crypté. Mais ceci, n'implique pas qu'une estimation approprié ne peut être basée sur ces statistiques.

V- Conclusion :

La cryptographie reste une lutte éternelle entre les concepteurs de cryptosystèmes, et d'ingénieux cryptanalystes utilisant toutes les avancées techniques pour casser ces systèmes. Un défaut majeur caractérise les cryptosystèmes actuels : leurs propriétés ne sont pas connues, et leur sécurité n'est pas démontrée pas formellement. L'utilisation des outils de l'intelligence semble être le prochain pas pour sécuriser ces systèmes. Une autre tendance dans la domaine de la cryptologie est l'utilisation des méta-heuristiques déjà utilisées dans beaucoup de problèmes. Certains préconisent que les méthode de résolution connues en dans le domaine de l'optimisation combinatoire peuvent apporter une grande attribution dans le domaine de la cryptologie, et que le pouvoir des techniques méta-heuristiques dans les applications cryptologique est très sous estimé

Chapitre II : *Optimisation combinatoire et Complexité*

I- Introduction :

L'optimisation combinatoire est l'une des branches les plus jeunes et les plus actives des mathématiques discrètes. Elle occupe une place très importante en recherche opérationnelle et en informatique. Son importance se justifie d'une part par la grande difficulté de résolution des problèmes d'optimisation, et d'autre part par les nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire [58].

L'objectif de l'optimisation combinatoire est d'apporter des techniques efficaces pour la résolution de certains problèmes difficiles (comme ceux classés NP-Complets en théorie de la complexité), des problèmes qui exigent des ressources (en terme de temps de calcul et d'espace mémoire) augmentant de manière considérable par rapport à la taille de l'instance du problème.

L'optimisation combinatoire tente de résoudre ces problèmes en utilisant des méthodes autre que celles basées sur la force brute (c'est-à-dire l'énumération de toutes les solutions possibles). Ces techniques provenant d'autres domaines (le recuit simulé inspiré de la mécanique statique, les algorithmes génétiques de la biologie, colonies de fourmis…) permettent d'apporter des solutions satisfaisantes à de nombreux problèmes.

D'un autre coté, la théorie de la complexité traite l'aspect de difficulté ou de facilité du problème traité. La théorie de la complexité des algorithmes permet d'apporter une clarification conceptuelle du domaine de l'optimisation combinatoire.

Parmi les domaines où se posent des problèmes d'optimisation combinatoire, on trouve la conception VLSI, où le but est de placer des composants VLSI sur une puce en minimisant la surface, la robotique avec la planification de trajectoires de robots, la logistique avec par exemple les problèmes de satisifiabilité et celui du voyageur de commerce, etc…

II- La complexité :

La complexité a pour but de donner un contenu formel à la notion intuitive de difficulté de résolution d'un problème. L'existence d'un algorithme pouvant résoudre un problème ne permet pas d'affirmer que la résolution peut être appliquée en pratique, si par exemple, le temps de calcul se révèle prohibitif. La théorie de complexité apporte la possibilité de dégager plusieurs niveaux de difficulté, ce qui permet de déterminer s'il est raisonnable ou non de se lancer dans la résolution d'un problème, ou bien, chercher à le reformuler pour aboutir à un problème plus simple. Auparavant, cette étude se basait sur le comportement moyen des algorithmes face à une batterie aléatoire de problèmes, mais cela manquait d'assurance ; et donc, on s'est orienté vers l'étude des plus mauvais cas, en cherchant pour chaque algorithme à majorer le nombre d'opérations.

Les problèmes concernés par la complexité sont des problèmes formalisables et décidables [2]. Un problème est dit *formalisable* s'il peut être exprimé sous la forme d'une association entre les entrées et les sorties. Un problème peut être non-formalisable parce qu'il est intrinsèquement vague, ou bien parce qu'il est difficile d'obtenir un consensus sur sa spécification précise où plusieurs approches sont possibles. Exemple: *déterminer la santé financière d'une entreprise.*

Mais il ne suffit pas qu'un problème soit formalisable pour qu'il puisse être résolu sur machine. Les problèmes qui ne peuvent être résolus à l'aide d'une machine de *Turing* sont appelés des problèmes *non-décidables*. Exemple : *le problème de la terminaison des programme*s.

II.1. Définition de la complexité :

La complexité d'un algorithme résolvant un problème donné, est le nombre d'opérations (complexité temporelle) ou la quantité d'espace mémoire (complexité spatiale) nécessaire à la résolution d'une instance du problème en fonction de sa taille. Habituellement, la complexité en espace n'est pas prise en compte car les modèles computationnels sont supposés disposer d'une mémoire infinie [14]; c'est pourquoi, la notion de complexité d'un algorithme désigne généralement sa complexité temporelle.

La complexité dans le pire des cas d'un algorithme pour une donnée est le nombre maximum d'opérations élémentaires nécessaires au traitement de la donnée. Si la fonction de la complexité de cet algorithme peut être bornée par un polynôme alors l'algorithme est dit à temps polynomial, sinon, il est à temps exponentiel.

II.2. Les classes de complexité :

II.2.1. Les problèmes polynomiaux (P) :

L'ensemble des problèmes polynomiaux comporte des problèmes pouvant être résolus par un algorithme dont la complexité peut être majorée par un polynôme en fonction de la taille des données en entrée. Un tel algorithme est dit *efficace* pour le problème en question. Plus formellement, ce sont les problèmes résolus en un temps polynomial par une machine de Turing déterministe. C'est notamment le cas de certains problèmes tels le tri d'un tableau, la recherche du plus court chemin dans un graphe, le test de planarité d'un graphe, ainsi que certaines variantes du problème de satisfiabilité (SAT) comme Horn-SAT et 2-SAT, mais notons que MAX-2-SAT (satisfiabilité maximale d'une formule dont les clauses contiennent deux littéraux) n'est pas polynomial. Cependant, pour la majorité des problèmes d'optimisation combinatoire (ceux qui sont vraiment intéressants dans des applications réelles), aucun algorithme polynomial n'est connu actuellement.

La classe P comporte donc les problèmes les plus faciles à résoudre. Il existe aussi une sous-classe de problèmes polynomiaux dont la complexité est linéaire, et qui constitue une classe de problèmes encore plus simples que ceux de P.

II.2.2. Les problèmes non-déterministes polynomiaux (NP) :

La classe NP a une définition plus générale que la classe P. Elle englobe tous les problèmes dont les meilleurs algorithmes ne sont pas polynomiaux (exponentiels au pire des cas), mais pour lesquels toute solution proposée peut être vérifiée en temps polynomial. Ces problèmes peuvent être résolu en temps polynomial mais sur une machine non-déterministe. Un exemple classique de représentant de cette classe est le problème de la satisfiabilité d'une formule booléenne.

Une machine déterministe étant un cas particulier d'une machine non-déterministe, il ressort de la définition des deux classes P et NP que $P \subseteq NP$. La question si l'inclusion est stricte n'a pas encore de réponse, il s'agit d'un des problèmes les plus importants de l'informatique théorique.

II.2.3. Les problèmes non-déterministes polynomiaux complets (NP-Complets) :

C'est une sous-classe de la classe NP, mais à cause de son importance, elle est souvent traitée comme une classe à part entière. Elle comporte les problèmes les plus difficiles de la classe NP. Un problème NP est NP-Complet si tout problème NP peut se ramener à ce problème en un temps polynomial. Cette classe tire son importance du fait que si un de ses problèmes peut être résolu en un temps polynomial, alors toute la classe NP devient polynomiale ($NP = P$). Bien qu'il n'a jamais était prouvé, il est souvent supposé que $NP \neq P$ [10].

Les problèmes NP-Complets recouvrent un très large spectre de domaines de recherche : SAT (satisfaction d'expressions booléennes), CSP (problèmes de satisfaction de contraintes), Recherche Opérationnelle, Théorie des Graphes, Programmation par Contraintes, etc. Le premier problème à être prouvé NP-Complet est le problème de Satisfiabilité SAT, COOK démontra que chaque problème appartenant à NP peut être réduit au problème de satisfiabilité [13].

Le degré de difficulté des différentes classes de la complexité peut être illustré par le schéma suivant :

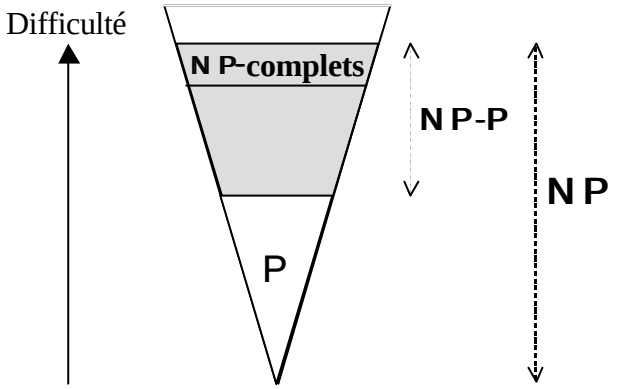

Fig II.1 : Les différentes classes de complexité ordonnées suivant leur degré de difficulté (sou l'hypothèse que $NP \neq P$)

III- Les méthodes de résolution :

Les algorithmes dédiés à la résolution de problème appartenant au domaine de l'optimisation combinatoire se divisent en deux grands groupes : les algorithmes *exacts* ou *complets*, et les algorithmes *approximatifs* ou *incomplets*.

Les algorithmes exacts sont plus lents mais garantissent la détermination d'une solution (ou de la solution optimale) si elle existe. Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent de manière implicite, l'ensemble des solutions possibles de l'espace de recherche. L'algorithme exact le plus basic est la recherche exhaustive. Pour améliorer

l'énumération des solutions, une telle méthode dispose de techniques pour détecter le plus tôt possible les échecs (calculs de bornes) et d'heuristiques spécifiques pour orienter les différents choix (comme c'est la cas pour la procédure de DAVIS et PUTNAM [16] pour la résolution du problème de satisfiabilité SAT).

Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) telles le Simplexe, les techniques de séparation et évaluation progressive (SEP) [39] ou les algorithmes avec retour arrière.

Les méthodes exactes permettent de déterminer des solutions optimales pour des problèmes de taille raisonnable. Malgré les progrès réalisés (notamment en matière de la programmation linéaire en nombres entiers), comme le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème, les méthodes exactes rencontrent généralement des difficultés face aux applications de taille importante, et donc, les recherches se sont dirigées vers un nouveau type d'algorithmes : les algorithmes *approximatifs* qui perdent la complétude pour gagner en efficacité.

Les méthodes approximatives constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale. Les algorithmes approximatifs tentent de trouver une solution *satisfaisante* au problème traité. Une solution satisfaisante peut être déterminée suivant une liste de critères pré-définie. Ainsi, en évitant de parcourir tout l'espace de recherche, ces algorithmes offrent un temps de réponse très intéressant.

Les méthodes approximatives sont très variées, et basées sur des concepts différents, il existent des méthodes basées sur l'approche de construction, d'autres basées sur le voisinage, et des méthodes évolutives. Parmi ce type d'algorithmes, beaucoup sont spécifiques à une des instances d'un problème donné. Mais, des progrès importants ont été réalisés avec l'apparition d'une

génération de méthodes approximatives puissantes et surtout générales, souvent appelées *méta-heuristiques* [57].

Une méta-heuristique est constituée d'un ensemble de concepts fondamentaux qui permettent d'aider à la conception de méthodes heuristiques spécifiques pour un problème d'optimisation. Ainsi les méta-heuristiques sont adaptables et applicables à une large classe de problèmes.

Les méta-heuristiques sont représentées essentiellement par les *méthodes de voisinage* (le recuit simulé, la recherche taboue…) et les *algorithmes évolutifs* (algorithmes génétiques, recherche dispersée, …), et récemment les méthodes constructionnistes (colonie de fourmis). Grâce à ces méta-heuristiques, on peut proposer aujourd'hui des solutions approchées pour des problèmes d'optimisation classiques de plus grande taille et pour de très nombreuses applications qu'il était impossible de traiter auparavant [39]. On constate, depuis ces dernières années, que l'intérêt porté aux méta-heuristiques augmente continuellement en recherche opérationnelle et en intelligence artificielle.

III.1. L'approche de construction :

L'approche de construction est la plus ancienne et occupe une place très importante en optimisation combinatoire et en intelligence artificielle [39]. Une méthode basée sur l'approche de construction consiste à développer étape par étape une solution à partir d'une solution partielle initialement vide. Elle cherche à étendre à chaque étape la solution partielle de l'étape précédente en déterminant la variable à ajouter et aussi la valeur assignée à cette variable, pour obtenir une nouvelle solution partielle. Ce processus est itéré jusqu'à ce qu'à l'obtention d'une solution complète où satisfaisant certaines conditions, ou bien tout simplement la fin du temps de calcul alloué à la recherche.

Pendant le processus de construction précédant, une méthode de construction peut faire intervenir des heuristiques pour effectuer les choix nécessaires : le choix de la variable suivante et le choix de la valeur assignée à cette variable. Les méthodes de cette classe diffèrent entre elles selon les

heuristiques utilisées. En général, les heuristiques portent plus souvent sur le choix de variables que sur le choix de valeurs car les informations disponibles concernant le premier choix semblent souvent plus riches. La performance de ces méthodes dépend des capacités des heuristiques employées à exploiter les connaissances du problème.

III.1.1. Exemples de méthodes de construction :

Un premier type de méthodes de construction est représenté par les *méthodes gloutonnes*. Une méthode gloutonne consiste à fixer à chaque étape la valeur d'une variable sans remettre en cause les choix effectués précédemment [55]. Les méthodes gloutonnes sont généralement rapides, mais fournissent le plus souvent des solutions de qualité médiocre. Elles ne garantissent la détermination de l'optimum que dans des rares cas particuliers.

Un autre type de méthode de construction comprend plusieurs algorithmes basés sur le principe de séparation et évaluation progressive [55]. Le plus connu des ces algorithmes l'algorithme A* pour la recherche d'un plus court chemin dans un graphe valué. Cet algorithme débute avec un nœud initial et prolonge ensuite -parmi l'ensemble de chemins partiels développés- le chemin dont la longueur est la plus faible, en utilisant une estimation de la longueur restante pour calculer la borne inférieure. Lorsqu'un chemin complet est trouvé, les chemins partiels dont la longueur est supérieure à celle du chemin complet sont éliminés.

Dans ce type d'approche, on trouve aussi des algorithmes spécifiques comme ceux de JOHNSON [41] pour le problème de satisfiabilité, et aussi une récente méta-heuristique inspiré du comportement des fourmis.

III.1.2. La méta-heuristique 'colonies de fourmis' (ACO) :

La méta-heuristique colonies de fourmis (*Ant Colonies Optimization*) est une méthode récente basée construction récente, elle s'inspire du comportement réel des colonies de fourmis. L'idée de s'inspirer du comportement des fourmis pour trouver de bonnes solutions à des problèmes d'optimisation combinatoire a été proposée par COLORNI, DORIGO et MANIEZZO [12]. Cette méthode se caractérise par la combinaison d'une approche de construction et des mécanismes d'apprentissage fondés sur la mémorisation [39].

Le principe de cette méta-heuristique est basé sur la manière dont les fourmis cherchent leur nourriture et retrouvent leur chemin pour retourner dans la fourmilière. Initialement, les fourmis explorent les environs de leur colonie de manière aléatoire. Sitôt qu'une source de nourriture est repérée par une fourmi, son intérêt est évalué, et la fourmi ramène un peu de nourriture à la colonie. Pour retrouver son chemin, elle suit une trace chimique (phéromone) déposée par elle. Durant le chemin du retour, elle dépose également une quantité de phéromone dépendant de l'intérêt de la source de nourriture. Comme toutes les fourmis membres de la colonies font de même, les traces laissées augmentent plus rapidement pour les sources de nourritures proches de la fourmilière ; et lorsque plusieurs traces mènent à la même source, les traces correspondant aux chemins les plus courts sont renforcées à un rythme plus élevé. Il en résulte qu'après un certain temps, les chemins les plus rapides menant aux sources de nourriture les plus importantes sont marqués par de fortes traces. De cette manière, les fourmis arrivent à optimiser leurs déplacements.

En résumé, malgré la vision très limitée de chaque fourmi, une colonie de fourmis parvient à minimiser la longueur du chemin conduisant à une source de nourriture, grâce aux traces chimiques (phéromones) laissées par chacune des fourmis. Un principe analogue a été utilisé pour développer une méta-heuristique en faisant une analogie entre l'aire dans laquelle les fourmis cherchent de la nourriture et l'ensemble des solutions admissibles du problème, entre la quantité ou la qualité de la nourriture et la fonction objectif à optimiser et enfin entre les traces et une mémoire adaptative [21].

Schématiquement, un algorithme basé sur une colonie de fourmis peut être décrit comme suit :

Procédure Colonie de Fourmis ;
Début
Générer les fourmis ;
Initialiser les traces ;
Tant que (*Critère_Arret non satisfait*) **faire**
Répéter pour chaque fourmi
Début
Construire une nouvelle solution à l'aide des informations contenues dans les traces et une fonction d'évaluation partielle ;
Évaluer la qualité de la solution ;
Mettre à jour les traces ;
Fin ;
Fin ;

Algorithme II.1 : Méta-heuristique Colonie de Fourmis

III.2. L'approche de voisinage :

Les méthodes de voisinage sont fondées sur la notion de voisinage. En général, une telle méthode démarre d'une configuration initiale, et réitère un processus qui consiste à remplacer la configuration courante par l'un de ses voisins en tenant compte de la fonction de coût. Ce processus s'arrête et retourne la meilleure configuration trouvée quand la condition d'arrêt est réalisée. Les méthodes de voisinage diffèrent essentiellement entre elles par le voisinage utilisé et la stratégie de parcours de ce voisinage. Un exemple simple d'une méthode de voisinage est la recherche locale.

III.2.1. La Recherche Locale '*Local Search*' :

La recherche locale est une très ancienne méthode. Elle consiste à essayer d'atteindre la solution à partir d'une configuration aléatoire en maximisant à chaque déplacement la valeur de la fonction objectif [5]. Cette procédure fait intervenir à chaque itération le choix d'un voisin pour améliorer la configuration courante. Plusieurs possibilités peuvent être envisagées pour effectuer ce choix : choisir le premier voisin améliorant la fonction objectif, ou bien rechercher le meilleur voisin. Comme l'espace des solutions est fini, cette procédure s'arrête toujours, et la dernière configuration est un optimum local [39].

Les principales étapes de la recherche locale peuvent être résumées comme suit :

Procédure Recherche Locale ;

Début

1) Choisir une solution aléatoire ;
2) Choisir parmi le voisinage de cette solution une solution qui améliore la fonction objectif ;
3) Répéter (2) jusqu'à l'obtention d'une solution satisfaisante ou qu'il n'y est aucun voisin qui améliore la fonction objectif ;
4) Si la solution obtenue n'est pas satisfaisante et le nombre d'itérations maximum n'est pas encore atteint, alors répéter (1) ;

Fin ;

Algorithme II.2 : Recherche locale

L'efficacité de la recherche locale dépend essentiellement du voisinage choisi. Un bon résultat peut être obtenu avec cette méthode avec de multiples exécutions démarrant de solutions différentes à chaque itération. Un défaut majeur est que l'algorithme ne tire aucun profit des optimums locaux déjà découverts.

III.2.2. Le Recuit Simulé *'Simulated Annealing'* :

Le recuit simulé est une méthode inspirée du processus de recuit physique [44][45]. Ce processus de physique statistique (étude physique de la thermodynamique) utilisé en métallurgie pour améliorer la qualité d'un solide cherche un état d'énergie minimale qui correspond à une structure stable du solide. En partant d'une haute température à laquelle le solide est liquide, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une diminution progressive de la température. Quand la température tend vers sa borne inférieure, seules les transitions d'un état à un état d'énergie plus faible sont possibles. Un refroidissement rapide bloquerait le système dans un état de forte énergie correspondant à un déséquilibre, tandis qu'un refroidissement lent conduirait le système vers un état équilibré de faible énergie.

Le recuit simulé répète une procédure itérative qui cherche des configurations de coût plus faible tout en acceptant de manière contrôlée des configurations qui dégradent la fonction de coût. A partir d'une première configuration générée aléatoirement et une température fixée, une nouvelle configuration est choisie parmi le voisinage de la configuration courante, si elle est meilleure, alors elle est retenue systématiquement. Dans le cas contraire, elle peut être acceptée avec une probabilité dépendant de la température et de l'importance de la dégradation de la fonction objectif (les dégradations plus faibles sont plus facilement acceptées, et une température élevée correspond à une probabilité plus grande d'accepter des dégradations). La température est contrôlée par une fonction décroissante qui définit un schéma de refroidissement.

Procédure Recuit Simulé ;
Début
Générer une solution et une température T ;
Tant que (T>0) **faire**
Répéter (*Nombre_Itérations*) fois
Début
Générer une solution dans le voisinage de la solution courante ;
Calculer la variation du la fonction objectif ΔE ;
Si (ΔE>0) **alors** remplacer la solution courante par la nouvelle solution
Sinon remplacer la solution courante par la nouvelle solution avec une probabilité $p = \exp\left(-\frac{\Delta E}{\delta T}\right)$ (δ constante) ;
Fin ;
Diminuer la température T ;
Fin ;

Algorithme II.3 : Procédure du recuit simulé

où :

- T : est la température.
- ΔE : la différence des valeurs de la fonction objectif.

La performance du recuit simulé dépend largement du schéma de refroidissement utilisé. Plusieurs schémas théoriques et pratiques existent [49] :

- Réduction par paliers : chaque température est maintenue égale pendant un certain nombre d'itérations, et décroît ainsi par paliers.
- Réduction continue : la température est modifiée à chaque itération.
- Réduction non-monotone : la température décroît à chaque itération avec des augmentations occasionnelles.

Le fait qu'une configuration peut être acceptée même si elle ne satisfait pas d'avantage la fonction objectif que la configuration précédente permet à l'algorithme de sauter les minima locaux et de converger l'optimum global dans l'espace des solutions.

III.2.3. La Recherche Taboue '*Tabu Search*' :

La recherche taboue est une méthode basée voisinage [30][37]. Elle utilise une stratégie générale pour guider la recherche en dehors des optimums locaux.

La recherche taboue est une technique itérative qui procède à chaque itération en choisissant la meilleure configuration voisine même si elle n'améliore pas la configuration courante. La recherche taboue ne s'arrête donc pas au premier optimum trouvé. Cependant, cette stratégie peut entraîner des cycles. Pour empêcher cela, les *k* dernières configurations visitées sont gardées dans une mémoire à court terme et tout mouvement qui conduit à une de ces configurations est prohibé. Cette mémoire est appelée la *liste taboue*, une des composantes essentielles de cette méthode. Elle permet d'éviter tous les cycles de longueur inférieure ou égale à *k*. La valeur de *k* dépend du problème à résoudre et peut éventuellement évoluer au cours de la recherche.

La mémorisation de configurations entières serait trop coûteuse en temps de calcul et en place mémoire ; donc, la liste taboue mémorise des *caractéristiques* des configurations au lieu de configurations complètes [31][32]. Plus précisément, lorsqu'un mouvement vient d'être effectué, c'est généralement la caractéristique perdue par la configuration courante qui devient taboue. Le résultat est que non seulement la configuration courante ne pourra pas réapparaître lors des *k* prochaines itérations, mais que de nombreuses autres configurations seront également interdites. Lorsque les listes taboues font intervenir des caractéristiques de modifications, les interdictions qu'elles engendrent peuvent s'avérer trop fortes et restreindre l'ensemble des solutions admises à chaque itération. Un mécanisme particulier, appelé *l'aspiratio*n, est mis en place afin de pallier cet inconvénient. Ce mécanisme permet de lever le statut tabou d'une configuration, sans pour autant introduire un risque de cycles dans le processus de recherche. La fonction d'aspiration peut être définie de

plusieurs manières. La fonction la plus simple consiste à révoquer le statut tabou d'un mouvement si ce dernier permet d'atteindre une solution de qualité supérieure à celle de la meilleure solution trouvée jusqu'alors.

```
Procédure Recherche Taboue ;
Début
  Générer une solution S ;
  Tant que (Critère_Arret non satisfait) faire
        Début
           Sélectionner la meilleure solution S' ;
           Si (S' ∉ Liste_Taboue) alors S = S' ;
           Mise_à_Jour (Liste_Taboue) ;
        Fin ;
Fin ;
```

Algorithme II.4 : Principe de la recherche taboue

Il existe d'autres techniques intéressantes pour améliorer la puissance de la méthode taboue, en particulier, *l'intensification* et *la diversificatio*n.

L'intensification se fonde sur l'idée d'apprentissage de propriétés favorables : les propriétés communes souvent rencontrées dans les meilleures configurations visitées sont mémorisées au cours de la recherche, puis favorisées. La diversification a un objectif inverse de l'intensification : elle cherche à diriger la recherche vers des zones inexplorées. Sa mise en œuvre consiste souvent à modifier temporairement la fonction de coût pour favoriser des mouvements n'ayant pas été effectués ou à pénaliser les mouvements ayant été souvent répétés. L'intensification et la diversification jouent donc un rôle complémentaire pour améliorer la performance de l'algorithme.

Pour la sélection de la configuration voisine à la configuration courante, plusieurs stratégies existent :

- Stratégie *Best Move* : où tout le voisinage de la configuration est passé en revue, et la meilleure configuration est sélectionnée.
- Stratégie *First Improve* : Lorsque le cardinal du voisinage est important, il est très coûteux d'utiliser la stratégie précédante, et donc, la première configuration améliorant la configuration courante est choisie, pour éviter de parcourir tout le voisinage.
- Stratégie *First and Best Admissible Move* : le même principe que la stratégie *Best Move* est adopté, sauf que dans le cas où aucune configuration n'améliore la fonction d'évaluation, alors la configuration qui détériore le moins cette fonction est choisie.

III.2.4. La méthode GRASP :

L'algorithme GRASP '*Greedy Randomized Adaptive Search Procedure*' [23][56] est une procédure itérative hybride, combinée à partir des heuristiques gloutonnes, de la recherche aléatoire et des méthodes de voisinage.

Chaque itération de GRASP est composée de deux phases : une phase de construction et une phase de recherche locale. La meilleure solution des deux phases est gardée comme résultat.

Dans la phase de construction, une solution réalisable est construite itérativement élément par élément. A chaque itération de la construction, le choix de l'élément à rajouter est déterminé en ordonnant les éléments contenus dans une liste candidate suivant une fonction Greedy. Cette fonction mesure l'avantage de la sélection chaque élément, et sa valeur est mise à jour à chaque itération. L'élément sélectionné de la liste n'est pas nécessairement le premier, mais un des meilleurs. Ce choix aléatoire de l'un des meilleurs éléments de la liste candidate (mais pas obligatoirement le meilleur) apporte une solution différente à chaque itération de GRASP. La liste des meilleurs candidats est dynamiquement mise à jour après chaque itération de construction.

La phase de construction ne garantit pas que la solution générée soit un optimum local, pour cette raison, on fait appel à une procédure de recherche locale pendant la deuxième phase pour essayer d'améliorer cette solution. Les deux paramètres de cette méthodes sont donc la longueur de la liste de candidats et le nombre d'itérations autorisées.

```
Procédure GRASP ;
Début
  Tant que (Conditions de fin non satisfaites) faire
    Début
      S ← GénérerSolutionAléatoire() ;
      AppliquerRechercheLocale(S) ;
      MémoriserMeilleurSolution() ;
    Fin ;
Fin ;
```

Algorithme II.5 : Procédure GRASP

III.3. Les méthodes évolutives :

Beaucoup d'algorithmes sont inspirés des sciences de la vie et des processus naturels, en adaptant des structures et des mécanismes du monde vivant pour développer des objets artificiels utilisables dans des contextes variés. Dans le domaine de l'optimisation combinatoire, Les méthodes évolutives que nous présentons dans cette section constituent la base d'un important champ de la programmation informatique.

Contrairement aux méthodes constructives et séquentielles qui font intervenir une solution unique (partielle ou non), les méthodes évolutives manipulent un groupe de solutions admissibles à chacune des étapes du processus de recherche. L'idée centrale consiste à utiliser régulièrement les propriétés collectives d'un ensemble de solutions distinguables, appelé *population*, dans le but de guider efficacement la recherche vers de bonnes solutions dans l'espace de recherche.

Les méthodes évolutives présentent trois aspects communs [3] :

- une *population* constituée de plusieurs individus représentant des solutions potentielles du problème donné.
- un *mécanisme d'évaluation* de chaque individu, qui va servir comme critère de sélection des individus qui vont pouvoir survivre et/ou se reproduire pour transmettre leurs caractéristiques à la génération d'individus suivante. Cette sélection se base généralement sur le principe de conservation des individus les mieux adaptés et d'élimination des moins adaptés.
- un *mécanisme d'évolution* composé d'opérateurs permettant d'éliminer certains individus et de produire de nouveaux individus à partir des individus sélectionnés.

III.3.1. Les Algorithmes Génétiques :

Les algorithmes génétiques [40][34] s'inspirent directement de l'évolution des espèces et la théorie de sélection naturelle. Ils s'appuient sur un codage des configurations sous forme de vecteurs binaires de longueur fixe représentant une configuration du problème, un critère de *sélection*, et un ensemble d'opérateurs génétiques : la *mutation* et le *croisement.* Ces opérateurs sont définis de manière à opérer aléatoirement sur un ou deux individus sans aucune connaissance sur le problème traité.

- L'objectif de la sélection est de choisir les parents qui vont être croisées. Il est nécessaire que la solution présentant la meilleure valeur de la fonction objectif ait une plus grande probabilité pour qu'elle soit sélectionnée qu'une autre solution.

- Le *croisement* permet de produire deux nouveaux individus (enfants) à partir de deux individus (parents). Le croisement réalise donc uniquement des recombinaisons de valeurs (gènes) existantes entre deux parents et ne permet pas d'introduire de nouvelles valeurs dans les individus enfants.
- L'opérateur de *mutation* permet d'introduire de nouvelles caractéristiques chez les individus, ce qui garantit un aspect de diversification des individus. La mutation consiste à changer aléatoirement la valeur de certains gènes dans un individu. Dans les algorithmes génétiques la mutation est considérée comme un opérateur secondaire par rapport au croisement. Plusieurs types de mutation existent : inverser la valeur d'un gène, échanger les valeurs de deux gènes,…etc ; parfois l'opérateur de mutation est beaucoup plus complexe, allant jusqu'à effectuer une recherche taboue [66].

Un cycle d'évolution complet d'un algorithme génétique est formé par l'application des opérateurs de sélection, croisement et mutation sur une population d'individus.

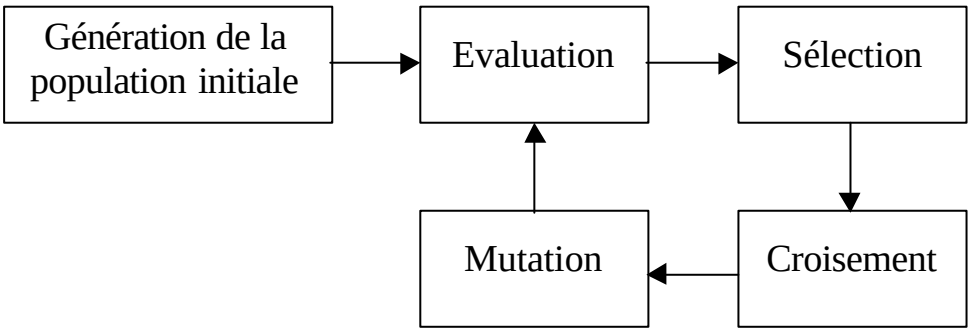

Fig II.2 : Schéma général des étapes d'un algorithme génétique

A chaque itération d'un algorithme génétique, une nouvelle population est générée contenant des individus créés en utilisant les meilleures parties des meilleurs individus de la génération précédante (croisement) ainsi que des parties innovatrices (mutation) mais avec une plus petite probabilité. A chaque génération, les individus vont tendre vers l'optimum de la fonction objectif.

Algorithme Génétique ;
Début
Générer une population initiale ;
Tant que (*Critère_Arret non satisfait*) **faire**
Début
Répéter (*Nombre_Opérations*) fois
Début
Sélectionner deux individus ;
Appliquer Croisement entre ces individus avec une probabilité P_C.
Appliquer Mutation sur l'individu croisé avec une probabilité P_M.
Evaluer l'individu produit ;
Fin ;
Remplacer les mauvais individus par de meilleurs individus produits avec une probabilité $P_{R.}$
Fin ;
Fin ;

Algorithme II.6 : Structure générale d'un Algorithme Génétique

L'étape la plus difficile pour implémenter un algorithme génétique est l'étape du codage des solutions. Les algorithmes génétiques nécessitent le codage des paramètres à optimiser sous forme d'individus. Ce codage doit être tel que les solutions obtenues doivent correspondre aux contraintes du problème.

III.3.2. La Recherche Dispersée '*Scatter Search*' :

La recherche dispersée [33][28] est une technique basée population. Elle consiste à chaque itération à générer une population de solutions diversifiée à partir d'une population semence, ensuite les meilleures solution sont combinées pour former une nouvelle population. Les principales étapes de cet algorithme sont :

- Construire un ensemble de solutions de bonne qualité et présentant un bon degré de diversité.
- Sélectionner des sous-ensembles de ces solutions.
- Appliquer un opérateur de combinaison à chacun de ces sous-ensembles pour générer de nouvelles solutions.

Contrairement aux algorithmes génétiques, la combinaison dans la recherche dispersée peut s'étendre à trois éléments ou plus. Une procédure (telle une recherche locale) est utilisée pour améliorer ou rendre admissibles les solutions produites par l'opérateur de combinaison. Les meilleurs éléments de cette population remplacent les mauvais éléments dans l'ensemble de référence, et cette procédure est itérée jusqu'à ce que l'ensemble des solutions ne change plus. Le terme 'meilleur' n'est pas retreint à la valeur de la fonction objectif comme c'est le cas pour la plupart des techniques, mais s'étend au degré de diversité des solutions considérées. La procédure peut être relancée en utilisant le meilleur élément de l'ensemble de référence comme semence.

IV- Conclusion :

Les recherche pour trouver de techniques de résolution efficaces traitant les problèmes d'optimisation combinatoire sont encore loin d'atteindre leurs objectifs. C'est pourquoi l'étude de ces méthodes est actuellement en plein développement. Afin de résoudre des instances de taille et de difficulté croissantes, il faut mettre au point des méthodes toujours plus puissantes. Pour atteindre cet objectif, au moins deux voies privilégiées se développent : l'hybridation des méthodes et la parallélisation.

Un point faible de ces méthodes, c'est qu'il n'existe pas de résultats théoriques garantissant la convergence d'une procédure. La raison principale de cet état de fait tient à la nature même des méthodes : étant souvent adaptatives et modulables, leur analyse par des outils formels est rendue encore plus difficile.

Dans le chapitre suivant, nous allons reprendre en détails une méthode de la famille des algorithmes évolutifs : la Recherche Dispersée.

Chapitre III : *La Recherche Dispersée*

I- Introduction :

La Recherche Dispersée (*Scatter Search*) fut proposée par GLOVER en 1977 dans une étude heuristique sur les problèmes de programmation linéaire discrète [29]. La Recherche Dispersée est une méthode basée population, et forme avec les algorithmes génétiques la grande famille des méthodes évolutives. Elle est basée sur les stratégies proposées dans les années 1960 pour la combinaisons des contraintes et des règles de décision.

L'approche de la Recherche Dispersée a évolué depuis sa première présentation, et un nombre de variantes et d'algorithmes basées sur ses principes furent récemment proposés pour différents problèmes d'optimisation combinatoire :

- Affectation multi-objectif [46].
- Affectation quadratique [15].
- Apprentissage des réseaux de neurones [42].
- Data-mining [1].
- Optimisation sans contraintes [24].
- Problème de coloration de graphe [36].
- Problème de satisfiabilité (SAT) [38].
- Problème de satisfiabilité maximale(Max-Sat) [22].
- Problème du routage des arcs [35].
- Problème du routage des véhicules [60].
- Problème d'arbres [9].
- Problème d'ordre linéaire [7].
- Problèmes de programmation non-linéaire [54].

La Recherche Dispersée partage avec la recherche taboue les mêmes origines. Initialement, la Recherche Dispersée était considérée comme un composant de la recherche taboue. Cependant, la plupart des travaux et des implémentations de la recherche taboue ont tout simplement ignoré ce composant, et les avantages de la Recherche Dispersée ne se sont faits connaître que récemment, quand certains travaux l'ont considérée comme une méthode à part entière dans le contexte des algorithmes évolutifs.

Similairement aux autres méthodes basées population, la Recherche Dispersée combine des configurations (vecteurs ou solutions) pour générer de nouvelles unes qui peuvent remplacer les anciennes dans la population. Mais la manière d'effectuer ces combinaisons et ces remplacements diffère des stratégies traditionnelles utilisées dans les algorithmes génétiques [36].

Les combinaisons opèrent sur plusieurs parents, et les remplacements dépendent de l'amélioration de la fonction objectif aussi bien que l'amélioration de la diversité de population. Encore, la Recherche Dispersée opère sur une population de taille relativement réduite ; et utilise à coté des mécanismes habituels, des processus heuristiques spécifiques pour renforcer le respect des contraintes du problème.

II- Description de la méthode de la Recherche Dispersée :

Ce qui suit, est la première description de la Recherche Dispersée faite par [29]. La Recherche Dispersée utilise une suite d'initialisations coordonnées pour générer des solutions. Ces solutions sont intentionnellement générer pour prendre en considération des caractéristiques dans diverses parties de l'espace des solutions.

La Recherche Dispersée oriente systématiquement ses exploitations vers un ensembles de points (solutions) de référence. Ces points de référence peuvent s'agir des meilleures solutions obtenues par une tentative précédente de résolution du problème, ou bien des points extrêmes dans des régions de l'espace de recherche.

L'approche commence par identifier une combinaison convexe, ou bien un centre de gravité pondéré des points de référence. Ce point central, avec un sous-ensemble des points de référence initiaux sont utilisés pour définir de nouvelles sous-régions. De la même manière, des points centraux des sous-régions sont aussi examinés. Finalement, les derniers points obtenus sont améliorés par un processus heuristique spécifique pour obtenir les solutions désirées (ce processus peut consister en une procédure itérative ou d'adjacence).

II.1. Principes généraux de la Recherche Dispersée :

Le processus de la recherche dispersée est basé sur des principes qui remplacent la conception orientée contraintes. Ce processus est conçu de telle façon à détecter l'information utile contenue dans les solutions, et à tirer profit de l'utilisation d'une méthode heuristique auxiliaire spécifique au problème pour évaluer les solutions combinées et pour créer de nouvelles unes [33].

D'une manière plus explicite, le processus de la recherche dispersée peut être schématisé comme suit :

1) Générer un ensemble initial de solutions de telle manière à assurer un niveau minimum de diversité, et tenter ensuite d'améliorer ces solutions en leur appliquant des processus heuristiques spécifiques au problème traité. Puis extraire de cet ensemble un sous-ensemble des meilleures solutions qui vont être les solutions de référence.

   La notion de '*meilleur*' dans ce contexte n'est pas limitée par une mesure donnée exclusivement par l'évaluation de la fonction objectif. Particulièrement, une solution peut être rajoutée à l'ensemble des solutions de références si la diversité de cet ensemble est améliorée, même si la valeur objectif de cette solution est inférieur aux valeurs d'autres solutions en concurrence pour l'admission à l'ensemble de référence.

2) Créer de nouvelles solutions par des combinaisons structurées des sous-ensembles de l'ensemble de solutions de référence courantes. Ces combinaisons structurées sont :
   a. choisies pour produire des solutions à l'intérieur et à l'extérieur des régions délimitées par les solutions de référence.
   b. modifiées pour rendre les solutions produites acceptables. Une solution acceptable peut être admissibles ou non à l'égard des contraintes exprimées par le problème.
3) Appliquer le processus heuristique déjà utilisé lors de la génération des solutions (étape 1) pour améliorer les solutions crées dans l'étape précédente. Un tel processus doit pouvoir opérer sur des solutions inadmissibles et peuvent ou non produire des solutions faisable.
4) Extraire un ensemble des meilleures solutions améliorées dans l'étape précédante et les ajouter à l'ensemble de référence. La notion de '*meilleur*' est une fois encore sommaire ; le mérite des solutions créées récemment est évalué par la fonction objectif à coté d'autres critère comme la diversité des solutions.
5) Répéter les trois dernières étapes (2, 3 et 4) jusqu'à ce que l'ensemble de référence ne change plus. Ici, cet ensemble a atteint un pont fixe.
6) Diversifier l'ensemble de référence en redémarrant le processus à partir de la première étapes, et arrêter quand une limite d'itérations spécifiée est atteinte

Le premier trait remarquable dans l'algorithme de la Recherche Dispersée est que la conception des combinaisons structurées est faite dans le but de créer de nouveaux centres de gravité dans les régions délimitées par les solutions à combiner. La notion de *'centre de gravité*' est -elle aussi- large, puisque ces centres peuvent être projetés en dehors de ces régions en appliquant des combinaisons non-convexe. Ce modèle de dispersion avec tels centres s'est avéré très utile dans plusieurs domaine d'application [28].

Un autre trait important est les stratégies de sélection des sous-ensembles des solutions à combiner. Ces stratégies sont définies pour permettre la construction de nouvelles solutions à l'intérieur ou l'extérieur des régions.

Aussi, la méthode est faite telle qu'elle utilise des mécanismes d'optimisation auxiliaires capables d'opérer sur des solutions inadmissibles au même titre que les solutions admissibles.

Les bases importantes de la Recherche Dispersée peuvent être résumées dans les points suivants [33] :

- L'information utile concernant la forme et l'emplacement des solutions optimales est typiquement contenue dans une collection des solutions d'élite diversifiée convenablement.
- Les stratégies de combinaison utilisées pour l'exploitation de telle information, doivent intégrer des mécanismes capables de construire des combinaisons pouvant extrapoler en dehors des régions formées par les solutions considérées. D'un autre coté, des processus heuristiques doivent être incorporés pour améliorer les solutions produites.

  L'utilité de la combinaison de ces mécanismes est d'assurer la diversité et la qualité des solutions tout au long le processus de recherche.
- La considération de plusieurs solutions simultanément, comme une base pour les combinaisons, augmente les possibilités d'exploiter l'information contenue dans l'union des meilleures solutions.

Les mécanismes utilisés dans la recherche dispersée ne sont pas restreints à une seule conception uniforme, ce qui permet l'exploration de plusieurs stratégies possibles qui peuvent être efficaces dans une implémentation particulière. Ceci permet de dresser un modèle général des procédures composant la Recherche Dispersée :

1) *Méthode de génération de diversification* : génère des solutions diverses à partir d'une ou plusieurs solutions (semence) en entrée.
2) *Méthode d'amélioration* : transforme une solution générée précédemment en une ou plusieurs solutions améliorées.
3) *Méthode de mise à jour de l'ensemble de référence* : crée et maintient un ensemble de solutions de référence formé par les meilleures solutions suivant les critères considérés. Le but est de combiner diversité et qualité des solutions.

4) *Méthode de génération de sous-ensembles* : génère des sous-ensembles de l'ensemble de référence formant chacun une base pour la construction des solutions combinées.
5) *Méthode de combinaison de solutions* : utilise des combinaisons structurées pondérées pour transformer chaque sous-ensemble de l'ensemble de référence en une ou plusieurs solutions.

Algorithme Recherche Dispersée ;
Début
$P = \varnothing$;
Tant que (*P est non rempli*) **faire**
Début
Construire une solution *s* et l'améliorer ;
Insérer *s* dans *P* ;
Fin ;

Insérer dans *BestSet* les b_1 meilleures solutions de *P* ;
Insérer dans *DivSet* b_2 solutions de *P* les plus diversifiées.
$RefSet = BestSet \cup DivSet$;
Tant que (*P est non rempli*) **faire**
Pour (chaque sous-ensemble *i* de *P*) **faire**
Début
Générer une solution s_i à partir de *i* ;
Si (s_i améliore *BestSet*) **alors** Insérer s_i dans *BestSet*
Sinon
Si (s_i diversifie mieux *DivSet*) **alors** Insérer s_i dans *DivSet ;*
Fin ;
Fin ;

Algorithme II.1 : Algorithme général de la Recherche Dispersée

Dans cet algorithme, une configuration est générée par la *méthode de génération de diversification*, puis elle est améliorée par la *méthode d'amélioration*, et insérée dans un ensemble initial *P* relativement grand.

Ensuite, un sous-ensemble de *P BestSet* est crée en sélectionnant les meilleures solutions de *P* en terme de fonction objectif.

Parmi les solutions restantes dans *P*, les plus diversifiées formeront un second sous-ensemble *DivSet*. L'ensemble de référence est formé par l'union de ces deux sous-ensembles : $RefSet = BestSet \cup DivSet$;

Après, la *méthode de génération de sous-ensembles* est utilisée pour générer des sous-ensembles de *RefSet* ; pour chaque sous-ensemble, une ou plusieurs solutions sont créées par la *méthode de combinaison de solutions* et améliorées par la *méthode d'amélioration*. La *méthode de mise à jour de l'ensemble de référence* décide de l'insertion de ces nouvelles solutions dans *RefSet*. Ces étapes sont itérées jusqu'à ce que *RefSet* ne change plus. La solution retournée est la meilleure solution de *BestSet* évidemment.

II.2. Les méthodes principales de la Recherche Dispersée :

Dans cette section, les méthodes présentées précédemment vont être reprises avec plus de détails.

II.2.1. La Méthode de génération de diversification :

La méthode de génération de diversification (*Diversification Generation Method*) génère une collection de diverses solutions à partir d'une ou plusieurs solutions arbitraires ou bien déterminées par une tentative de résolution précédante. Cette méthode a pour but de combiner la qualité des solutions générées avec un bon degré de diversité.

L'ensemble de référence n'est pas directement construit par cette méthode, mais plutôt un grand ensemble *P* est construit en premier, ensuite des solutions sont sélectionnés par la méthode de mise à jour de l'ensemble de référence pour créer *BestSet* et *DivSet* et en déduire ensuite *RefSet* (voir §II.2.3).

II.2.2. La Méthode d'amélioration :

La Méthode d'amélioration (*Improvement Method*) transforme une solution générée par l'étape précédante en une ou plusieurs solutions améliorées. Ni les solutions en entrée, ni celles retournées ne doivent être nécessairement faisables. Cette méthode est très liée à la représentation des solutions du problème et aux contraintes exprimées, ce qui la rend plus spécifique que les autres méthodes.

La plupart des méthodes d'amélioration dans le contexte de la recherche dispersée consistent en une simple procédure de recherche locale. Un premier conseil dans l'application de cette méthode est d'investir dans l'utilisation sélective de cette méthode au lieu de l'appliquer à toutes les solutions [48]. Dans le même contexte, une méthode d'amélioration cherchant la meilleure amélioration pour chaque solution est très coûteuse [52], il est préférable de la limiter à examiner une liste réduite de déplacement.

Notons aussi que l'application de cette méthode à toutes les solutions accélère la convergence de l'ensemble de référence *RefSet*, et une solution est prématurément retournée, malgré son avantage de trouver des solutions de bonne qualité assez rapidement. Il faut aussi trouver le juste équilibre entre le temps de calcul dépensé dans la génération des solutions et le temps dépensé en les améliorant.

II.2.3. La Méthode de mise à jour de l'ensemble de référence :

La Méthode de mise à jour de l'ensemble de référence (*Reference Set Update Method*) crée et maintient un ensemble de solutions de référence formé par les meilleures solutions (le cardinal de *RefSet* est relativement petit ($|RefSet| \approx 20$)). Le but est de combiner diversité et qualité des solutions, d'autres critères peuvent être pris en compte.

Un premier appel de cette méthode est lancé au début de l'algorithme de la Recherche Dispersée pour sélectionner les éléments de l'ensemble P à insérer dans l'ensemble de référence *RefSet*. Aussi, cette méthode est appelée pour

décider de l'insertion dans *RefSet* des solutions générées par la méthode de combinaison de solutions.

Dans les deux cas, la méthode procède de la même manière :

1) Tout d'abord, les solutions à insérer (appartenant à *P* ou bien celles combinées) sont triées suivant l'ordre croissant de la fonction objectif ; et les b_1 meilleures solutions sont insérées dans *BestSet*.
2) Pour chaque solution restante, la distance minimale entre cette solution et celles de *RefSet* = *BestSet* $\cup$ *DivSet* est calculée (à ce stade : *RefSet* = *BestSet* uniquement), et la solution avec la plus grande distance minimale est insérée dans *DivSet*.

   Donc, insérer la solution x dans *DivSet*, si $d_{\min}(x) = \max_{x_i}(d_{\min}(x_i))$

   avec $d_{\min}(x) = \underset{y \in RefSet}{Min}\{d(x,y)\}$ et $d(x,y)$ est la distance entre deux solutions.

   Ce processus est répété pour obtenir b_2 solutions et l'ensemble de référence *RefSet* est formé par $b = b_1 + b_2$ solutions de l'union de *BestSet* et *DivSet*, ce qui garantie la qualité et la diversité de *RefSet*.

Un juste équilibre entre l'intensification et la diversification est une caractéristique très recherchée dans la Recherche Dispersée. GLOVER conseille d'initialiser l'ensemble de référence *RefSet* avec la moitié de l'ensemble consistant de solutions de bonne qualité et l'autre moitié avec des solutions diversifiées, c'est à dire que $b_1 = b_2$ [28]. Aussi, la taille *p* de l'ensemble initial *P* est de l'ordre $p \approx 5 \times b$.

La mise à jour de l'ensemble de référence peut se faire de deux manière, l'une statique et l'autre dynamique. La mise à jour statique opère après que toutes les solutions candidates soient créées ; tandis que la mise à jour dynamique teste la possibilité d'insertion d'une solution dans l'ensemble de référence dés qu'elle est créée.

Par exemple, si $RefSet = \{s_1, s_2, s_3\}$ avec s_1 la meilleure solution et s_3 la mauvaise, et si la combinaison de s_1 avec s_2 a généré s_4 meilleure que s_3, alors s_3 est immédiatement remplacée par s_4 avant même de tester les autres combinaisons, ceci signifie que s_3 ne peut avoir l'opportunité de générer de nouvelles solutions. La mise à jour dynamique donc est plus 'agressive' en terme d'incorporation de solutions de bonne qualité, ceci est une caractéristique très utile au début de la recherche.

L'ensemble de référence peut être regénéré si la combinaison des solutions existantes n'apporte aucun renouvellement, ceci est fait en gardant *BestSet* intact, et générer de nouvelles solutions par un appel de la méthode de génération de diversification.

II.2.4. La Méthode de génération de sous-ensembles :

La Méthode de génération de sous-ensembles (*Subset Generation Method*) opère sur l'ensemble de référence, pour produire des sous-ensembles de ses solutions formant une base pour la création des solutions combinées. Cette méthode donc doit générer des sous-ensembles, en évitant évidemment la duplication de sous-ensembles générés précédemment. L'approche adoptée pour accomplir cette tache est d'organiser la génération en des catégories avec des caractéristiques différentes [33] :

- tout les sous-ensembles formés de deux (2) éléments.
- les sous-ensembles de trois (3) éléments formés par les sous-ensembles de deux (2) éléments augmentés par la meilleure solution non incluse dans ces sous-ensembles.
- les sous-ensembles de quatre (4) éléments formés par les sous-ensembles de trois (3) éléments augmentés par la meilleure solution non incluse dans ces sous-ensembles.
- les sous-ensembles formés par les *i* meilleures solutions, avec $i = \overline{5, b}$.

De cette manière, pour un ensemble de référence de cardinal *b*, au maximum $\left(b(b-1)/2 + b(b-1)/2 + b(b-1)/2 + (b-4) \right)$ sous-ensembles peuvent être générés, après éliminations des doublons.

Une étude portant sur la contribution de chaque catégories des solutions admises dans l'ensemble de référence sont celles créées par des combinaisons de sous-ensembles de deux (2) éléments [7]. Mais ce résultat, ne peut être interprété comme une justification pour éliminer complètement l'utilisation des autres catégories de sous-ensembles [48].

II.2.5. La Méthode de combinaison des solutions :

La méthode de combinaison des solutions (*Solution Combination Method*) est un élément de la Recherche Dispersée qui est très lié au contexte d'utilisation. Cette méthode dépend essentiellement de la manière de représenter les solutions dans le problème, et les contraintes exprimées.

Même s'il est possible de concevoir des procédures de combinaisons génériques, mais il est plus efficace de baser la conception sur des caractéristiques spécifiques du problème traité [47].

Mais malgré la spécificité de cette méthode, quelques principes généraux restent vérifiés : les meilleures solutions sont la plupart du temps générées par la combinaisons d'autres solutions de bonne qualité ; aussi, l'utilisation de paramètres aléatoires peut améliorer cette combinaison [48]. Il faut noter aussi que cette méthode peut retourner des solutions formées par des combinaisons convexes, de même que des combinaisons non-convexes, qui projettent les solutions créées en dehors des régions délimitées par les solutions parents, ce qui renforce l'aspect de diversification.

II.3. Algorithme général de la Recherche Dispersée :

Après que toutes les composantes de la Recherche Dispersée ont été présentées, un algorithme général de cette méthode peut être dressé comme suit :

Algorithme Recherche Dispersée ;
Début
P = ∅ ; *BestSet* = ∅ ; *DivSet* = ∅ ;
Tant que $|P| < p$ **faire**
Début
Générer une solution s par la méthode diversification ;
Appliquer à s la méthode d'amélioration pour obtenir s^* ;
Si $s^* \notin P$ alors $P = P \cup \{s^*\}$;
Fin ;

Tant que $|BestSet| < b_1$ **faire**
$BestSet = BestSet \cup \{s\}$ où $f(s) = \max_{t \in (P - RefSet)} \{f(t)\}$;
Tant que $|DivSet| < b_2$ **faire**
$DivSet = DivSet \cup \{s\}$ où $d_{\min}(s) = \max_{t \in (P - RefSet)} \{d_{\min}(t)\}$;

Répéter
Début
Générer les sous-ensembles de *RefSet* par la méthode de génération des sous-ensembles ;
Pour (chaque sous-ensemble i) **faire**
Début
Calculer les solutions combinées de i par la méthode de combinaisons ;
Pour (chaque solution s) **faire**
Si $(f(s) > f(s_{\min}))$ **alors** Remplacer $s_{\min}$ par s dans $BestSet$
/* où $f(s_{\min}) = \min_{s \in BestSet} (f(s))$ */
Sinon
Si $(d_{\min}(s) > d_{\min}(s_{div}))$ **alors** Remplacer s_{div} par s dans $DivSet$;
/* où $d_{\min}(s_{div}) = \max_{s \in DivSet} (d_{\min}(s))$ */
Fin ;
Fin ;
Jusqu'à ((*RefSet* ne change plus) où (*Conditions d'arrêts satisfaites*)) ;

Si (*Conditions d'arrêts non satisfaites*) **alors**
Garder *BestSet* et regénérer un nouveau ensemble *P* et refaire toutes les étapes de l'algorithme ;
Fin ;

Algorithme II.2 : Algorithme général de la Recherche Dispersée

II.4. Comparaison entre laRecherche Dispersée et les Algorithmes Génétiques :

Malgré que la Recherche Dispersée et les Algorithmes Génétiques appartiennent à la même famille, mais plusieurs traits les diffèrent. Après avoir présenté la méthode de Recherche Dispersée en détail, une comparaison avec les algorithmes génétiques s'impose, les principales différences sont contenues dans le tableau suivant :

Caractéristiques	La recherche Dispersée	Les algorithmes génétiques
Taille de la population	Petite, souvent de dépassant pas 20 solutions	Grande, pas moins de 100 en général
Génération de la population initiale	Suivant des critères de qualité et de diversité	Généralement aléatoire
Mise à jour de la population	Utilisant des règles déterministes combinant diversité et qualité	Utilisant le principe de '*la survie du meilleur*'
L'aspect aléatoire	Pratiquement inexistant, ou avec un caractère spécifique	Utilisé largement (sélection, mutation)
Sélection	Systématique	Aléatoire
Combinaison	2 ou plusieurs solutions	2 solutions exactement
Processus de recherche locale	Essentiel pour l'implémentation	Non utilisé dans le schéma général

Tableau II.1 : comparaison entre la Recherche Dispersée et les Algorithmes Génétiques

III- Exemple d'application de la Recherche Dispersée :

Pour mieux illustrer les différentes étapes de la Recherche Dispersée ,un exemple d'application de cette méthode est présenté pour la classe de problèmes d'optimisation suivants :

$$\textit{minimiser} \quad f(x)$$
$$\text{avec :} \quad l \le x \le u$$

où f est une fonction non linéaire de x. Pour l'exemple, la méthode sera appliquée à l'instance suivante :

$$\textit{minimiser} \quad 100\left(x_2 - x_1^2\right)^2 + (1-x_1)^2 + 90\left(x_4 - x_3^2\right)^2 + (1-x_3)^2 + 10{,}1\left((x_2-1)^2 + (x_4-1)^2\right) + 19{,}8(x_2-1)(x_4-1)$$
$$\text{avec :} \quad -10 \le x \le 10$$

Cette instance est le test n°6 présenté dans [54]. A titre d'information, une tentative de résolution de cette instance par un algorithme génétique a retourné la valeur de 0,0001333, après 500.000 itérations de l'algorithme. Trivialement, la valeur optimale est $f^* = 0$ pour le point $(x_1, x_2, x_3, x_4) = (1,1,1,1)$. Parmi les spécificité de cet exemple, c'est qu'il n'y a pas de notion de solution faisable ou non, puisqu'il n'y a qu'une simple contrainte unique. Aussi, la résolution ne démarre d'aucune solution existante.

a) Méthode de diversification :

La méthode de diversification emploie un processus aléatoire contrôlé pour générer des solutions. Ceci est effectué en divisant l'espace des solutions [-10,+10] en 8 intervalles de même taille. Ensuite, une solution est construite en deux étapes. Un premier choix aléatoire porte sur le sous-intervalle contenant la solution, et le second choix portera sur la valeur de la solution elle-même. La probabilité de choisir un sous-intervalle est proportionnelle au nombre de solution appartenant à cet intervalle, ceci est fait en associant un compteur à chaque intervalle, pour réduire les chance de sélection d'un intervalle contenant trop de solutions. Ensuite, une valeur est choisie aléatoirement à l'intérieur de

l'intervalle. Cette manière d'agir permet d'avoir une distribution assez uniforme des solutions considérées.

La taille adoptée pour l'ensemble de référence *RefSet* est $b = 20$, et donc un ensemble initial P de cardinal $p = 5 \times 20 = 100$ est généré.

	x_1	x_2	x_3	x_4	$f(x)$
01	-0,3828	5,0702	-0,9578	-2,1245	3.276,9
02	7,6893	1,0951	-0,8421	-6,3608	341.835,7
03	8,0235	2,8822	-9,2013	-1,4586	1.045.840,6
04	-1,1910	6,9440	1,6576	3,7290	3.898,2
05	6,1111	1,7527	-2,3644	3,1662	127.340,4
06	-4,3865	4,4766	9,2346	-3,2440	727.143,7
07	7,1459	3,6832	-2,8143	4,7062	225.877,3
08	0,0852	0,7816	0,5219	9,9347	9.231,6
09	-5,0430	-2,9260	7,7003	8,7945	310.183,8
10	9,9796	-6,0628	1,7781	7,1889	1.117.870,4

Tableau II.2 : Solutions de l'ensemble initial *P*

Dans le tableau ci-dessus, dix (10) des cents (100) solutions de *P* sont présentées. La valeur de la fonction *f* pour ces solutions varie de 3,2 à 1.566.668,6 avec une moyenne de 382.958,8, ce qui est très loin de la valeur optimale sachant qu'elle est nulle. Pour vérifier si cette méthode de génération garantie un certain seuil de diversité, un tableau de la distribution des solutions est dressé :

Intervalle	x_1	x_2	x_3	x_4
[-10 ;-7,5]	12	13	15	12
]-7,5 ;-5]	16	14	13	12
]- 5 ;-2,5]	13	11	8	15
]-2,5 ;0]	10	12	12	11
]0 ;2,5]	12	13	13	16
]2,5 ; 5]	09	11	12	09
]5 ;7,5]	16	13	13	14
]7,5 ; 10]	12	13	14	11

Tableau II.3 : Distribution des solutions générées

Les fréquences sont assez similaires pour chaque ligne et pour chaque variable, avec une valeur minimum égale à 08 et 16 comme valeur maximum.

b) Méthode d'amélioration :

La méthode d'amélioration est une simple recherche locale appliquée un nombre limité de fois. Cette procédure cherche dans le voisinage de chaque solution une meilleure solution, et s'arrête en trouvant un optimum local ou bien après un nombre d'itération fixé.

	x_1	x_2	x_3	x_4	$f(x)$
01	1,1072	3,5802	0,5322	-3,6145	1968,4
02	2,7893	1,5851	-0,3521	-6,8508	8756,2
03	3,1235	3,3722	-4,3013	-1,9486	41.751,8
04	-0,9910	6,5440	1,8576	2,7290	3.675,5
05	3,1711	0,7727	-1,8744	2,1862	8.798,6
06	-3,4065	3,0066	4,8246	-4,2240	75.601,0
07	6,1659	2,2132	1,5957	3,7262	128.508,4
08	-0,4548	-0,0284	2,9519	8,5847	450,2
09	-3,5730	-1,4560	3,2903	5,8545	22.541,0
10	5,0900	-4,5928	-2,6319	7,6789	93.138,2

Tableau II.4 : Solutions de l'ensemble initial *P* après amélioration

Après le passage de la procédure d'amélioration, la valeur de f varie de 14,7 à 422.148,2. Aucune solution ne semble converger vers l'optimum global $(1,1,1,1)$.

c) Méthode de mise à jour de l'ensemble de référence :

L'ensemble de référence *RefSet* contient des solutions de bonne qualité et des solutions diversifiées utilisées pour générer de nouvelles solutions. Cet ensemble est divisé en deux sous-ensembles : *BestSet* (de cardinal $b_1 = 3$) qui contient les solutions de la première catégorie, et *DivSet* (de cardinal $b_2 = 2$) qui contient les solutions diversifiées.

La construction de *BestSet* est la plus simple, il suffit de sélectionner les b_1 meilleures solutions.

	x_1	x_2	x_3	x_4	$f(x)$
(*a*)	0,5790	0,2287	1,1565	1,3474	3,2677
(*b*)	-0,8722	0,7412	1,1522	1,319	3,6427
(*c*)	0,1197	-0,0113	0,9765	0,8674	14,6714

Tableau II.5 : Les 3 meilleures solutions formant *BestSet*

Pour *DivSet*, des calculs sont nécessaires pour sa génération. Pour chaque solution *s* dans (*P-RefSet*), la distance minimale entre elle et tout les éléments de *RefSet* est calculée :

$$d_{\min}(s) = \min_{s' \in RefSet} \{d(s,s')\} \qquad \text{pour chaque } s \in (P - RefSet)$$

où $d(x,y)$ est la distance euclidienne entre les points *x* et *y*.

dans ce cas, la distance minimum pour une solution *s* serait :

$$d_{\min}(s) = \min\{d(s,a), d(s,b), d(s,c)\} \text{ où } a,b,c \in RefSet$$

La solution ayant la valeur $d_{\min}$ maximale est insérée dans *RefSet* (plus précisément dans *DivSet*), et les distance minimales du reste des solutions sont recalculées puisque *RefSet* a changé, c'est à dire, qu'après l'insertion de la solution (*x*), le calcul de la distance doit prendre en considération les solutions *a*, *b*, *c* et *x*.

	x_1	x_2	x_3	x_4	$f(x)$
(*x*)	1,0986	-7,5471	-7,7772	-6,0871	409.048,9
(*y*)	-7,3098	4,3925	-6,2441	-3,0843	399.943,7

Tableau II.6 : Les solutions diversifiées

Après que l'ensemble de référence initial est construit, la méthode de combinaison est appliquée pour générer de nouvelles solutions. L'ensemble de référence est mis à jour dynamiquement pendant cette méthode. Une solution nouvellement générée peut devenir membre de *RefSet* si :

- elle a une valeur de la fonction objectif *f* meilleur que la pire solution dans *BestSet*.
- elle a une valeur de la distance minimale $d_{\min}$ meilleur que la pire solution dans *DivSet*.

d) Méthode de génération des sous-ensembles :

Cette méthode a pour rôle de générer des sous-ensembles de *RefSet* utilisés pour la création de solutions combinées. Quatre types de sous-ensembles existent (voir §II.2.4), le tableau suivant contient toutes les combinaisons possibles de l'exemple en court :

Type 1	$\{a,b\},\{a,c\},\{a,x\},\{a,y\}$ $\{b,c\},\{b,x\},\{b,y\}$ $\{c,x\},\{c,y\}$ $\{x,y\}$
Type 2	$\{a,b,c\},\{a,x,b\},\{a,y,b\}$ $\{c,x,a\},\{c,y,a\}$ $\{x,y,a\}$
Type 3	$\{a,b,c,\mathbf{x}\},\{a,y,b,\mathbf{c}\}$ $\{x,y,a,\mathbf{b}\}$
Type 4	$\{a,b,c,x,y\}$

Tableau II.7 : Les sous-ensembles de l'ensemble de référence

e) Méthode de combinaison des solutions :

Cette méthode utilise les sous-ensembles générés précédemment pour combiner les éléments de chaque sous-ensemble afin de créer de nouvelles solutions. Généralement cette méthode est spécifique au problème, puisqu'elle est directement liée à la représentation des solutions. Dans ce cas, la méthode consistera à trouver des combinaisons linéaires des solutions de référence. Par exemple : pour les sous-ensembles du type 2, on peut définir les règles suivantes :

- C1 : $x = x' - r(x' - x'')$
- C2 : $x = x' + r(x' - x'')$
- C3 : $x = x'' - r(x' - x'')$
- C4 : $x = x'' + r(x' - x'')$

où : $x', x'' \in RefSet$

x est la nouvelle solution combinée

r est un réel aléatoire $\in]0,1[$

ces règles peuvent être combinées entre elles.

	r	x_1	x_2	x_3	x_4	$f(x)$
C1	0,0343	0,52910	0,24632	1,15635	1,34642	2,1465
C2	0,1848	0,84731	0,13394	1,15729	1,35265	36,9545
C3	0,6202	-1,77230	1,05908	1,14953	1,30139	442,5076
C4	0,8372	0,34278	0,31212	1,15580	1,34278	5,5454

Tableau II.8 : Les solutions combinées à partir des solutions (*a*) et (*b*)

La méthode d'amélioration est appliquée aux nouvelles solutions, la meilleure valeur obtenue après l'amélioration est pour la combinaison (C1) avec $f(0.6153\ ,\ 0.3756\ ,\ 1.2258\ ,\ 1.5050) = 0{,}4705$. Ensuite la méthode de mise à jour est appelée pour décider de l'insertion de ces solutions dans l'ensemble de référence. Ce processus est itéré jusqu'à l'atteinte d'un point fixe par l'ensemble de référence ou bien que les conditions d'arrêt soient satisfaites (valeur acceptable pour la fonction objectif, limite d'itération…).

IV- Le Chemin guidant '*Path Relinking*' :

L'approche du Chemin Reliant (*Path Relinking*) est une généralisation de la Recherche Dispersée. Le processus de génération des combinaisons linéaires des solutions d'un ensemble de référence peut être considéré comme une génération de chemins entre et autours ces solutions, et où les solutions sur tels chemins servent aussi comme sources pour produire des chemins supplémentaires. Cela mène à une conception plus générale de la signification de créer des combinaisons de solutions [28]. Ce formalisme est renforcé par le fait qu'un chemin entre deux solutions dans un espace de voisinage peut générer de nouvelles solutions partageant une partie significative des attributs contenus dans les solutions d'origines.

La recherche explore l'espace des solutions en créant des chemins menant de solutions initiales vers des solutions d'élites dites '*solutions guidantes*'. Les solutions initiales et les solutions guidantes sont appelées les '*solutions de référence*'. Pour générer de tels chemins, il suffit juste de sélectionner les déplacements qui respectent le rôle suivant : en démarrant d'une solution initiale, les déplacements doivent introduire progressivement les attributs des solutions guidantes (ou réduire la distance entre les attributs des solutions

initiales et les solutions guidantes). Les rôles des solutions initiales et solutions guidantes sont interchangeables.

La procédure essaie d'isoler les attributs qui apparaissent fréquemment dans les solutions de bonne qualité ; ensuite, insérer ces attributs dans de nouvelles solutions, ceci est fait en créant des incitations qui favorisent la sélection des déplacements vers les solutions contenants ces attributs. Mais au lieu d'encourager simplement l'inclusion de tels attributs, l'approche dépasse ces considérations à l'objectif de choisir des déplacements qui introduisent les attributs des solutions guidantes, afin de créer une bonne composition d'attributs dans la solution courante.

La création des chemins reliants deux solutions données, se fait par la génération de séquences de solutions reliant ces deux solutions. Pour réduire le nombre de chemins à considérer, le choix de la solution suivante à la solution courante se fait en réduisant au maximum le nombre de déplacements à la solution guidante. Même si ce critère est appliqué sans exception, il permet un nombre considérable de choix pour générer la solution suivante à chaque étape.

V- Conclusion :

La Recherche Dispersée est une méthode évolutive dont les mérites ne se sont faits connaître que récemment Il est impossible avec la vue limitée de ce chapitre de présenter tous ses aspects. Un des points forts de cette méthode est sa souplesse et la possibilité de la connecter à d'autres méthodes de recherche.

Dans le chapitre suivant, un nouveau domaine d'utilisation pour cette méthode sera présenté : la cryptologie. En effet, l'utilisation des méta-heuristiques dans la cryptanalyse est une des nouvelles tendances dans ce domaine. Des schémas de cryptanalyse des chiffres simples utilisant la Recherche Dispersée seront présentés.

Chapitre IV : *Cryptanalyse des chiffres simples*

I- Introduction :

Une des techniques les plus simples de chiffrement est la substitution. Par sa simplicité et son efficacité, ce type de chiffres à coté de la transposition, a dominé la cryptographie durant des siècles. Les chiffres simples ont longuement résisté aux efforts des cryptanalystes jusqu'au IX^{ème} siècle quand AL KINDI met au point *l'Analyse des fréquence* dans son traité intitulé « *Manuscrit sur le déchiffrement des messages cryptographiques* ». Cette technique est depuis la base pour toute tentative de cryptanalyse des chiffres simples.

Malgré que ces systèmes de chiffrement ne sont plus d'actualité, mais beaucoup de travaux les traitent encore, ceci vient du fait que la plupart des cryptosystèmes modernes adoptent la substitution et la transposition comme moyens pour incorporer la *confusion* et la *diffusion*, afin gommer les redondances dans un texte en clair [63].

La cryptanalyse des chiffres simples n'est pas un but en soi, mais plutôt un moyen pour mettre au point et estimer l'efficacité des techniques de cryptanalyse, avant de passer à des systèmes beaucoup plus complexes.

La tendance actuelle dans le domaine de la cryptologie est d'utiliser les techniques issues du domaine de l'intelligence artificielle pour casser des cryptosystèmes, aussi bien que pour élaborer de nouveaux systèmes cryptographiques. Certains préconisent que les méthode de résolution connues en Intelligence Artificielle peuvent apporter une grande attribution dans le domaine de la cryptologie, et que le pouvoir des techniques méta-heuristiques dans les applications cryptologiques est très sous estimé [11]. Encore, l'utilisation des outils du raisonnement automatisé pour l'analyse des cryptosystèmes semble être la prochaine étape vers une plus grande assurance de sécurité [50].

On trouve beaucoup de travaux récents présentant des schémas de cryptanalyse utilisant des méta-heuristiques : les premiers à être utilisés sont les Algorithmes génétiques [51][65][4], on trouve même un Algorithme génétique parallèle [19]. D'autres schémas de cryptanalyse ont utilisé le Recuit Simulé [25][27] et la Recherche Tabou [17], et récemment l'ACO [61] (méta-heuristique de colonie de fourmis). D'autres études ont même tenté d'établir une comparaison entre ces techniques [11][20].

Dans ce chapitre, nous allons essayer de dresser un schéma de cryptanalyse des systèmes de chiffrement simples en utilisant la Recherche Dispersée (*Scatter Search*). Tout d'abord, le problème de cryptanalyse est formalisé sous la forme d'un problème d'optimisation, ensuite la méta-heuristique est appliquée pour le résoudre. Nous présenterons aussi une implémentation d'un algorithme génétique pour pouvoir ensuite établir une étude comparative entre ces deux techniques évolutives.

L'implémentation de la Recherche Dispersée présentée dans ce chapitre n'est pas spécifique seulement à la cryptanalyse, mais peut être utilisé comme un cadre général pour l'utilisation de la Recherche Dispersée pour les problèmes de permutations.

II- Cryptanalyse de systèmes de chiffrement par substitution simple :

Le chiffrement par substitution simple procède en général en remplaçant chaque symbole du texte en clair par un symbole dans le texte chiffré. Ce processus garde intact les statistiques de distribution des caractères. Pour cela, la plupart des schémas de cryptanalyse des substitutions simples tentent d'utiliser cette caractéristique en reliant les statistiques des distributions des caractères du texte crypté avec les statistiques connues de la langue du texte en clair. A chaque langue correspond une distribution des fréquences unique et généralement stable pour de longs textes (voir Annexe I).

Le paramètre le plus important dans ce schéma de cryptanalyse est la longueur des textes chiffrés disponibles ; du moment où l'attaque se base sur les statistiques des distributions, ces statistiques se rapprocheront des statistiques réelles autant que le texte chiffré est long.

Avant d'entamer la présentation du schéma de cryptanalyse, quelques notions doivent être bien explicitées : tout d'abord la représentation des solutions ; ensuite, la manière de mesurer de la qualité des solutions, et aussi la distance entre ces solutions. Ces deux grandeurs (mesure de qualité et de distance) sont les composantes de base de la procédure de la Recherche Dispersée.

II.1. Représentation des solutions :

Les solutions dans notre cas sont des clefs, et donc des permutations de l'alphabet du texte en clair considéré. Les problèmes de permutations en général peuvent être divisés en deux grandes classes suivant le cas où la fonction objectif est plus sensible aux positions absolues des éléments dans les solutions (permutations), ou à leurs positions relatives. On distingue donc :

- Les problèmes de *A*-permutations : pour lesquels la position absolue d'un élément est plus importante.
- Les problèmes de *R*-permutations : où les positions relatives des éléments par rapport aux autres est plus importante.

Notons qu'un même problème peut appartenir à ces deux classes en même temps suivant la représentation des solutions adoptée, comme c'est la cas pour le problème du voyageur du commerce [8]. La recherche d'une clef d'une substitution est un problème de *A*-permutations dans la mesure où la position exacte de chaque élément est primordiale.

Les solutions sont donc des permutations de l'alphabet. Chaque solution correspond aux substitutions utilisées pour chiffrer ce texte. Les caractères de l'alphabet sont triés suivant l'ordre décroissant de leur fréquence d'apparition (voir Annexe I), par exemple en français on a

(_,e,s,a,n,t,i,r,u,l,o,d,c,m,p,v,q,f,g,h,b,x,j,y,z,k,w). Donc si la solution est (v,y,t,l,a,k,g,m,z,x,n,w,i,e,u,o,p,_,b,q,c,r,s,j,d,h,f) ceci signifie que le caractère '_' a été subsitué par 'v', le 'e' par le 'y',…etc. L'utilité de cet ordre apparaîtra clairement dans la méthode de combinaison.

II.2. Mesure de la qualité des solutions :

Dans ce contexte, la solution à rechercher est la clef du système de chiffrement, une clef est représentée par une permutation des symboles de l'alphabet utilisé. Cette clef est appliquée au texte crypté intercepté, puis l'évaluation de sa qualité se fera par la formule suivant :

$$\begin{aligned} f_C(K) = \alpha_1 \cdot & \sum_{i \in \mathsf{A}} \left| P(i) - C(i) \right| \\ + \alpha_2 \cdot & \sum_{i_1, i_2 \in \mathsf{A}} \left| P(i_1, i_2) - C(i_1, i_2) \right| \\ + \alpha_3 \cdot & \sum_{i_1, i_2, i_3 \in \mathsf{A}} \left| P(i_1, i_2, i_3) - C(i_1, i_2, i_3) \right| \\ + & \cdots \end{aligned} \qquad \text{(IV.1)}$$

d'où la forme générale de la fonction :

$$f_C(K) = \sum_{j=1}^{j=k} \left(\alpha_j \cdot \sum_{i_1, \ldots, i_j \in \mathsf{A}} \left| P(i_1, \ldots, i_j) - C(i_1, \ldots, i_j) \right| \right) \qquad \text{(IV.2)}$$

où :

- A : Alphabet du texte en clair
- α_i : des constantes réelles où chaque $\alpha_i \in [0,1]$ et $\sum \alpha_i = 1$.
- $P(i_1, \ldots, i_n)$: fréquence standard du *n*-gramme $(i_1, \ldots, i_n)$.
- $C(i_1, \ldots, i_n)$: fréquence du *n*-gramme $(i_1, \ldots, i_n)$ dans le texte crypté.

Cette formule fournit une estimation de la distance entre les fréquences des *n*-grammes dans le texte crypté intercepté et les fréquence du langage du texte en clair. Le but donc est de minimiser cette fonction pour obtenir un texte ayant des statistiques les plus proches des statistiques réelles. Le rôle des constantes

(α_i) est de permettre de favoriser certains *n*-grammes par rapport aux autres en leur donnant un plus grand poids.

Toutes les attaques connues contre les chiffres simples n'ont allé au-delà de $j = 3$ (c'est à dire qu'elles se sont limitées à l'utilisation des statistiques des unigrammes, bigrammes et trigrammes uniquement) [17][19][27][61], même les trigrammes sont rarement utilisés à cause du nombre important de données à traiter ($27^3 \approx 2 \times 10^4$). MATHEWS [51] proposa une manière d'agir qui permet d'utiliser efficacement les statistiques des bigrammes et des trigrammes en évitant de prendre ne considération toutes les informations les concernant. Au lieu d'utiliser toutes les statistiques des *n*-grammes, juste sont considérées les *n*-grammes les plus fréquent et les moins fréquents ; en attribuant aux *n*-grammes apparaissant le plus dans la langue considérée des poids positifs, et à ceux qui n'apparaissent jamais ou que très rarement des poids négatif.

Le problème posé avec l'heuristique précédente (formule IV.1) est que parfois plusieurs clefs peuvent fournir la valeur optimale, ou pire, la clef associée à la valeur minimum n'est pas la bonne clef ; pour cela, le schéma de cryptanalyse sera renforcé par une seconde heuristique.

Une autre heuristique est utilisée pour évaluer les solutions, celle-ci valide une clef donnée en essayant de calculer le nombre de mots reconnus du texte décrypté avec cette clef. Pour valider une clef *K*, cette clef est appliquée au texte crypté intercepté $C = E_K(P)$ pour obtenir un texte P^*. P^* est le texte en clair correspondant réellement à P si le nombre de mots appartenant à P* et qui sont présent dans une liste pré-définie (une sorte de dictionnaire formé par les mots les plus répandus ou un analyseur lexical) est maximum.

Donc, cette heuristique peut être définie comme suit

$$Mot\_Reconnu\left(M_P^l\right) = \begin{cases} 1 ; & \text{si } M_P^l \in \text{Dictionnaire} \\ 0 ; & \text{sinon} \end{cases} \qquad \text{(IV.3)}$$

Pour chaque sous-chaîne M_P^l de longueur l du texte P, la fonction *Mot_Reconnu* renvoie la valeur 1 si ce mot est présent dans le dictionnaire, 0 dans le cas contraire. Donc, la valeur de cette fonction heuristique pour un texte P de longueur totale L est :

$$Mot(P) = \frac{1}{L} \sum_{M_P^l \subset P} \left(l \times Mot\_Reconnu\left(M_P^l\right)\right) \qquad \text{(IV.4)}$$

Cette heuristique est le rapport de la somme des longueurs des mots reconnus sur la longueur totale du texte ; notons que la longueur des mots est bornée par des valeurs qui diffère d'une langue à une autre. Notons aussi qu'il n'est pas nécessaire d'établir une liste exhaustive de tout les mots de la langue considérée, ce qui est presque impossible et surtout inutile, puisque la majorité des mots utilisés provient généralement d'un vocabulaire réduit. Cette heuristique évite ainsi toute intervention humaine pour valider la détermination de la clef. On peut remarquer que *Mot* peut être vue comme étant la forme extrême de la fonction (IV.2) dans la mesure ou elle dépasse l'utilisation des *n*-grammes à l'utilisation de mots entiers.

Il est possible d'utiliser cette heuristique (*Mot*) comme fonction objectif dans l'opération de cryptanalyse, au lieu de la formule (IV.2) qui estime les différence entre les fréquences ; mais le temps de calcul nécessaire à son exécution la rend très coûteuse. Pour cela, cette heuristique est utilisée juste pour valider l'authenticité d'une clef, ou pour départager entre plusieurs clefs dans la mesure où la fonction objectif ne le permet pas toujours.

II.3. Mesure de la distance entre les solutions :

La manière d'exprimer la distance entre deux solutions données est un élément clef dans l'implémentation de la Recherche Dispersée, puisque l'aspect de diversification des solutions est essentiellement basé sur cette grandeur.

Dans le contexte des problèmes de permutations, on trouve une formule donnée par CAMPOS [08] pour calculer la distance entre deux solutions données :

- Pour les problèmes de *A*-permutation : la distance entre deux solutions $p=(p_1, p_2,..., p_n)$ et $q=(q_1,q_2,...,q_n)$ est donnée par la formule suivante :

$$d(p,q)=\sum_{i=1}^{n}|p_i - q_i| \qquad \text{(IV.5)}$$

- La distance pour les problèmes de *R*-permutation est définie par :

$$d(p,q)=\text{le nombre de fois où } p_i \text{ ne suit pas immédiatement } p_{i+1} \text{ dans } q, \text{ pour } i=\overline{1, n-1} \qquad \text{(IV.6)}$$

Dans notre cas, qui consiste en un problème de *A*-permutation, la mesure de distance fournie par la formule (IV.5) est imprécise et grossière, car la formule manque de sémantique ; et par conséquent, cette mesure ne reflète pas le degré de différence entre les solutions considérées. Ceci est mieux illustré dans l'exemple IV.1 où deux solutions s_2 et s_3 sont à distance égale de la 1ère solution, malgré que la seconde solution paraît plus proche que la 3ème, puisqu'elles ne diffèrent que par deux positions.

Exemple IV.1 :

$s_1=(2,3,1,4)$

$s_2=(2,3,4,1)$ $\quad d(s_1,s_2)=0+0+3+0=6$

$s_3=(3,4,2,1)$ $\quad d(s_1,s_3)=1+1+1+3=6$

$s_4=(3,4,1,2)$ $\quad d(s_1,s_4)=1+1+0+3=4$

Encore, s_4 est la plus proche de s_1 (toujours en terme de la fonction de calcul de la distance) mais elle la solution qui présente le plus de différence. Pour palier à ce problème, nous allons essayer de proposer une formule pour la distance, exprimant mieux les différences entre les solutions (les permutations).

La fonction de calcul de la distance doit refléter au mieux le degré de ressemblance (et de différence) entre les permutations ; c'est-à-dire, le nombre d'éléments différents, mais aussi en prenant en compte qu'une permutation de deux éléments voisins doit engendrer une distance plus petite qu'une

permutation de deux éléments éloignés dans une solution donnée. Par exemple, en considérant la solution $s_1 = (1,2,3,4)$, la solution $s_2 = (2,1,3,4)$ doit être plus proche d'elle que $s_3 = (4,2,3,1)$, malgré que toutes les deux ont été obtenues en appliquant une simple permutation des éléments de s_1. Une autre caractéristique de la fonction de distance, qui est triviale mais importante, est qu'elle soit symétrique, c'est-à-dire : $d(p,q) = d(q,p); \forall p,q$.

Une définition de la distance entre deux solutions (permutations) peut être donnée en considérant cette distance comme étant le nombre minimum de permutations d'éléments voisins nécessaires pour transformer une solution des en l'autre.

Donc, pour deux solutions $p = (p_1, p_2, ..., p_n)$ et $q = (q_1, q_2, ..., q_n)$, la distance est donnée par la formule :

$$d(p,q) = \text{le nombre de permutations de } p_i \text{ et } p_{i+1} \quad \left(i = \overline{1, n-1}\right) \text{ pour obtenir } p = q \qquad \text{(IV.7)}$$

Cette définition est mieux illustrée par l'exemple suivant :

Exemple IV.2 :

- En considérant les solutions suivantes : $s_1 = (1,2,3,4)$, $s_2 = (2,1,4,3)$ et $s_3 = (4,2,3,1)$, on a les distances suivantes :
  - $s_2 = (\underline{2,1},4,3) \rightarrow (1,2,\underline{4,3}) \rightarrow s_1 = (1,2,3,4) \Rightarrow \boxed{d(s_1, s_2) = 2}$
  - $s_3 = (\underline{4,1},3,2) \rightarrow (1,4,\underline{3,2}) \rightarrow (1,\underline{4,2},3) \rightarrow (1,2,\underline{4,3}) \rightarrow s_1 = (1,2,3,4)$
    $\Rightarrow \boxed{d(s_1, s_3) = 4}$
- Et en reprenant les solutions utilisées dans l'exemple (IV.1), nous obtenons les distances suivantes :
  - $s_2 = (2,3,\underline{4,1}) \rightarrow s_1 = (2,3,1,4) \Rightarrow \boxed{d(s_1, s_2) = 1}$

- $s_3 = (3,\underline{4,2},1) \rightarrow (\underline{3,2},4,1) \rightarrow (2,3,\underline{4,1}) \rightarrow s_1 = (2,3,1,4) \Rightarrow \boxed{d(s_1, s_3) = 3}$
- $s_4 = (3,4,\underline{1,2}) \rightarrow (3,\underline{4,2},1) \rightarrow (\underline{3,2},4,1) \rightarrow (2,3,\underline{4,1}) \rightarrow s_1 = (2,3,1,4)$
  $\Rightarrow \boxed{d(s_1, s_4) = 4}$

Cette fonction exprime mieux la distance que celle définie en (IV.5), et évite les anomalies vues dans l'exemple (IV.1). Dans le tableau ci-dessous, pour une solutions $s_0 = (1,2,3,4)$ formée par 4 éléments, le reste des solutions possibles $(4!-1 = 23)$ sont classées suivant leur distance de cette solution :

s_0	$d = 1$	$d = 2$	$d = 3$	$d = 4$	$d = 5$	$d = 6$
(1,2,3,4)	(2,1,3,4) (1,3,2,4) (1,2,4,3)	(2,3,1,4) (2,1,4,3) (3,1,2,4) (1,3,4,2) (1,4,2,3)	(3,2,1,4) (2,3,4,1) (2,4,1,3) (3,1,4,2) (1,4,3,2) (4,1,2,3)	(3,2,4,1) (2,4,3,1) (4,2,3,1) (3,4,1,2) (4,1,3,2)	(3,4,2,1) (4,2,3,1) (4,3,1,2)	(4,3,2,1)

Tableau IV.1 : Toutes les permutations possibles classées suivant leur distance de s_0

On peut remarquer que la solution la plus éloignée de s_0 est celle obtenue en inversant l'ordre des éléments de s_0.

En général, pour une solution de taille *n*, on a :

- L'espace des solutions est formé par *n*! solutions (toutes les permutations possibles de *n* éléments).
- Chaque solutions possède exactement (*n*-1) voisins, où un voisin est une solution se trouvant une distance égale à 1, c'est-à-dire, en permutant deux éléments p_i et p_{i+1} pour $i = \overline{1, n-1}$.
- La distance maximale pouvant exister entre deux solutions données est de $d_{\max} = \frac{n(n-1)}{2}$: en considérant le cas le plus extrême où les

éléments de la 2ème solutions suivent l'ordre inverse de la 1ère, le 1er élément nécessite (*n*-1) permutations pour retrouver son emplacement, le 2me nécessite (*n*-2) et ainsi de suite jusqu'au dernier. Donc la distance maximale est donnée par : $d_{\max} = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2}$.

Distance ($S = (s_1,...,s_n)$: solution**) : entier ;**
Début
  $d = 0$;
  Pour $i = 1$ **à** N-1 **faire**
    Pour $j = 1$ **à** $N - i$ **faire**
      Si $(Pos_{S_1}(S_2[j]) > Pos_{S_1}(S_2[j+1]))$ **alors**
        Début
          $x = S_2[j]$; $S_2[j] = S_2[j+1]$; $S_2[j+1] = x$;
          $d = d + 1$;
        Fin ;
  Retourner(d) ;
Fin ;

Algorithme IV.1 : Calcul de la distance entre deux permutations

II.4. Application de la Recherche Dispersée pour la cryptanalyse des substitutions :

Dans cette section, nous allons présenter les implémentations des méthodes de la Recherche Dispersée, certaines sont plus spécifiques au problème traité que d'autres. L'évaluation de la qualité des solutions se fait par la fonction (IV.2) et la distance les séparant par la formule (IV.7). L'algorithme est itéré jusqu'à ce que l'ensemble de référence ait atteint un point fixe, ou bien que le temps alloué est écoulé, ou bien qu'une solution satisfaisante est trouvée. A chaque itération, l'heuristique *Mot* (IV.4) évalue les nouvelles solutions insérées dans l'ensemble de référence. Notons qu'il n'est pas obligatoire d'avoir une clef entièrement juste pour avoir un texte lisible (du fait que certains caractère sont rarement utilisés). Si aucune solution acceptable n'est trouvée, l'algorithme

reprend dés le début, c'est à dire de la méthode de génération en prenant comme semence les meilleures solutions dans l'ensemble de référence.

II.4.1. Méthode d'amélioration :

La méthode d'amélioration tente d'améliorer la solution en entrée. Cette méthode est basée sur une simple procédure de recherche locale explorant le voisinage de la solution. Dans ce contexte, une solution voisine consiste en une solution obtenue par une permutation de deux éléments voisins de la solution courante. La recherche de la meilleure amélioration pour chaque solution est très coûteuse [52], donc cette méthode se limitera à explorer la 1ère amélioration et s'arrêter quand un optimum local est trouvé ou bien après dépassement d'un seuil fixé d'itérations.

Méthode d'amélioration ($S=(s_1,...,s_n)$: solution**) : Solution ;**
Début
1) $Iter=0$;
2) $i=1$;
3) **Répéter**
   $S'=(s_1,...,s_{i+1},s_i,...,s_n)$; //permutation de s_i et de s_{i+1}
   $i=i+1$;
   $iter=iter+1$;
   Tant que $((f(S')>f(S))$ et $(i\le n-1)$ et $(iter\le MaxIter))$;
4) **Si** $(f(S')\le f(S))$ **alors**
   Début
   $S=S'$;
   Aller à (2) ;
   Fin ;
5) Retourner(S) ;
Fin ;

Algorithme IV.2 : Méthode d'amélioration

où :

- *MaxIter* : nombre d'itérations maximum de la procédure ;

- f : fonction objectif à minimiser ;

II.4.2. Méthode de génération de diversification :

La méthode de génération de diversification a une grande importance dans la Recherche Dispersée, puisque toutes les étapes suivantes se baseront sur les solutions générées par le générateur de cette méthode. L'objectif de la méthode est de trouver le juste équilibre entre la qualité et la diversité des solutions générées.

Le générateur qui va être utilisé génère la population initiale par deux approches différentes, où chacune génère une partie de cette population :

- La 1ère approche utilise une solution de bonne qualité existante (solution semence) pour la génération, elle parcoure le voisinage de cette solution (les solutions se trouvant à une distance relativement petite de cette solution), mais en évitant les voisins immédiats puisque ceux-là seront parcourus par la procédure d'amélioration).

  La solution semence peut être obtenue par une tentative de résolution précédante, ou bien suivant une heuristique comme l'heuristique suivante : ordonner les caractères suivant l'ordre décroissant de leurs fréquences d'apparition dans le texte crypté, ceci permet de minimiser la différence des fréquences des unigrammes. Cette 1ère approche peut d'obtenir des solutions présentant une bonne qualité.
- La 2ème approche a pour but de créer des solutions très éloignées les unes des autres, et utilise un processus aléatoire contrôlé, comme celui vu dans l'exemple d'application de la Recherche Dispersée (voir Chapitre III §III.a).

Toutes les solutions générées seront passées par la procédure d'amélioration avant d'être insérées dans l'ensemble initial. Le but à atteindre est trouver une partition ce cet ensemble (taille de chacune des trois parties) qui combine qualité et diversité. On peut considérer que la 1ère approche se base juste sur la qualité, et la 2ème juste sur la diversité.

Générateur 1 (S_0 : solution semence) ;
Début
$P_1 = \varnothing$;
Répéter
Générer une solution *S* telle que $d(S_0, S) = d$;
// permuter *d* fois deux éléments voisins de S_0
*S** = Méthode d'amélioration (*S*) ;
Si ($S^* \notin P_1$) **alors** $P_1 = P_1 \cup \{S^*\}$;
Tant que ($|P_1| < TaillePopulation/3$) ;
Si (on ne peut plus générer de solution *S*) **alors** $d = d + 1$;
Fin ;

Algorithme IV.3 : 1er Générateur de diversification

avec :

- *d* : distance minimum entre la solution semence et ses voisins pris en considération.
- *TaillePoppulation* : taille totale de l'ensemble initial *P*.

Générateur 3 ;
Début
$P_3 = \varnothing$;
Diviser l'alphabet en *p* sous-ensembles égaux ; // à un élément près
Répéter
$S = (\ )$;
$CompteurChoix = (1,2,...,p)$;
Répéter
Choisir *Choix* parmi les éléments de *CompteurChoix* ;
Pour *i* =1 à *p* **faire**
Si ($i \neq Choix$) **alors**
$CompteurChoix = CompteurChoix + Choix$;
Choisir un caractère *c* parmi les éléments de $SousEnsemble_{Choix}$;
$SousEnsemble_{Choix} = SousEnsemble_{Choix} - \{c\}$;
$S = S + \{c\}$;
jusqu'à (*S* est complète**) ;**
*S** = Méthode d'amélioration (*S*) ;
Si ($S^* \notin P_3$) **alors** $P_3 = P_3 \cup \{S^*\}$;
Tant que ($|P_3| < TaillePopulation/3$) ;
Fin ;

Algorithme IV.4 : 2ème Générateur de diversification

avec :

- *CompteurChoix* : est utilisé pour pénaliser les sous-ensembles déjà choisis, en ajoutant les indices des sous-ensembles non choisis à chaque choix.

II.4.3. Méthode de mise à jour de l'ensemble de référence :

La Méthode de mise à jour de l'ensemble de référence est appelée au début de l'algorithme après la génération de l'ensemble initial pour sélectionner les solutions allant former l'ensemble de référence *RefSet* ; et elle est appelée aussi après la méthode de combinaison pour mettre à jour *RefSet* après la création de nouvelles solutions combinées par cette dernière.

La mise à jour de l'ensemble de référence peut être statique ou dynamique. La mise à jour dynamique est plus 'agressive' en terme d'incorporation de

solutions de bonne qualité (voir Chapitre III §II.2.3), et donc, accélère la convergence de l'ensemble de référence.

Dans cette implémentation, pour éviter la grande influence des solutions de qualité dans la phase initiale, une méthode de mise à jour statique est adoptée ; ensuite, après un nombre *InitialIter* fixé d'itérations, la mise à jour devient dynamique, puisque dans la majorité des cas, l'incorporation de solutions diversifiées à la fin de la procédure ne change en aucun cas l'optimum auquel elle va convergé, mais ne fait juste que ralentir cette convergence [48].

II.4.4. Méthode de génération des sous-ensembles :

Cette méthode a pour but de générer des sous-ensembles de l'ensemble de référence qui seront utilisés par la prochaine procédure pour la création de solutions combinées. Quatre types de sous-ensembles existent (voir Chapitre III §II.2.4).

- **Type 1 :** tous les sous-ensembles formés de deux (2) solutions, ce qui fait exactement $b(b-1)/2$ sous-ensembles, où b est le cardinal de l'ensemble de référence.
- **Type 2 :** chaque sous-ensemble parmi la catégorie précédante sera augmenté par la meilleure solution n'appartenant pas à ce sous-ensemble, ce qui fait $(b-1)(b-2)/2$ sous-ensembles.
- **Type 3 :** les sous-ensembles de trois (4) éléments formés par les sous-ensembles de deux (3) éléments augmentés par la meilleure solution non incluse dans ces sous-ensembles, ce qui fait $(b-2)(b-3)/2$ sous-ensembles.
- **Type 4 :** les sous-ensembles formés par les i meilleures solutions, avec $i = \overline{5,b}$, ce qui fait $(b-4)$ sous-ensembles.

Générateur Type 1 ;
Début
$Type1 = \varnothing$;
Pour $i = 1$ **à** $b - 1$ **faire**
Pour $j = i + 1$ **à** b **faire**
$Type1 = Type1 \cup \{(RefSet[i], RefSet[j])\}$;
Fin ;

Algorithme IV.5 : 1er Générateur de sous-ensembles

où :

- $RefSet[i]$: dénote la $i^{ème}$ solution de l'ensemble de référence.

Générateur Type 2 ;
Début
$Type2 = \varnothing$;
Pour $i = 2$ à $b - 1$ **faire**
Pour $j = i + 1$ à b **faire**
$Type2 = Type2 \cup \{(RefSet[1], RefSet[i], RefSet[j])\}$;
Fin ;

Algorithme IV.6 : 2ème Générateur de sous-ensembles

Générateur Type 3 ;
Début
$Type3 = \varnothing$;
Pour $i = 3$ à $b - 1$ **faire**
Pour $j = i + 1$ à b **faire**
$Type3 = Type3 \cup \{(RefSet[1], RefSet[2], RefSet[i], RefSet[j])\}$;
Fin ;

Algorithme IV.7 : 3ème Générateur de sous-ensembles

Générateur Type 4 ;
Début
$Type4 = \varnothing$;
Pour $i = 5$ à b **faire**
$Type4 = Type4 \cup \{(RefSet[1], RefSet[2], ..., RefSet[i])\}$;
Fin ;

Algorithme IV.8 : 4ème Générateur de sous-ensembles

II.4.5. Méthode de combinaison des solutions :

La méthode de combinaison est un élément de la Recherche Dispersée fortement lié au contexte d'application. Cette méthode doit générer –à partir des éléments de chaque sous-ensemble créé précédemment- une ou plusieurs solutions. Dans le cas des problèmes de permutations, la notion de combinaison linéaire utilisé dans la forme générale de la Recherche Dispersée n'a aucun sens.

Dans notre cas, la méthode de combinaison adoptée utilise un mécanisme de vote présentant des similitudes avec celui utilisé dans [8]. La procédure parcourt chaque solution à combiner de gauche à droite (suivant l'ordre établi dans §II.1). La nouvelle solution est construite élément par élément : à chaque étape, le mécanisme de vote détermine l'élément suivant à ajouter. Le nombre de voix accordées par une solution à son élément dépend de la position de cet élément dans sa solution. Par exemple, Un élément se trouvant dans la 1ère position d'une solution, et après 3 itérations ne figurant pas encore dans la nouvelle solution va recevoir 3 voix de sa solution.

Au début, chaque solution offre une voix à son premier élément. L'élément élu est celui ayant obtenu un plus grand nombre de voix. Au cas où plusieurs éléments aient le même score, celui correspondant à la solution de meilleure qualité (en terme de fonction objectif) sera choisi, et les autres verront leur scores augmenter.

Méthode de Combinaison ($S_1,...,S_p$: solution**) : Solution ;**

Var

Vote : Tableau [1..*N*] d'entiers initialisés à 0 ;

F : Tableau [1..*N*] de réels initialisés à 0 ;

AncienElement : Tableau [1..*p*] de listes de caractères ;

Début

Pour $i=1$ **à** *N* **faire**

début

Pour $j=1$ **à** p **faire**

Si $(S_j[i] \notin \{S_{New}[1],...,S_{New}[i-1]\})$ **alors**

Début

Déterminer $k \setminus S_j[i] = a_k$;

// $S_j[i]$ est le $k^{ème}$ caractère de l'alphabet

Vote [*k*]++ ;

Si $(f(S_j) > F[k])$ **alors** $F[k] = f(S_j)$;

Fin ;

Déterminer $m \setminus Vote[m] = \underset{j=1}{\overset{N}{Max}}(Vote[j])$ et $F[m] = \underset{j=1}{\overset{N}{Max}}(F[j])$;

$S_{New}[i] = a_m$;

$Vote[m] = 0$; $F[m] = 0$; //éliminer cet élément des prochains votes

Pour $j=1$ **à** p **faire**

Début

Retirer a_m de la liste $AncienElement[j]$;

Pour (chaque élément *c* dans $AncienElement[j]$**) faire**

Vote [*k*]++ \ $c = a_k$

Si $(S_j[i] \notin \{S_{New}[1],...,S_{New}[i]\})$ **alors**

Ajouter $S_j[i]$ dans la liste $AncienElement[j]$;

Fin ;

Fin ;

Retourner (S_{New}) ;

Fin ;

Algorithme IV.9 : méthode de combinaison des solutions

où :

- *Vote* : contient les scores de vote.
- *F* : contient pour chaque élément, la valeur maximale de la fonction objectif obtenue par les solutions à lesquelles il appartient.
- *AncienElement* : contient pour chaque solution, la liste des éléments contenus dans cette solution n'étant pas encore élus.
- $(a_1,...,a_N)$: l'alphabet considéré.
- $S_j[i]$: dénote le $i^{ème}$ élément de la $j^{ème}$ solution.

On peut varier les combinaisons, en utilisant une seconde méthode de combinaison similaire à la précédante, mais qui parcours les solutions de droite à gauche, en adoptant toujours le même mécanisme de vote. Malgré qu'apparemment, elle ne présente pas de grande différences avec la $1^{ère}$ méthode, mais beaucoup de cas, elle fournit des solutions différentes, surtout pour les sous-ensembles du $3^{ème}$ et du $4^{ème}$ type.

Méthode de Combinaison ($S_1,...,S_p$: solution**) : Solution ;**

Début

Pour $i = N$ **à** 1 **faire**

Début

Pour $j = 1$ **à** p **faire**

Si $(S_j[i] \notin \{S_{New}[i+1],...,S_{New}[N]\})$ **alors**

Début

Déterminer $k \setminus S_j[i] = a_k$;

// $S_j[i]$ est le $k^{ème}$ caractère de l'alphabet

Vote [*k*]++ ;

Si $(f(S_j) > F[k])$ **alors** $F[k] = f(S_j)$;

Fin ;

Déterminer $m \setminus Vote[m] = \underset{j=1}{\overset{N}{Max}}(Vote[j])$ et $F[m] = \underset{j=1}{\overset{N}{Max}}(F[j])$;

$S_{New}[i] = a_m$;

$Vote[m] = 0$; $F[m] = 0$; //éliminer cet élément des prochains votes

Pour $j = 1$ **à** p **faire**

Début

Retirer a_m de la liste $AncienElement[j]$;

Pour (chaque élément *c* dans $AncienElement[j]$**) faire**

Vote [*k*]++ \ $c = a_k$

Si $(S_j[i] \notin \{S_{New}[i],...,S_{New}[N]\})$ **alors**

Ajouter $S_j[i]$ dans la liste $AncienElement[j]$;

Fin ;

Fin ;

Retourner (S_{New}) ;

Fin ;

Algorithme IV.10 : $2^{ème}$ méthode de combinaison des solutions

II.5. Utilisation d'un algorithme génétique pour la cryptanalyse des substitutions :

Dans cette section, nous présenterons une implémentation simple d'un Algorithme Génétique pour la cryptanalyse d'un système de chiffrement par substitution. Les algorithmes génétiques opèrent sur une population de solutions. La population initiale est créée aléatoirement. A chaque génération, un nombre de solutions sont sélectionnées pour jouer le rôle des parents. L'opérateur de croisement crée de nouvelles solutions à partir des solutions parents. Ces solutions peuvent alors subir une mutation. Les meilleures solutions remplaceront les mauvaises pour maintenir la taille de la solution constante.

Ce processus continue jusqu'à ce que des critères d'arrêt soient satisfaits (obtention de la solution désirée ou l'écoulement du temps alloué). Dans notre, cas, à chaque itération, la fonction *Mot* évalue les nouvelles solutions, et suivant le résultat retourné décide d'arrêter ou non.

Dans ce qui suit, nous présenterons les éléments de l'algorithme génétique : la méthode de sélection et les opérateurs de croisement et de mutation. Les procédures présentées seront intégrées dans la structure générale des algorithmes génétiques (Algorithme II.6) présentée précédemment, pour fournir un algorithme complet.

II.5.1. Sélection :

La procédure de sélection adoptée est la *Roulette* présentée par GOLDBERG [34]. La probabilité de sélection d'une solution est en rapport contraire avec sa qualité (fonction d'évaluation). Donc pour chaque solution, un nombre aléatoire est choisi, si ce nombre dépasse sa probabilité alors la solution est sélectionnée.

Séléction ($S_1,...,S_p$: solution**) ;**
Début
  Pour $i=1$ **à** p **faire**
    Début
      $P = \dfrac{f(S_i)}{\sum_{j=1}^{p} S_j}$ **;**
      Choisir un nombre aléatoire $r \in [0,1]$ **;**
      Si $(r \geq p)$ **alors** Sélectionner(S_i) ;
    Fin ;
Fin ;

Algorithme IV.11 : Procédure de sélection

II.5.2. Croisement :

L'opérateur de croisement doit combiner les informations contenues dans les deux parents. Pour les problèmes de permutations, les opérateurs traditionnels ne peuvent être appliqués. L'opérateur retenu ressemble à la méthode de combinaison utilisée dans le Recherche Dispersée (voir §II.4.5). L'opérateur parcourt les deux solutions, et à chaque position choisit l'élément à insérer suivant la fréquence des éléments des parents.

Croisement 1 (S_1, S_2 : solution**) : Solution ;**
Début
  Pour $i=1$ **à** N **faire**
    Si $(f(S_1[i]) \geq f(S_2[i]))$ **alors**
      Si $(S_1[i] \notin \{S_{New}[1],...,S_{New}[i-1]\})$ **alors** $S_{New}[i] = S_1[i]$
      Sinon
        Si $(S_2[i] \notin \{S_{New}[1],...,S_{New}[i-1]\})$ **alors** $S_{New}[i] = S_2[i]$
        Sinon $S_{New}[i] =$ élément $\notin \{S_{New}[1],...,S_{New}[i-1]\}$ aléatoire

    Sinon
      Si $(S_2[i] \notin \{S_{New}[1],...,S_{New}[i-1]\})$ **alors** $S_{New}[i] = S_2[i]$
      Sinon
        Si $(S_1[i] \notin \{S_{New}[1],...,S_{New}[i-1]\})$ **alors** $S_{New}[i] = S_1[i]$
        Sinon $S_{New}[i] =$ élément $\notin \{S_{New}[1],...,S_{New}[i-1]\}$ aléatoire ;

  Retourner (S_{New}) ;
Fin ;

Algorithme IV.12 : 1er opérateur de croisement

où :

- $f(c)$: dénote la fréquence d'apparition du caractère c dans le texte crypté.

De la même manière, le second opérateur crée un autre fils, mais en parcourant les solutions parents de droite à gauche.

Croisement 2 (S_1, S_2 : solution**) : Solution ;**
Début
Pour $i = N$ **à** 1 **faire**
Si $(f(S_1[i]) \geq f(S_2[i]))$ **alors**
Si $(S_1[i] \notin \{S_{New}[i+1],...,S_{New}[N]\})$ **alors** $S_{New}[i] = S_1[i]$
Sinon
Si $(S_2[i] \notin \{S_{New}[i+1],...,S_{New}[N]\})$ **alors** $S_{New}[i] = S_2[i]$
Sinon $S_{New}[i] =$ élément $\notin \{S_{New}[i+1],...,S_{New}[N]\}$ aléatoire

Sinon
Si $(S_2[i] \notin \{S_{New}[i+1],...,S_{New}[N]\})$ **alors** $S_{New}[i] = S_2[i]$
Sinon
Si $(S_1[i] \notin \{S_{New}[i+1],...,S_{New}[N]\})$ **alors** $S_{New}[i] = S_1[i]$
Sinon $S_{New}[i] =$ élément $\notin \{S_{New}[i+1],...,S_{New}[N]\}$ aléatoire;

Retourner (S_{New}) ;
Fin ;

Algorithme IV.13 : 2ème opérateur de croisement

II.5.3. Mutation :

La mutation est importante pour maintenir la diversité de la population. Deux des plus fréquentes méthodes de mutation seront utilisées :

- Sélectionner deux positions aux hasard et échanger les éléments de ces positions.
- Choisir une séquence de la solution au hasard, et la déplacer vers une nouvelle position choisie au hasard elle aussi.

II.6. Cryptanalyse de systèmes de chiffrement par substitution polyalphabétique :

Le schéma de cryptanalyse pour la substitution polyalphabétique est identique à celui de la substitution simple puisque la substitution polyalphabétique n'est qu'une combinaison de plusieurs substitutions simples. La cryptanalyse débute par lé détermination de la période *b*, en utilisant les méthodes présentées dans le premier chapitre (voir Chapitre I §IV.2.1). Ensuite, casser le chiffre revient à déterminer *b* clefs de substitutions simples, exactement que dans les sections précédantes. Dans ce cas, on peut paralléliser les procédure de recherche, puisqu'elles sont indépendantes les unes des autres.

III- Cryptanalyse de systèmes de chiffrement par transposition :

Le système de chiffrement par transposition présenté précédemment (voir Chapitre I § IV.3) peut être vu comme une permutation des colonnes du texte en clair et où le nombre de colonne (la période) est connu. Ceci est la forme générale des chiffres de transposition.

Exemple IV.3 :

123456
UN_EXE
MPLE_D
E_SIMP
LE_SUB
STITUT
IONXXX

⟹

436125
E_EUNX
ELDMP_
ISPE_M
S_BLEU
TITSTU
XNXIOX

Les fréquences des caractères restent stables par rapport à la transformation associée au chiffrement, puisque aucune substitution n'est appliquée ; pour cela,

la cryptanalyse de tels systèmes se base sur les fréquences des n-grammes où $n \geq 2$ (bigrammes ou trigrammes).

La cryptanalyse des transpositions est un aussi un problème de A-permutations. Ce problème peut être approché de deux manières. La 1ère consiste à appliquer les clefs sur le texte crypté intercepté et ensuite agir de la même manière que pour les substitutions simples. La seule différence est que les statistiques des n-grammes ne sont pas utilisées. La fonction d'évaluation (IV.2) utilisée devient dans ce cas:

$$f_C(K) = \sum_{j=2}^{j=k} \left(\alpha_j \cdot \sum_{i_1,...,i_j \in \mathsf{A}} \left| P(i_1,...,i_j) - C(i_1,...,i_j) \right| \right) \qquad \text{(IV.8)}$$

où :

- A : Alphabet du texte en clair
- α_i : des constantes réelles où chaque $\alpha_i \in [0,1]$ et $\sum \alpha_i = 1$.
- $P(i_1,...,i_n)$: fréquence standard du n-gramme $(i_1,...,i_n)$.
- $C(i_1,...,i_n)$: fréquence du n-gramme $(i_1,...,i_n)$ dans le texte crypté.

Nous nous intéressons à la 2ème approche qui consiste à retrouver le texte colonne par colonne.

L'opération de cryptanalyse peut être formulée comme un problème d'optimisation dans un graphe : Retrouver le texte en clair à partir du texte crypté intercepté revient à réarranger les colonnes de ce dernier de façon à obtenir un texte lisible ; en d'autre terme, maximiser la valeur de l'heuristique *Mot* (voir §II.1). Les colonnes du texte crypté sont représentées par les sommets d'un graphe orienté, et un arc reliant deux sommets signifie que les deux colonnes se succèdent dans le texte clair correspondant. Le problème revient alors à retrouver un chemin passant par tous les arcs du graphe et qui maximise *Mot*.

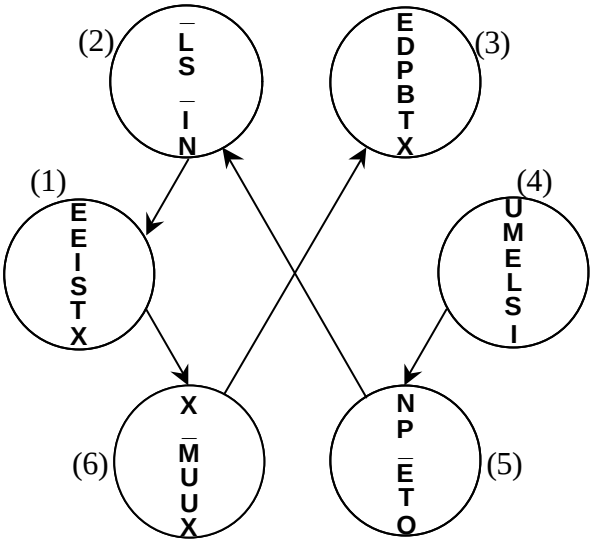

Fig IV.1 : Représentation de la clef de l'exemple (IV.3) par un graphe

Comme pour le chiffrement par substitution, cette heuristique ne sera utilisée que pour valider une clef donnée ou pour départager, à cause de son coût aussi ; et encore dans ce cas, parce qu'elle ne donne pas la possibilité de construire progressivement les solutions, puisqu'elle ne peut fournir une estimation que pour une clef complète et non pour une partie seulement (2 ou trois colonnes), ce qui présente un grand inconvénient.

Les arcs retenus pour former un tel chemin doivent correspondre à des relation d'adjacence admissibles entre les colonnes représentées par les sommets de ces arcs. Deux colonnes se succèdent si elles comportent les bigrammes les plus fréquents dans le texte en clair ; et si elles ne sont pas réellement voisines, les bigrammes formés par leur adjacence seront rares. L'heuristique évaluant cette relation utilise les fréquences standard des *n*-grammes (du texte en clair) où *n* est supérieur ou égal à 2. Donc, en se basant sur les statistiques des fréquences des bigrammes (voir Annexe II), deux colonnes seront supposées voisines si elles obtiennent un score élevé de la fonction suivante :

$$f_{bi}(C_i, C_j) = \frac{1}{l} \sum_{x=1}^{l} P(C_{xi}, C_{xj}) \qquad \text{(IV.9)}$$

où :

– l : nombre de lignes du texte intercepté.

- C_i : la $i^{\text{ème}}$ colonne du texte intercepté.
- $P(C_{xi}, C_{xj})$: fréquence standard du bigramme formé par le $x^{\text{ème}}$ caractère de la $i^{\text{ème}}$ colonne et de la $j^{\text{ème}}$ colonne.

La fonction précédente, peut être étendue pour inclure les statistiques des trigrammes :

$$f_{tri}(C_i, C_j, C_k) = \frac{1}{l} \sum_{x=1}^{l} P(C_{xi}, C_{xj}, C_{xk}) \qquad \text{(IV.10)}$$

De la fonction (IV.9), on peut déduire une fonction estimant la qualité d'une clef entière :

$$f(K = (k_1, ..., k_b)) = \frac{1}{b} \sum_{i=1}^{b-1} f_{bi}(C_{k_i}, C_{k_{i+1}})$$

$$f(K = (k_1, ..., k_b)) = \frac{1}{bl} \sum_{i=1}^{b-1} \sum_{x=1}^{l} P(C_{xk_i}, C_{xk_{i+1}}) \qquad \text{(IV.11)}$$

L'avantage de cette fonction par rapport à la fonction (IV.8) est qu'elle prend en compte que le texte est divisé en colonnes, et aussi elle peut éviter de recalculer toutes les probabilités en sauvegardant les probabilités des couples de colonnes.

Après le calcul de la valeur de cette fonction d'évaluation pour chaque couple de colonnes, seuls les arcs ayant obtenu un score supérieur à un seuil fixé sont gardés (ou bien dont le score sont nettement supérieur à ceux de leurs voisins). Ces arcs sont considérés comme des relations d'adjacence plausibles entre les colonnes correspondantes. Un arc est éliminé si dans les deux colonnes réunies, on trouve des bigrammes inexistants dans la réalité (tels '`qz`', '`xj`',…).

Dans le tableau ci-dessous, tous les scores des combinaisons possibles de l'exemple (IV.3) sont calculés :

Sommets	1	2	3	4	5	6
1		17,5895	4,5157	3,5654	5,4115	9,1969
2	9,7359		5,3062	6,8207	12,2155	2,1697
3	3,0136	9,1886		2,3843	3,2912	1,0882
4	3,6216	7,8293	3,6049		13,1722	2,2143
5	6,9101	15,8830	4,9624	4,5855		2,5545
6	5,7207	6,6463	5,7976	4,6333	5,1127	

Tableau IV.2 : Les scores de tous les arcs possibles de l'exemple (IV.3)
(les scores sont les moyennes des fréquences des bigrammes $\times$ 1000)

On remarque que dans le tableau (IV.3), chaque ligne possède une ou deux valeurs nettement supérieur aux autres valeurs, sauf pour la ligne (6). Deux choix se présentent, ou bien retenir les arcs dont le score dépassent un certain seul, en sachant que le score moyen pour de textes réels est de 11,97‰[(*)] ; ou bien retenir les arcs dont le score est nettement supérieur aux arcs ayant pour origine lu même sommet. Dans les deux cas, un seuil doit être fixé pour le choix, si ce seuil est peu restrictif, beaucoup d'arcs seront retenus et ce qui rendrai la tache de rechercher la clef plus complexe ; dans le cas contraire, on aura bien sûr moins d'arcs, mais il est fort probable qu'un arc ou plus appartenant la solution ne seront pas retenus. Dans le cas où les scores sont très proches (comme c'est le cas pour la dernière ligne du tableau (IV.3)), tous les arcs doivent être pris en considération. L'exemple (IV.4) essaie d'illustrer cette procédure en considérant deux seuils différents.

Exemple IV.4 :

Les données de l'exemple (IV.3) sont reprises. Dans le graphe à gauche (Fig IV.4.a), on a considéré seulement les arcs dont le score dépasse $\alpha = 6$, sauf pour le sommet (6) où tous les arcs ont été pris en considération. Dans le graphe à droite (Fig IV.4.b), le critère de sélection a été légèrement renforcé, puisque pour chaque somment, les arcs sont ordonnés suivant l'ordre décroissant de leurs

[*]. Cette valeur est calculée à partir des données utilisées dans les statistiques présentées dans l'Annexe I et II.

scores, ensuite, le premier arc est sélectionné, les autres peuvent être sélectionnés seulement s'ils ont obtenu un score supérieur à 60% du score du premier.

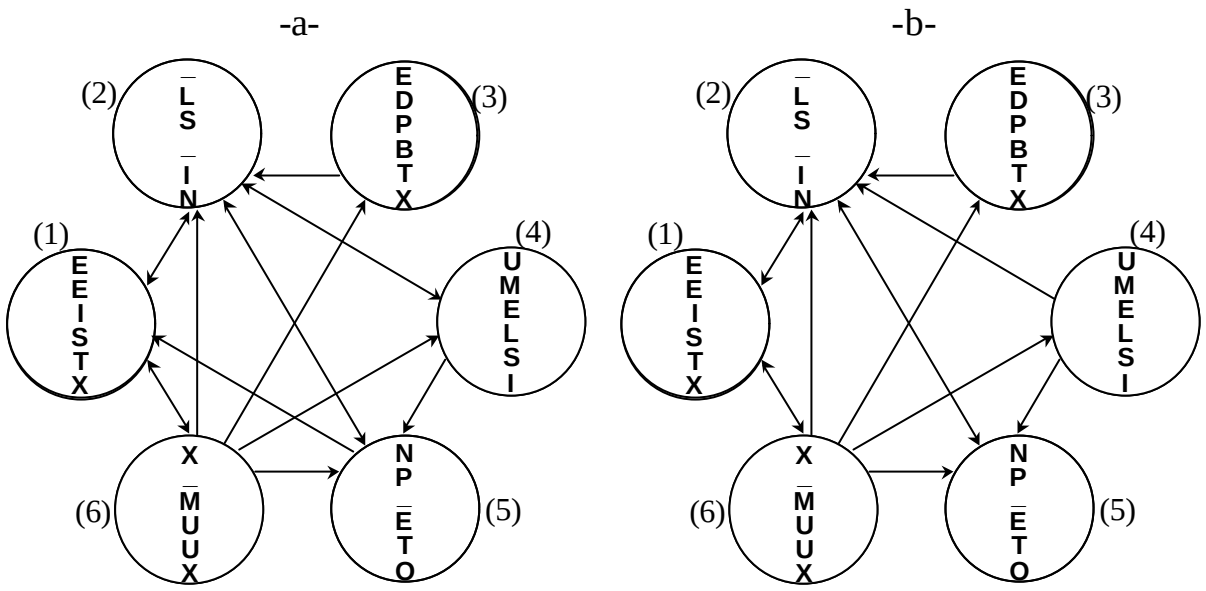

Fig IV.2 : Les arcs retenus suivant deux critères différents

On remarque que pour les deux cas, on a les même arcs (14 arcs au total pour la 1er graphe), sauf que l'arc (2,4) est présent juste dans le 1er cas (Fig IV.4.a). Dans le 1er cas, huit (8) chemins possibles ont été énumérés, dans le 2nd cas, juste deux ((3,2,1,6,4,5) et (4,5,2,1,6,3)). Il est ensuite aisé de vérifié que (4,5,2,1,6,3) est la bonne solution. On voit clairement que la légère différence entre les deux graphes a permit de réduire le nombre de solutions possibles de (8) à (2). Ceci montre que le choix du critère de sélection est primordial.

III.1. Application de la Recherche Dispersée pour la cryptanalyse des transpositions :

Les implémentations des méthodes de la Recherche Dispersée pour les transpositions ne diffèrent pas beaucoup des implémentations précédentes pour les substitutions. Le calcul de la distance se fait toujours par la formule (IV.7), et la fonction (IV.11) évalue la qualité des solutions. Quand l'ensemble de

référence a atteint un point fixe, l'heuristique *Mot* (IV.4) évalue alors les solutions de bonne qualité pour en déterminer la meilleure. Si aucune solution acceptable n'est trouvée, l'algorithme reprend dés la méthode de génération en prenant comme semence les meilleures solutions dans l'ensemble de référence.

Dans ce qui suit, nous allons détailler les méthodes de la Recherche Dispersée, mais seulement celles qui diffèrent des méthodes utilisées pour les substitutions. Les méthodes d'amélioration, de mise à jour de l'ensemble de référence et de la génération des sous-ensembles restent inchangées. Les autres méthodes présentent des différences mais sans toucher le principes.

III.1.1. Méthode de génération de diversification :

Pour la méthode génération de diversification, les mêmes générateurs sont utilisés (Algorithmes IV.3, IV.4, IV.5). Une légère différence existe pour la génération des solutions semence dans le cas où elles ne pas disponibles.

Pour obtenir de telles solutions, tout d'abord on détermine la colonne ayant les plus bas scores d'adjacence quand elle est suivie par d'autres colonnes (comme pour la colonne 4 dans le tableau IV.2). Ceci apporte une indication que cette colonne est probablement la 1ère colonne. Ensuite, les autres colonnes sont choisies une à une en sélectionnant à chaque fois celle qui a un le plus grand score d'adjacence quand elle suit la colonne courante.

On peut aussi générer une solution semence d'une manière inverse. On sélectionne au début la colonne dont la ligne correspondante dans le tableau d'adjacence comporte les plus bas scores (c'est le cas de la colonne 3 dans le tableau IV.2). Ensuite, à chaque fois on sélectionne la colonne présentant un fort score quand elle est suivie par la solution courante.

III.1.2. Méthode de combinaison des solutions :

Pour la méthode de combinaisons, le même principe est utilisé : chaque solution offre une voix à son premier élément (colonne). L'élément élu est celui ayant obtenu un plus grand nombre de voix. La différence se situe au cas où plusieurs éléments ont le même score. Dans ce cas, la colonne choisie est celle présentant le plus grand score d'adjacence avec la colonne courante qui sera élue. Les autres éléments verront leurs scores augmenter.

III.2. Utilisation d'un algorithme génétique pour la cryptanalyse des transpositions :

Pour l'algorithme génétique, il n'y a pas une grande différence avec l'algorithme précédent. Le critère de sélection et l'opérateur de mutation restent inchangés. L'opérateur de croisement quant à lui subit une légère modification, puisque dans l'opérateur de croisement précédant, les fréquences d'apparition des caractères ont été utilisées, et ceci est impossible pour les transpositions du moment où ces fréquences ne changent pas.

Croisement 1 (S_1, S_2 : solution**) : Solution ;**
Début
  Pour $i=1$ **à** N **faire**
    Si $(S_1[i] \notin \{S_{New}[1],...,S_{New}[i-1]\})$ **alors** $S_{New}[i]=S_1[i]$
    Sinon
      Si $(S_2[i] \notin \{S_{New}[1],...,S_{New}[i-1]\})$ **alors** $S_{New}[i]=S_2[i]$
      Sinon $S_{New}[i]=$ élément $\notin \{S_{New}[1],...,S_{New}[i-1]\}$ aléatoire
  Retourner (S_{New}) ;
Fin ;

Algorithme IV.14 : 1er opérateur de croisement

De la même manière que pour les substitutions, on peut avoir un second opérateur procédant dans l'ordre inverse :

Croisement 2 (S_1, S_2 : solution**) : Solution ;**
Début
  Pour $i = N$ **à** 1 **faire**
    Si $(S_1[i] \notin \{S_{New}[i+1],...,S_{New}[N]\})$ **alors** $S_{New}[i] = S_1[i]$
    Sinon
      Si $(S_2[i] \notin \{S_{New}[i+1],...,S_{New}[N]\})$ **alors** $S_{New}[i] = S_2[i]$
      Sinon $S_{New}[i] =$ élément $\notin \{S_{New}[i+1],...,S_{New}[N]\}$ aléatoire
  Retourner (S_{New}) ;
Fin ;

Algorithme IV.15 : 2ème opérateur de croisement

VI- Conclusion :

Dans ce chapitre, nous avons présenté les implémentations des méthodes de la Recherche Dispersée pour la cryptanalyse des substitutions et des transpositions. Une des difficultés principales de l'utilisation de cette méthode est le grand nombre de paramètres influençant ses performances.

Dans le chapitre suivant, nous allons effectuer un ensemble d'expérimentation pour attribuer des valeurs optimales à ces paramètres, et aussi pour démonter la validité de l'algorithme.

Chapitre V : *Tests et évaluations*

I- Introduction :

Dans ce chapitre, nous allons exposer les résultats des évaluations de l'implémentation de la Recherche Dispersée présentée dans le chapitre précédent. Ce chapitre sera divisé en deux grandes parties, la première est consacrée à l'attaque contre les systèmes de chiffrement par substitution mono-alphabétique, la seconde partie -quant à elle- est consacrée aux transpositions.

Les textes utilisés sont des portions de longs textes divers (articles, grands classiques de la littérature) choisies au hasard, et découpé de telle manière à approcher la longueur fixée à chaque expérimentations.

II- Cryptanalyse des substituions :

Dans cette section, tout d'abord nous débuterons par présenter les expérimentations qui nous ont permis de fixer les valeurs des paramètres de la Recherche Dispersée pour les substitutions. Dans un second lieu, nous présenterons les résultats concernant les performances de la procédure de la Recherche Dispersée, en même temps qu'une comparaison avec un Algorithme Génétique. Les clefs des substitutions sont des permutations de l'alphabet crées par un générateur aléatoire.

II.1. Détermination des paramètres de la Recherche Dispersée :

Les expérimentations suivantes ont pour but de déterminer les valeurs optimales pour les paramètres de la Recherche Dispersée. Tout au début, nous avons commencé avec les valeurs par défaut suivantes :

- Taille de l'ensemble de référence : $b = 20$.
- Taille de l'ensemble initial P : $|P| = 200$.
- Nombre maximum d'itérations de la méthode d'amélioration : $MaxIterAmelioration = 200$.
- Nombre maximum d'itérations de la procédure de la Recherche Dispersée : $MaxIterRD = 50$.

Au fur et à mesure que nous avançons, ces paramètres seront fixés aux valeurs déterminées par les tests.

II.1.1. Détermination des valeurs des constantes de la fonction d'évaluation :

Nos tests se baseront sur l'utilisation des statistiques de fréquence des unigrammes, bigrammes et trigrammes. La fonction d'évaluation (IV.2) utilise des constantes ($a_i \in [0,1]$) où chacune est associée à un type de n-grammes pour donner plus de poids à quelque-uns par rapport aux autres.

Pour déterminer la contribution de chaque type de n-grammes dans la recherche de la clef, nous allons calculer le nombre d'éléments de clef retrouvés en variant à chaque fois ces constantes. Nous utiliseront 100 textes d'une longueur moyenne d'environ 1000 caractères. Des clefs de substitution simple générées aléatoirement sont appliquées à ces textes pour obtenir 100 cryptogrammes. A partir de ces cryptogrammes, des tentatives de déterminations des clefs sont effectuées (en minimisant la fonction (IV.2)) pour différentes combinaisons des a_i.

Pour ce test, nous avons restreint les valeurs des a_i aux multiples de 0,1 pour simplifier la tache, et avoir un nombre raisonnable des combinaisons des

a_i. Dans ce cas, nous avons 66 combinaisons de α_1, α_2 et α_3 vérifiant les conditions :

$$\alpha_1, \alpha_2, \alpha_3 \in \{0 ; 0{,}1 ; 0{,}2 ; 0{,}3 ; 0{,}4 ; 0{,}5 ; 0{,}6 ; 0{,}7 ; 0{,}8 ; 0{,}9 ; 1\}$$

$$\alpha_1 + \alpha_2 + \alpha_3 = 1$$

Pour chaque *n*-gramme, nous avons fixé son poids, et varié les poids des deux autres n-grammes ; et nous avons ensuite la moyenne du nombre d'éléments corrects des clefs retrouvées :

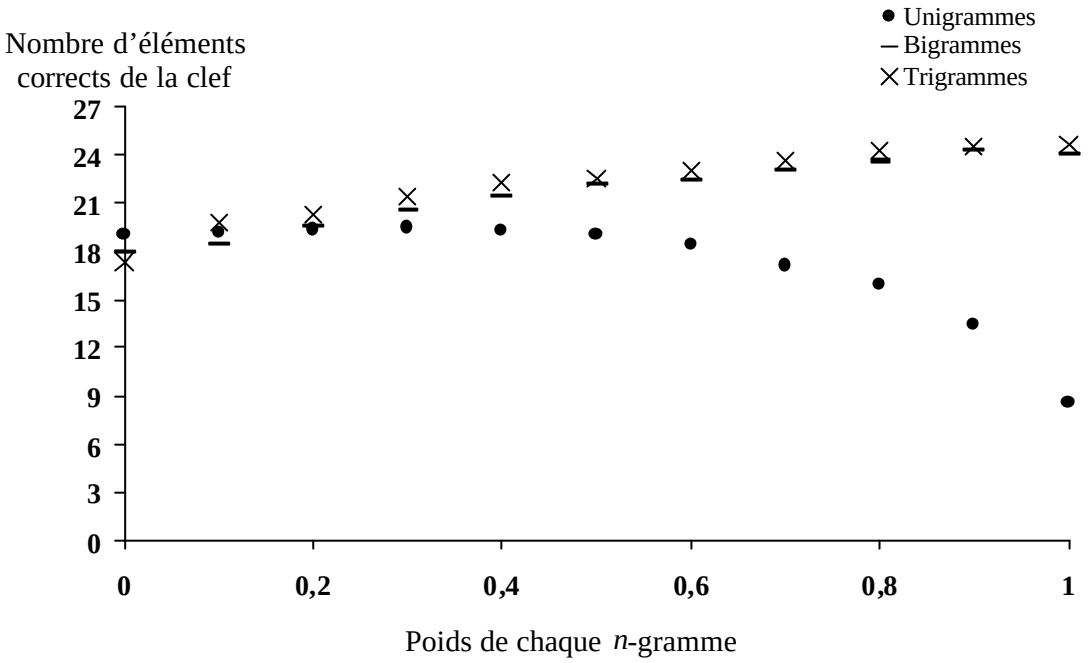

Fig V.1 : La moyenne des nombres d'éléments corrects des clefs retrouvées en fonction de la contribution de chaque *n*-gramme

Le graphe ci-dessus montre clairement que quand la fonction d'évaluation est entièrement basée sur les unigrammes ($\alpha_1 = 1,\ \alpha_2 = 0,\ \alpha_3 = 0$) le nombre moyen d'éléments corrects des clefs retrouvées décroît considérablement. Par contre, ce nombre augmente quand la fonction est basée sur les bigrammes ($\alpha_1 = 0,\ \alpha_2 = 1,\ \alpha_3 = 0$) ou les trigrammes ($\alpha_1 = 0,\ \alpha_2 = 0,\ \alpha_3 = 1$). Pour ces derniers, la majorité des éléments des clefs est retrouvée pour la quasi-totalité des tests, la plupart des cas d'erreur consistent dans l'inversement des positions de caractères apparaissant rarement dans le texte en clair ('*w*', '*k*', '*z*' le plus souvent).

Les trigrammes présentent les meilleurs résultats, et qui sont très légèrement supérieurs à celles des bigrammes. Cependant, les ressources (temps de calcul et espace mémoire) nécessaires pour établir les statistiques des trigrammes sont énormes par rapport à celles des bigrammes. Il est clair que le profit tiré de l'utilisation des trigrammes ne compense pas les ressources nécessaires, donc, dans le reste des tests, la fonction d'évaluation sera entièrement basée sur l'utilisation des bigrammes.

II.1.2. Détermination de nombre maximum d'itérations de la méthode d'amélioration :

L'objectif de ce test est d'estimer la meilleure valeur du nombre maximum d'itérations de la méthode d'amélioration (*MaxIterAmelioration*) utilisé dans l'algorithme IV.2. Pour cela, nous avons généré 100 ensembles initiaux pour 100 clefs différentes, où chaque ensemble est formé de 100 solutions. Nous avons appliqué à ces solutions la méthode d'amélioration, et estimé le taux d'amélioration en fonction du nombre d'itérations.

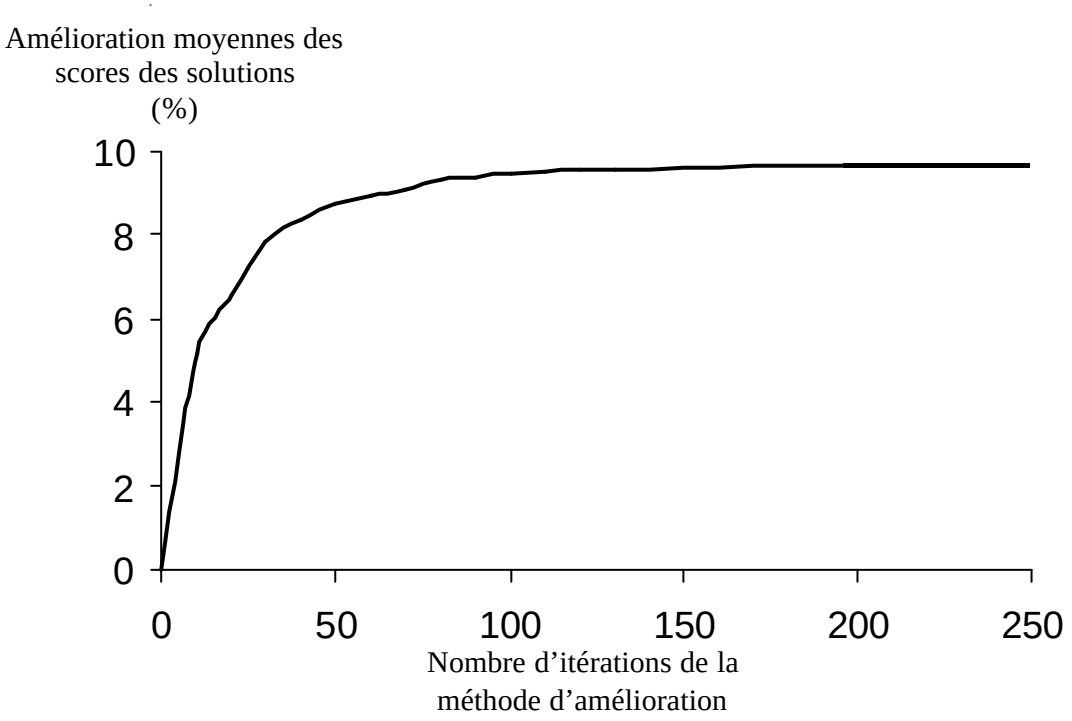

Fig V.2 : Taux moyen d'amélioration des solutions en fonction du nombre d'itération de la méthode d'amélioration

Dans cette évaluation, nous avons remarqué que le taux moyen d'amélioration a atteint 9,32% pour 80 itérations, ensuite ce taux augmente d'une manière très réduite pour atteindre 9,63% pour 180 itération : ce qui fait une amélioration de 3% dans la moyenne d'amélioration pour une augmentation de 125% dans le nombre d'itérations. Après 180 itérations, la moyenne d'amélioration reste stable (à 9,63%) même pour de grands nombres d'itérations (1000).

A partir de ces résultats, nous avons décidé de fixer le nombre d'itération de la méthode d'amélioration à la valeur de 80.

II.1.3. Contributions des types de sous-ensembles :

Le but de cette évaluation est de déterminer les types de sous-ensembles qui contribuent le mieux dans la génération des solutions de l'ensemble de référence, et par la suite, d'éliminer les types de sous-ensembles qui semblent inertes.

Pour ce tests, nous avons tout d'abord déterminé les meilleures solutions pour 100 textes (de 500 caractères) en utilisant les 4 types de sous-ensembles. Ensuite, nous avons calculé le nombre d'itérations nécessaires pour que la procédure arrive à cette solution, pour plusieurs combinaisons de types de sous-ensembles.

Dans le graphe ci-dessous, nous avons illustré le nombre d'itération pour chaque type à part. Nous avons testé toutes les combinaisons possibles, et nous avons illustré juste la meilleure (2,3) pour ne pas surcharger le graphe.

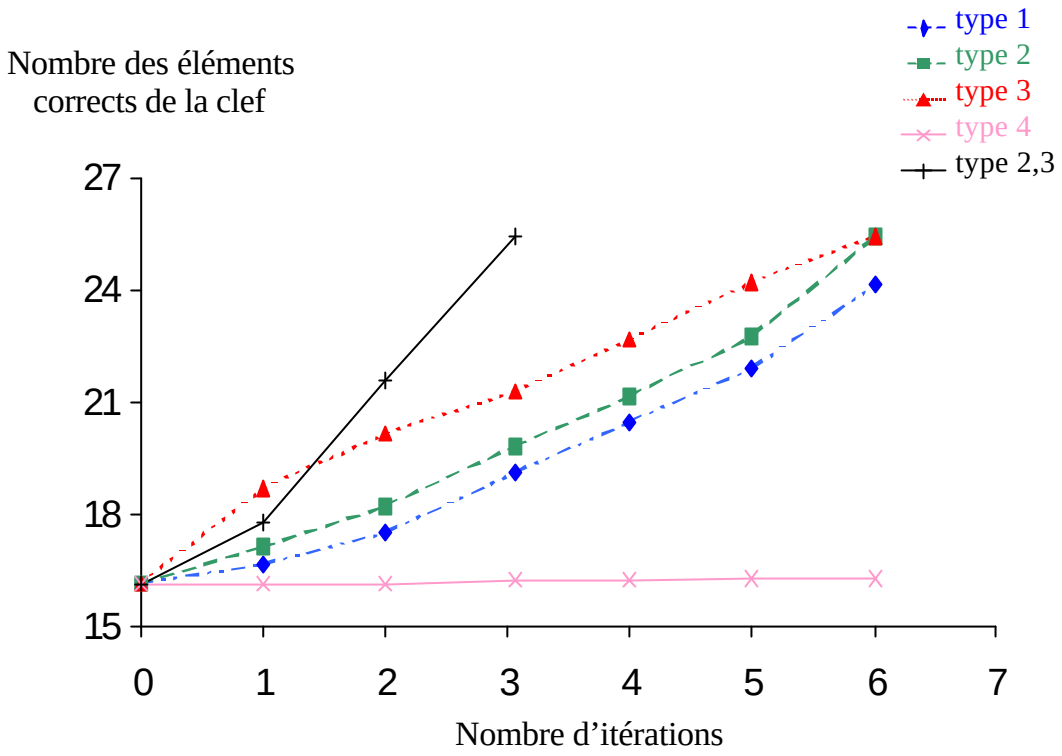

Fig V.3 : Nombres d'itérations nécessaires à la détermination de la meilleure solution pour plusieurs types de sous-ensembles

La première remarque qu'on peut tirer de ce graphe, est que le sous-ensembles de type 2 et 3 génèrent de meilleures solutions que les sous-ensembles de type 1, contrairement à ce qui est avancé dans [48]. Ceci revient à la nature des solutions du problème et au mécanisme de la méthode de combinaison : les solutions de bonne qualité sont souvent très proches, et la combinaison de deux solutions de ce genre, retourne dans la plupart des cas une des solutions en entrée ; par contre, pour 3 ou 4 solutions en entrée, la méthode de combinaison opère mieux et retourne de nouvelles solutions de meilleure qualité.

Après l'essai de plusieurs combinaisons de types de sous-ensembles, la meilleure combinaison est l'utilisation des sous-ensembles des types 2 et 3. En adoptant cette combinaison, la procédure détermine la meilleure solution avec le même nombre d'itération que celui nécessaire quand tout les sous-ensembles sont utilisés. Mais le gain principal est que l'itération dans cette combinaison nécessite moins temps de calcul puisqu'il y a moins de sous-ensemble à

examiner : $\left(3b^2 - 7b\right)/2$ sous-ensembles pour les 4 types, contre $\left(2b^2 - 8b + 8\right)/2$ sous-ensembles pour les types 2 et 3, ce qui fait par exemple une réduction de 40% quand la taille de l'ensemble de référence $b = 20$.

II.1.4. Taille de l'ensemble de référence :

L'objectif de cette expérimentation est de trouver la valeur optimale de la taille de l'ensemble de référence *RefSet* (*b*). Ce paramètre est primordial pour l'efficacité de la procédure de la recherche dispersée. Un ensemble de référence de petite taille convergera rapidement vers une solution le plus souvent très éloignée de la solution optimale. Par contre, un ensemble de référence de plus grande taille fournira des solutions de meilleure qualité, mais en entraînant des temps de réponses exorbitants, puisque le nombre de solutions examinées à chaque itération est en relation directe avec la taille de l'ensemble de référence.

L'idéale serait de trouver la plus petite valeur de la taille de *RefSet* à partir de laquelle il n'y a pas de grand gain en matière de la qualité de la solution retrouvée.

Pour cette expérimentation, nous avons calculé le temps de réponse nécessaire pour la détermination des meilleures solutions (déjà calculées) pour 100 textes en variant à chaque fois la taille de l'ensemble de référence *RefSet*. Les résultats sont montrés dans le graphe suivant :

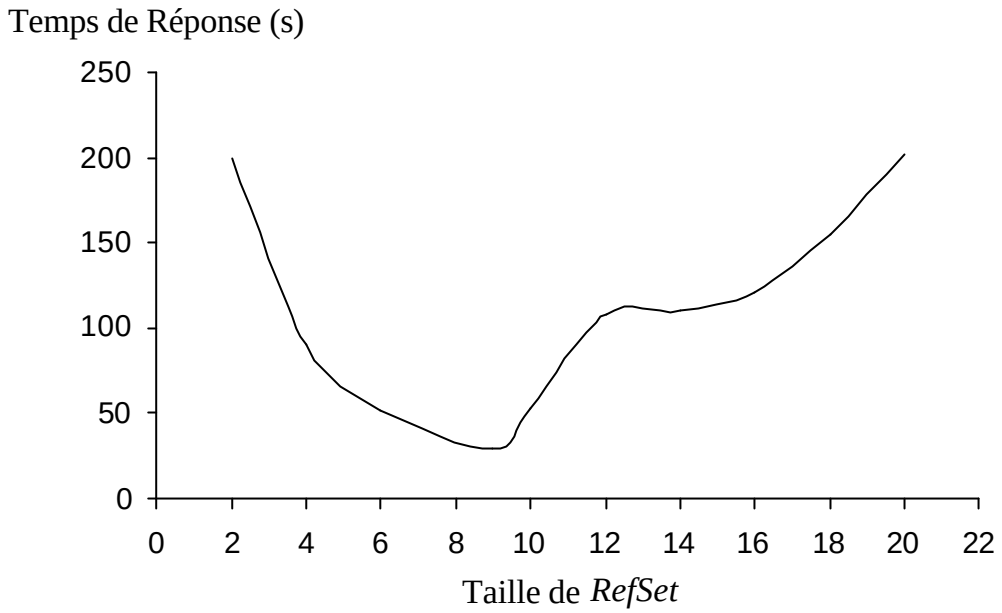

Fig V.4 : Temps nécessaire à la détermination de la solution optimale en fonction de la taille de l'ensemble de référence

Les résultats de cette expérimentation ont montré que pour une taille inférieure à 10, le nombre d'itérations pour trouver la meilleure solution est très grand, pire encore, pour des petites valeurs (2 à 4) la procédure converge prématurément vers une solution de qualité médiocre. A partir de la valeur 10, le nombre d'itérations nécessaires reste le même, mais le temps de calcul augmente du fait que le temps de calcul utilisé par chaque itération augmente en fonction de la taille de *RefSet*. A partir de ces résultats, nous adopterons la valeur 10 comme taille de l'ensemble de référence.

II.1.5. Taille de l'ensemble initial :

Le but de cette expérimentation est de fixer une valeur pour la taille de l'ensemble initial P. Ce paramètre est très important, dans la mesure où toutes les méthodes de la procédure reposent sur les solutions de cet ensemble, et d'un autre coté, la génération d'un tel ensemble est à l'origine d'une bonne partie du temps de réponse (environ 40% pour $|P| = 200$).

Pour l'évaluation, nous avons estimé la moyenne des éléments corrects des clefs retrouvées par la procédure, en variant à chaque fois le taille de la population initiale, les résultats sont résumés dans le graphe suivant :

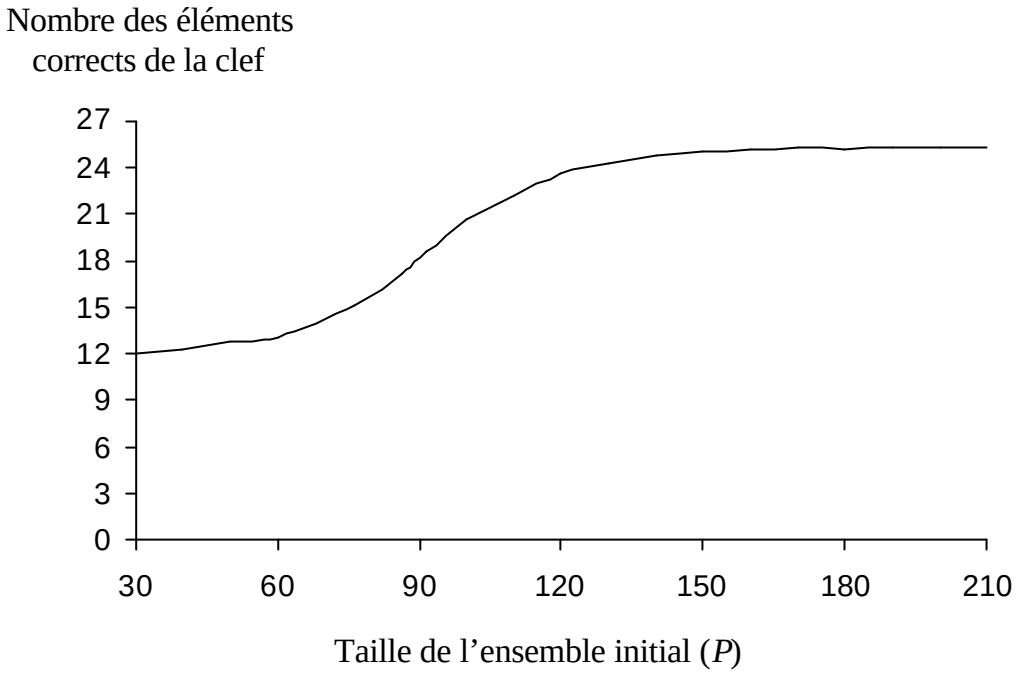

Fig V.5 : Nombre moyen des éléments corrects des clefs en fonction de la taille de l'ensemble initial (*P*)

A partir des résultats du graphe ci-dessus, la valeur retenue pour la taille de l'ensemble initial *P* est 120.

II.1.6. Nombre d'itérations de la Recherche Dispersée :

Le dernier paramètre à fixer est le nombre d'itérations de la Recherche Dispersée. La procédure de cryptanalyse s'arrête quand l'ensemble de référence atteint un point fixe, ou bien quand le nombre maximum d'itérations de la procédure est dépassé. Pour déterminer la valeur idéale à ce paramètre, nous l'avons varié et calculé le nombre moyen des éléments des clefs retournées par la procédure.

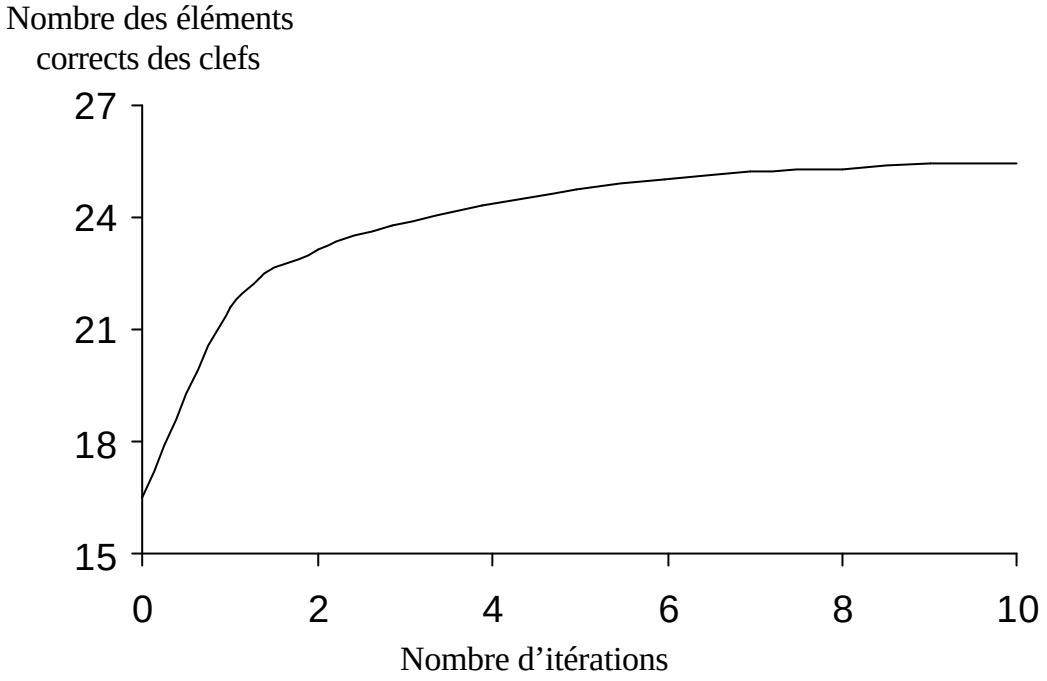

Fig V.6 : Nombre moyen des éléments corrects des clefs en fonction du nombre d'itérations de la Recherche Dispersée

De la figure ci-dessus, nous remarquons bien que le nombre moyen des éléments des clefs retrouvées par la procédure reste relativement constant après 7 itérations. Les expérimentations ont montré que dans la majorité des cas l'ensemble de référence atteint un point fixe avant cette valeur.

Lors de l'implémentation de la méthode de la mise à jour de l'ensemble de référence, nous avons supposé que la recherche dispersée commence par une mise à jour statique, et après un certain nombre d'itérations, la mise à jour devient dynamique pour accélérer la convergence de *RefSet* (voir Chapitre IV § II.4.3). Puisque le nombre maximum d'itérations est relativement petit, nous adopterons une mise à jour statique pour toute la procédure.

II.2. Comparaison entre les performances de la Recherche Dispersée et les Algorithme Génétique :

Dans cette section, nous allons présenter les résultats concernant les performances de la procédure de la Recherche Dispersée, en même temps qu'une comparaison avec un Algorithme Génétique. Pour l'implémentation de l'algorithme génétique, on a repris les paramètres déterminés dans l'implémentation de [65] : une population de 1000 individus, et un taux de mutation fixé à 0,1.

II.2.1. Influence de la taille du texte chiffré :

Pour cette évaluation, nous avons varié la taille des textes cryptés, et nous avons calculé le nombre moyen des éléments corrects des clefs trouvés. Les résultats sont montrés dans la figure V.7.

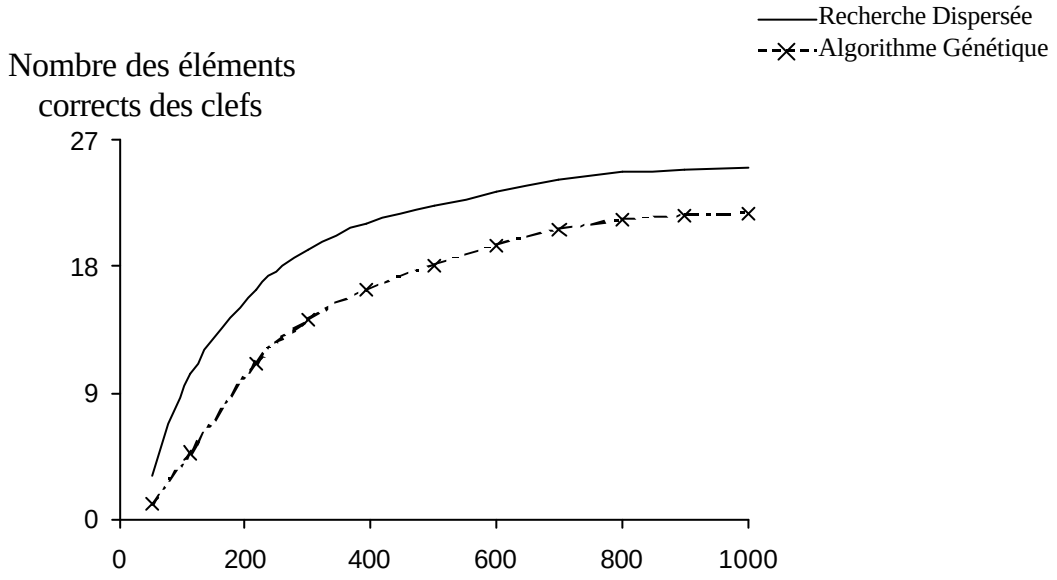

Fig V.7 : Le nombre moyen des éléments corrects de la clef en fonction de la taille des texte chiffrés (pour la RD et l'AG)

Le graphe de la figure V.7 montre clairement que la RD retourne des solutions de meilleure qualité que l'AG (environ 15%). Ceci est du en grande

partie à la méthode d'Amélioration qui permet d'explorer mieux le voisinage de chaque solution considérée. Mais en contre partie, cet avantage augmente considérablement le temps de réponse (en moyenne la RD consomme 75% de temps en plus que l'AG).

En reprenant l'expérimentation précédente, mais en désactivant la méthode d'amélioration pour la RD, nous obtenons les résultats suivants :

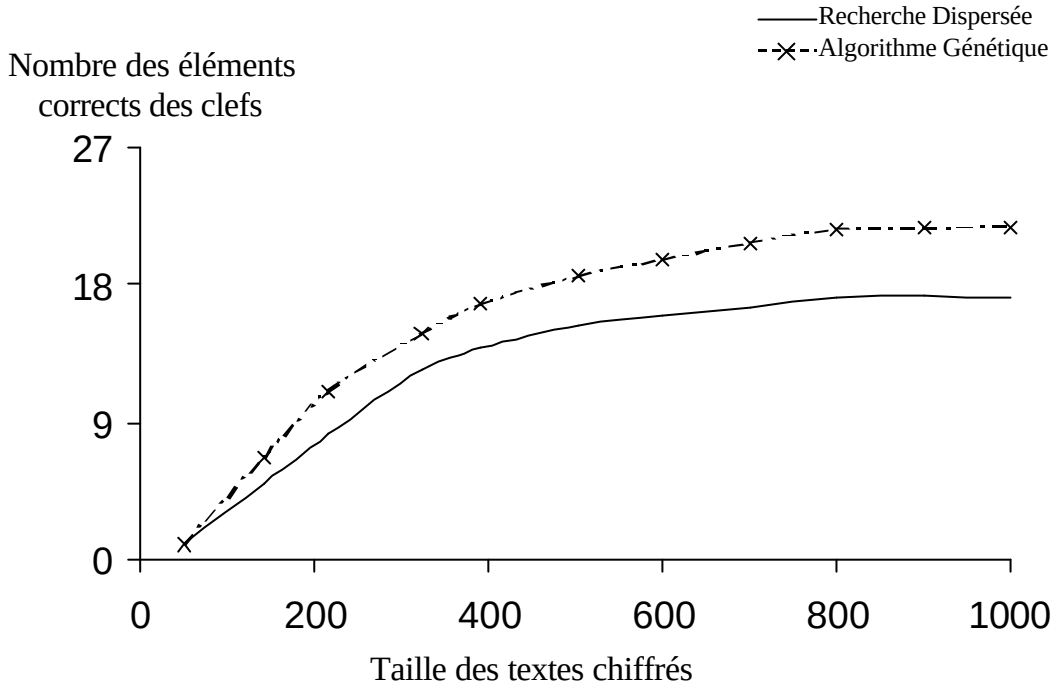

Fig V.8 : Le nombre moyen des éléments corrects de la clef en fonction de la taille des texte chiffrés (pour l'AG et la RD sans méthode d'amélioration)

Les résultats montrent clairement que les solutions retournées par la RD sont de qualité très inférieure à celles retournées précédemment. L'élimination de la méthode d'amélioration de la RD a diminué le taux moyen des éléments corrects des clefs trouvées, rendant la procédure de la Recherche Dispersée équivalente à l'Algorithme génétique, mais le fait que la RD opère sur un population de taille plus petite fait en sorte que l'ensemble de référence converge prématurément et souvent vers des solutions de qualité moyenne.

II.2.2. Influence du temps alloué :

Le but de cette expérimentation est de montrer l'évolution de la qualité des solutions trouvées au cours de la procédure. Pour cela, nous avons calculé le nombre moyen des éléments corrects des clefs en fonction du temps écoulé.

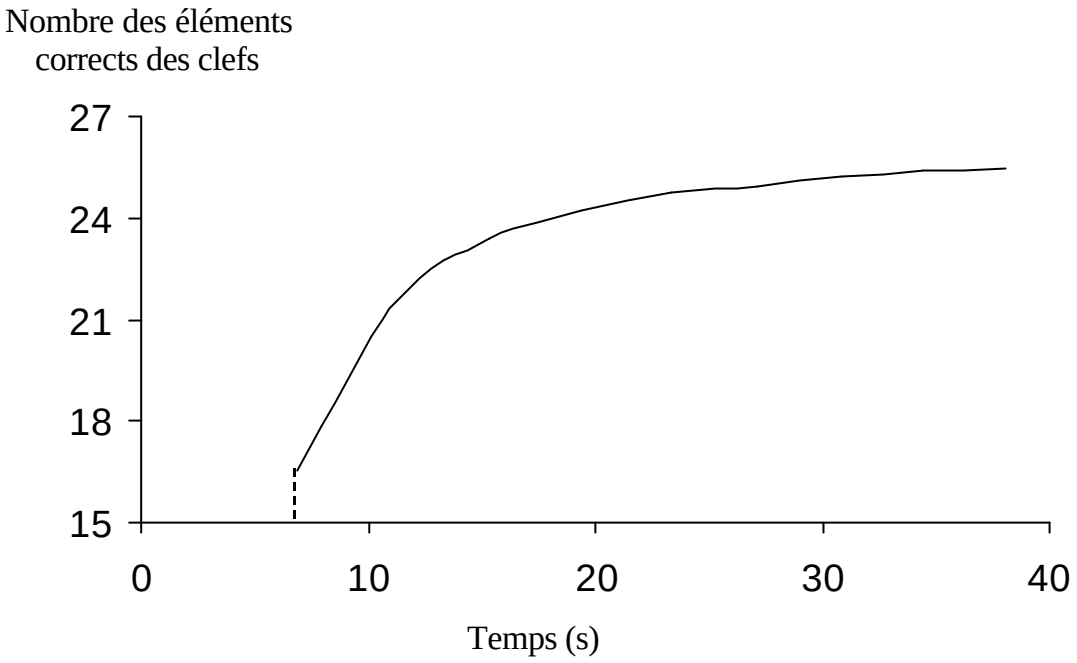

Fig V.9 : Le nombre moyen des éléments corrects de la clef en fonction du temps écoulé de la procédure

La première valeur est relevé à $t = 6{,}85\,s$, c'est le temps moyen que prend la méthode de génération de diversification pour générer l'ensemble initial P ; le nombre moyen des éléments corrects des meilleures clefs trouvées est 16,54. Le temps moyen que nécessite une itération est d'environ 3,88 *s*. Nous remarquons qu'à partir de $t = 30\,s$ l'amélioration moyenne de la qualité des clefs devient très légère.

III- Cryptanalyse des transpositions :

Dans cette section, nous allons reprendre les expérimentations précédentes pour les transpositions. Pour ces expérimentations, nous avons utilisé des textes contenant en moyenne 1000 caractères environ. La sélection des colonnes adjacentes se fait suivant le critère présenté dans Chapitre IV §III. Les clefs utilisées ont été générées aléatoirement.

III.1. Détermination des paramètres de la Recherche Dispersée :

Nous avons effectué des expérimentations similaires à celles présentées dans (§II.1) dans le but de déterminer les valeurs optimales pour les paramètres de la Recherche Dispersée. Pour les transpositions la tache est plus compliquée puisqu'il y un facteur en plus qui entre en jeu : la période de la transposition considérée. Pour cela, tous les paramètres seront fixés en fonction de la période.

III.1.1. Détermination de nombre maximum d'itérations de la méthode d'amélioration :

Nous avons effectué des expérimentations similaires à celle présentée dans la section §II.1.2 pour déterminer la limite maximum d'itérations de la méthode d'amélioration (*MaxIterAmelioration*), mais en variant à chaque fois la période de la transposition.

Les résultats sont résumés dans le tableau suivant :

Période	5	10	15	20	25	30
MaxIterAmelioration	4	15	30	50	70	100

Tableau V.1 : Variation du nombre maximum d'itération en fonction de la période de la transposition

On remarque du tableau précédant, que le rapport entre la période et la constante *MaxIterAmelioration* est autant plus grand que la période est grande. Ceci revient au fait que le nombre de voisins directs augmente ($n-1$ voisins

pour une période *n*), et donc la taille du voisinage directe d'une solution augmente énormément.

III.1.2. Taille de l'ensemble de référence :

L'objectif de cette expérimentation est de trouver la valeur optimale de la taille de l'ensemble de référence *RefSet* (*b*). Pour cela, nous avons calculé le temps de réponse nécessaire pour la détermination des meilleures solutions (déjà calculées) en variant à chaque fois la taille de l'ensemble de référence *RefSet*. Le tableau suivant contient les meilleures valeurs pour certaines période.

Période	5	10	15	20	25	30
Taille de RefSet	6	6	8	8	10	12

Tableau V.2 : Variation de la taille de l'ensemble de référence en fonction de la période de la transposition

III.1.3. Taille de l'ensemble initial :

Le but de cette expérimentation est de fixer une valeur pour la taille de l'ensemble initial *P*. Pour cela, nous avons estimé la moyenne des nombre d'éléments corrects des clefs retrouvées par la procédure, en variant à chaque fois le taille de la population initiale. Pour chacune des périodes testées, nous avons trouvé des données similaires à celle trouvée pour l'expérimentation II.1.5. Les résultats sont résumés dans le tableau suivant :

Période	5	10	15	20	25	30
Taille de *P*	15	80	100	120	120	160

Tableau V.3 : Variation de la taille de l'ensemble initial en fonction de la période de la transposition

On remarque La taille optimale de l'ensemble initiale a tendance a se stabiliser pour de grande période.

III.1.4. Nombre d'itérations de la Recherche Dispersée :

Dans cette expérimentation, on cherche à trouver la meilleure valeur pour le nombre maximum d'itérations de la Recherche Dispersée. Pour cela, nous l'avons fait varier et calculé le nombre moyen des éléments des clefs retournées par la procédure. Le tableau suivant contient les résultats trouvés pour certaines périodes.

Période	5	10	15	20	25	30
MaxIter RD	2	4	6	6	7	7

Tableau V.4 : Variation du nombre maximum d'itérations de la RD en fonction de la période de la transposition

III.2. Comparaison entre les performances de la Recherche Dispersée et les Algorithme Génétique :

Nous avons lancé les deux procédures de recherche : la recherche dispersée et l'algorithme génétique pour environ 200 textes chiffrés formés chacun d'environ 1000 caractères en moyenne. Nous avons compté le nombre moyen d'éléments corrects des clefs retrouvées, en variant à chaque fois la période.

Période	**RD**	**AG**
10	8,32	7,90
15	13,45	12,37
20	17,24	15,51
25	22,49	18,43
30	26,34	23,17

Tableau V.5 : Nombre moyen des éléments corrects des clefs retrouvées en fonction de la période de la transposition (pour la RD et l'AG)

La recherche dispersée présente toujours de meilleurs résultats, ceci est du en grande partie au processus spécifique utilisé pour améliorer les solutions à

chaque itération. Ceci peut être vérifié comme pour les substitutions en relançant la recherche dispersée sans la méthode d'amélioration.

Période	RD	AG
10	7,23	7,90
15	11,41	12,37
20	13,75	15,51
25	16,37	18,43
30	19,89	23,17

Tableau V.6 : Nombre moyen des éléments corrects des clefs retrouvées en fonction de la période de la transposition (pour la RD sans méthode d'amélioration et l'AG)

Cette expérimentation montre aussi la grande importance de la méthode d'amélioration dans cette implémentation de la recherche dispersée. L'exploration du voisinage des solutions permet un grand gain dans leur qualité.

IV- Conclusion :

Dans ce chapitre, nous avons présenté les résultats des expérimentations qui nous ont permis de déterminer les valeurs des paramètres de l'implémentation de la recherche dispersée pour la cryptanalyse des chiffres simples.

Les tests effectués ont montré que la méthode est valable et exploitable dans la mesure où les textes disponibles ne sont très courts, comme c'est le cas pour la plupart des autres schémas de cryptanalyse. Les expérimentations ont montré que le cas de la cryptanalyse, nécessite des paramètres bien particuliers pour la recherche dispersée, surtout au niveau des types des sous-ensembles. Nous avons remarqué aussi la grande importance de la méthode d'amélioration dans l'implémentation pour les deux cas de cryptanalyse, et la grande importance de l'exploration du voisinage des solutions dans l'obtention de solutions optimales.

Conclusion générale

Plusieurs travaux ont été réalisés sur la cryptanalyse des chiffres simples. Certaines méthodes ont été plus efficaces que d'autres. Dans cette thèse, nous avons réussi à démontrer l'efficacité de l'utilisation de la méthode méta-heuristique '*Recherche Dispersée*' pour la cryptanalyse des chiffres simples.

Nous avons commencé cette thèse en présentant le domaine de la cryptologie en en définissant ses deux composantes : la cryptographie et la cryptanalyse, ensuite en présentant en détail les systèmes de chiffrement simple. D'un autre coté, nous avons exposé les principales notions du domaine l'optimisation combinatoire et de la complexité, en détaillant les principaux algorithmes de résolution, et les méta-heuristiques les plus utilisées. Après, la méta-heuristique en question -la '*Recherche Dispersée*'- est définie et ses principales méthodes sont présentées en détail et illustrées avec un exemple d'application.

Ensuite, nous avons dresser un schéma de cryptanalyse des systèmes de chiffrement simples utilisant la Recherche Dispersée, en commençant tout d'abord par formaliser le problème de cryptanalyse sous la forme d'un problème d'optimisation, ensuite nous avons appliqué la méta-heuristique pour le résoudre, en détaillant les implémentations des méthodes. Et enfin, nous avons présenté les résultats des expérimentations sur les paramètres de l'algorithme, et aussi les évaluations des performances de la méthode de cryptanalyse pour la valider.

Notre objectif était d'explorer la contribution de recherche dispersée pour la cryptanalyse des chiffres simples. Il est clair que les méthodes heuristiques ont un rôle important à jouer dans le domaine de la cryptanalyse. Le prochain pas à franchir est de passer à des systèmes beaucoup plus complexes, où l'unité de cryptage ne se situe plus au niveau caractère mais plutôt au niveau bit. Dans ce cas là, une attaque basée sur les caractéristiques linguistiques du texte et l'analyse fréquentielle n'est plus envisageable, mais plutôt des attaques du type texte chiffré-texte en clair sont utilisées.

Annexes

I- Fréquences des caractères (unigrammes) :

Lettre	Français	Anglais	Espagnole	Allemand
_	18,01%	19,92 %	19,04 %	18,84 %
A	6,81%	6,71 %	9,96 %	5,10 %
B	0,74%	1,24 %	0,83 %	1,62 %
C	2,66%	1,68 %	3,64 %	2,42 %
D	3,14%	4,09 %	4,08 %	4,09 %
E	14,22%	9,69 %	11,08 %	13,73 %
F	0,89%	1,61 %	0,62 %	1,31 %
G	0,81%	1,83 %	0,84 %	2,53 %
H	0,73%	5,00 %	0,53 %	3,66 %
I	5,82%	4,91 %	6,30 %	6,02 %
J	0,34%	0,07 %	0,23 %	0,24 %
K	0,02%	0,89 %	0,02 %	1,18 %
L	4,47%	3,08 %	4,73 %	2,89 %
M	2,40%	2,16 %	2,30 %	2,06 %
N	5,93%	5,81 %	6,00 %	8,28 %
O	4,38%	6,58 %	7,03 %	2,33 %
P	2,26%	1,21 %	2,13 %	0,62 %
Q	1,04%	0,06 %	0,83 %	0,02 %
R	5,36%	4,17 %	5,21 %	6,04 %
S	6,80%	4,75 %	5,64 %	5,37 %
T	5,93%	7,45 %	3,90 %	4,83 %
U	5,20%	2,44 %	3,23 %	3,56 %
V	1,31%	0,67 %	0,84 %	0,87 %
W	0,01%	2,11 %	0,02 %	1,23 %
X	0,36%	0,09 %	0,13 %	0,02 %
Y	0,20%	1,75 %	0,53 %	0,08 %
Z	0,16%	0,03%	0,28 %	1,01 %

Les statistiques ont été obtenues en analysants une grande variété de texte (dépassant le 10 millions de caractères) tirée des grands classiques de la littérature de chaque langue.

II- Fréquences des bigrammes :

Fréquences des bigrammes en langue française (‰) :

	-	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
-	12,752	0,6939	7,1522	0,0215	0,3940	5,6021	0,3175	0,1335	0,1573	49,849	4,0721	1,7287	0,0395	12,261	0,0000	0,0000	16,480	0,0000	3,2378	2,2303	13,502	0,0594	22,684	16,740	4,9197	0,0000	0,0000	0,0000
A	2,6591	4,0017	8,2351	0,0046	0,3596	0,8140	1,8535	0,9917	1,8997	1,0995	2,9743	2,2927	0,9216	0,0016	0,0000	0,0000	0,0016	0,0000	0,0000	1,3153	3,2309	2,0889	0,7845	0,0229	0,2510	1,6877	2,3550	0,0000
B	0,0018	0,0000	0,0273	0,0002	0,0000	0,0000	0,0000	0,0002	0,0004	0,2518	0,0008	0,0008	0,0187	0,0004	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0004	0,0028	0,9218	0,0009	0,0000	0,0000	0,0000
C	2,9874	0,0000	0,0683	0,0000	0,0000	0,0000	0,0010	0,0000	0,0000	3,5624	0,0002	0,4506	0,0000	0,0667	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0003	0,0007	0,0005	0,0000	0,0000	0,0000
D	3,8492	0,0000	0,0449	0,0000	0,0000	0,0184	0,0000	0,0019	0,0000	0,5222	0,0029	0,0000	0,0000	0,5609	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0007	0,0007	0,0000	0,0000	0,0000
E	7,8182	8,7134	16,588	0,0589	1,5959	7,1990	2,8210	2,4429	1,4581	1,6637	15,314	16,864	0,9218	0,0229	1,5708	1,5470	2,5438	0,0000	1,1532	6,4206	5,0174	1,5708	8,4091	0,0384	0,2726	1,9908	0,0000	0,0000
F	0,5663	0,0000	0,0040	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,6535	0,0075	0,0005	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
G	1,1478	0,0000	0,0582	0,0002	0,0000	0,0092	0,0000	0,0000	0,0000	0,8246	0,0070	0,0000	0,0000	1,6877	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
H	0,0261	0,0002	0,0788	0,0012	0,0000	0,0005	0,0000	0,0504	0,0002	0,1314	0,0100	3,6724	0,0005	0,0090	0,2726	1,9908	0,0003	1,3925	13,637	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
I	1,5647	2,3767	1,9974	0,0628	0,0005	0,0516	0,4107	0,4893	1,2664	0,9326	3,8436	1,4325	1,1783	12,574	5,4889	7,3433	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
J	0,0302	0,0003	0,0000	0,0003	0,0000	0,0028	0,0000	0,0000	0,0000	0,1745	0,0055	0,0000	0,0436	0,1250	2,6786	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
K	0,0064	0,0002	0,0015	0,0000	0,0000	0,0007	0,0000	0,0000	0,0000	0,0015	0,0000	0,0667	0,0004	0,0048	0,0594	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
L	0,0798	0,0005	4,3464	0,0028	0,0000	6,1753	0,0038	0,3062	4,6885	5,4021	0,0978	0,0007	1,5472	2,5438	18,734	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
M	0,0097	2,1481	10,040	0,0000	0,0000	1,2586	0,0065	0,0682	0,0002	25,402	0,0000	0,0002	0,5602	1,5708	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
N	1,6189	0,0205	0,0029	0,0012	0,0000	6,5705	0,0067	1,2229	0,0000	16,451	0,0002	0,0002	1,5472	8,4091	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
O	2,3429	2,5953	1,8205	0,0051	0,6729	2,7705	1,4345	0,2451	1,6959	0,2477	1,5426	5,9569	1,0361	0,0309	4,1357	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
P	0,0002	1,7811	0,0236	0,0003	0,0000	0,2105	0,0000	0,0000	0,0000	1,5814	0,0000	0,0008	0,0003	1,3924	13,637	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
Q	0,3176	0,0004	0,4071	0,0000	0,0000	0,8451	0,0000	0,0000	0,0000	0,1881	0,0000	0,0340	0,0000	0,3619	7,0433	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
R	0,1635	0,0000	0,0038	0,0000	0,0000	3,5959	0,0339	1,3398	0,5540	10,428	0,9041	1,1562	1,1532	6,4206	5,0174	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
S	6,1033	0,0219	0,7362	0,0039	0,0000	7,1813	0,0038	0,0273	0,0738	22,864	0,2888	0,0582	0,2200	2,8332	12,341	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
T	14,153	0,0216	0,3671	0,0004	0,0000	10,109	0,0045	0,1740	0,0002	10,063	0,0018	1,2999	0,0344	3,3907	5,5780	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
U	0,7157	0,4279	2,6561	0,0000	0,3491	0,0172	0,2677	0,5526	0,4159	6,5772	2,1529	1,0481	1,2772	4,8274	4,6136	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
V	0,3154	0,0000	0,0085	0,0000	0,0000	1,1521	0,0000	0,0000	0,0000	1,6271	0,0146	0,0001	0,0007	2,9169	4,8894	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
W	0,0010	0,0000	0,0019	0,0002	0,0000	0,0007	0,0000	0,0009	0,0000	0,0092	0,0053	0,0000	0,0007	0,0026	0,0942	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
X	0,0053	0,0000	0,0028	0,0000	0,0000	0,3542	0,0007	0,0001	0,0000	1,0174	0,0000	0,0000	0,0000	0,0165	0,0315	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
Y	0,0119	0,0219	0,0552	0,0019	0,0000	0,0008	0,1234	0,0041	0,0000	0,0138	0,1874	0,0120	0,0038	0,3204	0,5578	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
Z	0,0495	0,0004	0,0128	0,0008	0,0000	0,0397	0,0008	0,0002	0,0000	1,3436	0,0000	0,0007	0,0005	0,0344	0,0676	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000

O	P	Q	R	S	T	U	V	W	X	Y	Z
0,0428	0,4143	0,0826	9,4625	38,2009	25,394	6,8316	0,0190	0,0120	2,4432	0,6926	1,3596
0,0105	0,9945	0,0000	5,7302	13,3073	4,8457	0,8899	3,1204	0,0408	0,1319	0,3736	0,0554
0,4403	0,0002	0,0000	0,2359	0,0087	0,0003	0,2806	0,0000	0,0002	0,0006	0,0513	0,0005
0,6033	0,0192	0,0000	0,6877	0,5162	0,0239	0,9509	0,0000	0,0006	0,1101	0,0041	0,0000
0,5658	0,0005	0,0000	1,4764	0,0025	0,0016	0,5309	0,0000	0,0002	0,0003	0,0187	0,0000
0,2284	4,1319	0,0000	16,823	7,6703	10,708	7,3027	4,2619	0,0169	0,2379	0,4587	0,1029
0,1888	0,0007	0,0000	0,1958	0,0364	0,0008	0,2736	0,0003	0,0004	0,0008	0,0006	0,0005
0,3008	0,0006	0,0000	0,5386	0,0023	0,0000	0,2164	0,0000	0,0000	0,0000	0,0064	0,0000
0,0868	0,2621	0,0000	0,0200	0,0231	0,6505	0,0079	0,0000	0,0019	0,0030	0,0002	0,0000
0,3642	0,7835	0,0000	4,3072	3,4379	4,9463	4,5876	1,7082	0,0533	0,2628	0,0012	0,0382
0,0546	0,0006	0,0000	0,0008	0,0002	0,0006	0,2569	0,0003	0,0004	0,0003	0,0008	0,0005
0,0128	0,0005	0,0000	0,0054	0,0028	0,0000	0,0002	0,0000	0,0000	0,0000	0,0000	0,0000
0,7577	2,5117	0,0000	0,2075	0,0048	0,0097	1,5457	0,0000	0,0002	0,0041	0,0269	0,0000
0,5085	0,0007	0,0000	1,1747	0,0998	0,0136	0,4222	0,0000	0,0005	0,0001	0,0205	0,0000
0,0685	0,0002	0,0000	0,4728	0,0039	0,0092	0,2535	0,0000	0,0091	0,0000	0,0066	0,0002
0,3155	3,7807	0,0000	3,7548	3,5697	2,6306	0,1838	3,1866	0,0059	0,0041	0,1127	0,0482
0,5791	0,9325	0,0000	0,1738	1,0864	0,0008	0,9501	0,0000	0,0000	0,2225	0,0572	0,0000
0,1494	0,0006	0,0000	0,2256	0,7306	0,0000	0,0693	0,0000	0,0000	0,0161	0,0005	0,0000
0,4026	3,2631	0,0000	1,3747	0,0013	0,4420	7,7483	0,5740	0,0009	0,0007	0,0148	0,0000
0,5321	0,3650	0,0010	2,3908	3,5108	1,5503	5,6978	0,0003	0,0007	0,0000	0,1789	0,0000
0,1016	0,3486	0,0000	3,1575	3,5440	3,7303	1,7065	0,0000	0,0007	0,1539	0,0141	0,0000
0,1165	0,8153	10,312	0,6787	2,1795	1,2254	0,0009	0,1910	0,0007	0,0172	0,0003	0,0010
0,0937	0,0005	0,0000	0,4238	0,0024	0,0005	1,6089	0,0000	0,0000	0,0079	0,0000	0,0003
0,0207	0,0000	0,0000	0,0041	0,0010	0,0019	0,0009	0,0000	0,0000	0,0006	0,0006	0,0000
0,0138	0,0000	0,0000	0,0005	0,0003	0,0004	2,1790	0,0000	0,0000	0,0100	0,0033	0,0000
0,4947	0,0064	0,0000	0,0481	0,0469	0,0808	0,0459	0,0002	0,0005	0,0089	0,0004	0,0004
0,0082	0,0008	0,0000	0,0092	0,0008	0,0067	0,0315	0,0000	0,0004	0,0000	0,0029	0,0007
2	0	0	4	0	3	3	0	0	0	7	7

Fréquences des bigrammes en langue anglaise (‰) :

	-	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
-	0,0000	25,565	8,6762	6,3167	6,1128	5,0648	6,1407	4,0881	16,146	11,024	0,5387	1,5976	5,2826	9,2328	5,0129	14,227	4,1935	0,4108	4,7362	14,449	27,962	3,4267	0,8069	14,665	0,0078	3,5440	0,0047
A	5,1136	0,0140	1,3441	1,8790	3,7757	0,0016	0,6046	1,3302	0,1093	3,5354	0,0178	1,1472	3,7618	1,7007	15,022	0,0085	0,7992	0,0008	5,1834	7,1213	8,5584	0,5852	1,8495	0,7465	0,0217	2,7890	0,0605
B	0,1450	0,7883	0,1705	0,0000	0,0062	4,2858	0,0054	0,0000	0,0054	0,3480	0,0884	0,0000	1,4720	0,0070	0,0070	1,8441	0,0000	0,0000	0,7651	0,1550	0,0845	1,4131	0,0047	0,0085	0,0000	0,8310	0,0000
C	0,1023	1,9836	0,0008	0,2698	0,0000	2,4564	0,0000	0,0000	0,0254	0,5287	0,0000	1,9627	0,6860	0,0016	0,0023	3,5913	0,0000	0,0380	0,5907	0,0062	0,9736	0,5193	0,0000	0,0000	0,0000	0,0698	0,0000
D	27,110	0,8449	0,0411	0,0093	0,5000	3,6323	0,0434	0,2217	0,0140	2,0720	0,0070	0,0256	0,3977	0,0674	0,7286	2,5991	0,0078	0,0008	0,6511	0,8193	0,0194	0,3201	0,0574	0,0271	0,0000	0,7170	0,0000
E	37,399	4,4788	0,1248	1,7115	7,9848	2,9812	0,8914	0,4643	0,2139	0,7527	0,0155	0,1116	3,4502	1,9774	7,4747	0,1853	0,9302	0,0775	13,339	4,9075	0,0518	0,0186	1,5914	0,5876	0,0000	1,4565	0,0109
F	6,2353	0,8333	0,0062	0,0062	0,0039	1,2023	0,7953	0,0008	0,0031	1,2898	0,0008	0,0000	0,3496	0,0016	0,0085	2,9487	0,0147	0,0000	0,9271	0,0217	0,0463	0,6434	0,0000	0,0054	0,0000	0,0295	0,0000
G	6,9259	1,0061	0,0101	0,0070	0,0070	2,4440	0,0023	0,5224	0,5014	0,6472	0,0008	0,0008	0,3883	0,0163	0,1798	1,8836	0,0132	0,0000	0,8279	0,4077	0,0481	0,3682	0,0000	0,0597	0,0000	0,0186	0,0000
H	4,7113	9,5808	0,0310	0,0078	0,0171	21,767	0,0264	0,0016	0,0093	6,9042	0,0000	0,0031	0,0496	0,0674	0,0256	3,6192	0,0070	0,0000	0,4558	0,0775	1,8154	0,4930	0,0000	0,0093	0,0000	0,2938	0,0000
I	0,0589	0,3442	0,2232	2,0200	3,3564	1,3960	1,2441	0,2193	0,0016	0,0000	0,0008	0,5333	1,9976	2,8138	14,707	1,6782	0,5744	0,0078	1,7542	5,6252	7,5399	0,0225	0,9736	0,0016	0,1442	0,0000	0,0752
J	0,0000	0,0504	0,0000	0,0000	0,0000	0,1589	0,0000	0,0000	0,0000	0,0279	0,0000	0,0000	0,0000	0,0000	0,0000	0,1240	0,0000	0,0000	0,0000	0,0000	0,0000	0,3752	0,0000	0,0000	0,0000	0,0008	0,0000
K	0,0215	0,0364	0,0039	0,0062	0,0023	2,9138	0,0705	0,0016	0,0047	1,1433	0,0008	0,0070	0,0860	0,0023	0,9658	0,1380	0,0008	0,0000	0,0062	0,3721	0,0031	0,0062	0,0000	0,0132	0,0000	0,0760	0,0000
L	4,5617	2,6712	0,0209	0,0233	3,1192	5,1865	0,6651	0,0147	0,0054	3,0200	0,0008	0,3170	4,6989	0,0543	0,0264	2,8076	0,1341	0,0000	0,0395	0,3767	0,3860	0,3302	0,1132	0,2023	0,0000	2,0681	0,0000
M	4,2230	2,3758	0,4969	0,0085	0,0000	6,2903	0,0636	0,0000	0,0023	1,8216	0,0008	0,0000	0,0853	0,5449	0,0457	1,7456	0,7162	0,0000	0,0147	0,4736	0,0109	0,6814	0,0008	0,0000	0,0000	1,9999	0,0000
N	14,879	0,7403	0,0178	1,4968	13,024	4,4759	0,2403	7,8236	0,0271	1,4247	0,0558	0,6767	0,3356	0,0209	0,4039	3,8781	0,0186	0,0372	0,0264	1,4053	5,8508	0,2333	0,1682	0,0295	0,0000	0,7410	0,0016
O	10,155	0,6519	0,3620	0,6348	1,2100	0,6705	4,9160	0,2806	0,0295	0,6573	0,0031	1,5681	1,6666	3,5254	8,3902	3,5719	1,1240	0,0016	5,8276	1,3061	3,2347	10,958	0,8523	3,8246	0,0411	0,3139	0,0302

	P	Q	R	S	T	U	V	W	X	Y	Z
P	2,0162	0,0000	11,831	19,258	24,982	2,4719	0,0202	2,5378	0,1085	11,318	0,0473
Q	1,2232	0,0000	2,1983	3,1386	4,1417	0,3085	0,2488	5,6485	0,0713	1,1271	0,0109
R	0,0124	0,0000	0,3147	0,0194	0,0217	0,3294	0,0000	0,0109	0,0000	0,1961	0,0008
S	0,0016	0,0000	0,3511	0,4682	0,4938	0,9705	0,0000	0,0078	0,1077	0,0109	0,0000
T	0,0008	0,0000	1,2705	0,0279	0,0047	0,4093	0,0000	0,0612	0,0000	0,0395	0,0000
U	0,1627	0,0000	9,5452	5,1609	5,4811	5,4008	0,1555	3,0959	0,0806	0,7224	0,1248
V	0,0016	0,0000	0,1341	0,0612	0,0519	0,1023	0,0000	0,0442	0,0023	0,0248	0,0000
W	0,0000	0,0000	0,3690	0,0256	0,0062	1,1030	0,0000	0,0093	0,0000	0,0047	0,0000
X	0,1194	0,0000	0,0690	0,1486	3,1434	0,0000	0,0000	0,0727	0,0171	0,0140	0,0000
Y	0,7418	0,0000	2,5812	2,0208	3,7463	0,5411	1,0751	2,8239	0,0628	0,2961	0,0271
Z	0,0008	0,0000	0,0016	0,0008	0,0016	0,0000	0,0000	0,0000	0,0000	0,0016	0,0000
	0,0039	0,0000	0,4457	0,3938	0,0016	0,0767	0,0000	0,0054	0,0000	0,0039	0,0000
	1,0643	0,0000	0,3666	0,5302	1,0782	2,8680	0,0000	0,1395	0,0000	0,0729	0,0155
	0,0062	0,0000	0,4465	0,3155	0,0519	0,6922	0,0000	0,0031	0,0000	0,0527	0,0000
	0,0023	0,0000	1,1046	0,1589	0,0884	0,6153	0,0000	1,0495	0,0000	0,0341	0,0016
	1,3759	0,0000	3,3207	2,7347	3,2747	0,0178	0,1589	2,0557	0,0054	3,0083	0,0054
	0,7891	0,0000	0,1899	0,7775	0,0132	1,5356	0,0000	0,0008	0,2248	0,0124	0,0000
	0,0000	0,0000	0,0000	0,0341	0,0000	0,0000	0,0000	0,0000	0,0016	0,0000	0,0000
	1,2263	0,0000	0,5984	0,0078	1,5193	2,9239	0,0016	0,2186	0,0008	0,0488	0,0000
	0,3837	0,0000	2,3169	1,8983	1,3922	2,5479	0,0000	0,1876	0,0023	1,2116	0,0000
	0,3046	0,0000	1,7704	5,7524	1,5487	4,3463	0,0000	0,0085	0,2155	0,2170	0,0000
	0,5961	0,6100	0,6883	1,1348	0,9007	0,0023	0,0031	0,0372	0,0070	0,0023	0,0000
	0,0000	0,0000	0,2256	0,0062	0,0000	0,0349	0,0023	0,0000	0,0000	0,0194	0,0000
	0,0062	0,0000	0,0814	0,2558	0,5170	0,0000	0,0000	0,0093	0,0031	0,0550	0,0000
	0,0000	0,0000	0,0008	0,0000	0,0000	0,0054	0,0000	0,0000	0,0000	0,0000	0,0000
	0,0000	0,0000	1,4689	0,1349	0,7519	0,0155	0,0411	0,0798	0,0023	0,0016	0,0124
	0,0000	0,0000	0,0000	0,0000	0,0039	0,0581	0,0000	0,0008	0,0000	0,0000	0,0194
	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000

Bibliographie :

[1] H. A. ABBASS, R. A. SARKER, C. S. NEWTON.
Data Mining: A Heuristic Approach.
Eds., Idea Group Publishing, 2001.

[2] J.M. ALLIOT, T. SCHIEX.
Intelligence Artificielle et Informatique Théorique.
Cépaduès édition, Toulouse, 1994.

[3] T. BÄCK.
*Evolutionary Algorithms in Theory and Practic*e.
Oxford University Press, New York, 1995.

[4] A. J. BAGNALL.
The Application of Genetic Algorithms in Cryptanalysis.
A Master of Scince Thesis, University of East Anglia, 1996.

[5] R. BATTITI, M. PROTASI.
Reactive Search, a History-Based Heuristics for MAX-SAT.
Journal of experimental algorithmic, 1997.

[6] E. BIHAM, A. SHAMIR.
Differential Cryptanalysis of DES-like Cryptosystems.
Journal of Cryptology, vol. 4, n°1, 1991.

[7] V. CAMPOS, F. GLOVER, M. LAGUNA, R. MARTÍ.
An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem.
*Journal of Global Optimizatio*n, vol. 21, 2001.

[8] V. CAMPOS, M. LAGUNA, R. MARTÍ.
Context-Independent Scatter and Tabu Search for Permutation Problems.
INFORMS Journal on Computing, 2004.

[9] S. A. Canuto, M. G. C. Resende, C.C. Ribeiro.
Local Search with Perturbations for the Prize-Collecting Steiner Tree Problem in Graphs.
AT&T Labs Research Technical Report, 1999.

[10] A. J. CLARK.
Optimisation Heuristics for Cryptology.
PhD Thesis, Queensland University of Technology, 1998.

[11] A. J. CLARK, E. DAWSON.
Optimisation Heuristics for the Automated Cryptanalysis Classical Ciphers.
*JCMMC*C, vol. 28, 1998.

[12] A. Colorni, M. Dorigo & V. Maniezzo.
Distributed Optimization by Ant Colonies.
*European Conference on Artificial Lif*e, 1991.

[13] S. A. COOK.
The Complexity of Theorem Proving Procedures.
Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing, 1971.

[14] M. M. CORSINI.
Réseaux de Neurones Artificiels : Un Survol et une Bibliographie Commentée.
Rapport Interne, Université V. Segalen Bordeaux 2, 2000.

[15] V. D. Cung, T. Mautor, P. Michelon, A. Tavares.
Scatter Search for the Quadratic Assignment Problem.
Rapport technique, Laboratoire PRiSM-CNRS URA, 1996.

[16] M. DAVIS, H. PUTNAM.
A Computing Procedure for Quantification Theory.
Journal of the ACM, vol. 7, n° 3, 1960.

[17] E. DAWSON, A. CLARK.
Optimisation Heuristics for the Automated Cryptanalysis Classical Ciphers.
JCMMCC, vol. 28, 1998.

[18] W. DIFFIE, M.E. HELLMAN.
New Directions in Cryptography.
IEEE Transactions on Information Theory, vol. IT-22, n°6, 1976.

[19] A. DIMOVSKI, D.GLIGOROSKI.
Attack On The Polyalphabetic Substitution Cipher Using A Parallel Genetic Algorithm.
Technical Report, Swiss-Macedonian scientific cooperation trought SCOPES project, Macedonia, 2003.

[20] A. DIMOVSKI, D. GLIGOROSKI.
Attacks on the Transposition Ciphers Using Optimization Heuristics
Proceedings of ICEST, Bulgaria, 2003.

[21] M. DORIGO, G. DI CARO, L. M. GAMBARDELLA.
Ant Algorithms for Discrete Optimization.
Artificial Life, 5(2), 1999.

[22] H. DRIAS, M. KHABZAOUI.
Scatter Search with Random Walk Strategy for SAT and MAX-W-SAT Problems.
Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Hungary, 2001.

[23] T. A. FEO, M. G. C. RESENDE, S. H. SMITH.
A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set.
Operations Research, 1994.

[24] C. Fleurent, F. Glover, P. Michelon, Z. Valli.
A Scatter Search Approach for Unconstrained Continuous Optimization.
Proceedings of the IEEE International Conference on Evolutionary Computation, 1996.

[25] W. S. FORSYTH, R. SAFAVI-NAINI.
Automated cryptanalysis of substitution ciphers.
Cryptologia, n°17, vol. 4, 1991.

[26] W. F. FRIEDMAN.
The Index of Coincidence and its Applications in Cryptography.
Riverbank Publication, n°22, 1920.

[27] J. P. GIDDY, R. SAFAVI-NAINI.
Automated Cryptanalysis of Transposition Ciphers.
The Computer Journal, n°17, vol. 4, 1994

[28] F. GLOVER, M. LAGUNA, R. MARTÍ.
Fundamentals of Scatter Search and Path Relinking.
Control and Cybernetics, vol. 39, n° 3, 2000.

[29] F. GLOVER.
Heuristics for Integer Programming Using Surrogate Constraints.
Decision Sciences, vol. 8, 1977.

[30] F. GLOVER.
Future Paths for Integer Programming and Links to Artificial Intelligence.
Computers and Operations Research, 1986.

[31] F. GLOVER.
Tabu Search : A Tutorial.
Special Issue on the Practice of Mathematical Programming, vol. 20, n°1, 1990.
[32] F. GLOVER.
Tabu Search Fundamentals and Uses.
Technical Report, University of Colorado, 1994.
[33] F. GLOVER.
A Template for Scatter Search and Path Relinking.
Notes in Computer Sciences, Spinger-Verlag, 1998.
[34] D. E. GOLDBERG.
Genetic Algorithms in Search, Optimisation and Machine Learning.
Addison-Wesley, 1989.
[35] P. GREISTORFER.
Computational Experiments with Heuristics for a Capacitated Arc Routing Problem.
Operations research proceedings, Springer-Verlag, 1994.
[36] J. P. HAMIEZ, J. K. HAO.
Scatter Search for Graph Coloring.
Artificial Evolution, 2001.
[37] P. HANSEN.
The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming.
*Congress on Numerical Methods in Combinatorial Optimizatio*n, Capri, Italie, 1986.
[38] J. K. HAO, F. LARDEUX, F. SAUBION.
Evolutionary Computing for the Satisfiability Problem.
EvoWorkshops, 2003.
[39] J. K. HAO, P. GALINIER, M. HABIB.
Métaheuristique pour l'optimisation combinatoire et l'affectation sous contraintes.
Revue d'Intelligence Artificielle, vol. , n°, 1999.
[40] J.H. HOLLAND.
Adaptation in Natural and Artificial Systems.
The University of Michigan Press, Ann Arbor, 1975.
[41] D. S. JOHNSON.
Approximation algorithms for combinatorial problems.
Journal of Computer and System Sciences, vol. 9, 1974.
[42] J. Kelly, B. Rangaswamy, J. Xu.
A Scatter Search-Based Learning Algorithm for Neural Network Training.
Journal of Heuristics, vol. 2, 1996.
[43] A. KERCKHOFFS.
La cryptographie militaire.
Journal des sciences militaires, vol. 9, 1883.
[44] S. KIRKPATRICK, JR. C. D. GELATT, M. P. VECCHI.
Optimisation by Simulated Annealing.
*Scienc*e, 220(4598), 1983.
[45] P. J. M. VAN LAARHOVEN, E. H. L. AARTS.
Simulated annealing: theory and applications.
Kluwer Academic Publishers, 1987.
[46] M. Laguna, H. Lourenço, R. Martí.
Assigning Proctors to Exams with Scatter Search.
Computing Tools for Modeling, Optimization and Simulation, Kluwer Academic Publishers, 2000.

[47] M. LAGUNA, R. MARTÍ.
Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions.
Technical report, University of Colorado, 2002.

[48] M. LAGUNA AND V. ARMENTANO
Lessons from Applying and Experimenting with Scatter Search.
Adaptive Memory and Evolution: Tabu Search and Scatter Search, Cesar Rego and Bahram Alidaee Editions, 2003.

[49] S. LUNDY, A. MEES.
Convergence of an Annealing Algorithm.
MathematicalProgramming, 1986.

[50] F. MASSACCI, L. MARRARO.
Logical Cryptanalysis as a SAT Problem.
Journal of Automated Reasoning, vol. 24, 2000.

[51] R. A. J. MATHEWS.
The Use of Genetic Algorithms in Cryptanalysis.
Cryptologia, n°17, vol. 2, 1993.

[52] R. MAZZINI.
A meta-heuristics Study for a parallel machine program.
PhD thesis, Campinas State University, 1998.

[53] R. C. MERKLE, M. HELLMAN.
Hiding and Signature in Trapdoor Knapsacks.
IEEE Transactions on Information Theory, vol. 24, n°5, 1978.

[54] Z. MICHALEWICZ.
Evolutionary Computation Techniques for Nonlinear Programming Problems.
International Transactions in Operational Research, vol. 1, n°2, 1994

[55] C. H. PAPADIMITRIOU, K. STEIGLITZ.
Combinatorial Optimization Algorithms and Complexity.
Prentice Hall, 1982.

[56] L. S. PITSOULIS, M. G. C. RESENDE.
Greedy Randomized Adaptive Search Procedures.
Handbook of Applied Optimization, Oxford University Press, 2001.

[57] C. R. REEVES.
Modern Heuristic Techniques for Combinatorial Problems.
Blackwell Scientific Publications, Oxford, 1993.

[58] C. C. RIBEIRO, N. MACULAN.
Applications of combinatorial optimization.
Annals of Operations Research 50, 1994.

[59] R. RIVEST, A. SHAMIR, L. ADLEMAN.
A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.
Communications of the ACM, vol. 21, n°2, 1978.

[60] Y. Rochat, E. D. Taillard.
Probabilistic Diversification and Intensification in Local Search for Vehicule Routing.
Journal of Heuristics, 1995.

[61] M. RUSSELL, J. A. CLARK, S. STEPNEY.
Using ants to attack a classical cipher.
Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), 2003.

[62] C. E. SHANNON.
A Mathematical Theory of Communication.
Bell System Technical Journal, vol. 27, n° 4, 1948.

[63] C. E. SHANNON.
Communication Theory of Secret Systems.
Bell System Technical Journal, vol. 28, n° 4, 1949.

[64] A. SINKOV.
Elementary cryptanalysis. *The American Association of America*, 1966,

[65] R. SPILLMAN, M. JANSSEN, B. NELSON, M. KEPNER.
Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers.
Cryptologia, n°17, vol. 1, 1993.

[66] E. D. TAILLARD.
Programmation à Mémoire Adaptative et Algorithmes Pseudo-Gloutons.
Thèse de Habilitation à Diriger des Recherches, Université de Versailles Saint-Quentin-en-Yvelines, 1998.

[67] W. TUCHMAN.
Hellman Presents No Shortcut Solutions to DES.
IEEE Spectrum, vol. 16, n°7, 1979.

[68] W. ZIMMERMANN.
PGP User's Guide, 1992.

Résumé. Dans ce travail, nous tentons d'explorer la contribution pour la cryptanalyse des chiffres simples d'une méta-heuristique récente et souvent méconnue : la *'Recherche Dispersée'* (*Scatter Search*). Nous essayons d'illustrer la faisabilité de son utilisation en tant qu'outil de cryptanalyse. Cette méta-heuristique évolutive basée population, est basée sur une approche pour la programmation discrète développé par Fred GLOVER. Elle utilise des combinaisons linéaires de parties de la population pour créer de nouvelles solutions.
Nous avons implémenter les méthodes de la recherche dispersée pour la cryptanalyse des chiffres simples ; ensuite, nous avons effectué des expérimentation pour fixer les paramètres des procédures, et pour évaluer les performances de la méthode ; enfin, nous avons dressé une comparaison avec un algorithme génétique.

Mots-clé. Cryptanalyse. Chiffres simples. Substitution. Transposition. Recherche Dispersée. Recherche heuristique. Approche évolutive. Problème d'optimisation.

Abstract. In This work, we tempt to explore the contribution of a recent meta-heuristic –the *Scatter Search*– for the cryptanalysis of simple ciphers. We try to illustrate the feasibility of its use as a cryptanalysis tool. This evolutionary meta-heuristic is based on an approach for the discreet programming developed by Fred GLOVER. It use linear combinations of population's subsets in order to generate new ones.
We've implemented scatter search's methods for the cryptanalysis of simple ciphers; then, we did the experimentation to fix parameters of the procedures, and to evaluate the method's performances. finally, we raised a comparison with a genetic algorithm.

Keywords. Cryptanalysis. Simple ciphers. Substitution. Transposition. Scatter Search. Meta-heuristic. Evolutionary Algorithm. Optimisation problems.