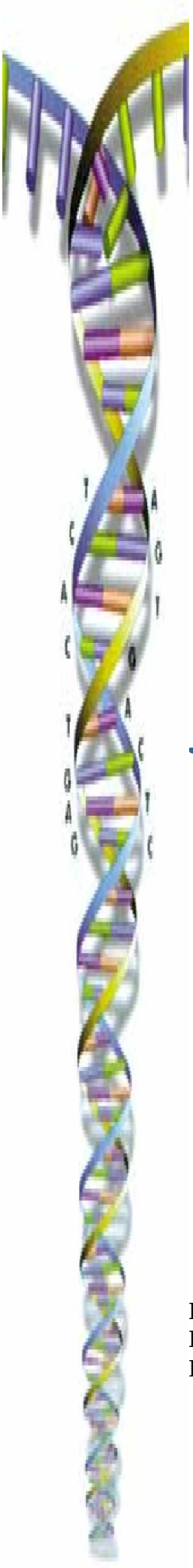

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPOLAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SIENTIFIQUE

Université Elhadj Lakhder, Batna
Faculté des sciences de l'ingénieur
Département d'informatique
Option : Informatique industrielle

Mémoire

Pour l'obtention du diplôme de magister

Systèmes bio-inspirés pour le traitement de l'information : *Application du DNA computing à la résolution de problèmes NP-complets*

Présenté et soutenu par

AKSA KARIMA

Le :

Directeur de Mémoire : Pr M.C.Batouche

Composition du JURY

Dr A.Zidani	**Maître de conférence**	**Président**
Dr M.Benmohammed	**Maître de conférence**	**Examinateur**
Dr D.E.Saidouni	**Maître de conférence**	**Examinateur**

Remerciements

Tous mes remerciements s'adressent tous d'abord à tout puissant ALLAH, d'avoir guidé mes pas vers le chemin de savoir.

Je tiens à remercier Mr M.C..Batouche, Professeur à l'université Mentouri de Constantine, pour son encadrement et ses encouragements à travers son attention, sa patienté, et ses conseils sérieux

Je tiens à remercier également Mr Dr A.Zidani,Maître de conférence à l'université de Batna,d'avoir accepté d'être le président de jury.

Je remercie vivement Mr Dr M.Benmohammed, Maître de conférence à l'université de Constantine, et Mr Dr D.E.Saidouni, Maître de conférence à l'université de Constantine, d'avoir accepté d'être examinateurs.

Enfin, que toutes les personnes qui ont participé de prés ou de loin à la réalisation de ce travail trouvent ici l'expression de mes sincères remerciements.

Résumé

La solution d'un problème combinatoire, quand elle existe, peut être déterminée par l'énumération d'un ensemble fini E de possibilités. S'il y a un algorithme simple pour tester si un élément de cet ensemble est une solution, on dispose d'un algorithme pour trouver une solution au problème en testant toutes les possibilités. Mais le cardinal de E est souvent exponentiel par rapport à la taille des données, ce qui rend l'énumération impossible dans un temps raisonnable.

Il existe de très nombreux problèmes d'un grand intérêt pratique pour lesquels on ne connaît pas d'algorithme plus efficace qu'un test portant sur tous les sous ensembles d'un ensemble, ce qui implique un temps de calcul exponentiel par rapport à la taille de l'ensemble.

De nos jours, certains problèmes nécessitent, pour être résolue une capacité de calcul phénoménale. L'ordinateur actuel étant limité dans sa puissance de calcul constitue une barrière dans la résolution de tels problèmes. C'est dans cette optique que de nouveaux concepts d'ordinateurs sont développés. Ainsi on tente de mettre sur pied des ordinateurs quantiques et des ordinateurs moléculaires. Ces derniers représentent certainement l'alternative de demain aux ordinateurs actuels.

En utilisant l'ADN, contrairement aux ordinateurs actuels, cela permet de réaliser des opérations en parallèles (multitask) avec des vitesses de calculs phénoménales. De plus, l'ADN constitue un excellent moyen de stockage de données et ne demande qu'un apport infime en énergie.

L'expérience qui a éveillé le monde des ordinateurs à ADN est celle d'Adleman. En 1994, Adleman, qui est un mathématicien de l'université de Californie du sud et qui possède également des connaissances en biologie, à résolut grâce à un ordinateur moléculaire le problème de chemin haméltonien (HPP : Hamiltonian Path Problem).

Depuis l'expérience originale d'Adleman, des inondations d'idées ont été proposées pour la résolution des différents problèmes NP-complets. Dans ce mémoire, deux nouveaux algorithmes ont été proposés pour résoudre deux problèmes NP-complets : le Problème de Voyageur de Commerce (TSP : Traveling Salsmen Problem), et celui de la Satisfaisabilité (SAT : Satisfiability problem). Théoriquement, ces deux algorithmes proposés et d'autres types d'algorithmes peuvent être exécutés également avec succès. Mais pratiquement ils seront très difficiles pour l'application, car ici on travaille avec la biologie qui nécessite la prudence pour appliquer des opérations biologiques, tel que la dénaturation, sur des molécules vivantes.

Malgré le fait que les ordinateurs moléculaires (ordinateurs d'ADN) semblent être très performants, ils possèdent de nombreux aspects négatifs. Ils ne peuvent résoudre que des problèmes combinatoires (pas de possibilité de traitement de texte ou de jeu sur de tels ordinateurs), ils peuvent être très lents dans la résolution de problèmes simples pour des ordinateurs classiques. Les réponses qu'ils fournissent peuvent être extrêmement compliquées. D'autre part la fiabilité de ces ordinateurs peut être remise en cause du fait de la capacité de la mutation de l'ADN.

MOTS CLES : Calcul moléculaire, Calcul de l'ADN, Ordinateur à l'ADN, Bio-calcul, Problèmes NP-complets, Bio-informatique, Systèmes bio-inspirés, Biologie moléculaire, ADN.

Abstract

The solution of a combinatorial problem, when it exists, can be determined by the enumeration of a finished set E of possibilities. If there is a simple algorithm to test if an element of this set is a solution, one has an algorithm to find a solution of the problem by testing all the possibilities. But the cardinal of E is often exponential with regard to the size of data, what returns impossible enumeration in a reasonable time.

There are very numerous problems of a big practical interest for which one does not know algorithm more effective than a test concerning all the sub-sets of a set, what implies a time of exponential calculation with regard to the size of the set.

Nowadays, certain problems require, to be resolved a phenomenal capacity of calculation. The current computer being limited in its power of calculation constitutes a barrier in the resolution of such problems. It is in this optics that new concepts of computers are developed. So one tries to set up quantum computers and molecular computers. These last ones represent certainly the alternative of tomorrow to the current computers.

By using the DNA, contrary to the current computers, it allows to realize operations in parallel (multitask) with phenomenal speeds of calculations. Furthermore, the DNA constitutes an excellent means of stocking of data and asks only for a tiny contribution in energy.

The experience which has to wake the world of computers to DNA is that of Adleman. In 1994, Adleman, who is a mathematician of the University of South California and who also possesses knowledge in biology, has resolved due to a molecular computer the Hamiltonian Path Problem (HPP).

Since the original experience of Adleman, floods of ideas were proposed for resolutions of various NP-complets problems. In this memoir, two new algorithms were proposed to resolve two NP-complets problems: Commercial traveller's problem Traveling Salesmen Problem (TSP), and that of Satisfiability problem (SAT). Theoretically, these two algorithms proposed and other types of algorithms can be executed successfully. But practically they will be very difficult for application, because here one works with biology which necessitates prudence to apply biologic operations, such as denaturation, on alive molecules.

In spite of the fact that the molecular computers (DNA computers) seem to be very successful, they possess numerous negative aspects. They can resolve only combinatorial problems (no possibility of text processing or game on such computers), they can be very slow in the resolution of simple problems for classic computers. The answers which they supply can be extremely complicated. On the other hand the reliability of these computers can be questioned because of the capacity of the DNA mutation.

WORD KEY : Molecular computing, DNA computing, DNA computer, Biocomputing, NP-completes problems, Bio-informatics, Bio-inspired systems, Molecular biology, DNA.

Table de matières

Chapitre2 : L'étude biologique de l'ADN

Chapitre3 : L'expérience d'Adleman & les ordinateurs à l'ADN

Chapitre4 : Résolution des problèmes NP-complets par le calcul d'ADN

Liste des figures

Liste des tableaux

INTRODUCTION GENERALE

La biologie regroupe l'ensemble des « Sciences de la Vie » (Life Sciences). Les sciences de la vie traitent des microorganismes (microbiologie), des êtres unicellulaires (protistes), des végétaux (botanique), des animaux (zoologie) et de l'homme (anthropologie). La biologie étudie les propriétés des êtres vivants, telles que leur morphologie et leur structure, leur cycle reproductif ainsi que l'évolution des espèces vivantes; elle peut être scindé en grandes disciplines [1].

La vie est essentiellement l'histoire de deux familles de molécules, toutes deux des polymères : les protéines et l'ADN. Il existe bien sûr beaucoup d'autres familles de molécules biologiquement importants : les membranes de cellules sont formées de lipides, le monde végétale n'est rien sans la cellulose (polymère de sucre), ... Mais ils ont quand même des seconds rôles [2].

L'apparition de l'informatique a permis à l'homme de résoudre un grand nombre de problèmes ; la puissance de calcul exigée les rendant impossibles à réaliser à la main. Dès lors est apparu l'algorithmique, science informatique proposant les solutions aux problèmes posés. Néanmoins, on ne connaît pas d'algorithmes efficaces pour certains problèmes. On développe tout de même des algorithmes d'approximations permettant de les résoudre mais pas de manière optimale. Ainsi, on connaît un algorithme pour la recherche du réseau le plus court, s'exécutant en temps acceptables mais le résultat peut présenter une erreur d'au plus 17% [70].

Depuis longtemps, les chercheurs cherchent à échapper à la logique binaire de l'informatique et aux limites de l'électronique. Parmi les pistes explorées : l'ADN, la signature génétique du vivant.

Difficile pour un ordinateur, si puissant soit-il, de tout garder en mémoire. Pourquoi ? Parce que l'outil que l'on connaît raisonne uniquement en binaire. Pourtant, ses capacités grimperaient en flèche s'il pouvait utiliser un langage plus élaboré. Le langage binaire ne code en effet que deux valeurs, à partir d'impulsions électriques : 0 - le courant ne passe pas - et 1 - le courant passe. Les instructions sont exprimées avec des successions de 0 et de 1... D'où les limites du binaire.

Il existe pourtant un moyen de s'en affranchir : la biologie. Dans l'imagination hypertrophiée de chercheurs s'est progressivement développée l'idée de substituer la génétique à l'électricité. Car l'alphabet de l'ADN est, lui, composé de quatre lettres : A, C, G, T, abréviation des quatre constituants formant les *" briques "* du matériel génétique : adénine, cytosine, guanine et thymine.

Le calcul d'ADN (DNA computing), nommé aussi le calcul moléculaire (molecular computing), c'est une nouvelle approche vers le calcul massivement parallèle basé sur le travail révolutionnaire d'Adleman (qui sera bien expliqué au cours de ce mémoire). Ce dernier a utilisé le principe d'ADN pour résoudre le problème de chemin hamiltonien (HPP) avec sept nœuds, un cas spécial du problème NP-complet qui tente à visiter chaque nœud dans le graphe exactement une seule fois.

Des chercheurs israéliens ont réussi de mettre au point un ordinateur biologique, qui carbure à l'ADN et aux enzymes. Ce la a été fait le 03/12/2001 où Ehud Shapiro et ses collègues de l'Institut Weizmann, à Réhovot, ont fabriqué un ordinateur dans lequel l'ADN remplace la traditionnelle puce en silicium [2].

Dans cet appareil, les enzymes font office de circuits électroniques. Elles coupent les séquences d'ADN quand apparaît une séquence spécifique composée des quatre lettres.
La machine ainsi créée peut effectuer un calcul mathématique très simple avec mille milliards de chiffres. Toutefois, l'objectif d'Ehud Shapiro n'est pas de concurrencer l'informatique, afin de résoudre des problèmes mathématiques complexes, mais de maîtriser la biologie [2].

Enfin, Il est escompté que de nouvelles technologies pour la production des machines pour le calcul moléculaire seront disponibles, dans les marchés, en 2050 [116].

Organisation de mémoire

Le but principal de ce mémoire est de montrer comment résoudre des problèmes complexes (problèmes NP-complets) avec le calcul d'ADN ou le calcul moléculaire.

Le mémoire se compose de quatre chapitres. Le premier chapitre décrit les systèmes biologiques qui représentent la source d'inspiration pour les applications technologiques. Ce chapitre va nous répondre à des questions qui seront posées par toute personne qui entend par les systèmes bio-inspirés : pourquoi étudier les systèmes biologiques ? Pour quoi on est intéressé à ce genre de systèmes pour l'inspiration ? Quelle est la relation entre le domaine de l'informatique et celui de la biologie ?....toutes ces questions et d'autres, leurs réponses seront éclaircies avec des exemples et des schémas dans ce premier chapitre.

Le deuxième chapitre représente une trajectoire au troisième chapitre. Cela signifie que sans passer à travers ce chapitre, le troisième ne sera plus compréhensible ! Donc,le but de ce chapitre, en cours de description, est de présenter un nouveau type d'ordinateur qui sera utilisé pour résoudre des problèmes complexes (les problèmes NP-complets) dans un temps très réduit. Ces ordinateurs sont appelés « ordinateurs d'ADN », leurs noms indiquent qu'ils sont basés sur un principe biologique qui est celui de la molécule d'ADN. Pour cela ce deuxième chapitre sera purement biologique. Il va nous prendre à travers une vingtaine de page dans un voyage surprenant pour savoir un autre monde qui ne sera pas vu par l'œil nue ; c'est le monde de la biologie moléculaire et plus précisément le miracle d'ADN.

Le troisième chapitre sera consacré aux ordinateurs d'ADN. Après l'étude biologique de la molécule d'ADN, dans le deuxième chapitre, nous pouvons maintenant présenter cette nouvelle machine de calcul, qui est basée sur des molécules d'ADN et des enzymes. Et pour présenter cette nouvelle machine, nous somme obligés de présenter l'expérience de son fondateur Lionard Adleman. L'expérience de ce dernier se présente dans la résolution de problème de chemin Hamiltonien (HPP) qui est un des problèmes NP-complets les plus connus. Pour bien et mieux comprendre cette expérience, un exemple d'application sera bien détaillé par la suite. Comme toute les inventions, les ordinateurs d'ADN ont des limites qui seront mentionnées à la fin de ce chapitre.

La résolution des problèmes NP-complets par le calcul d'ADN sera le but du dernier chapitre. Pour faire cela, on est obligé de passer par quelques généralités sur la complexité des problèmes tel que : la notion de la théorie de complexité, problèmes de décision, d'optimisation, P, NP et NP-complets…Quelques exemples sur les problèmes NP-complets seront mentionnés, par la suite, tel que le TSP (Traveling Salesman Problem) qui est le problème de voyageur de commerce et celui de la satisfaisabilité (SAT). Ces deux derniers problèmes (TSP et SAT) seront résolus par le calcul d'ADN en utilisant deux algorithmes bien détaillés et avec des exemples. Deux nouveaux algorithmes pour la résolution de ces deux problèmes, bien sûr avec le calcul d'ADN, seront proposés à la fin de ce chapitre et avec des exemples d'application.

1.1 Introduction

Des avancées récentes dans les techniques d'IA ont été réalisées en s'inspirant de phénomènes physiques (recuit simulé), ou biologiques (réseaux de neurones, algorithmes génétiques, sociétés d'insectes, [9]). Plus récemment, des tentatives sont menées pour utiliser les capacités de calcul massivement parallèle d'entités biologiques pour résoudre des problèmes NP-Complets [84]. Ces différentes approches procèdent de la démarche suivante. Certains phénomènes naturels sont capables de mettre en œuvre des heuristiques originales susceptibles de résoudre des problèmes pour lesquels il est difficile de trouver des solutions de manière déterministe par des algorithmes classiques. De plus, ces heuristiques sont robustes, une défaillance d'un élément constitutif de l'heuristique ne mettant pas en péril l'heuristique dans son ensemble.

Ce premier chapitre est consacré pour l'inspiration biologique. On va parler un petit peu de l'intelligence artificielle et de la vie artificielle et d'une façon générale de systèmes bio-inspirés dans lesquels on va voir quelques système biologiques très intéressant tel que l'intelligence d'essaim, le système immunitaire, les réseaux de neurones …et comment utiliser les principes de ces systèmes pour trouver des nouvelles idées qui seront utilisées par la suite pour trouver de nouvelles techniques et résoudre différents problèmes.

1.2 Intelligence artificielle

L'intelligence artificielle ou l'IA (Artificial Intelligence ou AI) peut être définie comme:[3]

1 - Quatre vues:
- Penser comme un humain.
- Agir comme un humain.
- Penser raisonnablement.
- Agir raisonnablement.

2 - Agir raisonnablement: Faire la bonne chose.

3 -Etudier des agents intelligents

L'intelligence artificielle apparaît dans les années cinquante, lorsque le développement des premiers ordinateurs donne à penser qu'ils seront rapidement capables de simuler la pensée. Le mathématicien britannique Alan Mathison Turing propose ainsi, en 1950, un test (appelé test de Turing) permettant d'évaluer l'intelligence d'une machine : un ordinateur est qualifié d'intelligent si, en communiquant avec lui à distance et par écrit, un utilisateur ne peut deviner s'il s'agit ou non d'un être humain [4]. Turing et d'autres chercheurs, comme Donald Michie, pensent qu'un tel ordinateur peut voir le jour avant la fin du XXe siècle. Mais les résultats ne sont pas à la hauteur des attentes, car les difficultés pour reproduire les mécanismes de l'intelligence humaine se révèlent considérables [4]. Toutefois, les recherches menées en intelligence artificielle permettent de concevoir des ordinateurs capables d'effectuer des déductions, d'apprendre à partir d'une expérience passée, ou de reconnaître des formes complexes. Aujourd'hui, les travaux réalisés sur la modélisation des connaissances et du raisonnement expert sont effectués dans le but de développer des systèmes experts et plus généralement des systèmes à bases de connaissances [4].

L'IA a connu de nombreux succès dans des applications extrêmement variées et particulièrement dans la conception de systèmes experts. Cependant au fil du temps, si elle a acquis le statut d'une science à part entière, elle s'est éloignée de son objet premier : créer des systèmes artificiels capables de reproduire les comportements intelligents de systèmes biologiques [5].

Initialement, les ambitions de l'IA étaient particulièrement élevées. On espérait pouvoir créer des automates capables de traiter des problèmes très généraux comme la reconnaissance du langage, la traduction des langues naturelles ou la résolution universelle de problèmes. Il était donc question de modéliser le savoir-faire de l'expert [5].

L'acquisition du langage naturel par une machine est un des enjeux essentiels de l'IA et indique clairement ses orientations théoriques. Cependant, la possibilité même qu'un ordinateur puisse « apprendre à parler » reste tout à fait hypothétique et l'idée qui sous-tend cette possibilité est que la machine peut et doit imiter le vivant. Un autre point de vue fait ressortir que, si l'ordinateur dispose de capacités remarquables pour résoudre certaines classes de problèmes, il est tout à fait impossible de le doter des caractéristiques perceptives et cognitives des êtres vivants [5].

1.3 Vie Artificielle

Pour préciser le domaine de la vie artificielle, distinguons la de l'intelligence artificielle (IA) [6]: l'IA prend la *cognition humaine* comme référence et s'intéresse à la conception de machines pouvant simuler la cognition humaine. La vie artificielle quand à elle tend à se rapprocher des *fonctionnements biologiques* et traite plus de comportements "réflexes" que de raisonnements logiques ou d'actes cognitifs. Son domaine d'étude est plus vaste, elle explore les caractéristiques du vivant en général. La Vie Artificielle exploite les concepts issus des courants cognitiviste et connexionniste qui partagent les sciences cognitives. Elle propose donc des solutions qui s'appuient tantôt sur l'approche symbolique, tantôt sur l'approche auto-organisationnelle, ou construit ses modèles à partir d'une combinaison des deux méthodologies. On peut dès lors distinguer deux types de travaux issus de la VA [6]. D'une part, les simulations qui utilisent exclusivement l'ordinateur. Il est admis, dans ce cas, que tout système est modélisable par ordinateur. C'est-à-dire qu'un système formel est susceptible de représenter de manière satisfaisante un système physique. D'autre part, les réalisations, où le caractère concret et matériel du système est primordial. On considère, ici, que la dimension physique d'un système est irréductible à une représentation symbolique [6].

1.3.1 Les critères de la vie artificielle

A partir de quand parle-t-on de vie artificielle ? Voici les *propriétés minimales* que nous retrouvons dans tout système de vie artificielle [7] :

- L'être humain a contribué au processus d'apparition de tout système de vie artificielle ;
- Un système de vie artificielle est autonome ;
- Un système de vie artificielle est en interaction avec son environnement ;
- Il y a émergence de comportements dans un système de vie artificielle ;

Ces 3 propriétés ne sont pas indispensables mais reste néanmoins très présentes [7]:

- Un système de vie artificielle peut se reproduire lui-même ;
- Un système de vie artificielle possède une capacité d'adaptation ;
- Un système de vie artificielle n'est pas une unité. A l'opposé de la vie, un système de vie artificielle peut être réparti en plusieurs endroits : exemple, un robot et un ordinateur peut effectuer des calculs reliés par ondes. Même à l'intérieur d'un ordinateur, rien ne garantie que les octets de ce système sont tous regroupés.

1.3.2 Les domaines de la vie artificielle

On peut aussi présenter la vie artificielle comme un ensemble de domaines que Claus Emmeche a classées en deux groupes principaux [7]: les versions triviales et non triviales de formes de vie artificielle. Pour chacun de ces groupes, nous pouvons y trouver la vie artificielle faible (recherche d'imitation de la vie) ou forte (recherche de création de vie).

Version triviale	Version non triviale
Simulation et modèles: Simulations informatiques fondées sur des modèles mathématiques, conceptuels ou physiques des systèmes biologiques. Elle ne peut aboutir à des formes réelles de vies	**Les systèmes computationnels:** Ces réalisations sont par définition strictement informatiques. Elles ont pour objectif la création d'organismes virtuels considérés comme vivants (du moins par leur auteur). C'est la voie forte la plus contestée de la vie artificielle.
Les organismes modifiés: Il s'agit d'êtres vivants réels, modifiés par l'homme au moyen de manipulations génétiques par exemple.	**Les expériences relevant de la biochimie:** Cette dernière catégorie inclut les synthèses de processus prébiotiques et d'organismes primitifs grâce à des expérimentations physico-chimiques in vitro.
	La robotique évolutive: Cela regroupe les robots autonomes et évolutifs.

Tableau 1.1 : Les domaines de la vie artificielle [7].

Ces formes de vie artificielle diffèrent fortement les unes des autres. En ces domaines les frontières sont mouvantes: celle entre vie naturelle et artificielle ainsi que celle entre vie artificielle et programme doté d'intelligence artificielle.

Il semble donc qu'il y ait plusieurs formes de vie artificielle ; et nous voyons qu'elles diffèrent énormément les unes des autres. Il serait donc peut-être plus opportun de ne pas les rassembler sous le même terme, mais de créer de nouveaux noms pour chacune de ces catégories.

1.4 Les systèmes bio-inspirés

L'évolution des relations entre l'Homme et les Sciences du vivant peut être envisagée en quatre grandes phases: le passage d'une biologie "descriptive" (classement des espèces), à une biologie "explicative" par suite de l'essor de la biologie moléculaire, puis "transformatrice" par le génie génétique et les biotechnologies, et désormais "impliquante" en raison des progrès de la génomique conduisant l'homme à devenir sujet et objet de ses propres expériences [66].

1.4.1 pourquoi étudier les systèmes biologiques ?

Pour répondre à cette question, on peut citer quelques raisons qui nous conduisent à étudier les systèmes biologiques :

- Face à des problèmes technologiques qu'on peut qualifier de « hard: difficiles », notre tendance s'oriente plutôt vers les systèmes biologiques où nous pouvons trouver matière d'inspiration.

- L'ambition de développer des machines Intelligentes.
- Des méthodes précédentes ont été partiellement réussies (Systèmes Experts, Agents Intelligents….).
- La révolution d'ordinateur a changé les sociétés humaines : communication, transport, production industrielle, administration et comptabilité, avances technologiques…etc. Cependant, certains problèmes ne peuvent pas être traités avec un matériel et un logiciel traditionnel !

Donc on a besoin de *nouvelle théories*, concentrant sur : la théorie de contrôle robuste, les systèmes dynamiques, l'analyse numérique, la théorie d'opérateur, la théorie de complexité informatique et logiciel.

Les systèmes de calcul inspirés par les systèmes biologiques (Biocomputing) sont une des alternatives possibles. Il y a deux définition pour la Biocomputing [8], soit utiliser le traitement biologique comme : métaphore, inspiration, ou permettant de développer une nouvelle technologie de calcul et un nouveau secteur d'informatique. Ou au contraire : l'utilisation de concepts et outils de science de l'information pour explorer la biologie et ces différentes perspectives théoriques.

1.4.2 Les types principaux des systèmes bio-inspirés

L'homme a exploré un monde naturel superbe plein de secrets et de miracles, il a réussi de comprendre et d'inspirer des idées à partir de quelques systèmes biologiques, soit pour la résolution de problèmes ou pour l'invention. Dans cette section, quelques systèmes biologiques seront mentionnés ainsi que leurs inspiration dans le domaine de la technologie et de l'informatique.

1.4.2.1 L'intelligence d'essaim 'Swarm Intelligence'

On peut définir l'intelligence des essaims " Swarm intelligence" par ce que Eric Bonabeau a dit : "The emergent collective intelligence of groups of simple agents." (Bonabeau et al, 1999)[9].

Cette définition de l'intelligence d'essaim se résume en deux principes :

1. l'auto-organisation chez les insectes qui est basée sur quatre mécanismes principaux [10] :
    1. L'existence d'interactions multiples.
    2. L'amplification par la rétroaction positive.
    3. La rétroaction négative.
    4. L'amplification des fluctuations.
2. la stigmergie 'stigmergy' qui vient du mot latin 'stigmaergon' et qui veut dire stigma (piqûre 'sting') + ergon (travail 'work') = stimulation par travail (stimulation by work). Elle est basée sur :

    - Un travail comme réponse comportementale à l'état environnemental,
    - Un environnement qui sert comme une mémoire d'état de travail,
    - Un travail qui ne dépend pas d'agents spécifiques.

D'une façon générale, la stigmergie permet par une communication indirecte[1] de coordonner les activités des insectes.

[1] **Communication directe** : contact par les antennes, contact visuel. **Indirecte** : comme précédemment mentionné, les insectes peuvent inter-agir quand l'un des individus modifie son environnement et que les autres en trouvent leur comportement modifié.

Si on veut répondre à la question pour quoi on est intéressé à l'intelligence d'essaim, nous pouvons, tous simplement, dire qu'il y a une analogie entre les principes de la technologie et ceux d'insectes qui se présentent par exemple dans : le système distribué d'une interaction entre des agents autonomes, la performance et la robustesse, l'auto-organisation de contrôle et de coopération (décentralisée), des interactions indirectes…etc. Et pour faire une conception d'un système de l'intelligence d'essaim on peut suivre 3 étapes : identification de l'analogie entre la biologie d'essaim et les système (IT), la compréhension par la modélisation d'ordinateur de la biologie d'essaim réelle, et enfin l'inginerie, dont la quelle on va simplifier le modèle et le régler pour des applications IT.

Comme exemple de l'intelligence d'essaim on site : le groupe de fourrage d'insectes sociaux, transport coopératif, division du travail, la construction de nid d'insectes sociaux…etc.

Une colonie d'insecte sociale est :

1- **Flexible :** la colonie peut répondre aux perturbations internes et des défis externes.
2- **Robuste** : les tâches sont achevées même si quelques individus échouent.
3- **Décentralisé** : il n'y a aucun contrôle ou contrôleur central dans la colonie.
4- **Auto-organisé** : les chemins aux solutions sont émergents plutôt que prédéterminé.

Les colonies d'insectes les plus utiliser dans le domaines informatique sont : les fourmis, les abeilles, les termites et même les araignées. Il existe une quinzaine d'espèces d'araignées sociales dans le monde. Les araignées de l'espèce *anelosimus eximius*, que l'on trouve en Guyane Française, partagent la même toile (qu'elles construisent collectivement), et coopèrent dans des activités telles que l'élevage des petits, la capture et le transport des proies [11]. Le processus de construction collective de toile a inspiré des travaux sur la détection de région dans des images [12].

1.4.2.1.1 L'optimisation par Colonies de fourmis

Quand on cherche à savoir la notion de 'Swarm Intelligence', on rencontre toujours les colonies de fourmis dans toutes les références. La question qui se pose est : pourquoi on est intéressé à ce genre d'insectes ? tous simplement, parce que les fourmis résolvent des tâches complexes par des moyens simples locaux, la productivité de fourmi est meilleure que la somme de leurs activités simples, enfin les fourmis sont 'des grands maîtres' dans la recherche et dans l'exploitation. Les mécanismes importants de fourmis se représentent dans : la coopération et division du travail, allocation de tâche adaptative, stimulation de travail par culture et enfin les phéromones.

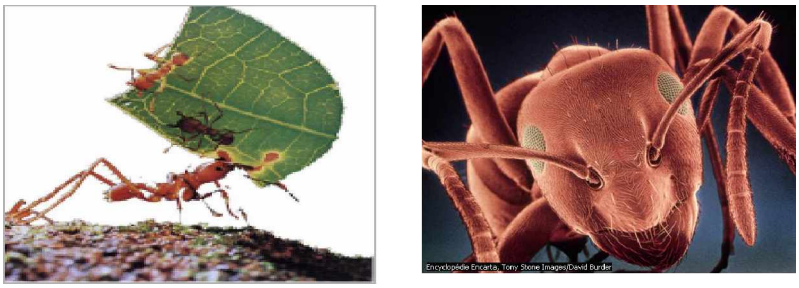

Figure 1.1 : Les fourmis [4].

1.4.2.1.1.1 Contexte et applications

L'optimisation par colonies de fourmis est une technique d'optimisation biomimétique inspiré par un travail d'un biologiste [13] repris par des informaticiens [14] et largement exploité et développé par Marco Dorigo dans les années 90 [15][16]. L'idée consiste à imiter le comportement

des fourmis réelles qui collaborent, par exemple pour la recherche de sources de nourriture en mélangeant comportement d'exploration aléatoire et suivi des traces chimiques laissées par leur consoeurs. Ces traces chimiques, les « phéromones », sont utilisées par les fourmis pour communiquer entre elles de manière indirecte, par le biais de l'environnement, une technique générale connue par les entomologistes sous le nom de stigmergie (précédemment mentionnée) [17]. C'est cette forme de communication ainsi que l'idée de faire coopérer une foule d'agents simples et localisés qui forme la base de l'heuristique développée par Dorigo [17].

1.4.2.1.1.2 Le fourrage des fourmis

Il existe deux chemins possibles pour atteindre la source, un chemin long et l'autre court. Les fourmis explorent aléatoirement les deux chemins. Lorsqu'elles trouvent la source, elles se chargent de nourriture, et retournent au nid d'où elles sont venues en libérant des phéromones tout au long de leur chemin [17]. Le chemin le plus court étant aussi le plus rapide, sa concentration en phéromones augmentera plus vite et les fourmis, qui suivent les traces chimiques, seront très vite, par un phénomène de renforcement, encouragées à suivre le chemin le plus court [17]. Le chemin du nid à la source a été optimisé [17].

Pour mieux et bien comprendre la coopération entre les fourmis, l'exemple suivant illustré par des images va nous expliquer les mécanismes de recherche utilisés, selon les étapes :

1. Pour cette première étape, nous avons un point de départ qui se représente dans la fourmilière (le nid de fourmis) et un point d'arrivée qui est la nourriture, et nous avons aussi deux fourmis qui vont prendre deux chemins différents (voire figure 2)

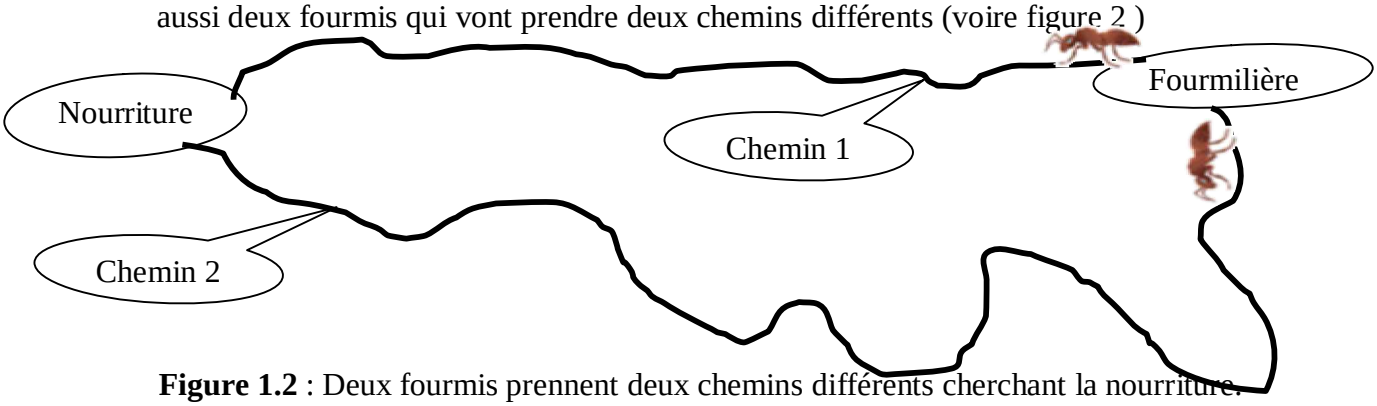

Figure 1.2 : Deux fourmis prennent deux chemins différents cherchant la nourriture.

2. Pour cette deuxième étape, la figure nous explique que la fourmi du '**chemin 1**' est en retour vers le nid en posant une quantité de phéromone. Par contre pour le deuxième chemin '**chemin 2**' la fourmi n'a pas encore arrivé à la nourriture. (voire figure 3)

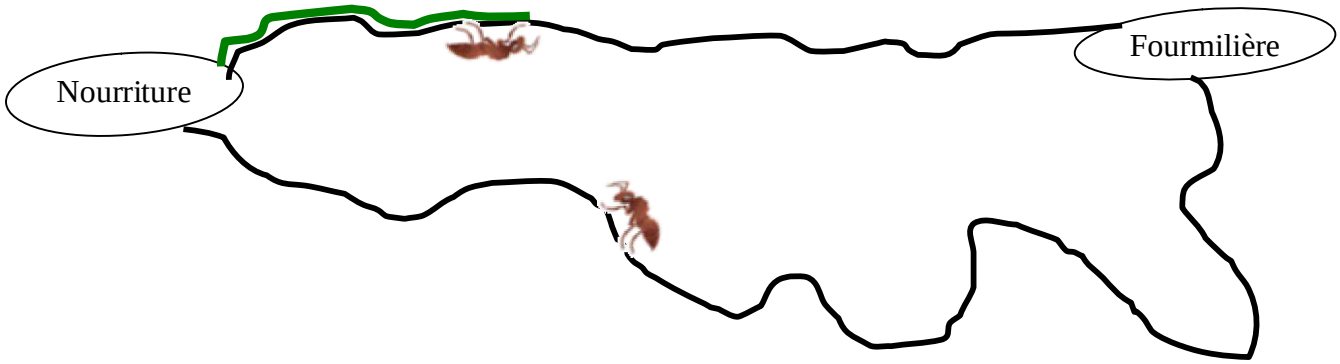

Figure 1.3 : La fourmi de 'chemin 1' retourne au nid en mettant une quantité de phéromone. La fourmi du 'chemin 2' n'a pas encore arrivé à la nourriture.

3. Après le retour de la fourmi du ‘chemin 1’, au nid, d’autres fourmis vont prendre ce premier chemin en suivant la phéromone déposée par cette fourmi qui a réussi de trouver la nourriture. Par contre la deuxième fourmi n’a pas encore arrivé au nid.

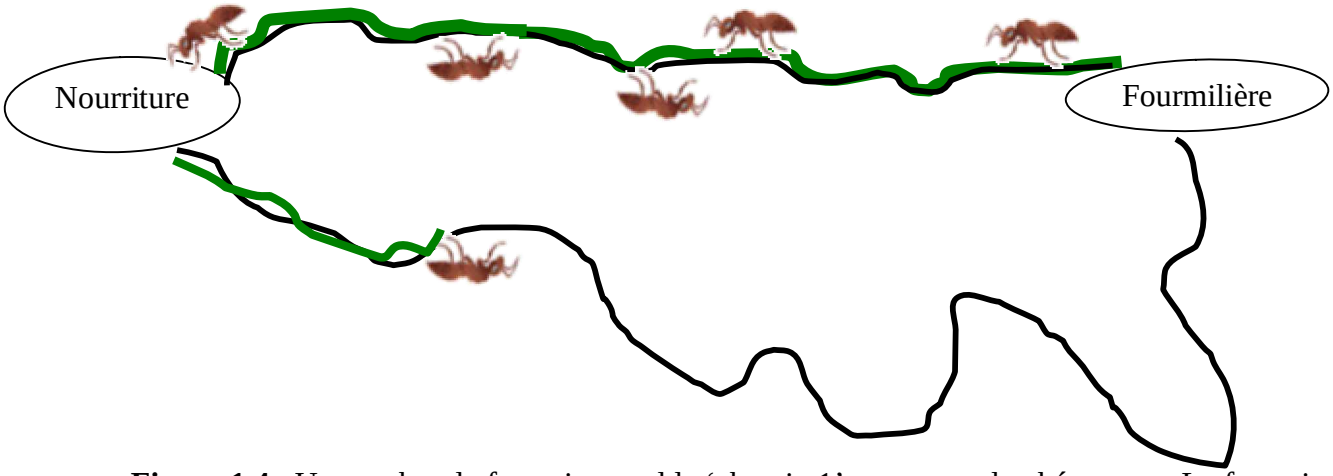

Figure 1.4 : Un nombre de fourmis prend le ‘chemin 1’ en posant de phéromone. La fourmi de chemin 2 retourne en mettant une quantité de phéromone.

4. Après un certain temps, il n’y a aucune fourmi qui suit le deuxième chemin et cela signifie que le premier chemin est le chemin le plus cours comme le montre la figure suivante :

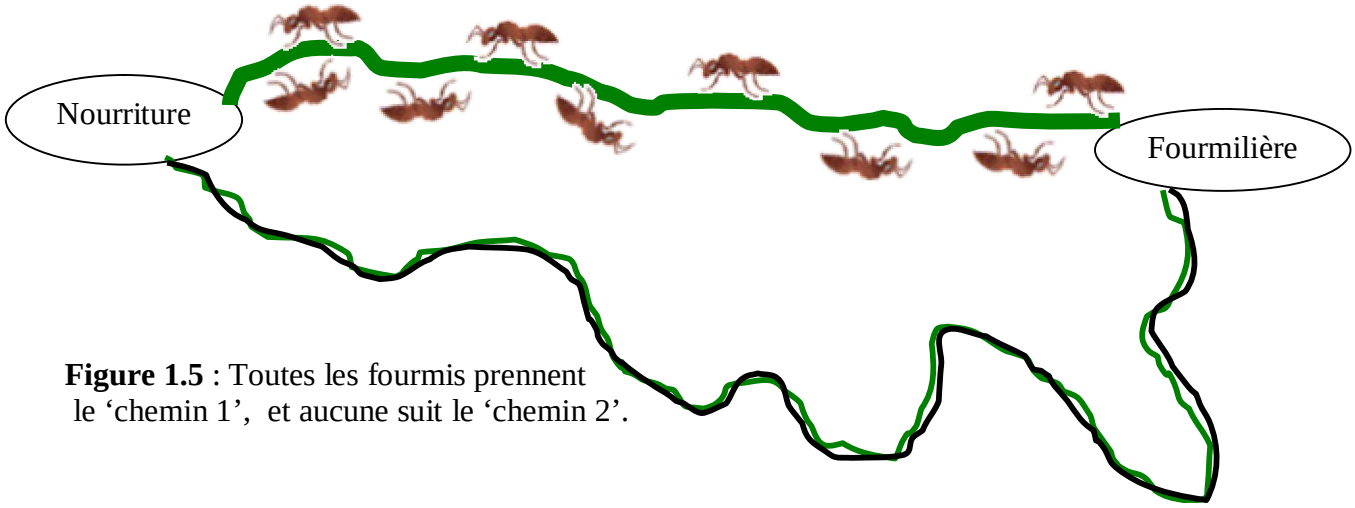

Figure 1.5 : Toutes les fourmis prennent le ‘chemin 1’, et aucune suit le ‘chemin 2’.

Comme résultat nous pouvons dire que si on a plusieurs chemins pour atteindre la nourriture (la ressource), l’accumulation de phéromones sera nécessairement plus rapide sur le chemin le plus court. C’est ainsi que certaines espèces de fourmis tendent à optimiser le recueil de nourriture [18]. Une piste présentant une plus grande concentration en phéromone est plus attirante pour les fourmis, elle a une probabilité plus grande d'être empruntée. La piste courte va alors être davantage renforcée que la longue, et, à terme, sera choisie par la grande majorité des fourmis [19].

1.4.2.1.1.3 Exemple d’application : problème de voyageur de commerce

Pour illustrer la manière dont on peut transformer l'observation de colonies de fourmis réelles en algorithme d'optimisation, on reprend ici l'exemple traditionnel de la résolution du

problème du voyageur de commerce avec un algorithme à base de fourmis artificielles [20] [21] (voir figure 7, 8).

Les fourmis vont travailler en parallèle à essayer diverses solutions à ce problème jusqu'à en trouver une acceptable. On procède de la manière suivante [17] :

Chacune des fourmis de la colonie est placée au hasard sur un des noeuds du graphe. Chacune commence alors sa « tournée » : elle se rend de noeud en noeud sans jamais en visiter un qu'elle a déjà vu et jusqu'à ce qu'elle les ait tous visités. Le choix du passage d'un noeud **i** à un noeud **j** (voir figure 1.6) se fait aléatoirement en fonction d'une probabilité donnée par l'équation 1. d y décrit la distance entre les noeuds **i** et **j** (information heuristique locale, la fourmi a tendance à aller au plus près) et **t** la quantité de phéromones déposées sur l'arc correspondant. a et b servent à régler l'importance relative que l'on donne à ces deux facteurs.

$$p_{ij}^{k} = \frac{\left([\tau_{ij}]^{\alpha}[d_{ij}]^{-\beta}\right)}{\sum_{l \in J_k}[\tau_{il}]^{\alpha}[d_{il}]^{-\beta}}$$

Équation 1 : probabilité de passer de la ville i à la ville j en fonction de la distance d et de la concentration en phéromones τ.

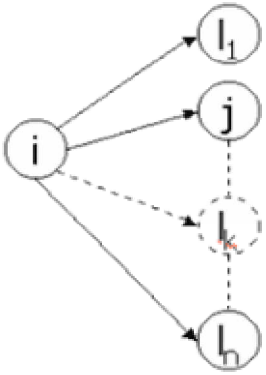

Figure 1.6 : Arrivant à la ville i, notre voyageur de commerce (ou notre fourmi) doit choisir la ville suivante [17].

Après *n* itérations, lorsque toutes les fourmis ont construit un chemin complet, chacune d'elles calcule la longueur totale parcourue (somme des distances d'une ville à l'autre) et libère des phéromones tout au long du chemin en quantité inversement proportionnelle à la longueur : plus le chemin est court, plus il sera chargé en phéromones. Le procédé est alors recommencé jusqu'à ce l'on obtienne une solution optimale ou jugée acceptable. Régulièrement par ailleurs, les quantités de phéromones portées par les arcs sont multipliées par un facteur d'évaporation typiquement proche de 0.99 [5].

On peut résumer l'algorithme de fourmis comme suit :

- dij= Distance entre la ville i et la ville j.
- tij= Phéromone virtuelle sur la liaison (i, j).
- m agents, chacun construit un tour.
- A chaque pas d'un tour, la probabilité pour aller de la ville i à la ville j est proportionnel à $(tij)^{a}(dij)^{-b}$
- Après la construction d'un tour de longueur L, chaque agent renforce les bords par une quantité de phéromone proportionnelle à 1/L.
- La phéromone virtuelle s'évapore : t ⟶ (1-p) t

Cette façon d'explorer l'espace de recherche a le mérite d'être efficace et d'offrir un compromis entre l'exploitation de l'information apportée par la collectivité (les phéromones préalablement déposées là où les fourmis ont intérêt à passer), l'utilisation des heuristiques locales (allons au plus près !) et l'exploration aléatoire [17]. Les méthodes ACO (Ant Colony Optimisation) hybridées avec des heuristiques de recherche locale bien choisies (type Lin-Kerningan ou 3-opt) sont parmi les meilleures pour les problèmes TSP (Travelling Salesman Problem) de grandes tailles [16][17].

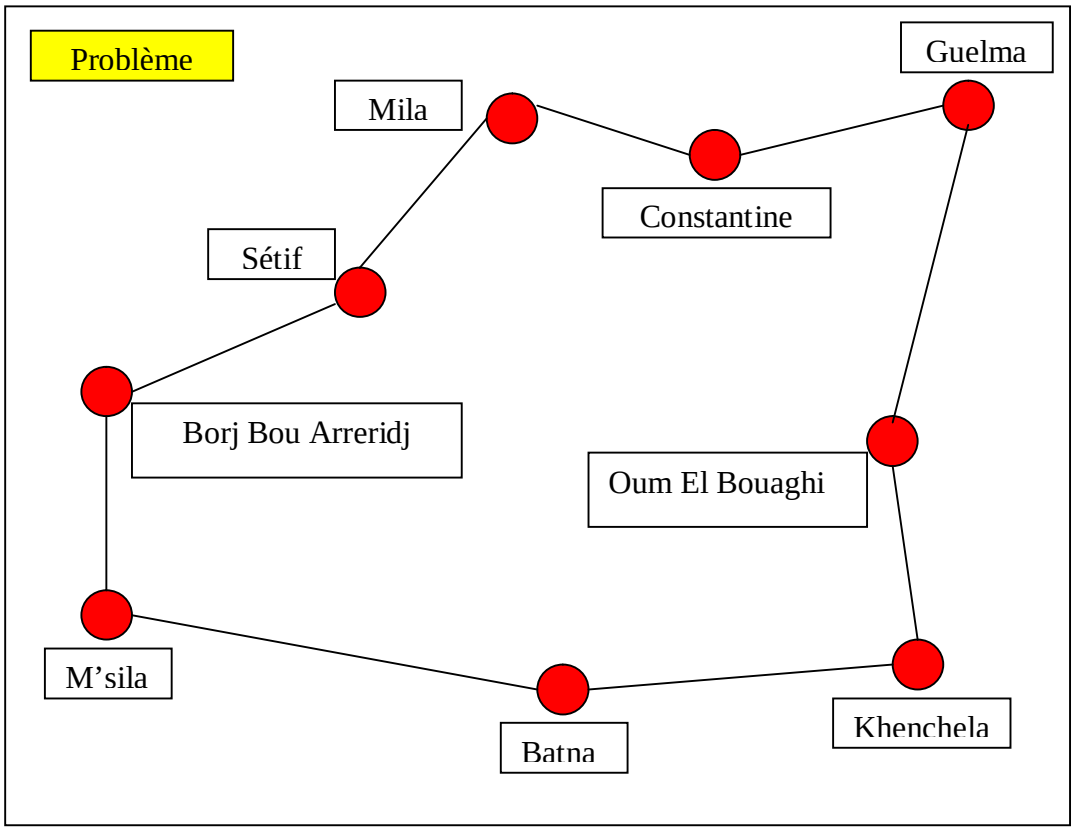

Figure 1.7: Un graphe de 9 villes représentant le problème de voyageur de commerce. Le nœud de départ se présente dans la ville de Khenchela qui est elle-même la ville d'arrivée, passant par chaque ville une seule fois.

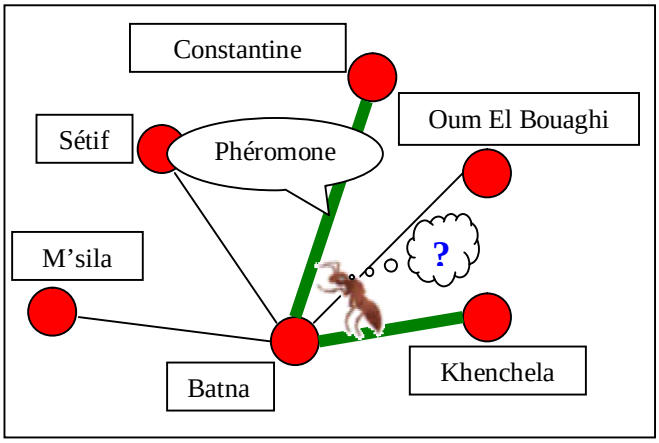

Figure 1.8 : Utilisation de l'algorithme de fourmis (ant algorithm) pour la résolution du problème de voyageur de commerce décrit dans la figure 1.7

D'après Eric Bonabeau, on peut résoudre, en utilisant cette méthode ou cet algorithme, d'autre types de problèmes tel que le problème de nomination quadratique (Quadratic assignment problem), planification de magasin de travail (Job-shop scheduling), coloration de graphe (Graph coloring), planification de véhicule (Vehicle scheduling).

1.4.2.1.2 La communication et l'auto-organisation chez les colonies d'abeilles

Un essaim d'abeilles est un ensemble d'abeilles sociales vivent en colonies hiérarchisées dans des nids appelés ruches. Dans la ruche d'une région tempérée, le nombre d'ouvrières est compris entre **8 000** et **15 000** individus au printemps, mais peut dépasser **80 000** au début de l'été[4]. Les abeilles comptent parmi les rares espèces d'insectes à être domestiquées par l'homme. Leurs colonies sont très structurées ; les différentes castes qui la composent présentent des adaptations comportementales et morphologiques à la tâche qui leur est assignée dans la société (voir figure 1.9) [4].

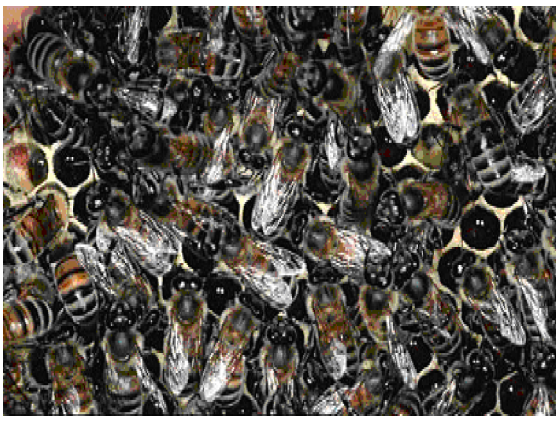

Figure 1.9 : L'auto-organisation d'un essaim d'abeilles dans la ruche [4].

La communication entre les abeilles (les ouvrières) se réalise à travers un type spécifié de la danse ! Qui est expliquée comme suit :

a. Danse circulaire :

Si les fleurs sont à moins de **90m** de la ruche, la découvreuse effectue des cercles alternativement dans un sens et dans l'autre comme le montre cette figure.

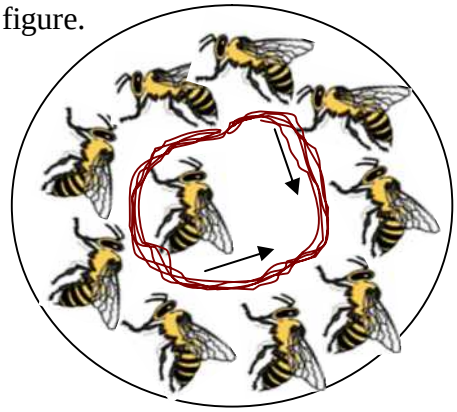

Figure 1.10 : Danse circulaire de l'abeille indiquant que la distance entre la ressource et la ruche est moins de **90m**

b. Danse frétillante :

Si les fleurs sont situées à plus de **90m**, l'abeille exécute un huit et elle frétille pendant la danse.

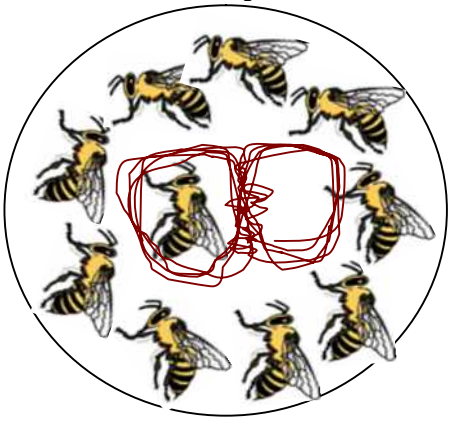

Figure 1.11 : Danse frétillante indiquant que la distance entre la ressource et la ruche est plus de **90m**

Si les fleurs sont très éloignées, le frétillement est accentué et la danse est très lente. Le point de changement de direction (pour les sources proches) ou le parcours en ligne droite du huit (pour les sources éloignées) est dirigé, par rapport à la verticale, de la même façon que la source de nourriture l'est par rapport au Soleil (avec le même angle).

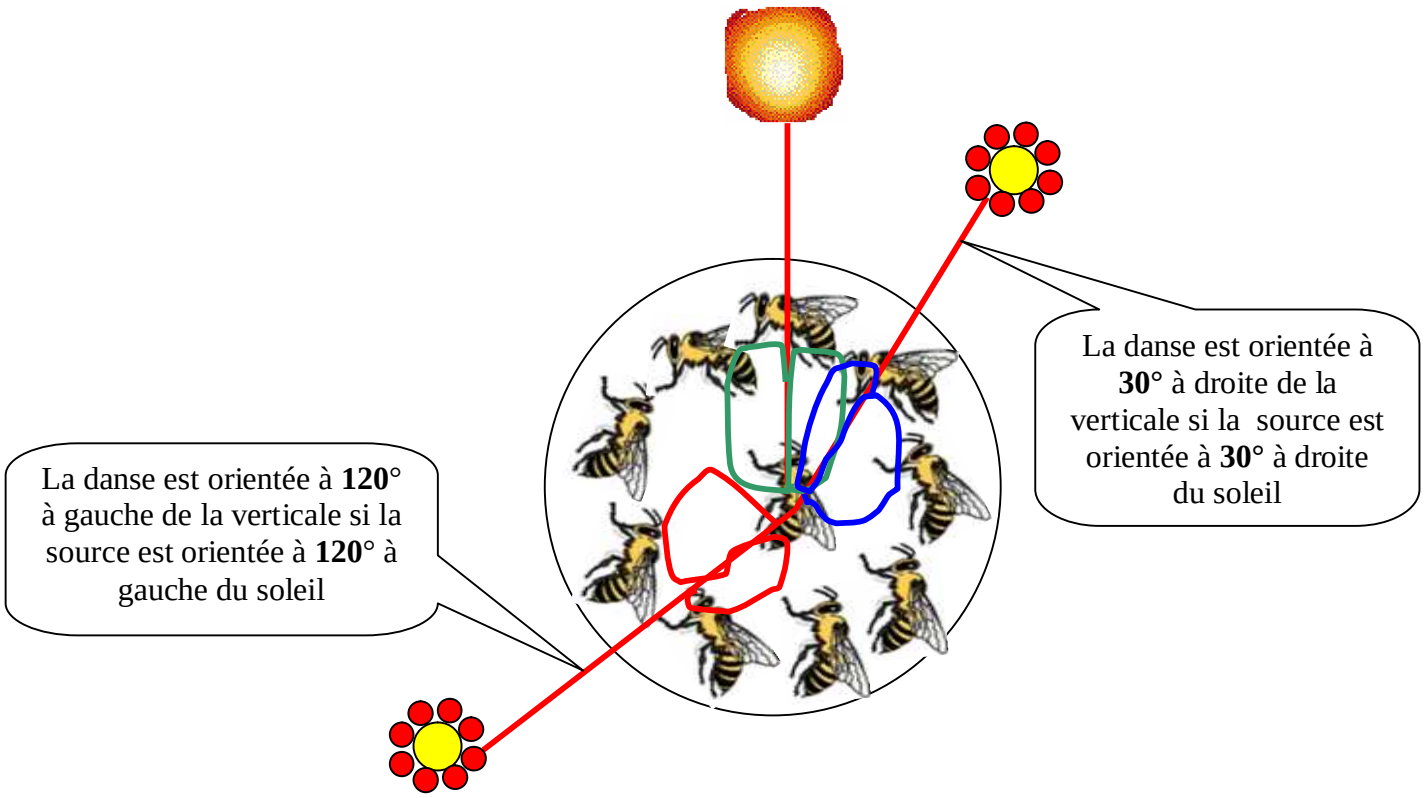

Figure 1.12: Danse frétillante orientée selon un angle spécifiée indiquant l'emplacement de la ressource

La figure 1.13, nous explique aussi clairement ce type de communication (la danse d'abeille) o ù il y a deux abeilles, chacune d'elle danse vers une source bien spécifiée. Les abeilles de la ruche, à travers une danse donnée, peuvent trouver la source A, en suivant la danse de l'abeille vers A, ou la source B, en suivant celle qui danse vers la source B.

L'intelligence des abeilles nous donne des leçons dans : les techniques de recherche de la source, les techniques de communications et aussi dans l'auto-organisation. Les abeilles exécutent des tâches basées sur les besoins de la ruche : en étudiant la voie dans laquelle ces emplois sont assignés, les scientifiques espèrent développer de meilleures voies pour programmer un équipement à une usine automatisée.

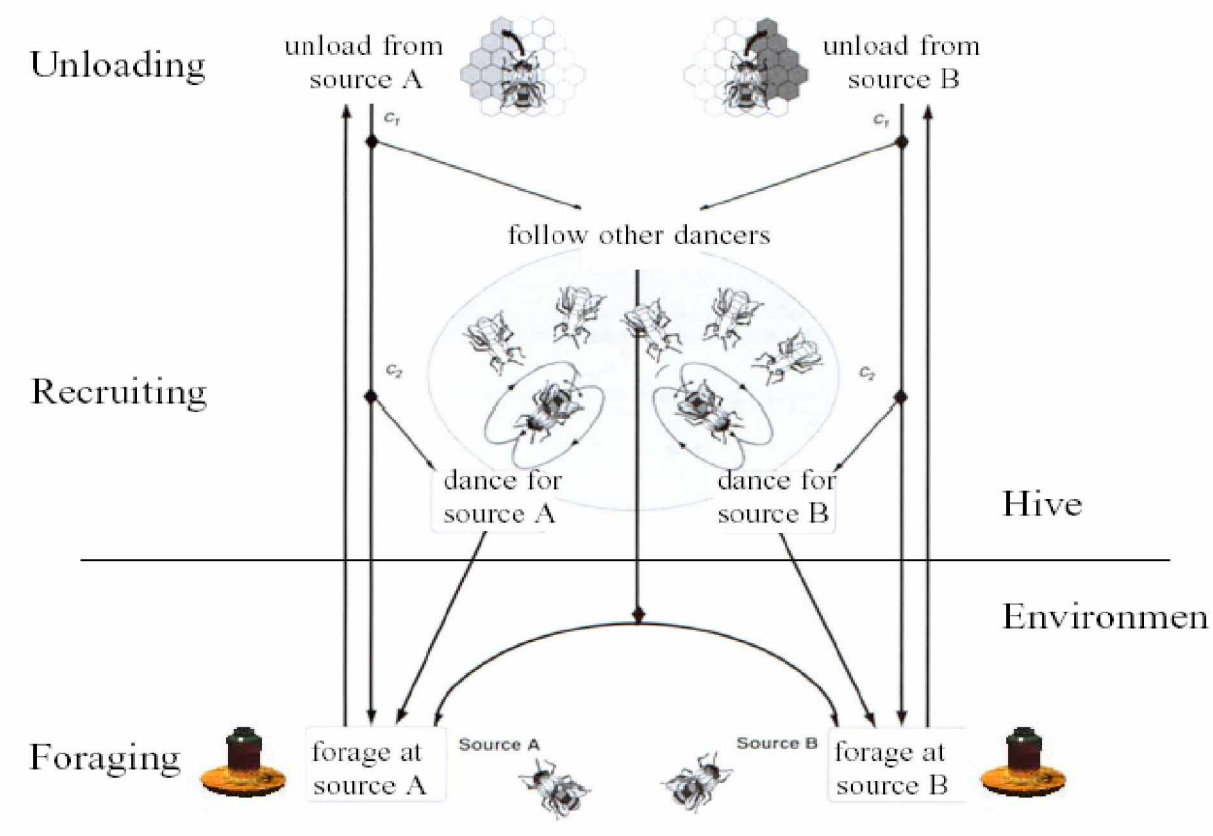

Figure1.13 : Communication à travers la danse d'abeille.

1.4.2.2 Les boids de Reynolds

Les boids sont des agents autonomes qui cohabitent dans un monde à 2 ou 3 dimensions. Leur vie est réglée par un certain nombre de comportements minimaux. En 1987 Reynolds se demandait comment ils pouvaient se déplacer efficacement dans le ciel sans guide ou direction centralisée. Pour cela il a réussi de concevoir un oiseau virtuel capable de voler appelé Boid. Et cela, lui permet de créer un ensemble de Boids qui peuvent vivre dans un monde totalement artificiel et qui peuvent être des moineaux, des chauves-souris ou des bancs de poisson. Ce genre d'agents 'Boids' est apparu dans des filmes spécialisés et des animations d'ordinateur (par exemple le Retour de Batman et le Lion Roi). Les idées de ce genre de travail ont été appliquées dans des domaines divers tel que l'analyse géographique [22] et l'exploration spatiale.

Un modèle comportemental complexe a été employé dans les premières expériences. Il a inclus l'action d'éviter des obstacles qui a permis au boids de voler dans des environnements simulés en s'esquivant des objets statiques (voir Figure 1.14).

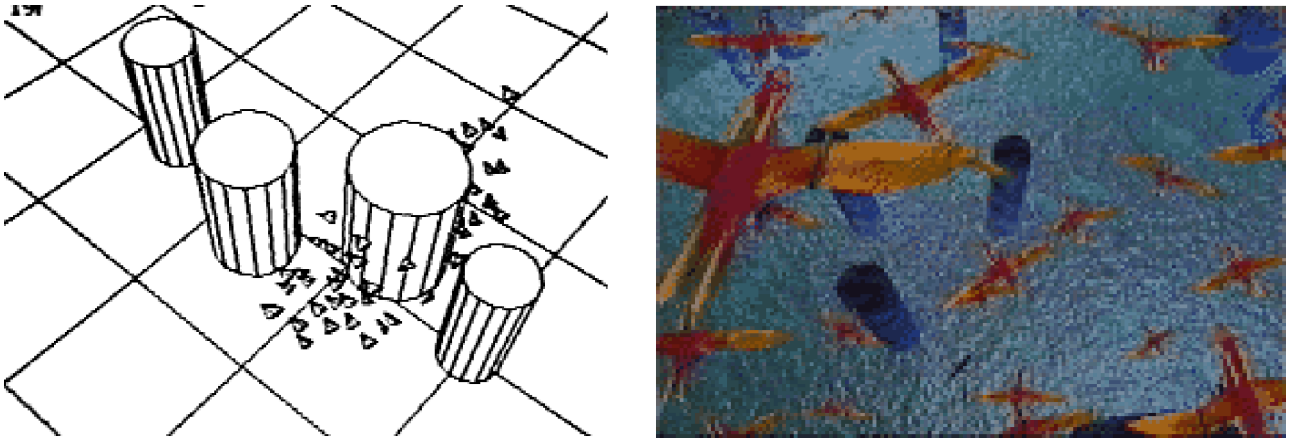

Figure 1.14: Un ensemble de boid simulé évitant des obstacles cylindriques (1986) [23].

Dans son expérience, Reynolds a utilisé 3 règles de conduites (steering behavior) et qui sont :

1. **La séparation** : 2 Boids ne peuvent pas se trouver au même endroit au même moment.
2. **La cohésion**: Pour former un groupe, les Boids se rapprochent les uns des autres.
3. **L'alignement**: Pour rester grouper, les Boids essaient de suivrent un même chemin.

A ces 3 règles minimales, une règle de **répulsion** est ajoutée. Par exemple :

- De manière standard, un Boid d'une couleur rouge, par exemple, n'interagit pas avec des Boids d'une couleur différente sauf pour s'en éloigner.
- Mais si dans un voisinage, il y a au moins 2 fois plus de boids de couleur vert, par exemple, que de couleur rouges, les verts attaquent les rouges.

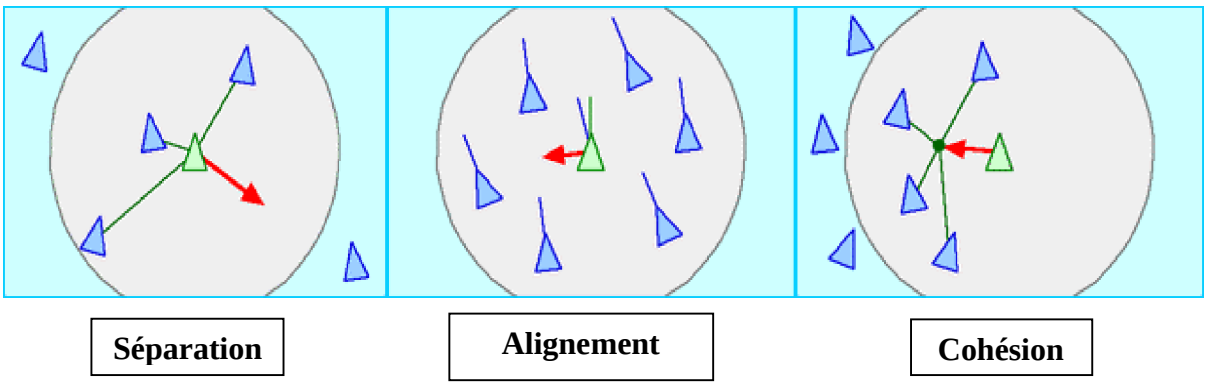

Figure 1.15 : Les trois comportements de direction trouvés par Reynolds pour produire comportement de masse vivant. Droit à gauche : séparation, alignement et cohésion et [23].

Chacune des 4 lois permet de générer une force qui va s'appliquer sur le Boid pour le faire évoluer dans le plan. Le Boid possède un angle de vision, mais, pour chaque force, une distance d'application est définie. Grâce au couple *(angle, distance)* on trouve un certain nombre de Boids qui vont influencer le Boid en cours de traitement.

Une fois les Boids qui vont interagir trouvés, une politique différente à chaque fois est appliquée pour permettre de donner un caractère à chaque force [24] [25] [26].

- Pour permettre la **Séparation**, une force est calculée de manière à repousser le Boid de l'ensemble des proches.
- La **Cohésion** est assurée par une force inverse à la précédente, qui a pour but de rapprocher le Boid du centre de gravité du groupe des proches.
- La force d'**alignement** est calculée comme étant une moyenne entre les vitesses des voisins.
- La force de **répulsion** est en fait une seconde force de séparation par rapport à certains Boids (les autres).

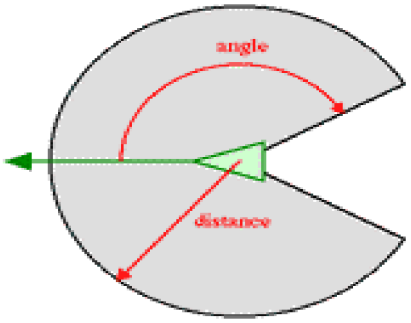

Figure 1.16 : Le voisinage local d'un boid. [23]

L'application de ces règles simples a permis de construire des simulations graphiques d'un réalisme étonnant. Des algorithmes de ce type ont été utilisés par l'industrie cinématographique dans des films tel que *Le retour de Batman* ou Cliffhanger.

1.4.2.3 Les essaims de particule (Particle Swarm)

Les algorithmes « d'*optimisation par essaim de particules* » (*Particle Swarm Optimization* - PSO) trouvent leur origine dans les *boids* de l'infographiste Craig REYNOLDS [25].
Les rassemblements animaux présentent divers avantages adaptatifs relevant au moins pour partie de l'échange d'informations [26]. De la même manière que les algorithmes évolutionnaires s'inspirent des mécanismes évolutifs pour mettre en œuvre des procédures d'optimisation, PSO s'inspire des phénomènes de rassemblement et de nuée, considérant qu'en tant que processus adaptatifs, ils sont potentiellement porteurs de capacités d'optimisation. PSO repose sur deux règles simples [26] :

1. Chaque individu se souvient du meilleur point (le plus proche de l'objectif) par lequel il est passé au cours de ses évolutions et tend à y retourner.
2. Chaque individu est informé du meilleur point connu au sein de la population prise dans son ensemble et tend à s'y rendre.

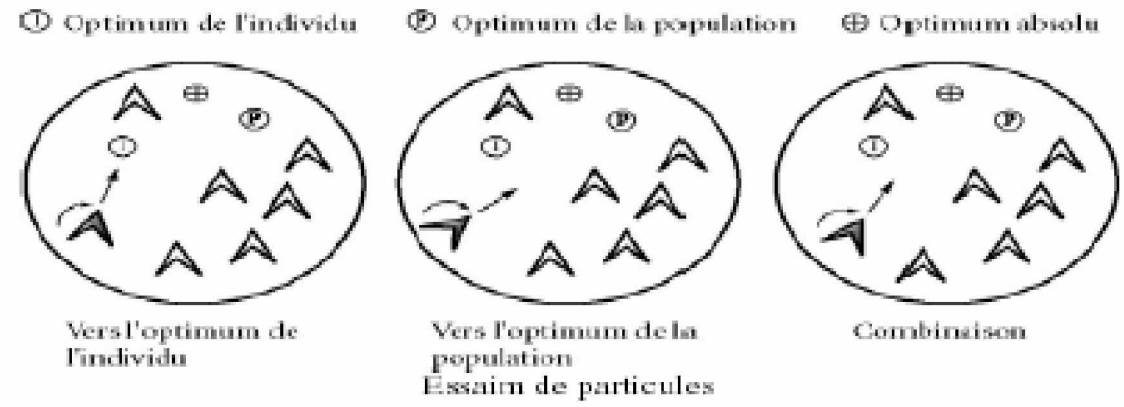

Figure 1.17 : Principe de fonctionnement des essaims de particules [25]

L'une des premières applications a été l'entraînement d'un réseau de neurones devant réaliser la fonction booléenne Xor (ou exclusif) et utilisant treize paramètres. L'essaim devait ainsi se déplacer au sein d'un hyperespace à treize dimensions. Chacune de ces dimensions représente alors l'espace des poids possibles pour une connexion neurale donnée [26].

Cette méthode a maintenant de nombreuses utilisations. On la retrouve comme alternative aux méthodes classiques d'entraînement des réseaux de neurones, mais aussi par exemple dans des procédures d'optimisation de l'alimentation de cultures bactériennes [26].

1.4.2.4 Le calcul à l'ADN (DNA computing)

En Novembre **1994**, Leonard Adleman fit une avancée spectaculaire dans le domaine des ordinateurs chimiques ou biochimiques. Il utilisa des fragments d'ADN pour résoudre informatiquement un problème complexe de la théorie de graphe. Tel un ordinateur comportant plusieurs processeurs, ce type d'ordinateur à ADN est capable de considérer simultanément plusieurs solutions à un problème. De plus, les brins d'ADN employés pour un tel calcul sont beaucoup plus nombreux et compacts que les processeurs électroniques des superordinateurs actuels. C'est le premier exemple de calcul effectué grâce à un ordinateur non-électronique.

Adleman a réussi de résoudre le problème de chemin hamiltonien en utilisant le calcul moléculaire (calcul d'ADN) **[1]**. Ce problème nommé aussi le problème de voyageur de commerce. Le problème de chemin hamiltonien est de déterminer s'il y a un chemin hamiltonien pour un graphe dirigé spécifiant le nœud de départ et celui d'arrivée en passant par chaque nœud seulement une et une seule fois. Dans son expérience Adleman a utilisé un graphe de 7 nœuds (voir figure 1.18). Si le nombre de noeuds augmente, même les super-ordinateurs ont la difficulté de découvrir le chemin hamiltonien.

Pour résoudre ce type de problème NP-complet, Adleman a proposé un algorithme dont ses étapes sont :

Etape1: A partir du graphe, générer tous les chemins aléatoires.

Etape2 : Garder seulement les chemins qui commencent par le nœud d'entrée spécifié et celui de sortie.

Etape3 : Si le graphe a n nœuds, alors garder seulement les chemins qui ont exactement n nœuds.

Etape4 : Garder seulement les chemins qui ont l'apparition de chaque nœud exactement une seule fois.

Etape5 : Après les étapes précédentes, s'il reste un chemin alors ce chemin représente le chemin hamiltonien, sinon aucun chemin hamiltonien existe.

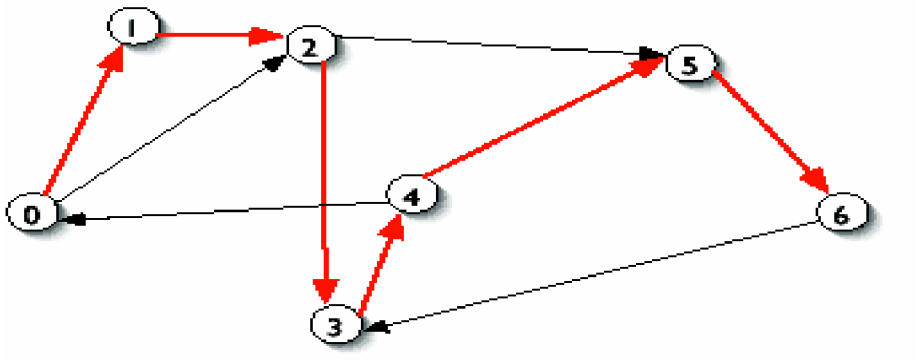

Figure 1.18 : Graphe de 7 nœuds utilisé dans l'expérience d'Adleman.

Le chapitre 3 va nous expliqué d'une façon simple, claire et en détaille la stratégie utilisée par Adleman pour résoudre ce genre de problème et même pour résoudre d'autres types de problèmes complexes (NP-complet).

1.4.2.5 Les réseaux de neurones artificiels (Artificial Neural Network)

La recherche de réseau neuronal a été inspirée par « neuroscience » [27]. Elle a été lancée comme "une contre-culture" à la recherche de l'intelligence artificielle qui a provoqué des systèmes experts. Selon Rosen [28], l'idée originale de réseau neural a été suggérée par Nicolas Rashevsky dans les années 1930 [29] et au plus tard, elle a était refondue en langage d'algèbre booléenne par McCulloch et Pitts [30]. Une description succincte de réseaux artificiels neuraux et une explication de leurs capacités cognitives peuvent être trouvées dans un article de Churchland [31]. Une liste courte de monographies est suggérée pour une lecture de plus dans [32], [33], [34], [35], [36].

1.4.2.5.1 Réseau de neurones naturel

On pense que le système nerveux compte plus de **1000** milliards de neurones interconnectés. Bien que les neurones ne soient pas tous identiques, leur forme et certaines caractéristiques permettent de les répartir en quelques grandes classes [37].

Un réseau neuronal humain (voir figure 1.18), composé de cellules nerveuses appelées neurones. Chaque neurone peut émettre des signaux vers ses voisins, par l'intermédiaire de prolongements nommées dendrites ; il peut également en recevoir, véhiculés par des fibres appelées axones [4]. Le corps cellulaire contient le noyau du neurone ainsi que la machine biochimique nécessaire à la synthèse des enzymes [5] (voir figure 1.19).

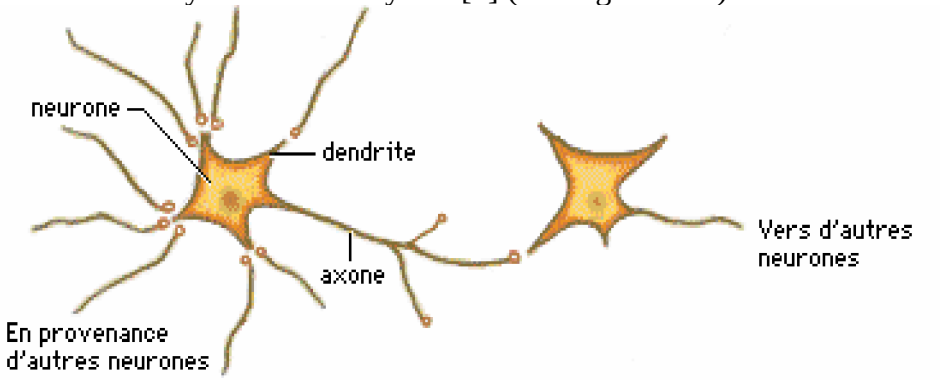

Figure 1.19 : Réseau de neurone naturel : exemple d'un réseau neuronal humain [4]

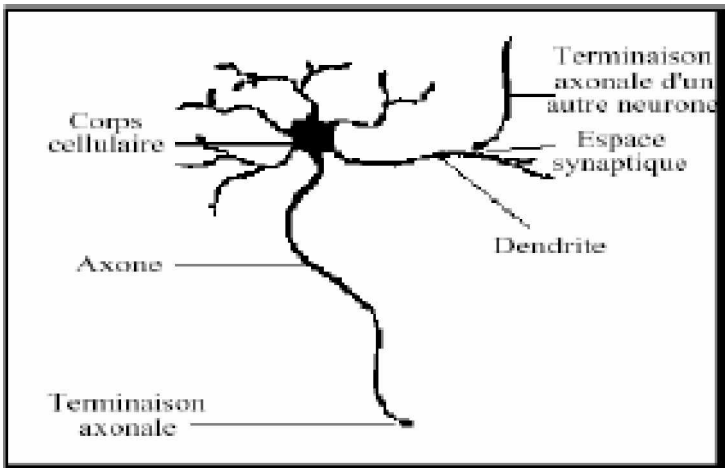

Figure 1.20 : Neurone naturel [5]

1.4.2.5.2 Réseau de neurones artificiel

Un réseau de neurones artificiel est constitué d'un ensemble de nœuds connectés entre eux. Dans un réseau de neurone artificiel il y a un modèle d'entrée qui se compose d'un ensemble de nœuds d'entrée, et un modèle de sortie qui se compose d'un ensemble de nœuds de sortie. Entre les nœuds d'entrée et ceux de sortie figurent de nombreux autres nœuds, appelés les nœuds cachés ou le modèle caché (voir figure 1.20). Un réseau neuronal artificiel peut comprendre plusieurs niveaux de nœuds cachés.

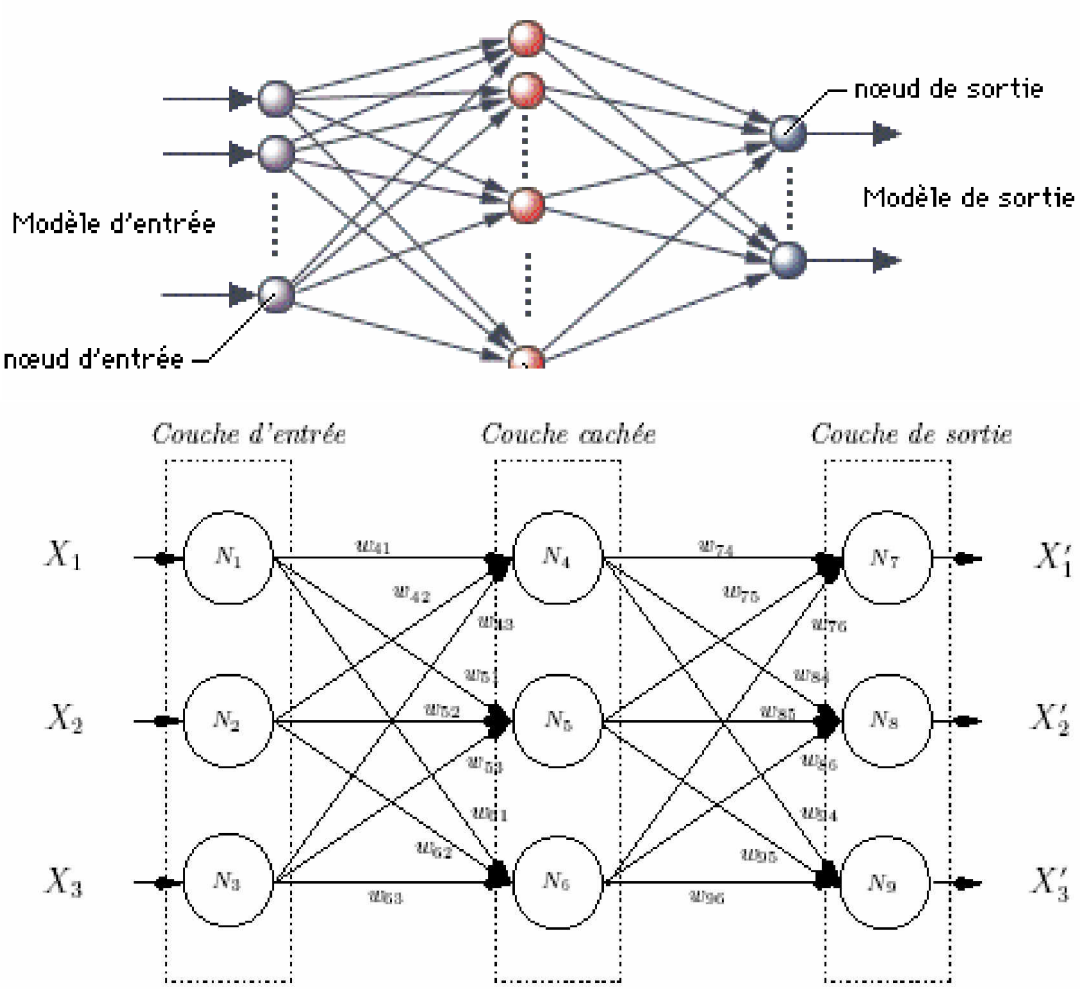

Figure 1.21 : Un réseau de neurones artificiel : haut [4], bas [123].

Les réseaux de neurones artificiels, pour les différencier des réseaux de neurones biologiques, peuvent être aussi définies comme un ensemble de neurones artificiels simples, petites fonctions mathématiques, qui permettent de former des fonctions complexes très utiles [5]. Par analogie aux neurones biologiques, les neurones artificiels ont pour but de reproduire des raisonnements « intelligents » d'une manière artificielle. Ces neurones peuvent adopter de certaines qualités habituellement propres au biologique, c'est-à-dire, la généralisation, l'évolutivité, et une certaine forme de déduction [37].

Un neurone formel peut être décrit comme un petit automate de décision. Il se compose de 2 parties ayant des fonctions distinctes :

- Evaluation de la stimulation reçue
- Evaluation de son état interne

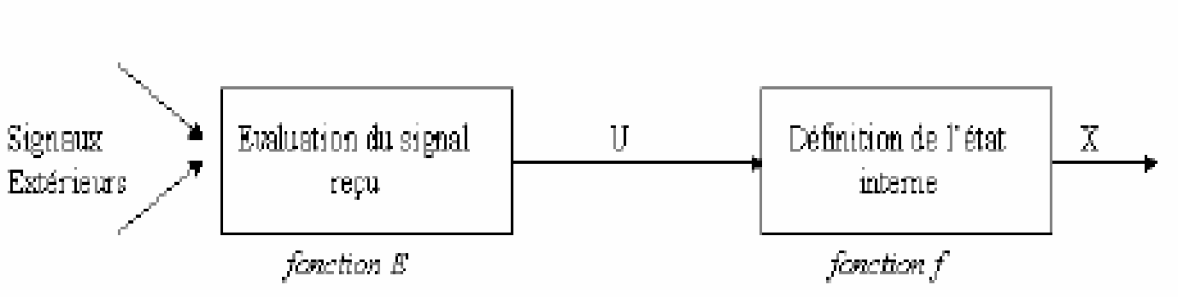

Figure 1.22 : Réseau de neurone formel [5].

Un neurone est donc caractérisé par :

- Son état *(X)*.
- Le niveau d'activation qu'il reçoit en entrée *(U)*.
- Sa fonction de transition *(f)*.
- Sa fonction d'entrée (une somme en général).

L'état que peut prendre un neurone est en général binaire mais peut aussi bien prendre une valeur discrète ou bien continue. Selon le type de cet état, on aura donc différentes formes pour la fonction de transition.

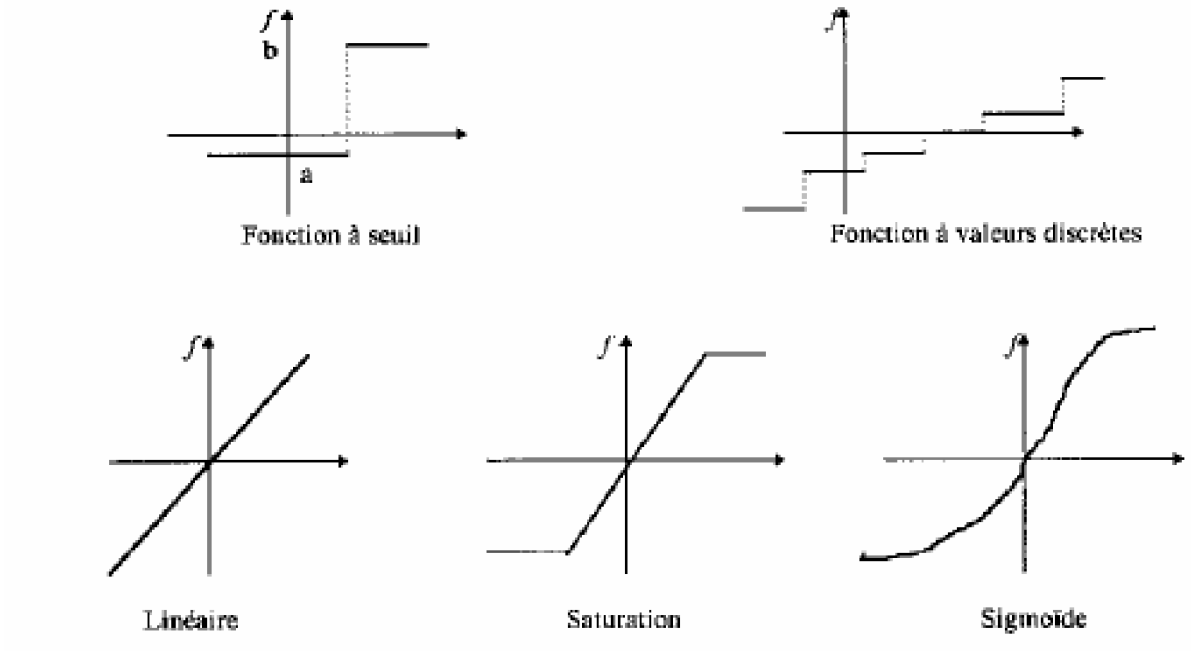

Figure 1.23 : Différentes formes pour la fonction de transition [5].

La topologie des réseaux de neurones peut être très variée (voir figure 1.24). On peut concevoir plusieurs types de réseaux seulement en modifiant les règles de connexion.

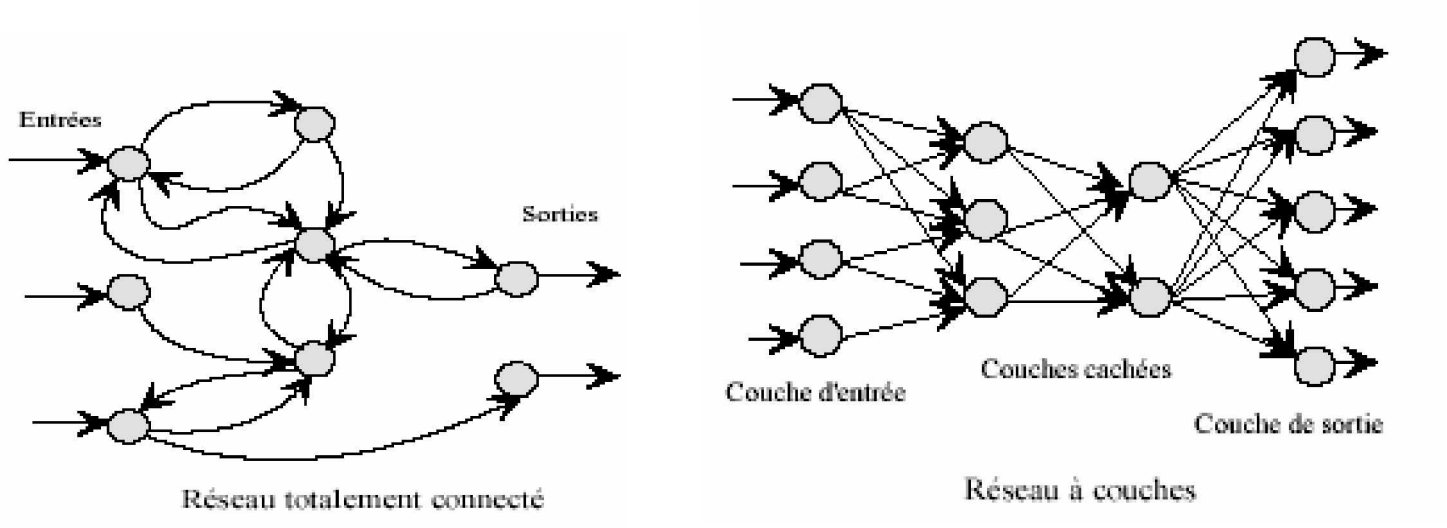

Figure 1.24 : Exemples de topologies des réseaux de neurones [5].

Dans le cas des réseaux de neurones artificiels, on ajoute souvent à la description du modèle l'algorithme d'apprentissage. Dans la majorité des algorithmes actuels, les variables modifiées pendant l'apprentissage sont les poids des connexions. L'apprentissage est la modification des poids du réseau dans l'optique d'accorder la réponse du réseau aux exemples et à l'expérience. Il est souvent impossible de décider à priori des valeurs des poids des connexions d'un réseau pour une application donnée. A l'issu de l'apprentissage, les poids sont fixés : c'est alors la phase d'utilisation [38].

Au niveau des algorithmes d'apprentissage, il a été défini deux grandes classes selon que l'apprentissage est dit supervisé ou non supervisé. Dans le cas de *l'apprentissage supervisé*, les exemples sont des couples (Entrée, Sortie associée) alors que l'on ne dispose que des valeurs (Entrée) pour *l'apprentissage non supervisé* [5].

De nombreux chercheurs travaillant sur l'intelligence artificielle reprochent aux réseaux neuronaux de ne servir qu'à la reconnaissance des formes. Maintenant, Les réseaux neuronaux sont utilisés pour diriger des missiles, commander des robots, mettre à la disposition des malentendants des appareils de correction auditive « intelligents » et pour modéliser des marchés financiers complexes [4]. La recherche se tourne vers des problèmes encore plus délicats. Certaines équipes essayent de produire une machine possédant une conscience artificielle, une machine ayant sa façon de penser. Une telle machine pourrait apprendre à parler de la même manière qu'un enfant. L'ordinateur pourrait relier les phrases à une représentation interne « apprise » du monde. Le langage aurait donc une véritable signification pour la machine. Cependant, certains chercheurs ne pensent pas que cela soit réellement possible.

1.4.2.6 Algorithmes évolutionnaires (EA : Evolutionary Algorithms)

Toujours l'être humain cherche des solutions, pour ces problèmes, mieux que celles qui sont entre ces mains et surtout si ces solution prend beaucoup du temps. Pour cela, il existe une catégorie de problèmes pour lesquels il est difficile, peut être impossible, de trouver une solution en un temps limité. Alors, il est utile de trouver une technique permettant la localisation rapide de solutions sous-optimales, sachant que l'espace de recherche a une taille et une complexité suffisamment importantes pour éliminer toute garantie d'optimalité [39]. Pour cela, un système capable de s'auto-modifier au cours du temps, tout en améliorant sa performance dans

l'accomplissement des tâches auxquelles il est confronté, semble ouvrir la voie à une recherche intéressante [40].

Les algorithmes évolutionnaires (EA) représente une technique basée sur des principes simples. En effet, peu de connaissances sur la manière de résoudre ces problèmes sont nécessaires, même si certaines peuvent être exploitées afin de rendre plus efficace l'évolution [40] (en effet, il n'est pas réaliste d'espérer obtenir une méthode d'optimisation raisonnablement efficace si aucune connaissance sur le domaine à traiter). C'est pourquoi, dans de nombreux domaines, les chercheurs ont été amenés à s'y intéresser.

Un certain nombre de travaux ont porté sur l'optimisation de fonctions mathématiques complexes telles celles proposées dans [41]. Les EA ont été appliqués à différents problèmes issus de la recherche opérationnelle: la coloration de graphes [42], le problème du voyageur de commerce [43] [44], la gestion d'emploi du temps [45], etc.

On trouve aussi des applications en robotique pour la recherche de trajectoire optimale [46] ou l'évitement d'obstacles [47], en vision pour la détection [48][49] ou la reconnaissance d'images [50][51], en recalage d'images médicales [52], en reconnaissance de formes [53], mais aussi en mécanique pour l'optimisation de formes [54], en chimie [55], économie et gestion de production [56], gestion de trafic aérien [57], etc.

Par ailleurs, pour se rapprocher des phénomènes trouvés dans la nature, divers travaux ont porté sur des individus "intelligents" basés sur des réseaux connexionnistes. Ils ont montré l'intérêt d'appliquer les EA à des réseaux de neurones [58] [59], et particulièrement dans l'association avec l'apprentissage par rétro-propagation. Ils permettent entre autres de choisir un bon ensemble de poids initiaux, ce qui permet non seulement d'améliorer la vitesse de convergence des réseaux, mais aussi d'éviter de les faire converger vers des états correspondant à des minima locaux qui ne donneraient pas des solutions optimales.

Enfin, un certain nombre de travaux ont été effectués sur l'application des EA dans des environnements changeants [40].

Principes généraux

Les EA sont une classe d'algorithmes d'optimisation par recherche probabiliste basés sur le modèle de l'évolution naturelle. Ils modélisent une population d'individus par des points dans un espace. Un individu est codé dans un génotype composé de gènes correspondant aux valeurs des paramètres du problème à traiter. Le génotype de l'individu correspond à une solution potentielle au problème posé, le but des EA est d'en trouver la solution optimale [40].

La plupart des problèmes peuvent être résolus par une méthode de type recherche locale. C'est le cas, par exemple, de l'apprentissage d'un réseau de neurones par rétro-propagation du gradient. Partant d'une configuration de poids initiale, la méthode va chercher la meilleure solution dans le voisinage de cette configuration. Cette solution est optimale localement mais peut ne pas correspondre à un optimal global car il est possible qu'il existe une meilleure solution qui n'est pas dans le voisinage de la configuration initiale. La Figure 25 illustre ce type de problème.

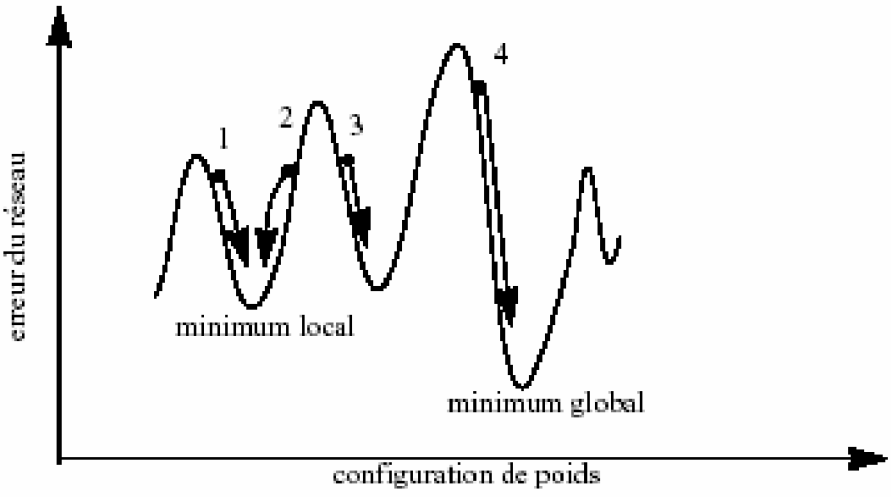

Figure 1.25 Exemple de convergence locale ou globale d'un réseau de neurones. Convergence vers des erreurs correspondant à des minima locaux (configurations 1, 2 et 3) ou à un minimum global (configuration 4). [40]

Les EA ont montré leur capacité à éviter la convergence des solutions vers des optima locaux, aussi bien lorsqu'ils sont combinés avec des méthodes de recherche locale comme la rétro-propagation du gradient [56] que lorsqu'ils sont seuls [60].

Quel que soit le type de problème à résoudre, les EA opèrent selon les principes suivants : la population est initialisée de façon dépendante du problème à résoudre (l'*environnement)*, puis évolue de génération en génération à l'aide d'opérateurs de *sélection*, de *recombinaison* et de *mutation*. L'environnement a pour charge d'évaluer les individus en leur attribuant une performance (ou *fitness*). Cette valeur favorisera la sélection des meilleurs individus, en vue, après reproduction (opérée par la mutation et/ou recombinaison), d'améliorer les performances globales de la population (voir figure 1.26).

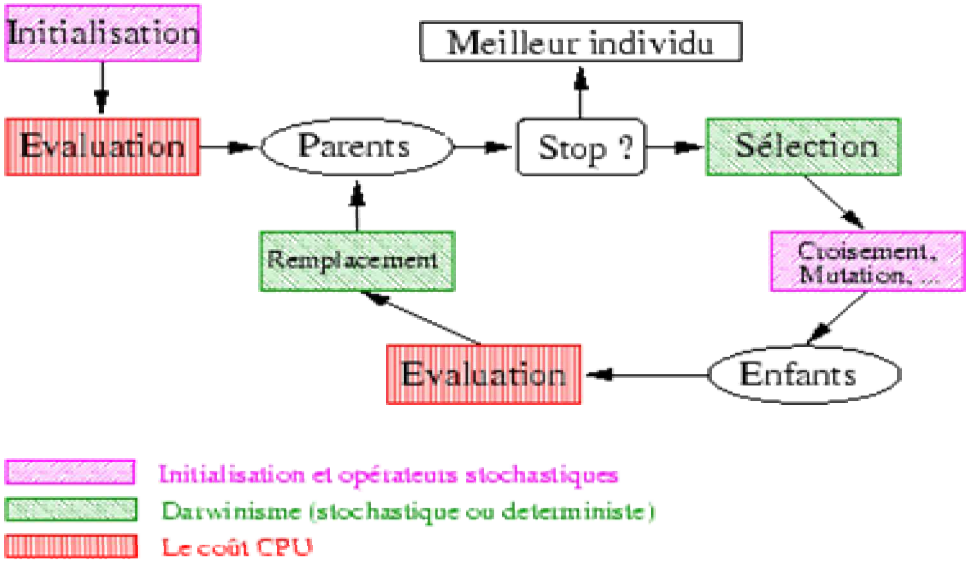

Figure 1.26: Squelette d'un algorithme évolutionnaire [61].

Plusieurs types d'évolution ont été développés, donnant naissance à trois grandes tendances : les Algorithmes Génétiques (ou *Genetic Algorithms* (GA)), les Stratégies d'Evolution (ou *Evolution Strategies* (ES)) et la Programmation Evolutive (ou *Evolutionary Programing* (EP)). Une

branche annexe, la Programmation Génétique (ou *Genetic Programing* (GP)) peut aussi rentrer dans ce type de systèmes.

De ces quatre méthodes classiques ont dérivé différentes techniques mélangeant les méthodes d'évolution des unes et des autres. Impossible à classer dans l'une des quatre familles citées ci-dessus, elles sont néanmoins considérées comme des EA.

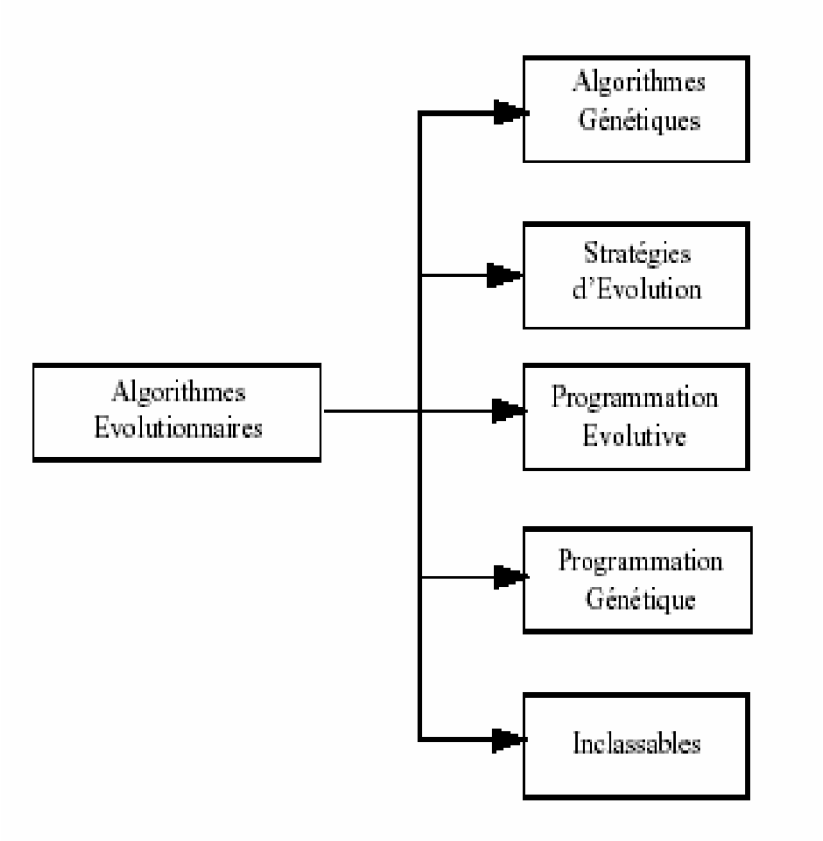

Figure 1.27 Différentes branches des algorithmes évolutionnaires [40].

- **Algorithmes Génétiques** (*GA*), *J. Holland*, **1975** et *D.E. Goldberg*, **1989**, *Michigan,USA.*
  Les GA ont été imaginés comme outils de modélisation de l'adaptation. Ils travaillent dans l'espace des chaînes de bits $[0,1]^n$ avec les schémas GGA et SSGA. Ce sont les plus connus des algorithmes évolutionnaires, et (malheureusement ?) souvent les seules variantes connues des chercheurs des autres disciplines [61].

- **Stratégies d'évolution** (*ES*), I. *Rechenberg* et *H.P. Schwefel*, **1965**, *Berlin*.
  Les ES ont été mises au point par deux jeunes ingénieurs travaillant sur des problèmes numériques. Le contexte était l'optimisation paramétrique, et les schémas d'évolution sont bien entendu les $(\mu, {}^{+}\lambda)$-ES. Un énorme progrès a été apporté par les techniques **adaptatives** d'ajustement des paramètres de mutation, et ce sont sans contestation les meilleurs algorithmes pour les problèmes purement numériques [61].

- **Programmation évolutionnaire** (*EP*), *L.J. Fogel*, **1966** et *D.B. Fogel*, **1991**, **1995**, *Californie, USA*. Imaginée par *Larry Fogel* pour la découverte d'automates à états finis pour l'approximation de séries temporelles, EP a rapidement travaillé sur des espaces de recherche très variés. Le schéma utilisé ressemble à s'y

méprendre à un (*P*+*P*)-ES -- quoique développé complètement indépendamment [61].

- **Programmation génétique** (*GP*), *J. Koza*, **1990**, *Californie, USA.* Apparue initialement comme sous-domaine des GAs, GP est devenu une branche à part entière (conférence, journal, ...). La spécificité de GP est l'espace de recherche, fait d'arbres représentant des programmes complets. GP cherche à atteindre (et réussit souvent!) un des vieux rêves des programmeurs, faire écrire le programme par un autre programme. Les schémas d'évolution utilisés sont souvent de type SSGA, mais avec des tailles de population énormes. Et les tendances récentes sont pour GP ... la parallélisation systématique et sur de grosses grappes de stations. Ainsi, les résultats récents les plus spectaculaires obtenus par *Koza* l'ont été avec **64** populations de **10000** individus, ou encore avec **1000** populations de **500** individus, utilisant le modèle en îlots sur des grappes *Beowulf* de stations Alpha ou Pentium sous Linux [61].

1.4.2.7 Système Immunitaire Artificielle (AIS : Artificial Immune System)

Pour comprendre le système immunitaire artificiel et son domaine d'application, on va d'abord présenter le système immunitaire naturel, sa structure, son rôle ……chez les êtres vivants.

1.4.2.7.1 Système Immunitaire Naturel (NIS : Natural Immune System)

Un système immunitaire est un ensemble des cellules, des tissus et des organes chargés de défendre l'organisme contre certains agents pathogènes, les substances étrangères ou anormales, notamment celles qui font partie des micro-organismes (bactéries, virus, etc.) et celles qui se trouvent à la surface des cellules cancéreuses[4]. Ces substances susceptibles d'induire une réponse immunitaire sont appelées ***antigènes*** [4]. Un système immunitaire joue un rôle fondamental, extrêmement complexe, il permet de réagir de façon appropriée à l'infinité d'antigènes différents et potentiellement pathogènes qui pénètrent dans l'organisme, et parfois l'envahissent. Les mécanismes physiologiques complexes mis en œuvre dans le système immunitaire ne sont pas encore complètement élucidés, mais ils sont chaque jour mieux compris [4].

Il existe trois grandes catégories de cellules immunitaires, qui sont des *leucocytes* (globules blancs) (voir tableau 2) [4]: les granulocytes neutrophiles, les monocytes / macrophages et les lymphocytes. Seule une minorité d'entre elles se trouvent en circulation dans le sang, qui ne leur sert, en fait, que de moyen de transport pour se rendre d'un point à l'autre de l'organisme avant de pénétrer dans un organe, à la rencontre des antigènes anormaux

Les lymphocytes, par certains aspects, sont les cellules les plus importantes du système immunitaire [4]. Ce sont les seules à avoir une spécificité d'action, chacune d'entre elles ne reconnaissant qu'un antigène, ce qui a l'avantage d'augmenter leur efficacité. Il en existe deux types principaux |4]: les lymphocytes *B* et les lymphocytes *T*. Les lymphocytes *B*, quand ils sont activés à la suite de l'introduction d'un antigène, se transforment en plasmocytes. Ils sont responsables de l'immunité humorale [4]. Cela signifie qu'eux-mêmes et surtout les plasmocytes sont les cellules responsables de la production des anticorps (ou immunoglobulines), qu'ils libèrent en général dans le sang [4]. Les lymphocytes *T* sont des globules blancs spécialisés dans la réponse immunitaire dite à médiation cellulaire. Certains d'entre eux, les lymphocytes *T4*, sont responsables de l'activation de la réponse immunitaire spécifique contre les agents pathogènes. Les lymphocytes *T8* sont des lymphocytes dits cytotoxiques, qui se fixent sur les cellules infectées par des virus et les détruisent [4].

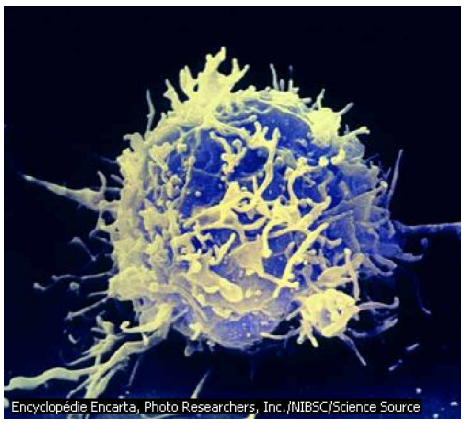

Figure 1.28 : Les lymphocytes T [4]

Forme de cellules							
Type de cellules	Globules rouges	Globules blancs Mononucléaires ⟷ Polynucléaires (granulocytes) Monocyt, Lymphocytes, Basophiles, Éosinophiles, Neutrophiles					plaquettes sanguines (thrombocytes)
Diamètre de cellules (Micron)	8-6	10-12	10-12	9-10	7-8	14-20	2-3
Nombre de cellules /mm ^{3}	Homme 4.5 - 6*10 ^{6} Femme 4 - 5.4*10 ^{6}	2000-7000 ou équivalent à 45-70%	50-300 ou équivalent à 1-3%	10-50 ou équivalent à 0.05%	1400-4000 ou équivalent à 20-40%	100-700 ou équivalent à 3-7%	150000 400000
Duré de vie de cellules	3 mois	6 heurs – des jours	8 – 12 jours	?	Quelques jours jusqu'à plusieurs années	Quelques mois jours jusqu'à plusieurs années	8-12 jours
Rôle de cellules	Transport de l'oxygène	Défenses de l'organisme					Contre hémorragie

Tableau 1.2 : Les cellules sanguines chez l'être humain [75].

1.4.2.7.2 Système Immunitaire Artificiel (SIA)

Le terme « système immunitaire artificiel » (« Artificial Immune System », AIS) s'applique à une vaste gamme de systèmes différents, notamment aux méta-heuristiques d'optimisation inspirées du fonctionnement du système immunitaire des vertébrés [5]. Un grand nombre de

systèmes ont été conçus dans plusieurs domaines différents tels que la robotique, la détection d'anomalies ou l'optimisation.

L'approche utilisée dans les algorithmes AIS est voisine de celle des algorithmes évolutionnaires, mais a également été comparée à celle des réseaux de neurones. On peut, dans le cadre de l'optimisation difficile, considérer les AIS comme une forme d'algorithme évolutionnaire présentant des opérateurs particuliers. Pour opérer la sélection, on se fonde par exemple sur une mesure d'affinité entre le récepteur d'un lymphocyte et un antigène ; la mutation s'opère quant à elle via un opérateur d'hyper-mutation directement issu de la métaphore [17].

1.4.2.7.3 Ordinateur immunitaire (Computer Immunology)

Il y a beaucoup de différences entre les organismes vivants et les ordinateurs, mais les ressemblances sont irrésistibles et pourraient diriger la voie à une sécurité d'ordinateur améliorée.

Les spécialistes dans le système immunitaire (Immunologistes) ont traditionnellement décrit le problème résolu par le système immunitaire comme la distinction entre le "***soi***" (self) et le dangereux "***non-soi***" (nonself) en éliminant ces derniers. Le ***soi*** représente les cellules internes et les molécules du corps, et le ***non-soi*** représente n'importe quel matériel étranger, en particulier les bactéries, les parasites et les virus. Le ***soi*** d'un système d'ordinateur peut être un modèle d'accès à une mémoire d'un autre hôte, réseau local, trafic du réseau …etc. Le ***non-soi*** dans les systèmes d'ordinateurs peut être un utilisateur non autorisé, un code étranger comme les virus ou un code imprévu comme les données endommagées. Le problème de protéger les systèmes d'ordinateur contre les intrusions malveillantes peut de la même façon être vu comme le problème de la distinction du soi (self) de non-soi (nonself).

Le système immunitaire d'ordinateur (CIS : Computer Immunology System) doit être capable d'avoir ces propriétés de NIS :

- La capacité de détecter les activités étrangères dangereuses.
- Avoir une mémoire des infections précédentes.
- Avoir une méthode pour la reconnaissance d'infections.
- Autonomie dans une réponse dirigeante.
- Avoir une méthode de protection du système immunitaire, lui-même, contre les attaques.

1.4.2.8 Les automates cellulaires

Un automate cellulaire est un réseau formé d'éléments simples, appelés cellules, connectés localement dans un espace à n dimensions. Chaque cellule d'un automate cellulaire possède un nombre fixe de cellules dites voisines, avec lesquelles elle est en contact direct. Une cellule se trouve à un temps t donné dans un état appartenant à un ensemble fini d'états possibles. L'état d'une cellule à un temps t + 1 dépend uniquement de l'état au temps t et de celui des cellules voisines à ce même temps t, et est déterminé selon une fonction de transition. On appelle configuration de l'automate l'ensemble des états de toutes les cellules le composant. La configuration d'un automate cellulaire évolue donc au cours du temps à partir d'une configuration initiale et selon les fonctions de transition des cellules.

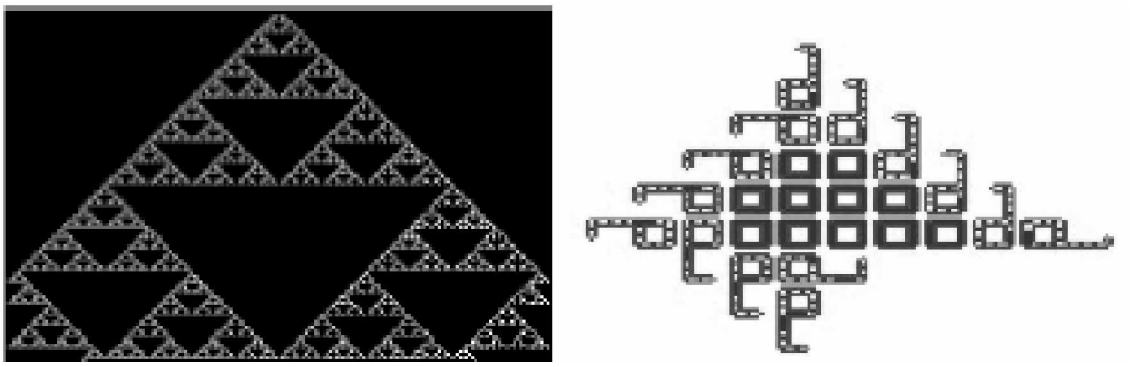

Figure 1.29 : (a) Exemple d'automate cellulaire à une dimension (Triangle de Pascal) [26].
(b) Les boucles de Langton (un automate cellulaire autoréplicateur) [26].

Les automates cellulaires ont été proposés par le mathématicien John von Neumann en **1966**. Ils sont utilisés pour modéliser des systèmes réels : interactions entre particules, formation de galaxies, systèmes biologiques, etc. Il a été démontré que les automates cellulaires constituent une bonne alternative aux équations différentielles.

On peut établir une classification des différents types de réseaux cellulaires selon leur dimension, le nombre d'états possibles pour chaque cellule, ou bien encore en fonction de leurs règles de transition. Stephen Wolfram propose quatre catégories pour les comportements des automates cellulaires [3] :

- ***Classe 1*** : Les automates cellulaires qui évoluent vers des états fixes et homogènes ;
- ***Classe 2*** : Les automates cellulaires qui aboutissent à des structures périodiques simples ;
- ***Classe 3*** : Les automates cellulaires qui évoluent vers des comportements chaotiques, caractérisés par des « attracteurs étranges » et des structures apériodiques ;
- ***Classe 4*** : Les automates cellulaires d'où émergent des « motifs » globaux complexes.

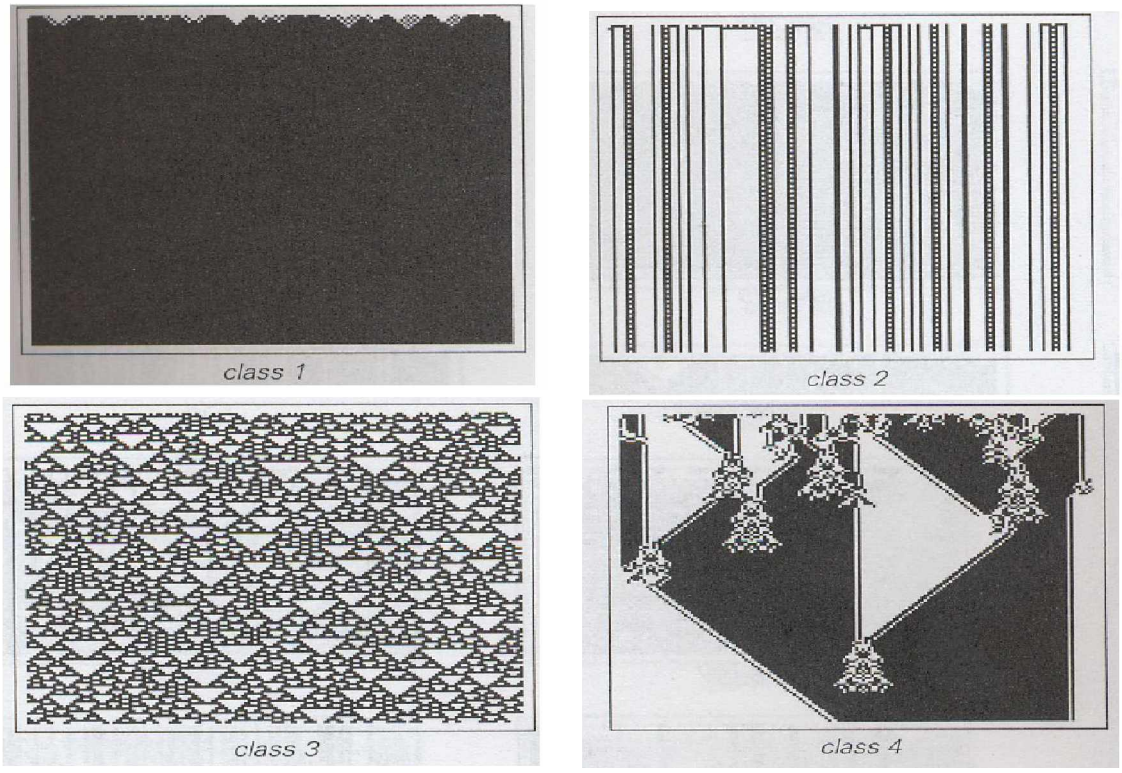

Figure 1.30: Exemples de classe 1, 2, 3, 4 [62]

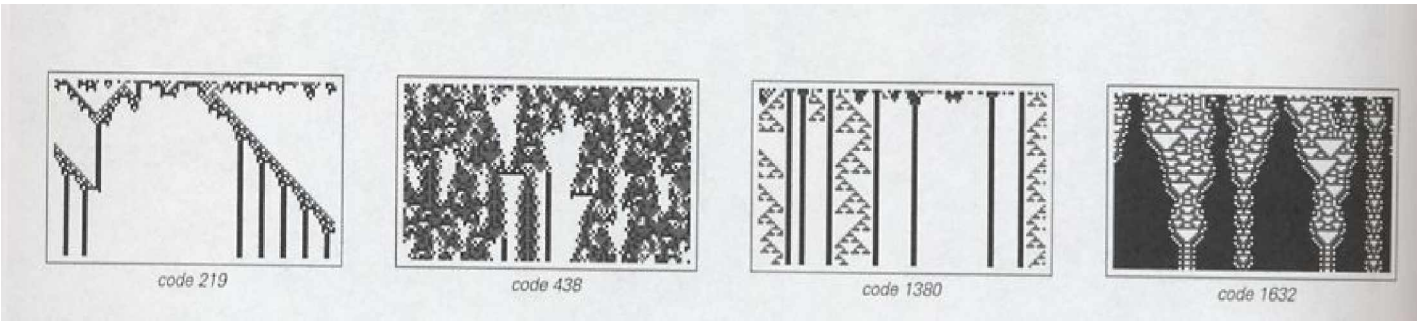

Figure 1.31: Exceptions, classes qui n'appartient pas aux classes 1, 2, 3, 4 [62]

Le « jeu de la vie » est un cas particulier d'automate cellulaire proposé par Conway en 1970. A l'origine, le Jeu de la vie fut présenté comme un jeu mathématique. Sa description va nous permettre de matérialiser et mieux comprendre ce que sont les automates cellulaires [63]. Le « jeu de la vie » c'est un automate cellulaire bidimensionnel, à deux états possibles par cellule. Une cellule possède alors huit cellules voisines, et peut être active ou non à un temps donné. Les règles de transition des cellules sont les suivantes :

- une cellule ayant deux cellules voisines actives ne change pas d'état ;
- une cellule ayant trois cellules voisines actives devient active ;
- une cellule ayant une ou plus de trois cellules voisines actives devient inactive.

Après quelques générations, il apparaît des comportements surprenant de la part du système. A partir de ces lois simples, certains types de formation de cellules émergent de la structure grouillante du départ [64].

Le jeu de la vie permet d'obtenir des configurations stables, périodiques ou non. Conway a prouvé en 1982 que le jeu de la vie est un moyen universel de calcul, au même titre que les machines de Turing.

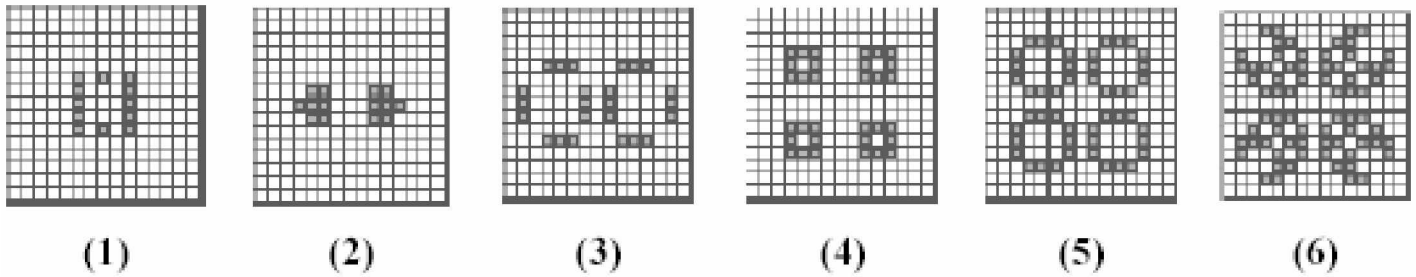

Figure 1.32 : Quelques générations successives d'un jeu de la vie [3]

1.4.2.9 L-système

Les processus de la morphogenèse restent mystérieux. Malgré les progrès de la génétique, de nombreux mécanismes nous restent inconnus. A. LINDENMAYER (1925-1989) a proposé une méthode de description formelle de la structuration des plantes. Basée sur une forme récursive de grammaire générative, elle a été approfondie et mise en œuvre graphiquement par P. PRUNSINKIEWICZ dans les années quatre-vingt [3].

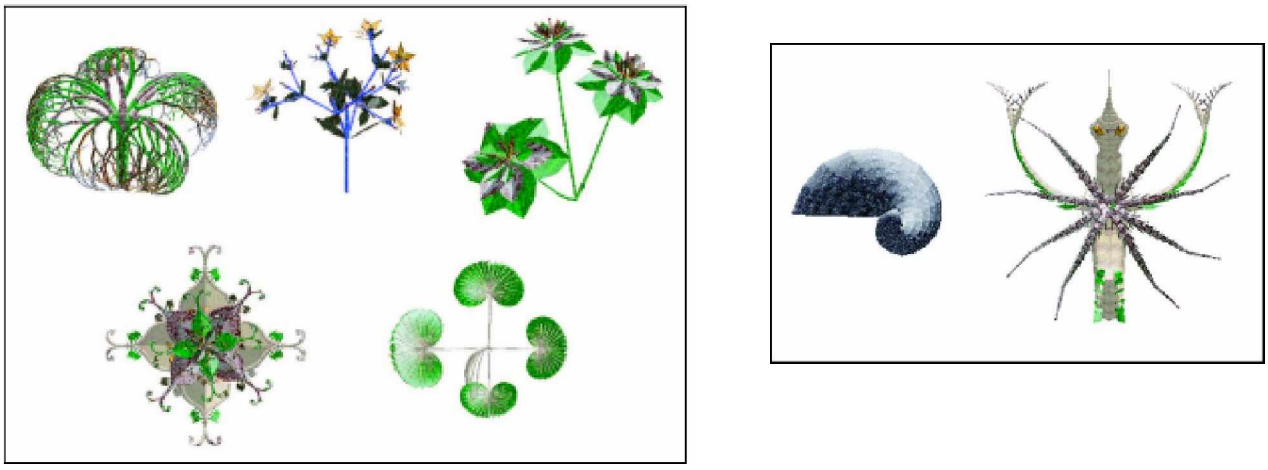

Figure 1.33 : Des images produites par les L-systèmes [65].

L'inspiration biomimétique de ces figures est indéniable. Elles ne sont pourtant que l'interprétation graphique de chaînes de caractères construites à partir d'un algorithme récursif [26].

Il est très difficile de cerner le lien entre la morphogenèse végétale et les L-systèmes [26]. De même que les paysages engendrés par les fractales, malgré leur ressemblance avec la réalité, n'ont en aucune manière suivi une évolution comparable aux paysages naturels, les similitudes entre plantes et L-systèmes ne signifient nullement que les génomes végétaux décrivent un mécanisme identique [26]. Les L-systèmes n'en confortent pas moins l'hypothèse selon laquelle les processus morphogénétiques mettent en œuvre des procédures répétitives, probablement souvent récursives [26].

1.5 Conclusion

D'après ce qu'on a vu et ce qu'on a lu, nous pouvons conclure que :

- Il y a des analogies entre le calcul distribué et les insectes sociaux.
- La biologie a trouvé des solutions aux problèmes informatiques difficiles.
- Le calcul biologiquement inspiré exige :
  - L'identification d'analogies.
  - La modélisation sur ordinateur des mécanismes biologiques.
  - L'adaptation de mécanismes biologiques pour les applications technologiques.

Cela ne va pas dire que l'inspiration de la biologie a réussi d'être la source des solutions de tout les problèmes actuellement rencontrés, par contre il y a des difficultés qui peuvent être apparues, tel que :

- La biologie échoue parfois.
- Quelques mécanismes naturels ne sont pas bien compris.
- Sans passer par des méthodes bio-inspirés, on peut résoudre des problèmes bien définis par des moyens meilleurs.

Dans le chapitre suivant, on va approcher profondément de la biologie moléculaire, et on va étudier plus précisément et en détaille la molécule d'ADN.

2.1 Introduction

La notion d'échelle est un concept flou : un objet à l'échelle du mètre par exemple peut avoir, *en gros*, entre un dixième et dix mètres [76].

Nous les humains vivons à l'échelle du *mètre* et c'est surtout à cette échelle que nous concevons la vie. La plupart des objets que nous construisons et manipulons, les animaux que nous mangeons etc., sont en gros de cette taille.

Zoomons mille fois ($^{a} \sim 10^{-3} m$) et nous arrivons à l'échelle du mm, les plus petites graduations que nous avons sur les règles. C'est l'échelle à laquelle vivent les petits insectes, les fourmis, les puces. C'est également l'échelle de la résolution de notre oeil : nous ne distinguons pas les objets plus petits que quelques dixièmes de millimètre [76]. Enfin, c'est à cette échelle que les phénomènes capillaires sont importants.

Zoomons encore mille fois ($^{a} \sim 10^{-6} m$), et nous arrivons à l'échelle du micron (μm). C'est l'échelle des cellules[76] : une bactérie a environ un micron de diamètre et quelques micron de long ; la taille typique des cellules de notre corps est de l'ordre de 10 à 20 microns, même si certains neurones peuvent pousser des extensions jusqu'à un mètre. C'est également la limite de résolution de nos microscopes optiques : depuis les travaux de M. Abe, nous savons que nos microscopes ne peuvent pas distinguer des objets plus petits que $\sim \lambda / 2NA$, où λ est la longueur d'onde de la lumière ($\approx 0.5\mu$m pour le vert) et NA l'ouverture numérique de l'objectif (≈ 1) [76]. Dans le meilleurs des cas, ceci met la résolution de nos microscopes optiques à $\approx 0.2\mu$m. Un virus mesure de l'ordre de 0.05μm, et c'est pourquoi leur existence n'a été découvert qu'au début du vingtième siècle [76].

Zoomons encore mille fois, et nous arrivons à l'échelle du nanomètre $^{a} \sim 10^{-9} m$, c'est l'échelle des molécules de la vie. Une molécule d'ADN fait environ un nanomètre de diamètre, une protéine globulaire de 5 à 10 nm, un microtubule (des tubes creuses pour maintenir la rigidité de la cellule) une vingtaine de nm de diamètres [76]. C'est également la limite de résolution des microscopes électroniques, qui peuvent voir, dans le meilleur des cas, jusqu'à 1/10ème de nm [76].

Notre voyage s'arrête ici, à l'échelle de nanomètre. Il existe d'autre échelle qui suit l'échelle de nanomètre, c'est celui d'angstrom ou l'échelle d'atomes. Dans l'échelle de nanomètre, nous nous intéressons à la molécule d'ADN. A travers ce chapitre, on va décrire la cellule brièvement, qui est l'emplacement de l'ADN. Ensuite, on va passer à la molécule d'ADN proprement dite, en présentant sa structure physique et chimique ainsi que ces caractéristiques. Enfin, ce chapitre sera achevé par les outils de la biologie moléculaire tel que les enzymes, la technique de PCR…

2.2 L'ADN, le miracle :

L'information conservée dans l'ADN ne doit en aucune manière être sous-estimée. Elle est tellement importante qu'une seule molécule d'ADN humain renferme assez d'information pour remplir une encyclopédie d'un million de pages ou pour remplir environ 1.000 livres [78]. Retenez bien ce fait: 1.000.000 de pages d'encyclopédies ou l'équivalent de 1.000 livres. Cela revient à dire que le noyau de chaque cellule contient assez d'information pour remplir une encyclopédie d'un million de pages; une information utilisée pour contrôler les fonctions du corps humain. Pour faire une analogie, nous pouvons considérer que même l'Encyclopaedia Britannica composée de 23 volumes, une des plus importantes sources d'information au monde, ne possède que 25.000 pages [78]. Ainsi, devant nos yeux apparaît une image incroyable. Dans une molécule trouvée dans un noyau, qui est lui-même beaucoup plus petit que la cellule microscopique qui l'abrite, existe une banque de données 40 fois plus

importante que la plus grande encyclopédie au monde qui contient elle-même des millions d'informations[78]. Ce qui signifie qu'il s'agit d'une immense encyclopédie en 1.000 volumes qui est unique et n'a pas d'équivalent dans le monde. Une encyclopédie dans laquelle chaque information, existante dans les gènes, serait lue à chaque seconde de façon continue, sans aucune pause, et qui prendrait un siècle pour en achever la lecture [78]. Et, si nous devions imaginer l'information de l'ADN sous forme de livres, les ouvrages qui seraient mis les uns sur les autres atteindraient une hauteur de 70 mètres[78]. Les dernières estimations ont même révélé que cette immense encyclopédie contenait trois milliards de "sujets" différents. Par ailleurs, si l'information de l'ADN devait être écrite, le papier utilisé irait du Pôle Nord à l'Equateur [78].

Ces exemples ne sont qu'une indication de l'impressionnante quantité d'information contenue dans l'ADN. Cependant, comment pouvons-nous parler de molécule contenant de l'information? Car il ne s'agit pas ici d'un ordinateur ou d'une bibliothèque mais seulement d'un bout de chair qui est cent mille fois plus petit qu'un millimètre et n'est constitué que de protéines, de lipides et de molécules d'eau. C'est un miracle aux proportions gigantesques que cet infinitésimal morceau de chair puisse contenir et emmagasiner la moindre information, alors que dire des millions d'autres[78].

Les ordinateurs sont, de nos jours, la forme la plus évoluée de conservation de l'information. Une masse de données qui, 30 ans auparavant, pouvait être conservée dans un ordinateur de la taille d'une salle, peut aujourd'hui être stockée dans de petits "disques", même si la dernière technologie inventée par l'intelligence humaine, après des siècles de savoir accumulé et des années d'efforts, est très loin d'atteindre les capacités de conservation d'un seul noyau cellulaire [78].

2.3 Où trouve-t-on l'ADN dans les organismes?

Les organismes sont constitués d'une ou plusieurs unités appelées cellules. Une cellule contient un cytoplasme, séparé de l'extérieur par une membrane. Il y a des cellules avec et sans noyaux. Le cytoplasme de toutes les cellules contient des ribosomes [77].

Dans le cytoplasme des cellules sans noyau (archaea et bacteria), on trouve au moins une molécule d’ADN circulaire attachée à la membrane [77].

Chez les Eucaria (eucaryote), cellules avec noyau, il y a plusieurs molécules d’ADN. Elles sont linéaires et confinées, sous le nom de chromosomes dans le noyau (une particule cytoplasmique) (voir figure 2.1). Les extrémités des chromosomes sont des télomères [77].

Certains organismes ne possèdent qu'une copie de chaque chromosome, on dit qu'ils sont haploïdes [77]. Les individus diploïdes ont deux copies de chaque chromosome, l'une d'origine paternelle et l'autre maternelle. Il existe des organismes polyploïdes [77].

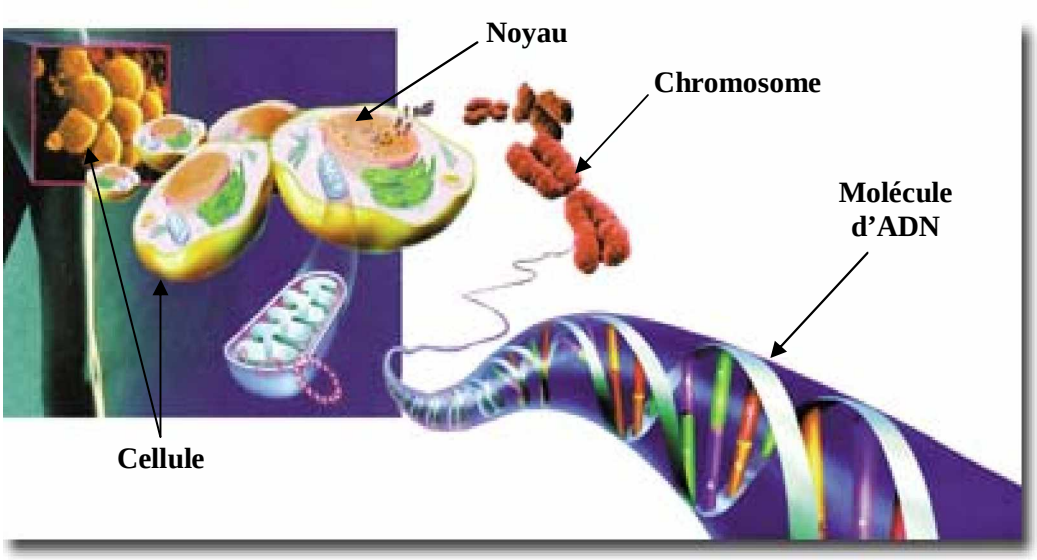

Figure 2.1 : L'emplacement de la molécule d'ADN dans la cellule [78].

La molécule d'ADN dans le noyau est recouverte de protections spéciales appelées chromosomes [78]. La longueur totale d'une molécule d'ADN enroulée dans un chromosome est de 1 mètre. Un chromosome a une épaisseur d'un nanomètre, c'est-à-dire un milliardième de mètre [78]. Comment une molécule d'ADN d'un mètre de long peut-elle tenir dans un espace aussi minuscule? [78]

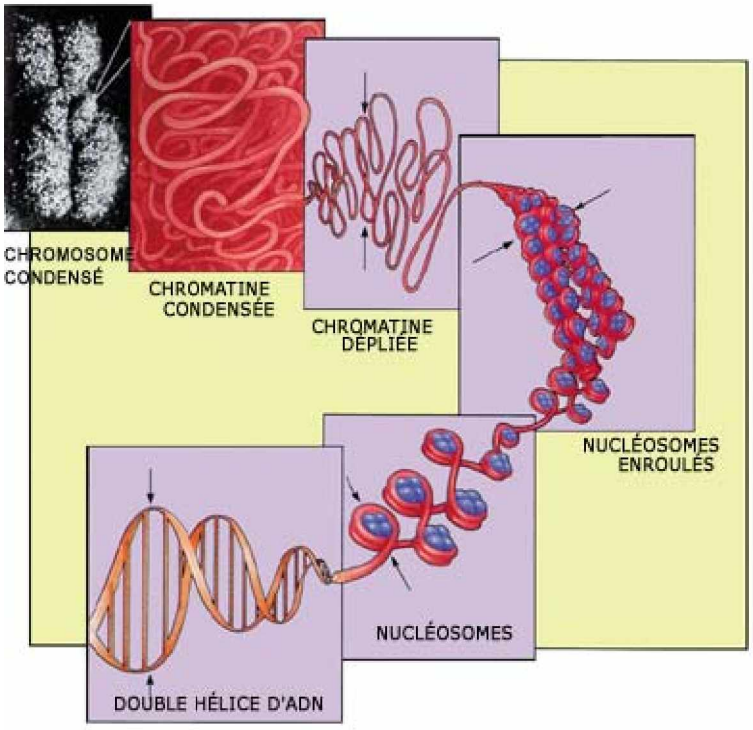

Figure 2.2 : Le chromosome et la molécule d'ADN [78].

Les chromosomes sont en fait constitués de systèmes de conteneurs particuliers encore plus petits [78]. La molécule d'ADN est d'abord enroulée, comme une pelote de laine (voir figure 2.2), autour de protéines spécifiques appelées histones. Des structures appelées

nucléosomes sont ainsi formées [78]. Ces nucléosomes sont conçus spécifiquement pour protéger l'ADN et l'empêcher d'être endommagé [78]. Quand les nucléosomes s'encordent les uns aux autres, ils forment la chromatine. Des boucles fermement enroulées sur elles-mêmes se forment dans la chromatine. De cette manière, une superbe structure compresse la molécule d'ADN dans un espace aussi minuscule qu'un milliardième de sa longueur [78].

2.4 La structure de l'ADN

Les acides nucléiques sont les dépositaires de l'information génétique. Le plus célèbre est l'ADN, l'acide désoxyribo-nucléique. C'est à travers cette molécule que l'information génétique est transférée entre les générations.

L'ADN dans le noyau cellulaire possède une structure en forme de spirale. Quand on le déroule, l'ADN devient un fil d'environ un mètre de long [78]. La manière dont un mètre d'ADN est pressé dans un noyau cellulaire minuscule est un sujet qui mérite plus d'attention.

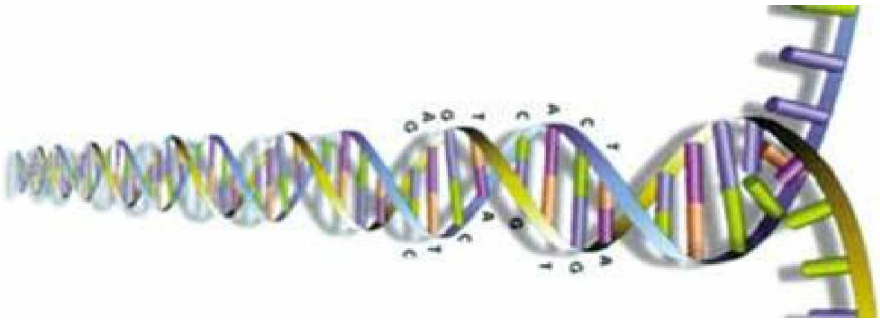

Figure 2.3 : L'ADN, après le déroulement devient un fil presque d'un mètre de long [78]

Les atomes constitutifs de l'ADN sont agencés de telle manière que la quantité maximum d'information peut être conservée dans le plus petit espace possible [78]. Trois éléments se retrouvent à chaque marche des deux échelles qui s'enroulent l'une autour de l'autre: un sucre, un phosphate et des bases organiques contenant de l'hydrogène, constituant les codes de l'ADN. Bien que les outils et les fonctions sont les mêmes en chaque être humain, les codes qui permettent aux gens d'être différents les uns des autres sont constitués de ces bases d'hydrogène [78]. Les différences dans l'organisation de ces bases sont à l'origine de toutes nos différences. Ces bases sont l'adénine, la guanine, la cytosine et la thymine (i.e : ces bases sont appelées aussi des nucléotides ou des monomères[1]). Elles sont liées les unes aux autres selon des règles précises. Tout comme une langue étrangère que les scientifiques commencent à peine à comprendre, ces quatre types de fondements organiques à base d'hydrogène contiennent la totalité du code de notre existence biologique [78].

Ces bases constitutives de la molécule d'ADN sont connues selon leurs initiales, A, G, C et T (leurs structure chimique est montrée dans la figure 2.4). L'information dans la banque de données du noyau cellulaire est ainsi conservée grâce à un alphabet constitué de ces quatre lettres [78].

[1] Un polymère est une longue chaîne linéaire formée d'éléments simple appelés monomère : la molécule d'ADN est un polymère constituée de quatre monomères : A, T, G, C.

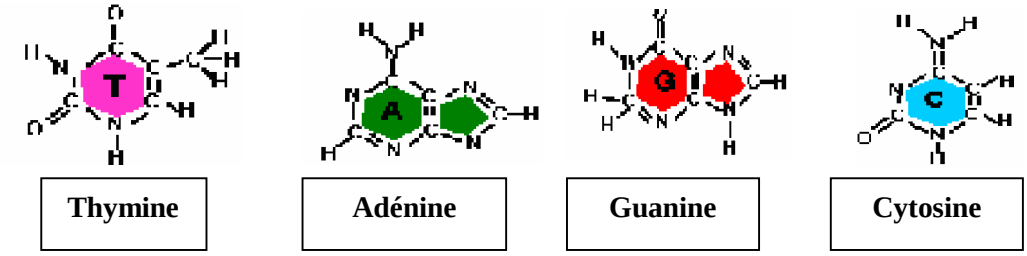

Figure 2.4: Structure chimique des 4 bases d'ADN [79]

Chaque gène, qui comprend une portion de la molécule d'ADN, détermine une caractéristique particulière du corps humain [78]. D'innombrables propriétés comme la taille, la couleur des yeux, la matière et la forme du nez, des oreilles et du squelette sont formées sous le commandement des gènes apparentés. Nous pouvons comparer chacun de ces gènes aux pages d'un livre. Sur chaque page sont écrites des phrases constituées des lettres A-T-G-C.

Il existe approximativement 30.000 gènes dans l'ADN d'une cellule humaine [78]. Chaque gène est composé d'une séquence spéciale de nucléotides, allant de 1.000 à 186.000 selon le type de protéines qui y sont associés [78]. Ces gènes contiennent les codes d'environ 100.000 protéines qui fonctionnent dans le corps humain et contrôlent la production de ces protéines [78].

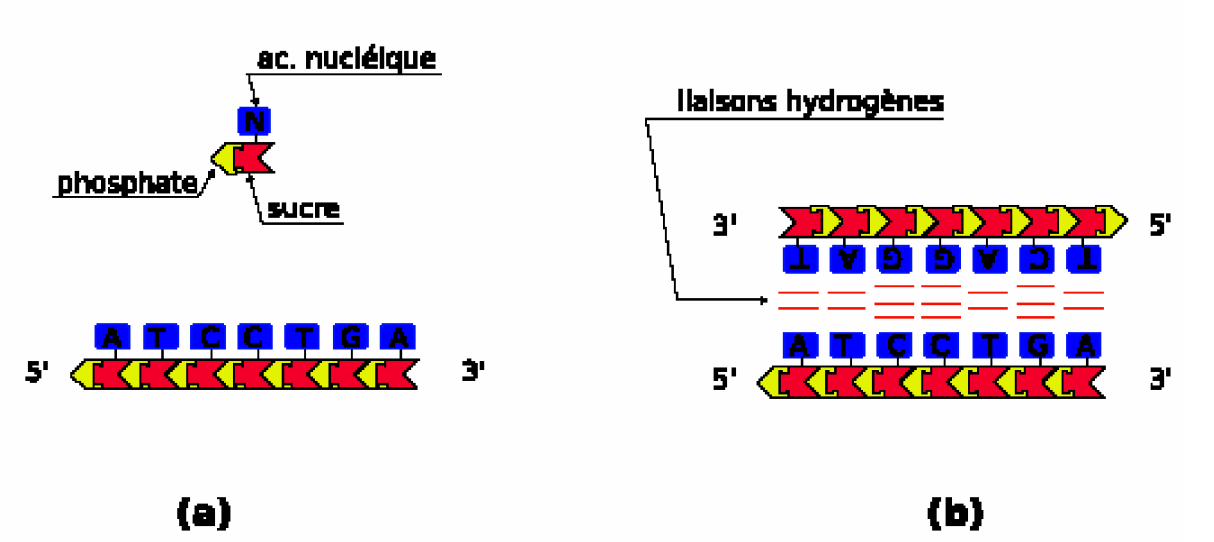

Figure 2.5: La structure d'une molécule d'ADN. (a) le monomère est formé d'un sucre (ribose), d'un groupe phosphate et d'un acide nucléique. Le binôme sucre-phosphate est capable de polymériser (en formant des liaisons covalentes). Le monomère est *orienté*, et possède un coté 5' et un coté 3' (du nom de la position des carbones sur le sucre). Lors de la polymérisation, les nouveaux monomères sont ajoutés toujours au coté 3'. C'est toujours dans le sens 5' à 3' qu'on lit la séquence de l'ADN ; ici par exemple, on doit lire ATCCTGA. (b) Une molécule d'ADN peut former des liaisons hydrogènes avec une autre si les deux brins ont des séquences *complémentaires* : A en face de T et C en face de G. L'appariement est *antiparallèle*. La séquence du brin complémentaire est ici TCAGGAT. C'est souvent sous cette forme de *double brin* que l'on trouve l'ADN dans la nature [76].

Les nucléotides sont des molécules orientés, et on appelle leur deux cotés 5' et 3', du nom de la position des atomes de carbones du sucre. La polymérisation, par formation des liaisons covalentes, se fait *toujours* par l'ajout des nucléotides au coté 3' de la chaîne. La lecture de l'ADN pour fabriquer des protéines se fait également dans ce sens. Les séquences sont toujours écrites donc dans ce sens : ACCGTGTT veut toujours dire 5'-ACCGTGTT-3'.

Les acides nucléiques d'un nucléotide sont capable de former des liaisons hydrogènes de façon spécifique : le A avec le T et le C avec le G. Le couple AT forme 2 liaisons hydrogènes, et le couple GC forme 3 liaisons. Une molécule d'ADN est donc capable de former beaucoup de liaisons hydrogènes avec une autre si les bases qui se font face sont complémentaires *et* les deux brins sont *antiparallèl*es (voir figure 2.5b). L'antiparallélisme est obligatoire si l'on veut avoir formation correcte des liaisons hydrogènes. De plus, les deux brins complémentaires s'enroulent autour de l'autre pour former une double hélice. La séquence complémentaire de 5'-ACCGTGTT-3' et donc 5'-AACACGGT-3'.

C'est très souvent sous la forme de double brins (ADNdb, ou dsDNA en anglais 'duble strand DNA') que l'on trouve l'ADN dans la nature. L'avantage de l'appariement est comme suit: pour produire fidèlement une nouvelle molécule d'ADN double brin (qui porte l'information génétique) à partir d'une ancienne, il suffit de séparer les deux brins de l'ancien, et utiliser chaque brin comme modèle pour fabriquer un brin complémentaire. On appelle ce procédé de duplication *semiconservative*, puisque la moitié d'une nouvelle molécule d'ADN vient d'une ancienne. C'est exactement comme cela que la cellule procède. C'est pour cette raison que l'ADN se trouve toujours sous forme double brin 'db'.

Les monomères d'un brin d'ADN sont reliés par des liaisons covalentes, extrêmement solide. Donc, les liaisons entre les deux brins complémentaires ne sont par contre que des liaisons hydrogènes, pas très solide. En augmentant la température vers 90°C, les deux brins se séparent facilement. On appelle cela la *fusion* de l'ADN et c'est un très joli problème de physique statistique. En abaissant la température, les deux brins peuvent s'apparier à nouveau, et on appelle cela *l'hybridation*.

2.5 Les outils de la biologie moléculaire

La majorité des expériences de la biologie moléculaire utilisent les quelques outils de base que nous verrons ci-dessous. Leur compréhension est nécessaire, et presque suffisante, pour travailler dans un laboratoire. Nous présentons d'abord ces outils (enzymes, électectrophorèse, PCR) et montrons ensuite quelques unes de leur utilisation (ADN recombinant, Séquençage et Synthèse). L'ensemble constitue le copier-coller-dupliquer de la biologie moléculaire [76].

2.5.1 Les enzymes

En physique, nous disposons d'instruments très sophistiqués, dont la maîtrise de certains demande un très long effort : des spectromètres très fins résolu en temps, la RMN, la diffusion de la lumière, ... [76] La visite de n'importe quelle laboratoire de physique vous en fournira une liste non-exhaustive. La biologie moléculaire au contraire possède très peu d'instruments. Ces instruments sont des enzymes extraits du monde vivant. Ces enzymes sont de véritables magiciens nanométriques, et on ne comprend pas toujours exactement leurs fonctionnements, mais on sait s'en servir. C'est un peu comme conduire une voiture : la plupart d'entre nous ne savons pas exactement ce qu'il y a sous le capot, même si les principes générales nous sont connus [76]. Cependant, nous savons parfaitement nous en servir pour aller d'un point à un autre.

Notons qu'un enzyme est une protéine agissant comme catalyseur de nombreuses réactions biochimiques [4] (voir figure 2.6). Actuellement, prés de 1 000 enzymes différents ont été identifiés. La science qui étudie les enzymes est l'enzymologie. Les enzymes sont des substances extrêmement efficaces [4].

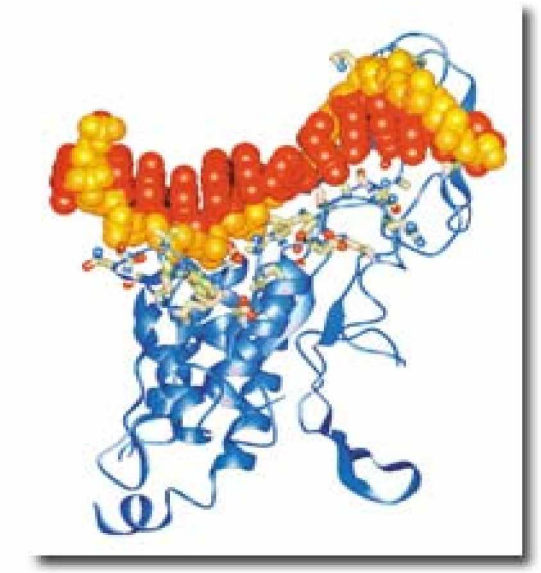

Figure 2.6 : Plusieurs enzymes doivent exister près de l'ADN durant la réplication et la synthèse protéique. Les zones rouges et jaunes dans l'image montrent les enzymes qui agissent avec l'ADN [78].

2.5.1.1 les polymérases :

Comment fabriquer une nouvelle molécule d'ADN ou d'ARN[2] ? Watson et Crick [80] ont mentionné dés le début, dans leur article de 1953 sur l'ADN, que la forme même de cette molécule peut être une indication pour sa synthèse : chaque brin devrait servir de *moule* pour fabriquer son complémentaire[3]. Nous connaissons aujourd'hui, grâce au travail acharné de nombreux biochimistes, les enzymes qui dirigent ces actions de synthèse [76].

Le premier enzyme de synthèse est découvert en 1955. Elle s'appelle PNPase, elle synthétise l'ARN en incluant de façon aléatoire les nucléotides présents dans la solution [76]. Niremberg et Matthaie, dans leur course pour briser le code génétique, l'ont utilisé pour synthétiser des ARN polyU en 1961 et établir ainsi la relation entre le triplet UUU et l'acide aminé phenylalanine[76].

[2] ARN: Acide Ribo-Nucléique, est un acides nucléiques comme la molécule d'ADN, il se compose de 4 nucléotides : A (Adénine), U (uracile), C (Cytosine), G (Guanine).

[3] "It has not escaped our notice that the specific pairing that we have postulated immediately suggests a possible copying mechanism for the genetic material"

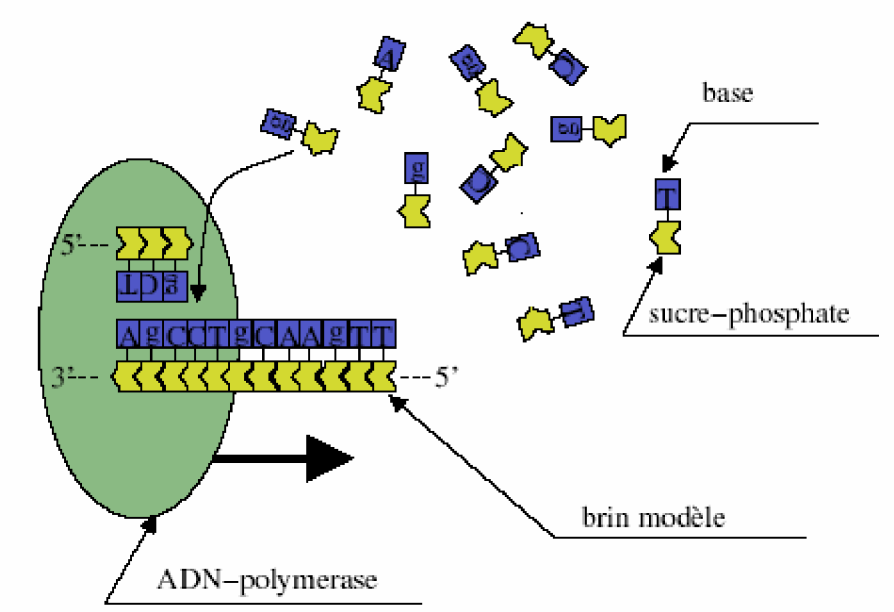

Figure 2.7: Synthèse d'un nouveau brin d'ADN à partir d'un modèle. Une molécule d'ADN polymérase *lit* le brin modèle à chaque étape et choisit et incorpore le nucléotide complémentaire présent en solution. La synthèse de l'ADN se fait toujours dans la direction 5' à 3', l'ADN polymérase lit le brin modèle et avance donc dans la direction 3' à 5'[76].

Mais le maître absolu, l'enzyme de réplication du génome (nommé l'ADN polymérase) a été découvert en 1958 après le travail minutieux des biochimistes Arthur Kornberg et Robert Lehman de l'Université de Stanford.

La réaction de base peut s'écrire $(ADN)_n + NTP \rightarrow (ADN)_{n+1} + P_i + P_i$ [76]
Une chaîne d'ADN de longueur *n* incorpore un nucléotide (présent en solution comme NTP un nucléotide tri-phosphate) et s'allonge d'une unité. Cette réaction est énergétiquement favorable. Le travail de l'ADN polymérase est de choisir le bon nucléotide parmi les 4 (ATP, CTP, TTP, GTP) présent dans la solution, en lisant le brin complémentaire. Nous verrons la biochimie de ce processus plus bas. Ce qui est essentiel à retenir ici est que l'ADN polymérase ne fonctionne qu'en présence d'un brin d'ADN utilisé comme modèle (voir figue 2.7). L'autre élément important est que l'ADN polymérase ne fonctionne que si il y a déjà une séquence synthétisée, et correctement hybridée au brin modèle *derrière lui* ! La longueur minimale de cette séquence est de trois à quatre bases.

L'ADN polymérase est là pour dupliquer le génome et le transmettre à la génération d'après. Il doit donc être le plus précis possible. Son taux d'erreur est de l'ordre de 10^{-19}, c'est à dire que tous les milliards de base, il commet une erreur. Essayez donc de recopier 1000 livres en ne commettant qu'une seule faute pour avoir une idée de la précision de cette machine [76]. Les considération énergétiques - un mauvais base pairing coûte deux à trois liaisons hydrogènes - donne un taux d'erreur de l'ordre de 10^{-4}. Il existe d'autres mécanismes de relecture qui permettent d'augmenter la précision. On peut constater que l'existence d'une séquence déjà hybridée derrière participe à ces mécanismes. En effet, supposons que l'ADN polymérase inclue un mauvais nucléotide et avance d'une séquence. La séquence qu'il a alors derrière lui n'est pas correctement hybridée et arrête le fonctionnement de l'ADN-pol. L'enzyme doit alors faire marche arrière, enlever la mauvaise base, et recommencer. Aussi surprenant que cela puisse paraître, c'est vraiment comme cela que ça se passe.

In vivo, pour dupliquer une molécule d'ADN, il faut séparer à un endroit les deux brins d'ADN et commencer à dupliquer chaque brin séparément. Le point de départ s'appelle

l'origine de réplication. *In vitro* le problème ne se pose pas, puisqu'en élevant la température (autour de 60 à 90°C) on sépare les deux brins.

L'ARN polymérase est un cousin très proche de l'ADN polymérase, à cette différence qu'il lit un brin d'ADN pour le copier en ARN. Il fut découvert en 1960 par plusieurs groupes de façon presque simultanée. Notons qu'ARN-pol est nettement moins précis que l'ADN-pol et ne possède pas plusieurs des mécanismes de correction d'erreurs de ce dernier. Enfin, le dernier enzyme de cette classe est appelé *reverse transcriptase* et qui polymérise l'ADN en lisant *l'ARN* ! Cet enzyme est propre aux retrovirus dont le matériel génétique est l'ARN et non l'ADN.

2.5.1.2 Ligase

Cet enzyme, découvert en 1967, est utilisé pour « coller » deux brins d'ADN en créant une liaison covalente entre l'extrémité 3' de l'un et 5' de l'autre [76]. Les deux morceaux doivent être immobilisés sur le brin complémentaire (voir Figure 2.8). *In vivo* cet enzyme est utilisé lors de la duplication d'ADN. Comme le chromosome est assez long, souvent sa duplication commence à plusieurs endroits en parallèle. La ligase intervient pour attacher ces copies partielles les unes aux autres.

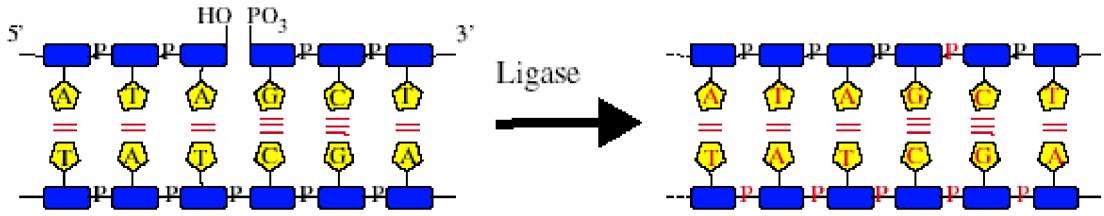

Figure 2.8: L'action de ligase est de former une liaison covalente phosphodiester entre deux morceaux d'ADN, immobilisés sur le brin complémentaire qui lui, est intact [76].

2.5.1.3 Enzymes de restrictions

Les enzymes de restriction sont découvertes à partir de 1973. Ils sont capables de reconnaître spécifiquement une courte séquence, de 4 à 10pb, et de cliver l'ADN au site reconnu. Ils permettent de fragmenter l'ADN en segments de taille réduite, ou de le couper à tel ou tel site désiré (voir figures : 2.9, 2.10). Certains enzymes coupent le site en son milieu et produisent deux fragments dont les extrémités sont franches. Cependant, la plupart réalisent une coupure dissymétrique : on parle dans ce cas d'extrémités cohésives (chaque fragment possède une chaîne qui dépasse l'autre de quelques bases). Plusieurs centaines de ces enzymes ont été caractérisés : ils reconnaissent une grande variété de sites de coupure.

Les enzymes de restriction sont utilisés pour établir une carte de restriction de toute molécule d'ADN que l'on souhaite caractériser [133]. Cela consiste à déterminer l'ordre des sites de restriction le long de cette molécule, qui vont produire, après "digestion enzymatique" de cette molécule, des fragments de tailles différentes dont la taille pourra être définie par électrophorèse.

Les enzymes de restriction appartiennent à la classe des endonucléases, c'est-à-dire des enzymes capables de cliver les liaisons phosphodiester entre deux nucléotides à l'intérieur d'un acide nucléique. Les endonucléases se différencient des exonucléases qui dégradent la molécule d'ADN à partir de l'une de ses extrémités (3' ou 5').

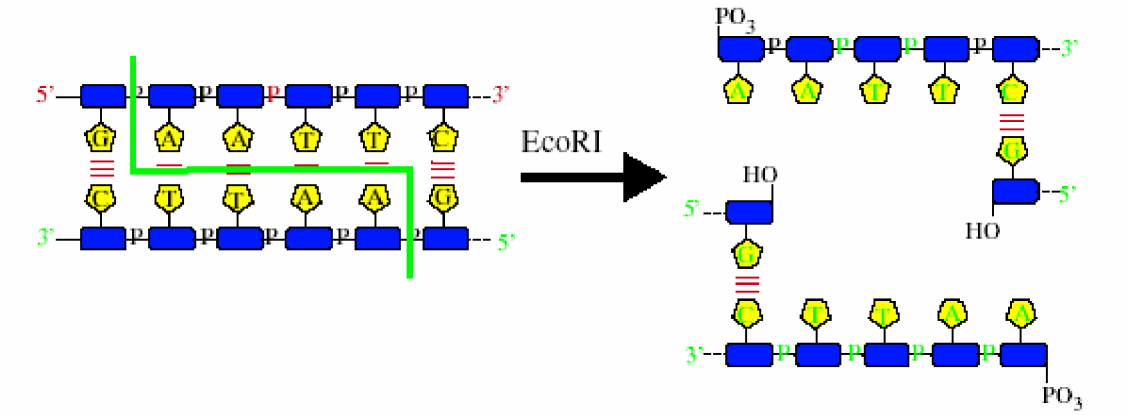

Figure 2.9: L'action des enzymes de restriction est de couper un double brin d'ADN à un endroit spécifique. Ici, nous montrons l'action de EcoRI et son site de reconnaissance. Cette enzyme produit des bouts collants [76].

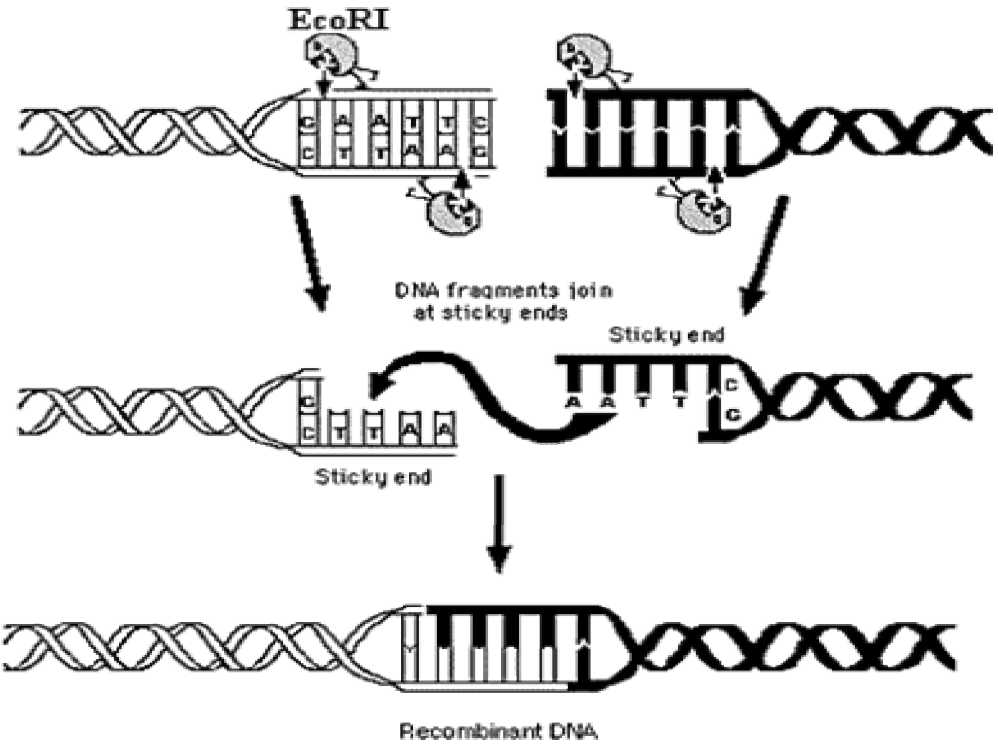

Figure 2.10 : L'effet de l'enzyme de restriction EcoRI sur la molécule de l'ADN [133]

Pour résumer, les enzymes de restriction sont les ciseaux moléculaires, coupant l'ADN à des endroits bien précis [76]. Nous en connaissons aujourd'hui des centaines de ces enzymes, chacun reconnaissant une séquence spécifique d'ADN pour le couper à cet endroit. La sélectivité de ces enzymes varie : certain n'ont besoin que de reconnaître une séquence de

5-6 paire de base, tandis que d'autres utilise plus d'une vingtaine. Certains produisent des bouts « collants », en laissant des petits morceaux d'ADN simple brin aux extrémités.

2.5.2 Electrophorèse

Le principe de l'électrophorèse, qui consiste à faire migrer dans un certain milieu des molécules chargées en présence d'un champ électrique, date des années trente. Cependant, elle est devenue une technique très répandue d'analyse et de séparation d'ADN (on l'utilisait initialement pour la migration de protéines) avec l'utilisation de gel d'agarose (1973) [81]. L'ADN est déposé dans un gel d'agarose en présence d'un marqueur qui devient fluorescent lorsqu'il s'intercale dans l'ADN. L'ADN est attiré vers l'électrode + (phosphates chargés négativement), mais les mailles du gel d'agarose permettent de séparer les molécules en fonction de leur taille [81]. Cette méthode permet de vérifier la taille de fragments d'ADN, et de les séparer ensuite. On peut aussi l'utiliser pour détecter la présence d'une séquence spécifique dans des fragments donnés. C'est la technique du Southern blot [81]: Après l'électrophorèse d'ADN double brin, le gel est transféré sur une membrane et l'ADN double brin dénaturé en simple brin. Cette étape permet de fixer l'ADN sur un support solide. La réplique du gel sur la membrane est ensuite mélangée à une sonde, c'est-à-dire un simple brin d'ADN portant un marqueur (radioactif ou fluorescent). Après rinçage, seules les bandes du gel ayant intégré la séquence voulue retiennent la sonde, et on peut les révéler (par autoradiogramme ou simple photo dans le cas d'un marqueur fluorescent). Cette technique est utilisable pour l'ADN (Southern), l'ARN (Northern), ou les protéines (Western) [81]. (Voir figures 2.11, 2.12)

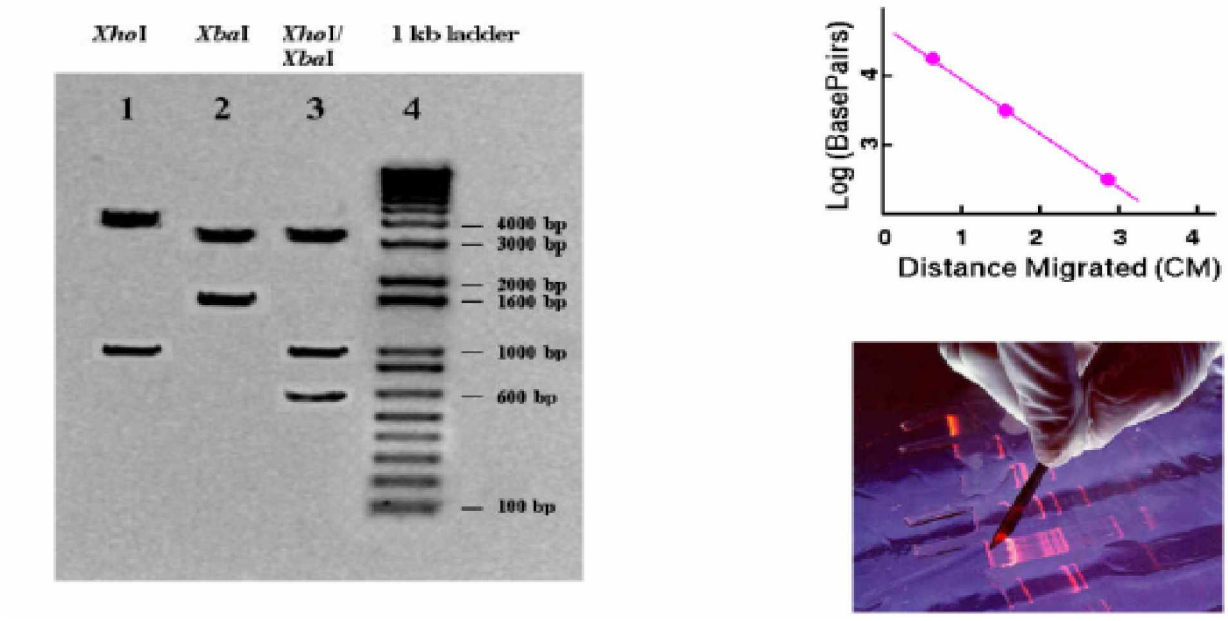

Figure 2. 11 : Gauche : Photo du résultat d'électrophorèse d'ADN marqué par fluorescence sur gel d'agarose. Les trois premières colonnes montrent le même fragment coupé par les enzymes XhoI et/ou XbaI. La dernière colonne est une solution contenant des fragments de tailles connues. Ainsi, on peut mesurer la taille et séparer différentes molécules d'ADN; droite haut: La distance de migration varie comme le logarithme de la taille de l'ADN; droite bas: on peut extraire la bande voulue du gel et en récupérer l'ADN débarrassé des fragments parasites; [81]

Pour résumer, l'électrophorèse en gel est un appareil qui nous permet de séparer un mélange de molécules d'ADN en fonction de leurs tailles. Si les pores du gel sont très petites, le pouvoir de résolution peut être *d'une seule base* : on peut séparer une séquence de 45 bases d'une séquence de 46, et c'est exactement cela qu'on utilise pour séquencer l'ADN. Le problème avec cela est qu'on ne peut pas faire cela sur des longues séquences d'ADN, disons plus que 300 bases. Cela est dû aux problèmes de piégeage dont un exemple est donné dans la figure (2.12a). En général, on n'a pas besoin d'autant de résolution et on prépare des gels avec des pores plus grands. Savoir préparer ses gels est pour le biologiste moléculaire ce que savoir préparer ses couleurs est pour le peintre.

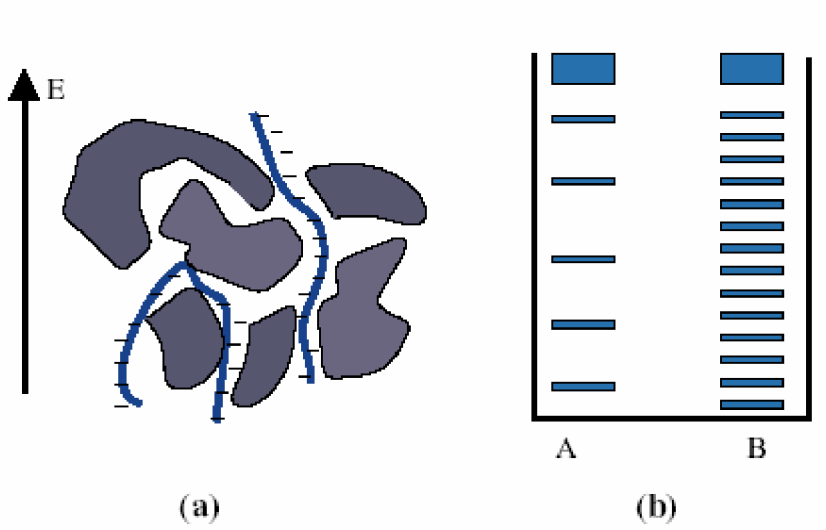

Figure 2.12: Principe de l'électrophorèse en gel. (a) le gel est montré à l'échelle de quelques nanomètres, où les chaînes d'ADN (en bleu) migrent dans le champ. En bas à gauche, on montre le cas d'une molécule piégée : les forces s'exerçant sur ses deux bouts son à peu prés égales, et elle peut rester bloquée dans cette position. (b) l'appareil de l'électrophorèse schématisé, d'une largeur d'environ 20 à 40 cm. Le haut et le bas sont branchés à des électrodes pour maintenir une différence de potentielle et créer un champ électrique à l'intérieur du gel. Au temps zéro, le mélange d'ADN (ou ARN, ou protéine) est chargé en haut (colonne A). Au bout d'un certain temps (de l'ordre de l'heure, toujours trop long de toute façon), les divers constituants migrent sous l'effet du champs, mais à des vitesses différentes, et se trouve donc à divers position du gel. Comme on ne connaît pas à priori leurs tailles, on charge toujours en parallèle (colonne B) un mélange de molécules d'ADN de longueurs connues. La colonne B constitue donc la règle qui nous permet de *lire* la taille des diverses bandes de la colonne A. Toutes ces molécules sont bien sûr transparentes, et il existe divers méthode pour les visualiser : soit on les a rendu radio-actives par avance, et il suffit alors de poser une plaque photographique sur le gel, soit on doit, après la fin de la migration, utiliser des marqueurs visibles qui se lient spécifiquement à l'ADN (ou ARN, ou...). La taille des pores est un paramètre ajustable et dépend des besoins de l'utilisateur en pouvoir de résolution [76].

2.5.3 PCR

Les outils décrit précédemment nous permettent maintenant d'achever la dernière étapes qui nous manque : l'amplification, ou l'art de copier des millions de fois un fragment d'ADN donné. La technique, appelé PCR (Polymerase Chain Reaction) est une amplification exponentielle. Elle est d'une telle simplicité que l'on s'étonne qu'il ait fallu attendre jusqu'à 1985 pour qu'elle soit découverte. Cette technique a révolutionné la biologie moléculaire.

Supposons que nous avons un fragment d'ADN double brin (ADNdb ou dsDNA en anglais) dont la séquence est connue, ou tout au moins la séquence de ses extrémités (quelques paire de base de chaque coté suffisent). Comment fabriquer une copie exacte *in vitro* ? Evidemment, nous devons utiliser l'ADN polymérase, mais l'ADN-Pol ne fonctionne que s'il y a déjà quelques séquences polymérisées derrière lui. Pour cela, on utilise des *amorces* (primers en anglais), des petites séquences de 4-5 pairs de base qui sont complémentaires aux extrémités 3' de chaque brin. Voilà comment on procède [76] :

On démarre avec une solution contenant l'ADN-Pol, beaucoup de NTP (les quatre bases, évidement) et beaucoup d'amorces, dans laquelle on balance notre fragment d'ADNdb. On chauffe alors notre solution autour de 90°C, ce qui a pour effet de séparer les deux brins. On refroidit alors la solution aux alentours de 60°C. Les amorces peuvent alors s'hybrider avec les extrémités 3' et l'ADN-Pol peut commencer son travail en polymérisant un brin tout neuf complémentaire du brin qu'il lit. Nous sommes donc maintenant en possession de *deux* ADNdb, strictement similaire à l'original avec lequel on avait commencé (Figure 2.13). Il suffit maintenant de chauffer à nouveaux à 90°C et recommencer le cycle pour avoir 4 copies et ainsi de suite. Le nombre de copie final sera 2^n, où n est le nombre de cycles.

Le PCR met grandement à profit la nécessité pour l'ADN-Pol de démarrer avec des amorces déjà hybridées. Si nous avons plusieurs fragments d'ADN différent dans la solution, seul le fragment qui correspond aux bonnes amorces sera amplifié. Par exemple, pour déterminer si un patient a été infecté par un virus ou une bactérie donné (bien avant que les symptômes de la maladie apparaissent), on purifie l'ADN issue du patient, et on amplifie avec des amorces spécifiquement préparées pour le génome du pathogène [76]. Des traces infimes de l'ADN du pathogène peuvent ainsi être détectées.

Nous avons parlé des températures de 60 à 90°C. La question qui se pose est comment la protéine ADN-Pol peut rester stable à ces températures, sans parler de son fonctionnement ? En fait, cela était la clef de la technique. Dans les versions primitives du PCR, ADN-Pol était ajouté à chaque cycle, pendant la phase basse température. Mais il existe des bactéries qui ne vivent que dans des conditions extrêmes, proches des sources chaudes au fond des océans et qui possèdent des versions thermostable de l'ADN polymérase. C'est la purification de ces polymérases (un des premiers et plus fameux s'appelle Taq-polymérase) qui a réellement lancé le PCR [76].

Le PCR a bien sûr ses limites. On ne peut amplifier des fragments de plus que quelques kilobases pour deux raisons :

a- *in vitro*, les polymérases manquent quelques mécanismes de correction d'erreur. Taq par exemple a un taux d'erreur de 10^{-4} (à comparer à10^{-9}).

b- La polymérase se détache de l'ADN au bout de quelques kilobases. *In vivo* il existe des mécanismes qui le garde accroché. Beaucoup d'efforts technologiques ont permis de pousser ces limites de plus en plus loin, mais on en est là de nos jours.

Nous avons dit également plus haut que les séquences des extrémités doivent être connues. Cela est souvent le cas, puisque les fragments d'ADN ont été produits en « coupant » un long fragment par des enzymes de restriction. Les extrémités du fragment à amplifier sont donc ceux qui sont reconnu par l'enzyme de restriction utilisé.

Enfin, notons que nous pouvons amplifier non pas tout le fragment d'ADN, mais seulement une fenêtre à l'intérieur qui nous intéresse. Il suffit pour cela que l'on utilise des amorces correspondant aux extrémités de la fenêtre en question.

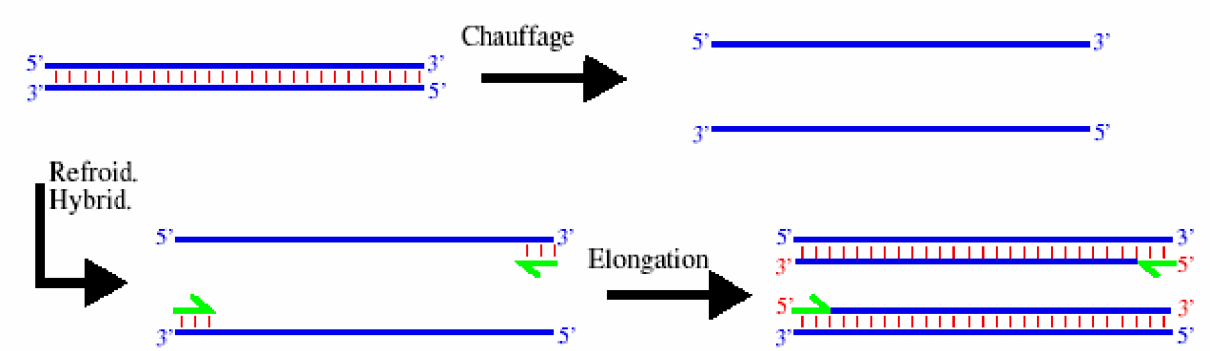

Figure 2.13: Le principe du PCR. A basse température, les deux brins du fragment d'ADN sont liés par des liaisons hydrogènes. On les sépare en les portant à haute température. En refroidissant ensuite, on permet aux amorces de s'hybrider (former des liaisons hydrogènes) avec les extrémités 3' de chaque brin. L'ADN polymérase peut alors utiliser ces amorces pour polymériser un nouveau brin, complémentaire de celui qu'il lit [76].

2.5.4 L'ADN recombinant

Comment insérer un fragment donné d'ADN au milieu d'un autre ? Par exemple, pour produire en grande quantité une protéine donnée, il suffit d'insérer la séquence d'ADN qui code pour cette protéine dans un plasmide[4]. Une bactérie munie de ce plasmide produira la protéine en question et deviendra ainsi une usine biologique. C'est également comme cela que l'on fabrique les organismes génétiquement modifiés, en incluant dans leur génome un gène intéressant provenant d'une autre espèce. Le maïs par exemple peut produire un insecticide dont le gène a été péché chez une algue. L'ADN recombinant est un outil inestimable pour la recherche. On peut par exemple fusionner le gène d'une protéine fluorescent avec le gène d'une protéine que nous sommes en train d'étudier et suivre en temps réel l'expression de cette dernière et sa localisation spatiale [76].

L'insertion d'un fragment d'ADN dans un autre s'appelle ADN recombinant. Elle a été réalisée la première fois en 1975 et a provoqué alors un vent de frayeur par le potentiel dévastateur qu'elle pouvait avoir [76]. Les diverses limites et dangers de la technique ont été depuis caractérisés et elle est courement pratiquée dans les laboratoires de biologie moléculaire à travers le monde.

Le principe de la technique est très simple. Supposons que nous voulons inclure le gène *G* d'un organisme *X* dans un fragment d'ADN *A* (un plasmide ou le génome d'un virus). On choisit d'abord un enzyme de restriction adéquat capable de couper une fenêtre incluant le gène *G*. Comme nous avons plusieurs centaines d'enzymes de restriction à notre disposition, cette étape ne pose pas de problème. Il est nécessaire que l'enzyme coupe l'ADN en laissant des « bouts collants » (voir plus haut, les enzymes de restriction). Une étape d'électrophorèse permet alors de purifier le gène *G* (Éventuellement suivi d'une étape de PCR pour l'amplifier).

On utilise ensuite le *même* enzyme de restriction pour couper l'ADN *A*. On dispose dans la solution les deux fragments issus de la coupure *avec* le fragment *G*. Puisqu'ils ont des extrémités complémentaires, le gène *G* peut s'hybrider avec les deux fragments. L'utilisation de la ligase permet ensuite de souder les jonctions (Figure 2.14). Il y a bien sûr plusieurs possibilités de recombinaison, et la bonne est sélectionnée par sa longueur lors d'une étape d'électrophorèse.

[4] Les plasmides sont de 'petits' fragments d'ADN circulaires. Ils peuvent être vus comme des jetons que les bactéries sont capables de s'échanger assez facilement et propager par exemple la résistance à un antibiotique donné.

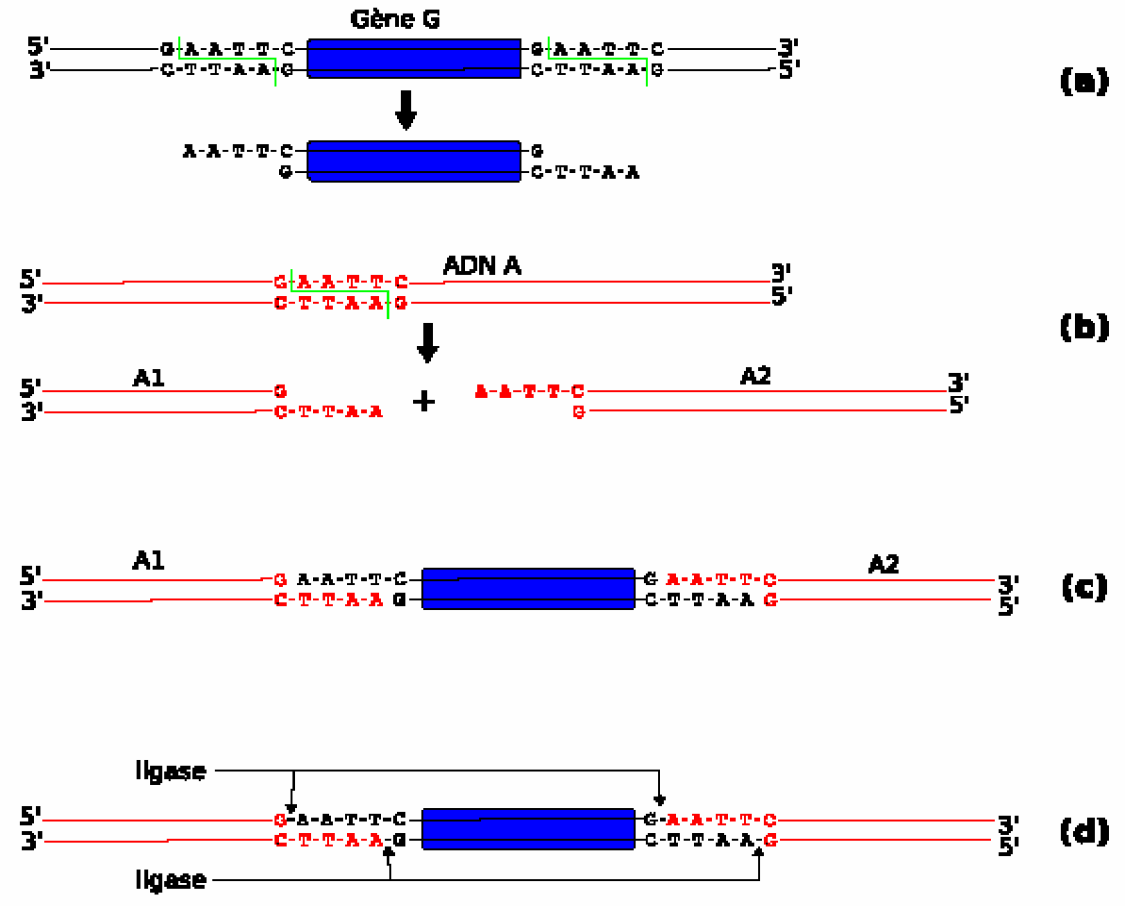

Figure 2.14: La technique d'ADN recombinant. (a) Un enzyme de restriction est choisi pour couper une fenêtre avec des extrémités collantes autour du gène intéressant *G*. (b) L'ADN *A* dans lequel on veut effectuer l'insertion est coupé avec le même enzyme de restriction. (c) Le gène *G* peut maintenant s'hybrider avec le fragment *A*1sur la gauche et le fragment *A*2 sur la droite. (d) Finalement, la ligase soude les jonctions [76].

2.5.5 Séquençage d'ADN

Comment connaître la séquence d'un fragment d'ADN donné ? De nos jours, de grand progrès sont fait dans le domaine du séquençage et le génome de beaucoup d'organisme, y compris l'humain, la souris, la mouche drosophile, la levure, la bactérie *E.Coli*, ... est entièrement décodé[76].

Le séquençage d'une molécule d'ADN consiste à déterminer l'enchaînement des bases qui le compose [81]. La technique utilisée de nos jours a été mise au point par Sanger en 1977. La version moderne de cette technique est la suivante :

Le séquençage d'ADN moderne utilise l'ADN polymérase, l'électrophorèse (comme filtre séparateur de longueur) et des nucléotides modifiés.

Les nucléotides dont ont parle sont des didesoxyriboses (voir Figure 2.15) où le groupe OH sur le carbone 3' a été remplacé par un simple H. Si un tel nucléotide est inséré dans une chaîne en cours de polymérisation, l'élongation s'arrête tout de suite (i.e : les nouveaux nucléotides sont toujours ajoutés à l'extrémité 3' d'une chaîne en cours d'élongation). De plus, on associe de petites molécules fluorescentes à ces ddNTP, et on choisit quatre couleurs différentes pour chacune des bases : par exemple ddATP en rouge, ddTTP en bleu, ddCTP en vert et ddGTP en magenta. On mélange alors ces ddNTP

fluorescents en faible proportion avec des NTP normaux dans une solution de polymérisation adéquate contenant l'ADN polymérase. On ajoute à cette solution l'ADN *simple brin* dont on veut connaître la séquence et des amorces complémentaires à son extrémité 3'. On laisse alors la réaction de polymérisation démarrer. Une fois l'amorce hybridée à l'extrémité 3', ADN-Pol prend des NTP dans la solution et les inclut à la chaîne en élongation. De temps en temps, au lieu de choisir un NTP, il choisit un ddNTP qu'il inclut dans la chaîne. La réaction pour cette chaîne en particulier s'arrête alors, l'ADN-Pol se détache et va se trouver un autre ADN à allonger. Une fois que le temps s'est suffisamment écoulé, nous disposons des chaînes de toutes les longueurs entre 1 et la longueur de l'ADN à séquencer. L'astuce est que les chaînes qui se sont arrêtées après l'inclusion d'un ddATP apparaissent en rouge, celles s'étant arrêtées après un ddTTP en bleu est ainsi de suite. Il suffit alors d'effectuer un électrophorèse pour séparer les longueurs et de les lire optiquement pour avoir la séquence (voir Figure. 2.15).

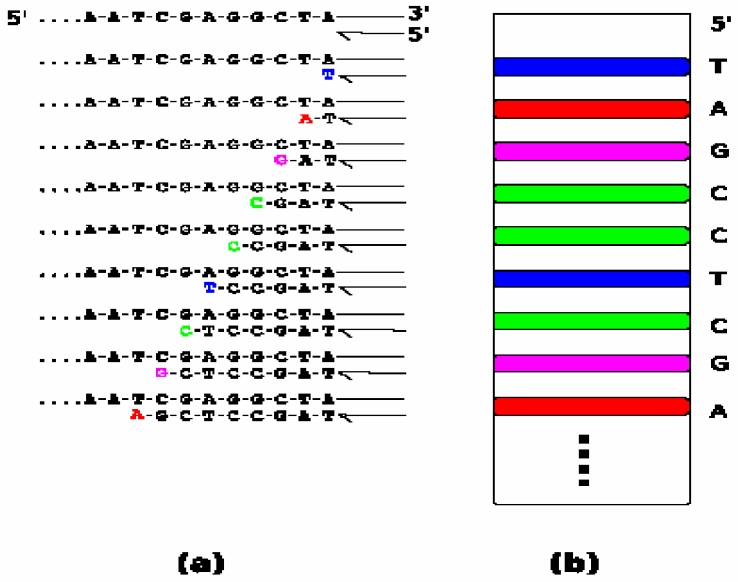

Figure 2.15: Principe du séquençage. (a) En haut est montré l'ADN simple brin à séquencer, hybridé à son amorce. A la suite de la réaction d'élongation, des chaînes de toutes les longueurs sont produites. Cela dépend de l'événement aléatoire de l'inclusion d'un ddNTP au lieu d'un NTP. (b) Ces divers fragments d'ADN peuvent être séparés par un électrophorèse en gel. Il suffit ensuite de se souvenir du code de couleur utilisé pour 'lire' la séquence dans le sens 5' à 3' [76].

2.5.6 Synthèse d'ADN

Souvent nous avons besoin de court (10-100 bases) fragment d'ADN simple brin de séquence bien déterminée. On peut synthétiser ces *oligonucleotides* chimiquement. L'astuce est d'utiliser des NTP dont le carbone 3' est protégé par un groupe chimique *amovible*. Quand ce « chapeau » est présent, la polymérisation d'ADN n'est pas possible [76].

On démarre alors avec de courtes amorces d'ADN simple brin immobilisé sur des billes. On dispose de quatre réservoirs contenant chacun un des quatre NTP protégés (qu'on notera ATP*,CTP*,...). Supposons que l'on veut synthétiser la séquence 5'-AGTCG... On ouvre d'abord le robinet du réservoir contenant les ATP*, et un A* s'inclut à la suite de l'amorce. A cause du groupe protecteur, l'élongation est arrêtée à ce niveau. On lave alors la solution en enlevant les ATP* restant dans la solution. Nous avons donc pour l'instant un 5'-A*. L'étape suivante consiste à laver la solution avec un produit qui enlève le groupe protecteur. Nous avons alors un 5'-A. Nous ouvrons alors le robinet des GTP*. Comme le A

n'est plus protégé, la synthèse avance d'un pas et nous obtenons 5'-AG*. Et on procède comme précédemment et effectue la synthèse pas à pas.

2.6 Conclusion

A travers ce chapitre, on peut conclure que :

- L'acide désoxyribonucléique (ADN) est la molécule qui contient toutes les informations génétiques. Elle se compose de deux brins complémentaires qui s'enroulent l'un sur l'autre à la manière d'une hélice.
- L'ADN est composé de quatre sous-unités (bases) différentes appelées Adénine (A), Guanine (G), Thymine (T) et Cytosine (C) qui sont liées les unes aux autres pour former une très longue chaîne. L'ordre d'apparence aléatoire des quatre bases détermine toutes les caractéristiques physiques et toutes les informations nécessaires au bon fonctionnement de la machine humaine. A l'exemple des encyclopédies, le code génétique s'inscrit lettre par lettre, avec ses mots, ses espaces, ses phrases, sa ponctuation et toutes les informations sont classées et organisées.
- L'ADN est exactement le même dans toutes les cellules de notre corps, mais unique à chaque individu (à l'exception des jumeaux identiques qui ont exactement le même profil génétique).
- Les multiples applications scientifiques et technologiques impliquant l'étude de l'ADN font appel à différentes techniques. Parmi ces techniques les plus répondues, la technique de l'électrophorèse, qui permet de séparer des molécules en fonction de leur taille et de leur charge en utilisant un courant électrique. On peut ainsi analyser et purifier dans un milieu gélifié (gel d'agarose, gel de polyacrylamide...) l'ADN, l'ARN, les protéines. La technique de PCR, l'amplification, ou l'art de copier des millions de fois un fragment d'ADN donné, est une technique extraordinaire, elle permet d'amplifier des séquences d'ADN de manière spécifique et d'augmenter de manière considérable la quantité d'ADN dont on dispose initialement. Elle nécessite de connaître la séquence des régions qui délimitent l'ADN à amplifier. Ces séquences serviront à synthétiser des amorces oligonucléotidiques complémentaires. Ces oligonucléotides serviront à délimiter la portion d'ADN à amplifier. L'ADN polymérase les utilisera comme amorces.

Le chapitre suivant va nous montrer ce que cette petite molécule peut faire dans le domaine de calcul, et comment elle peut résoudre des problèmes complexes dans un temps très réduit en utilisant quelques outils (opérations) biologiques (précédemment mentionnés).

3.1 Introduction :

Depuis des milliards d'années, les cellules végétales et animales se servent du grand pouvoir de stockage d'information de l'ADN pour effectuer toutes sortes d'opérations biologiques [82].

De tous les points de vue, elles sont plus efficaces que nos ordinateurs : elles sont exceptionnellement compactes et faciles à répliquer, et de plus elles peuvent faire des opérations en parallèle. Nos ordinateurs ne pourront jamais se mesurer à cela. En effet, ils seront toujours handicapés par une précision et une capacité de miniaturisations relativement limitées. De plus, le parallélisme que peuvent utiliser nos ordinateurs est très limité. En effet, il ne dépasse pas 4 processeurs en parallèle pour les ordinateurs individuels et de l'ordre de 5000 (en 2004) pour les plus gros [82]. Ceci signifie que nos ordinateurs ne peuvent tester que quelques solutions possibles à la fois.

Des mathématiciens se penchent donc sur la possibilité de répliquer l'élégance et la compacité des opérations effectuées dans les cellules pour un jour en arriver à créer des ordinateurs à l'ADN. Ils sont encore loin d'avoir réussi, mais leurs recherches commencent déjà à donner des résultats intéressants.

En 1994, Léonard Adleman a réussi à résoudre un problème de la théorie des graphes à l'aide des chaînes d'ADN. Son modèle de calcul à l'ADN est 1 200 000 fois plus rapide que nos ordinateurs, et peut effectuer $2 * 10^{19}$ opérations par joule (contre 10^9 pour un ordinateur conventionnel) [82]. De plus, son modèle est un très bon exemple du parallélisme des ordinateurs à l'ADN. Ce type de calcul est cependant très difficile à faire aujourd'hui, car il fait appel à plusieurs techniques biologiques mal comprises. Cependant, le succès de Adleman laisse espérer que l'ordinateur à l'ADN sera peut-être un jour une réalité.

Les recherches sur l'ordinateur à l'ADN ont aussi été un grand succès du point de vue théorique. En effet, des mathématiciens ont démontré que l'ADN pouvait trouver la solution de n'importe quel problème résoluble par une machine de Turing. En termes plus clairs, ceci veut dire que l'ordinateur à l'ADN serait capable (au moins en théorie) de résoudre n'importe quel problème exprimable par une fonction récursive [82].

Dans ce chapitre, nous commencerons par mentionner l'insertion-délétion, un type de modèle théorique de calculs avec l'ADN qui pourrait résoudre n'importe quel problème exprimable par une fonction récursive. Ensuite, nous montrerons comment Adleman a réussi à résoudre un problème de la théorie des graphes à l'aide des chaînes d'ADN.

3.2 Le langage

Avant de commencer l'étude des machines à l'ADN en tant que telle, on doit d'abord se munir de quelques outils de base. En particulier, il sera utile de se définir un langage, qui nous permettra de traduire les problèmes mathématiques que nous devrons résoudre en quelque chose de "compréhensible" en termes des chaînes d'ADN et de leurs opérations biologiques associées.

Pour ce faire, nous exposerons la théorie du langage classique, qui est à la base des sciences informatiques modernes, et nous l'adapterons graduellement au cas spécifique de l'ADN.

Au départ, on doit se munir d'un alphabet X, constitué d'un ensemble de symboles. Par exemple, dans le cas de l'ADN, l'alphabet sera constitué des quatre bases azotées. Nous aurons donc que X = {A, C, G, T}.

Cet alphabet peut sembler restrictif, mais rappelons que les ordinateurs conventionnels ne disposent que d'un alphabet de 0 et de 1.

On construit alors des chaînes avec les symboles de cet alphabet, et on définit X* comme l'ensemble des chaînes possibles. Si on considère λ comme la chaîne "nulle" (ne contenant aucun symbole), on peut construire chacune des chaînes possibles par la concaténation, que l'on définit par récursivité :

1. $\lambda \in X^*$
2. Si $a \in X$ et $c \in X^* \Rightarrow ca \in X^*$

Notons que l'étape 2 ne peut être appliquée qu'un nombre fini de fois à partir de λ, ce qui exclut les chaînes infinies.

Définition1 [82] : Pour $u \in X^*$, on dit que v est une sous-chaîne de u si $\exists x, y \in X^*$ tels que u = xvy. Si x = λ, v est un préfixe de u et si y = λ, v est un suffixe de u.

Dans le cas de l'ADN, X^* représente donc l'ensemble des chaînes simples d'ADN de longueur finie pouvant être construites avec les quatre bases azotées ainsi que la chaîne vide.

En principe, on pourrait définir un *langage* comme tout sous-ensemble de X^*. Cependant, ce n'est pas une définition très utilisée, car un langage complètement aléatoire n'est pas assez restrictif pour être efficace. En général, on préfère n'admettre que certaines chaînes répondant à un ensemble de critères précis.

C'est d'ailleurs le concept qu'on utilise intuitivement avec la langue française. En effet, on considère que notre alphabet est composé des 26 lettres minuscules, de leurs 26 majuscules, de l'espace blanc et des signes de ponctuation. Le langage lui-même est constitué de toutes les chaînes qui constituent des phrases complètes. Pour ce faire, il faut donner certaines restrictions au concept de phrase complète (c'est ce qu'on appelle une *grammaire*). En effet, on ne voudrait pas que la chaîne *afgbd fJhrbnd hdgfb*, soit un élément de notre langage. Il faut donc programmer dans la grammaire toutes les règles de la langue française : toute phrase débute par une majuscule et se termine par un point, on ne peut avoir qu'un mot défini dans le dictionnaire entre deux espaces blancs, le sujet vient avant le verbe, etc. ...

Pour l'ordinateur à ADN, nous devrons aussi définir un langage muni d'une grammaire restrictive qui corresponde à nos besoins. En effet, dans un ordinateur à l'ADN, on introduit différentes chaînes d'ADN dans une éprouvette (test-tube), et les calculs se font par une série d'opérations biologiques opérées le plus souvent par des enzymes. On veut donc un langage qui ne générera que les chaînes pouvant être créées par ces opérations.

Un des modèles les plus connus de calculs par ordinateur à ADN est celui de l'insertion-délétion [82]. L'idée est d'utiliser des enzymes pour

- couper une sous-chaîne d'ADN à un endroit précis (délétion)
- insérer une sous chaîne dans une autre à un endroit prescrit (insertion).

Mais comment traduire ces opérations biologiques en termes de langage et de grammaire ? Nous avons déjà donné une idée intuitive de ce qu'est une grammaire ci-haut, mais nous poursuivrons en donnant la définition plus rigoureuse, telle qu'elle est utilisée en informatique théorique. Nous pourrons ensuite l'adapter pour modéliser l'insertion-délétion.

Définition2 [82] : Une grammaire générative est un triple ordonné G = (X, A, P) où X est l'alphabet, $A \in X^*$ est l'axiome et P sont des lois de production (ou tout simplement productions).

Intuitivement, l'axiome représente la chaîne de départ. Dans le cas de l'ADN, l'axiome représente la chaîne introduite au départ dans l'éprouvette avant le début des calculs.

Quant à elles, les productions représentent toutes les modifications que l'on peut faire à la chaîne de départ pour générer toutes les chaînes du langage.

Exemple1 [82] : Dans le modèle d'insertion-délétion qui nous intéresse, nous devrons avoir deux productions : une qui modélise l'insertion et l'autre, la délétion.

Voyons maintenant comment on peut décrire l'insertion et la délétion de manière plus rigoureuse.

Définition3 [82] : Définition rigoureuse de l'insertion-délétion :

1. Si $x = x_1 x_2$ *est un mot dans* X^**, on peut insérer une chaîne* $u \in X^*$ *entre* x_1 *et* x_2*, ce qui donne le mot* $y = x_1 u x_2$*. On écrit alors que* $x \Rightarrow y$ *et on dit que x* **dérive** *y par la loi de production de l'insertion.*

2. Si, au contraire, $x = x_1 u x_2$ *est un mot dans* X^**, on peut retrancher la chaîne u, ce qui donne* $y = x_1 x_2$*. On écrit encore une fois* $x \Rightarrow y$ *et on dit que x* **dérive** *y par la loi de production de délétion.*

On voit donc que $x \Rightarrow y$ est la notation générale pour dire que *y* a été obtenu de *x* par une des lois de production. Si *y* a été obtenu à la suite de plusieurs lois de production appliquées une après l'autre à *x*, on utilise la notation $x \Rightarrow * y$.

Définition4 [82] : Etant donné une grammaire G, on lui associe un langage L (G) avec L (G)=$\{ c \in T^* \mid A \Rightarrow * c \}$.

Il reste maintenant à adapter la définition de grammaire afin que les productions modélisent bien les opérations d'insertion et de délétion. La grammaire sera donc le quadruplet ID = (X, I, D, A).

Ici, l'alphabet X = {A, G, T, C} contient les quatre bases azotées. $A \neq \lambda$ représente l'axiome de départ (la chaîne introduite dans l'éprouvette avant le début des opérations). On aura aussi $P = I \cup D$ où I représente les productions de l'insertion et D, celles de la délétion.

Chaque loi d'insertion est représentée par un triplet $(c_1, x, c_2)_I$, où $x \in X^*$ est la chaîne à insérer et $c_1, c_2 \in X^*$ représentent le contexte d'insertion. On dit donc que $I \subseteq (X^*)^3$.

De même, chaque loi de délétion est représentée par un triplet $(c_1, x, c_2)_D$, où $x \in X^*$ est la chaîne à retrancher et $c_1, c_2 \in X^*$ sont le contexte de délétion. L'insertion et la délétion sont représentées dans la Figure 3.1.

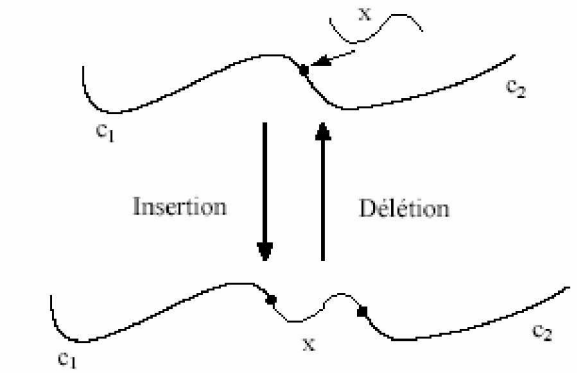

Figure 3.1: Schématisation de l'insertion et de la délétion [1]

Théoriquement, ce modèle est très efficace. En effet, nous prouverons plus loin qu'on peut calculer n'importe quelle fonction récursive et résoudre n'importe quel problème exprimable récursivement en utilisant l'insertion-délétion. Cependant, il est souvent assez difficile de trouver un algorithme pour résoudre des problèmes mathématiques à l'aide d'une suite d'insertions et de délétions.

Pensons par exemple à la multiplication. C'est une fonction toute simple, et pourtant il est difficile d'imaginer un algorithme court qui permettrait de l'effectuer simplement avec l'insertion et la délétion de chaînes d'ADN. En effet, nous savons que c'est possible de trouver un algorithme pour la multiplication et que c'est toujours le cas pour n'importe quelle fonction récursive. Cependant, rien ne nous garantit que cet algorithme sera "beau".

Il y a aussi un problème d'un autre ordre avec ce modèle théorique. En effet, nous avons supposé qu'il était possible de faire n'importe quelle insertion et n'importe quelle délétion. En d'autres mots, ceci veut dire que nous connaissons tous les enzymes et qu'on peut en placer un nombre illimité dans une éprouvette, où ils agiront sans interférence les uns avec les autres.

Or, on sait très bien que, dans la réalité, notre maîtrise de la biologie est très imparfaite et que nous ne comprenons pas assez bien l'action des enzymes pour pouvoir opérer n'importe quelle insertion ou n'importe quelle délétion permise par la théorie.

3.3 Les machines de Turing et les systèmes d'insertion-délétion

Nous établirons maintenant que, avec le système d'insertion-délétion, nous pouvons effectuer n'importe quel calcul opérable par une machine de Turing. Nous ferons une brève définition de la machine de Turing.

En fait, une machine de Turing est un outil théorique capable de faire n'importe quel calcul exprimable par une fonction récursive. On la représente comme un ruban partant de la gauche et s'étendant vers l'infini, sur lequel on peut écrire plusieurs symboles venant d'un alphabet $\{ X \cup \# \}$, où le # représente la fin des entrées sur le ruban. On a aussi un alphabet d'état Q. Les instructions sont de la forme $(q_1, x_1) \rightarrow (q_2, x_2, c)$ avec $x_i \in X, q_i \in Q$ et $c \in \{ 0, -1, 1 \}$ et s'interprètent comme suit : Si le pointeur est sur une case avec le symbole x_1 et la machine est dans l'état q_1 , alors la machine remplace x_1 par x_2 , change dans l'état q_2 et le pointeur bouge de c cases vers la droite (si c = -1 le pointeur bouge d'une case vers la gauche.)

Pour l'instant, la machine de Turing n'est sûrement pas très facile à comprendre. On peut cependant y arriver avec un exemple simple.

Exemple2 [82]: Prenons ici le cas de l'addition des deux nombres 2 et 3. Nous nous munirons de l'alphabet X = {1, B, #} et nous représenterons chaque nombre entier x par 1^{x+1}, ce qui veut dire que 2, par exemple, serait représenté par 111. Le symbole B représentera un blanc, utile pour séparer les deux nombres sur le ruban.

Au départ, le ruban de notre machine de Turing ressemblera à ceci : 111B1111#

Ici, on reconnaît les nombres 2 (111) et 3 (1111) séparés par un blanc. Le premier état de cette machine de Turing sera de se déplacer vers la droite et de remplacer chaque lettre par un 1, jusqu'à l'encontre du #. On obtient alors : 11111111#

Ensuite, à partir du #, la machine passe à un nouvel état : elle doit maintenant reculer à gauche de deux espaces et remplacer chacun des 1 qu'elle rencontre par un B, ce qui donne : 111111BB#

On passe finalement à l'état final à revenant vers la gauche jusqu'à avoir atteint le tout premier 1.

Et voilà, nous avons vu comment effectuer une addition avec la machine de Turing. Ceci nous donne une idée intuitive de ce qu'est une machine de Turing mais, avant de pouvoir faire la preuve que toutes les opérations qu'elle peut effectuer sont opérables par une machine à l'ADN, il faut d'abord donner une définition plus rigoureuse de la machine de Turing.

Définition5 [82] : Une machine de Turing est un triplet M = (Q, X,φ), où Q est l'alphabet d'état, X est l'alphabet et φ représente l'ensemble des opérations permises. Comme on l'a vu plus haut, ces opérations représentent l'ensemble du remplacement d'un caractère par un autre, du changement d'état et du choix entre un mouvement vers la droite ou vers la gauche, ou encore rester sur place.

Introduisons maintenant la définition d'un langage accepté par une machine de Turing.

Définition6 [1] : Soit M = (Q, X, φ) une machine de Turing. Une chaîne $\omega \in X^*$ *est acceptée par M si les calculs de M avec ω comme entrée se terminent dans un état final* $q_f \in Q_f$ *. Un calcul qui se termine abruptement, i.e. qui se termine dans un état* $q_i \notin Q_f$ *, rejette l'entrée. Ainsi, le langage accepté par M, noté L (M), est l'ensemble de toutes les chaînes acceptées par M. Plus rigoureusement, on écrit L (M) =* $\{\omega \in X^* \mid q_0\omega \Rightarrow *\omega_1 q_f \omega_2\omega_3\}$ *où* $q_f \in S_f, \omega_2 \in X, \omega_1, \omega_3 \in X^*$ *sont tels qu'il n'y ait pas de règle de transition dans ω pour la configuration* (q_f, ω_2)*. Notons ici que* $\omega_1 q_f \omega_2\omega_3$ *représente la configuration du ruban* $B\omega_1\omega_2\omega_3 B$ *pour laquelle le pointeur est dans l'état* q_f *sur la case avec* ω_2*.*

Exemple 3 [82]:Soit f une fonction totale de N dans N et calculable avec une machine de Turing M qui prend comme entrée la représentation unaire d'un nombre, L(M) =$\{\overline{x} : x \in N\}$, où $\overline{x}$ est la représentation unaire de x. Ici, ω est la configuration initiale du ruban, i.e. $B\overline{x}B$; puisque le pointeur doit être sur le symbole blanc qui précède le résultat du calcul de la machine $\omega_1 = \lambda$ et $\omega_2 = B$; finalement, $\omega_3 = \overline{y}B....$, où $y = f(x) \in N$.

Théorème1 [82] [3] : Si un langage est accepté par une machine de Turing, alors il existe un système d'insertion-délétion qui accepte le même langage.

Preuve [1]: Nous n'allons pas faire la preuve rigoureuse du théorème. Nous allons plutôt expliquer l'idée générale de la preuve et développer certaines parties plus intéressantes à nos yeux.

Soit M = (Q, X,φ) une machine de Turing. Une règle de transition de φ est de la forme $(q_i, x_i) \to (q_i, x_j, c)$, où $c \in (-1,0,1)$. Séparons cette règle selon que c = 0, c = 1, c = -1. Il faut vérifier que chacune de ces trois règles de transition possède un équivalent dans un système d'insertion-délétion ID = (N, I, D, A) pour lequel $N = X \cup Q \cup \{ L, R, O \} \cup \{ q_{i1} : q_i \in Q \} \cup \#$. L'ensemble $\{ q_{i1} : q_i \in Q \}$ est rajouté à l'alphabet N, car pour un certain nombre de q_i, nous aurons besoin d'un nouvel état intermédiaire que nous noterons q_{i1} . Dans l'ordinateur à l'ADN R, L et O sont des chaînes particulières, donc des éléments de X^*.

Rappelons qu'un langage accepté par un système d'insertion-délétion ID est défini comme suit : L (ID) = $\{\nu \in N^* \mid \nu \Rightarrow^* A\}$ où A est l'axiome.

Nous utiliserons diverses chaînes de caractères durant la preuve. Notons que $\mu\mu, \mu_1, \nu, x_i, x_j, \mu_2 \in X$ et $q_i, q_j \in Q$.

1. Pour toutes les règles de la forme $(q_i, x_i) \to (q_j, x_j, 0)$ nous devons ajouter à ID les trois règles suivantes : $(q_i x_i, q_j O x_j, \nu)_I$, $(\mu, q_i x_i, q_j O x_j)_D$, $(p\sigma q_i, O, x_j)_D$, où μ, ν, p, σ sont des caractères quelconques dans X. En fait, pour chaque caractère v dans X, il faut ajouter dans ID une règle de la forme $(q_i x_i, q_j O x_j, \nu)_I$, et de même pour les deux autres règles. Puisque la cardinalité de X est fini, nous avons un nombre fini de règles à ajouter dans ID et cela ne pose dons aucun problème. Ainsi, si nous avons une chaîne de la forme $\mu q_i x_i \nu$ nous allons avoir la suite d'opérations suivantes : $\mu q_i x_i \nu \Rightarrow \mu q_i x_i q_j O x_j \nu \Rightarrow \mu q_j O x_j \nu \Rightarrow \mu q_j x_j \nu$.

Décrivons les opérations précédentes. Au départ, nous avons la chaîne $\mu q_i x_i \nu$. Nous commençons par insérer la chaîne $q_j O x_j$ entre x_i v. Suivent alors deux délétions : la première permettant d'enlever $q_i x_i$ et la seconde enlevant le O restant compris entre q_j et x_j, permettant alors de se retrouver avec $\mu q_j x_j \nu$ qui est la chaîne que nous voulions obtenir. Rappelons que le caractère représentant l'état du pointeur, i.e. q_j, précède le caractère sur lequel le pointeur se trouve. Nous voyons alors que les opérations précédentes nous ont permis de passer de la configuration μxi_ dans l'état qi à la configuration $\mu x_j \nu$ dans l'état q_j.

2. Pour toutes les règles de la forme $(q_i, x_i) \to (q_j, x_j, 1)$ nous devons ajouter à ID les six règles suivantes :
$(q_i x_i, q_{i1} O x_j, \nu)_I, (\mu, q_i x_i, q_{i1}, O x_j)_D, (p\sigma q_{i1}, O, x_j)_D, (q_{i1} x_j, q_j R, \nu)_I, (\mu, q_{i1}, x_j q_j R)_D, (\tau x_j q_j, R, \nu)_D$
, où τ est un caractère quelconque dans X.
Ainsi, si nous avons une chaîne de la forme $\mu q_i x_i \nu$ nous allons avoir la suite d'opérations suivantes :

$\mu q_i x_i \nu \Rightarrow \mu q_i x_i q_{i1} O x_j \nu \Rightarrow \mu q_{i1} O x_j \nu \Rightarrow \mu q_{i1} x_j \nu \Rightarrow \mu q_{i1} x_j q_j R \nu \Rightarrow \mu x_j q_j R \nu \Rightarrow \mu x_j q_j \nu$.

On voit ici que les trois premières opérations sont une répétition de celles présentées pour les règles de la forme $(q_i, x_(\,) \to (q_j, x_j, 0)$. Ces trois insertion et délétions permettent en effet de changer x_i pour x_j sans déplacer la position du pointeur. On utilise un état artificiel q_{i1} pour signifier qu'on n'a pas terminé l'exécution de la commande de la machine de Turing. Les trois opérations suivantes permettent de déplacer le pointeur vers la droite et d'amener celui-ci dans l'état q_j désiré. La machine est alors en position d'effectuer une commande $(q_j, \nu) \to (q_k, x_k, c)$ avec $c \in \{-1, 0, 1\}$, si une telle commande existe.

3. Pour toutes les règles de la forme $(q_i, x_i) \to (q_j, x_j, -1)$ nous devons ajouter à ID les six règles suivantes :
$(q_i x_i, q_{i1} O x_j, \nu)_I, (\mu_2, q_i x_i, q_{i1} O x_j)_D, (p\sigma q_{i1}, O, x_j)_D, (\mu_1, q_j L, \mu_2 q_i, x_j)_I, (q_j L \mu_2, q_{i1}, x_j)_D, (q_j, L, \mu_2 x_j)_D$
Ainsi, si nous avons une chaîne de la forme μqixi_ nous allons avoir la suite d'opérations suivantes :

$\mu_1\mu_2 q_i x_i \nu \Rightarrow \mu_1\mu_2 q_i x_i q_{i1} O x_j \nu \Rightarrow \mu_1\mu_2 q_{i1} O x_j \nu \Rightarrow \mu_1\mu_2 q_{i1} x_j \nu \Rightarrow \mu_1 q_j L \mu_2 q_{i1} x_j \nu \Rightarrow \mu_1 q_j L \mu_2 x_j \nu$

Ce théorème montre qu'un système d'insertion-délétion a le même pouvoir de calcul qu'une machine de Turing. Ainsi, tout ce qui est calculable est calculable avec un système

d'insertion-délétion et, par conséquent, avec un ordinateur à l'ADN. Nous voyons ici toute la puissance de calcul que pourrait avoir un ordinateur à l'ADN.

3.4 Les machines à l'ADN concrètes

On a vu qu'en théorie, les machines à l'ADN peuvent, à l'aide des seules opérations d'insertion et de délétion, effectuer tous les calculs qu'on peut effectuer avec une machine de Turing. Seulement, la réalité biologique est beaucoup plus complexe. On est encore loin de pouvoir manipuler assez bien les enzymes en laboratoire pour effectuer n'importe quelle opération d'insertion ou de délétion [82].

De plus, même si on parvient un jour à maîtriser parfaitement bien toutes les techniques biologiques requises, il faut savoir que les opérations sur les chaînes d'ADN ont un haut taux d'erreur naturel, ce qui est très difficile (voire impossible) à prévoir et à contrôler. Pour contrer ce problème, certains ont suggéré de faire les opérations à l'ADN in vivo (à l'intérieur de cellules vivantes) puisque les cellules sont munies d'un dispositif de contrôle des erreurs [82]. La recherche est en cours.

3.4.1 L'expérience d'Adleman

3.4.1.1 Le problème du chemin hamiltonien (HPP : Hamiltonian path problem)

Même si bien des gens doutent qu'on pourra un jour bâtir un ordinateur à l'ADN viable, certains calculs simples ont déjà été effectués avec des chaînes d'ADN. En 1994, Léonard Adleman [84] a réussi à résoudre un problème concret avec l'ADN.

Le problème avait pour point de départ un graphe dirigé, comme celui qui est représenté dans la Figure 3.2. En fait, un graphe dirigé est un ensemble de sommets (ici numérotés de 1 à 7) qui sont reliés par des arêtes dirigées représentées par des flèches partant du sommet de départ et pointant vers le sommet d'arrivée.

Dans ce graphe, il s'agit de trouver un chemin qui part du début (sommet 1) et se rend vers le sommet de la fin (sommet 7), en passant par chaque sommet une et une seule fois. C'est ce qu'on appelle le problème du chemin hamiltonien.

Si on prend le graphe de la Figure 3.2, le problème se fait très facilement à la main. En effet, la solution est de passer par les sept sommets dans l'ordre suivant : 1 à 5 à 2 à 3 à 4 à 6 à 7. C'est encore plus simple avec un ordinateur conventionnel : même avec un algorithme rudimentaire le calcul prend une fraction de seconde.

En revanche, Adleman a passé sept jours en laboratoire pour arriver au même résultat. Quel est alors l'avantage d'utiliser un ordinateur à l'ADN ? C'est que le problème du chemin hamiltonien est un problème complexe : même s'il est facile de programmer un algorithme, en général le temps pour effectuer l'algorithme devient si grand dés que le nombre de sommets augmente qu'aucun ordinateur ne peut trouver la solution en un temps raisonnable (en termes techniques, on dit que le problème est NP-complet). Déjà, à 100 sommets, un ordinateur prend un temps beaucoup trop élevé. Ceci vient du fait qu'un ordinateur conventionnel fait ses opérations séquentiellement (l'une à la suite de l'autre), ce qui constitue sa limite quand le nombre d'opérations devient grand. Par contre, avec l'ADN on peut faire des milliards d'opérations en parallèle. C'est ce qui fait sa force. La partie plus longue de l'exécution d'un algorithme par un ordinateur à l'ADN est la suite d'opérations de laboratoire qui doivent être effectuées par un être humain. Mais, pour un grand graphe, ceci devient tout de même avantageux par rapport aux ordinateurs conventionnels.

Avec les calculs à l'ADN, le temps de résolution du problème croit de façon linéaire par rapport à la taille du graphe.

Notons que Adleman n'a pas résolu le problème avec la méthode d'insertion-délétion présentée précédemment. En effet, cette méthode, bien que théoriquement élégante, est difficile à appliquer dans une éprouvette. De plus, les algorithmes donnés par l'insertion-délétion pour résoudre de "vrais problèmes "comportent souvent beaucoup trop d'étapes. En général on a avantage à simplifier ces algorithmes avec des astuces plus efficaces.

3.4.1.2 Algorithme d'Adleman:

Le problème résolu est celui d'un graphe hamiltonien : soit donné *N* noeuds liés par des liens *directionnelles* (ce qui veut dire que *i* à *j* et *j* à *i* ne sont pas équivalent). Existe-t-il un chemin menant du noeud 1 au noeud *N* en passant par les autres noeuds *une et seulement une* fois ? La figure (3.2a) est un exemple de tels graphes. Le problème posé est classé dans la catégorie *NP* au niveau de son coût en terme de nombre d'opérations. Cela veut dire que si le nombre de noeud est *N*, nous ne connaissons pas d'algorithme capable de résoudre ce problème en *N* ou N^2 ou n'importe quelle autre puissance de *N* opérations. Par contre, nous savons vérifier en un temps polynomial si un chemin quelconque est solution ou non. Ce problème peut en principe être résolu par l'algorithme suivant (qui n'est pas le plus efficace si on programmait un ordinateur classique):

1. Générer *tous* les chemins, étant donnés les noeuds et les liens.
2. Sélectionner seulement les chemins qui commencent par 1 et finissent par 7 (dans le cas de la figure 3.2a)
3. Sélectionner ceux qui ont une longueur exactement de 6 (7-1).
4. Sélectionner ceux qui sont passés par chaque noeud au moins une fois.
5. Si à la fin de ce processus de sélection, il reste encore des chemins, la réponse est oui.

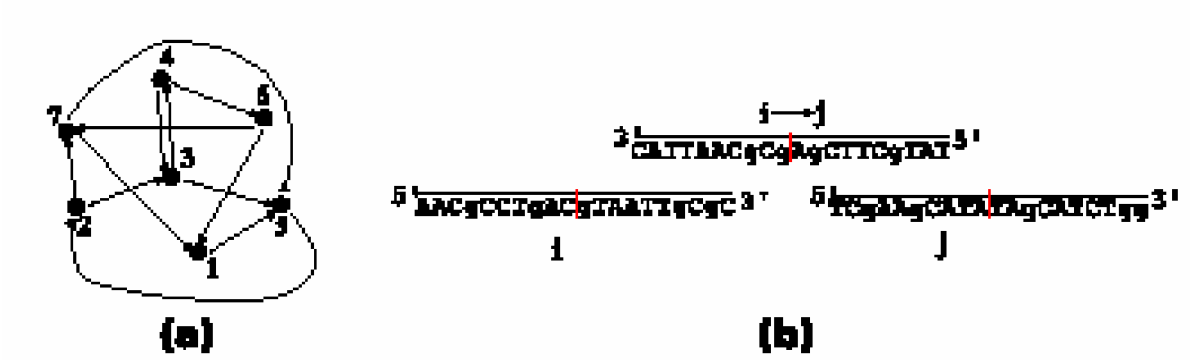

Figure 3.2 : (a) : Existe-t-il, dans ce graphe, un chemin menant de 1 à 7 en passant une et une seule fois par tous les noeuds ? La réponse est ici oui : 1 à 5 à 2 à 3 à 4 à 6 à 7. (b) : Dans l'expérience d'Adleman, chaque noeud est représenté par une séquence unique de 20-mere d'ADN. Si il existe un lien du noeud *i* vers le noeud *j*, alors ce lien est également représenté par une séquence de 20-mere. La séquence du lien est choisie pour être pour moitié complémentaire à *i* et pour moitié à *j*. Notez que comme l'ADN est un polymère avec une direction (un coté 5' et un coté 3'), les liens *i* à *j* et *j* à *i* sont représentés par deux séquences distinctes[76].

Pour implémenter cet algorithme, Adleman a utilisé des séquences de 20 bases (des 20-meres) d'ADN simple brin. Chaque noeud est représenté par une séquence unique. Si un lien de *i* à *j* existe, alors un autre 20-mere est ajouté à la sauce, qui est complémentaire pour moitié à *i* et pour moitié à *j*. L'astuce de l'expérience est d'utiliser le fait que l'ADN est un

polymère dirigé, avec un coté 5' et un coté 3'. En se référant à la figure (3.2b), on se rend vite compte que le lien *i* à *j* et *j* à *i* ont des séquences différentes. La dernière idée d'Adleman était pour les liens qui impliquent le premier ou le dernier noeud. Le lien 1 à *j*, si il existe, est représenté par un 30-meres, dont les 20 bases coté 3' sont complémentaire à la totalité de vingt bases de 1 ; les dix autres restantes sont complémentaires à la moitié de *j*, comme précédemment. De même, si le lien *i* à 7 existe, il est représenté par un 30-meres dont les 20 bases (cotés 5') sont complémentaire à la totalité de 7. Passons maintenant à l'implémentation des étapes 1 à 5.

Etape 1: C'est l'étape la plus facile, et celle qui fait pratiquement toute la puissance de la méthode. On mélange tous les noeuds et tous les liens ; on laisse gentiment les morceaux complémentaires s'hybrider ; on ajoute des ligases dans le milieu pour souder les liens pendants [76]. La figure 3.3 illustre ce processus pour deux noeuds et leurs liens. A la fin de cette étape, les chemins possibles sont représentés par des polymères *doubles brins* d'ADN dans la solution. Notez également que seul les chemins qui commencent par 1 et se terminent par 7 sont entièrement doubles brins. Les autres auront toujours, d'un coté ou de l'autre, une séquence de 10 bases simple brin.

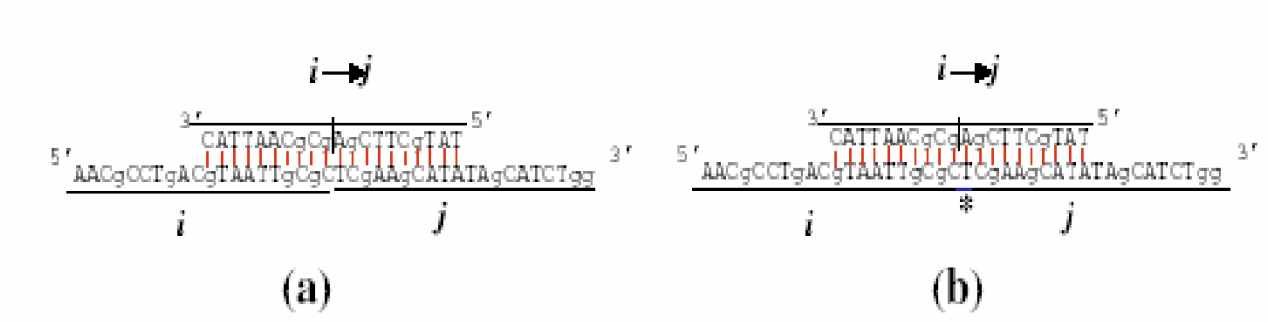

Figure 3.3: (a) étape d'hybridation : le lien s'hybride aux deux morceaux complémentaires des noeuds *i* et *j*, en rapprochant le coté 3' de l'un au coté 5' de l'autre ; (b) la ligase soude la liaison covalente entre la dernière base de *i* et la première base de *j*. La position de la soudure est marquée par une *[76].

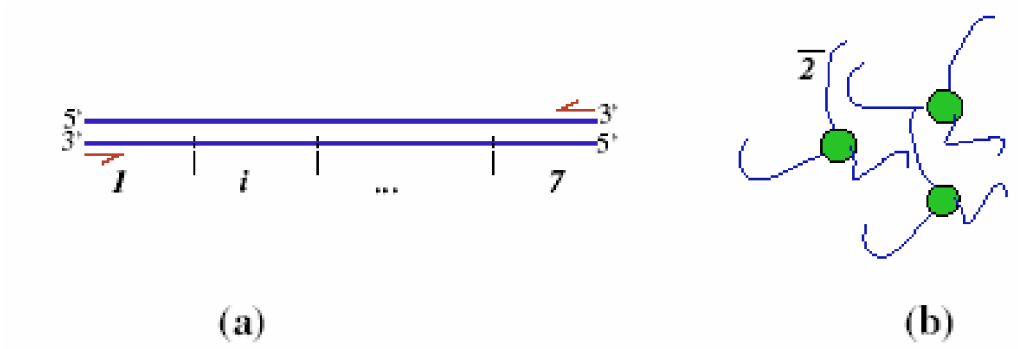

Figure 3.4: (a) Choix des amorces de PCR pour la sélection des chemins commençant par 1 et finissant par 7 ; (b) des séquences complémentaires au noeud 2 sont greffées sur des billes : leur hybridation avec des ‘‘2’’ permet de sélectionner les chemins qui contiennent ces derniers[76].

Etape 2: Pour sélectionner les chemins qui commencent par 1 et finissent par 7, on utilise le PCR, en donnant des amorces complémentaires au début de 1 et de $\overline{7}$ (figure 3.4a) [76].

Comme nous l'avons vu au chapitre sur le PCR, seul ces chemins seront amplifiés exponentiellement.
Etape 3: La sélection des chemins qui ont exactement 6 liens, et donc 120 bases, se fait par un simple électrophorèse : nous avons vu au chapitre précédent que cet outil permet essentiellement de distinguer les longueurs. La sélection des chemins à 120 bases peut être suivi d'un PCR pour une amplification supplémentaire.

Etape 4: C'est l'étape la plus fastidieuse. Pour sélectionner, dans tous les chemins qui nous restent, ceux qui contiennent le noeud 2, on greffe des simples brins $\overline{2}$ sur des billes micrométriques (figure 3.4b). On ajoute l'ensemble des chemins dont on dispose ; on chauffe pour séparer les doubles brins ; on laisse refroidir pour permettre aux brins complémentaires de s'hybrider à nouveau. Certains des chemins qui contiennent le noeud 2 vont alors s'hybrider avec les brins $\overline{2}$ sur les billes. On lave ensuite la solution pour enlever tous les ADN qui ne se sont pas hybridés. Finalement, on chauffe à nouveau la solution pour séparer les doubles brins et on récolte l'ADN en solution. Ces ADN contiennent sûrement la séquence du noeud 2. On procède à un PCR pour amplifier le signal. Et on continue de la même manière pour tester la présence des noeuds 3,4,... dans les chemins restants.

Etape 5 [76] : Finalement, à la fin de tous ces processus, il suffit de rouler un gel pour voir si il reste de l'ADN dans la solution. Si oui, la réponse à la question posé est ''oui, il existe des chemins allant de 1 à 7 en passant une et une seule fois par chaque nœud''.

On peut résumer l'algorithme d'Adleman, dans le tableau 1, comme suit :

	Algorithme	Réalisation	Opération
Etape1	A partir du graphe, générer aléatoirement tous les chemins possibles.	Mélanger tous les chemins binaires (entre 2 nœud) et tous les nœuds à l'exception ceux de départ et d'arrivée dans une éprouvette.	Ligation & hybridation
Etape2	Garder seulement les chemins qui commencent par le nœud d'entrée (de début) spécifié et celui de sortie (de fin).	Utilisation de la technique de PCR en donnant des amorces complémentaires au début de nœud d'entrée et de complément de nœud de sortie.	PCR
Etape3	Si le graphe a *n* nœuds, alors garder seulement les chemins qui ont exactement *n* nœuds.	employer le gel électrophorèse pour identifier et garder les molécules qui ont une longueur correcte, et éliminer les autres molécules.	électrophorèse
Etape4	Garder seulement les	Extraire toutes les	séparation d'affinité

	chemins qui ont l'apparition de chaque nœud exactement une seule fois.	séquences contenant les différents *i nœud* du graphe, tel que *i=1 : n*	'extraction'
Etape5	Après les étapes précédentes, s'il reste un chemin, alors ce chemin représente le chemin hamiltonien, sinon aucun chemin hamiltonien existe.		Séquençage

Tableau 3.1 : Algorithme d'Adleman.

3.4.1.3 Exemple d'application de l'algorithme d'Adleman :

Pour appliquer l'algorithme d'Adleman, on va prendre un simple exemple. Notre exemple est un cas du problème de chemin hamiltonien.

Considérant une carte de quatre villes connectées par des certains voyages sans arrêt (figure 3.5). Dans cet exemple, il est possible de voyager directement de Batna à Setif, mais non vice versa.

Le but est de déterminer s'il existe un chemin qui commence par la ville de départ (Batna), et qui fini par la ville d'arrivée (Setif) en passant par chacune des villes restantes exactement une seule fois.

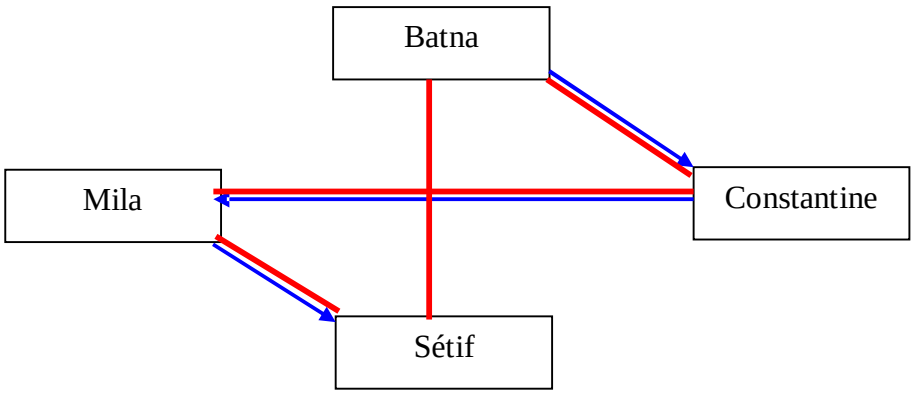

Figure 3.5 : Graphe de 4 villes reliées par des arcs. Le chemin hamiltonien est représenté en bleu, et les autres chemins en rouge.

Etape 1 : Génération de tous les chemins aléatoires possibles

- **Codage des nœuds et d'arcs du graphe par des séquences d'ADN**

Dans le calcul d'ADN, Adleman a pensé d'assigner chaque ville par une séquence d'ADN (par exemple, ATCGTCGA pour Batna) qui peut être considérée comme un prénom (ATCG) suivi par un nom de famille (TCGA) de la ville. Les numéros de voyages d'ADN peuvent alors être définis en enchaînant le nom de famille de la ville d'origine avec le prénom de la destination comme le montre la figure suivante :

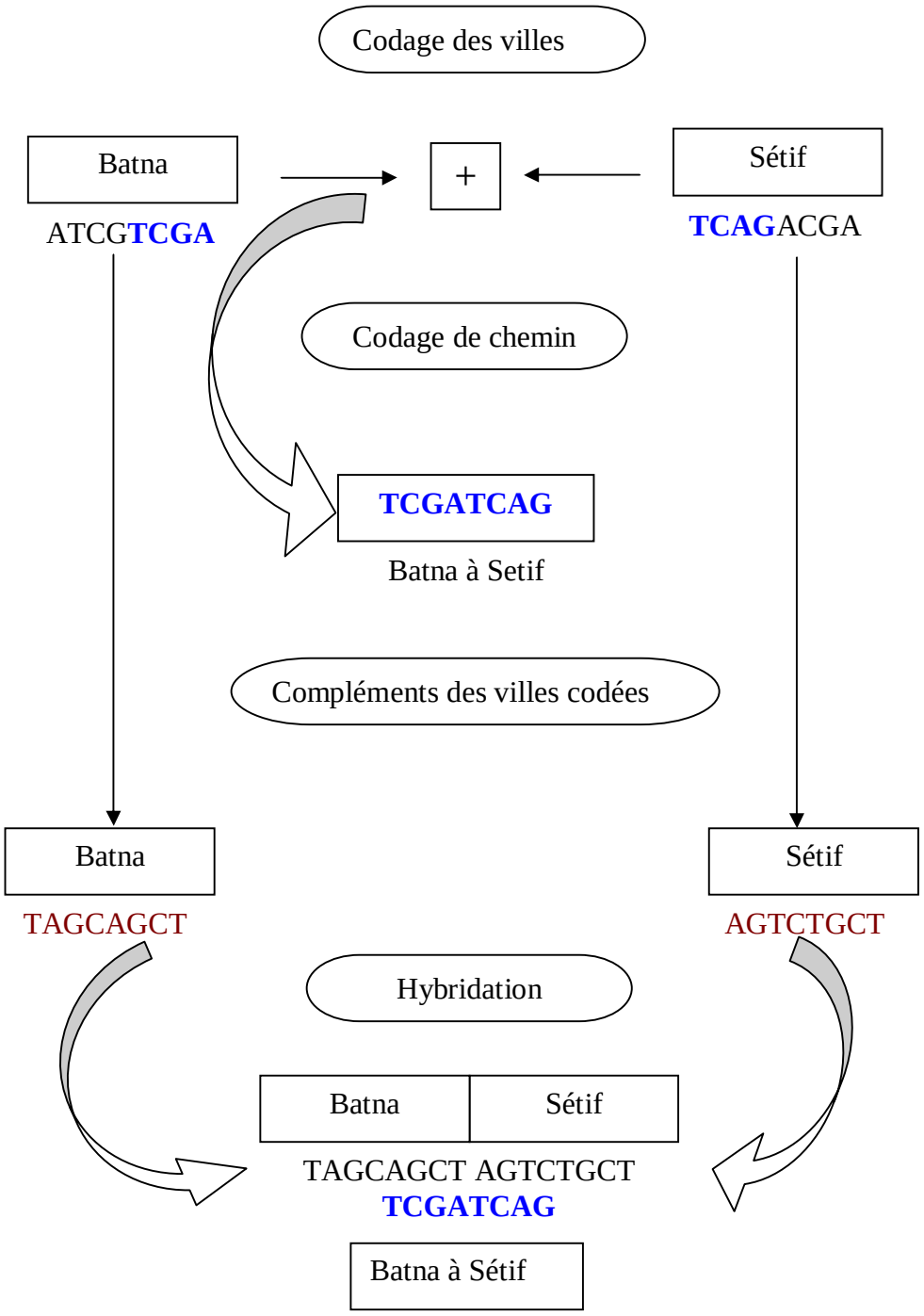

Figure 3.6 : Exemple de codage des deux villes 'Batna' et 'Sétif', ainsi que la route 'Batna vers Sétif' en utilisant des séquences d'ADN.

De cette façon nous pouvons assigner des séquences d'ADN pour toutes les villes et pour tous les numéros de voyage comme suit :

Ville	**code d'ADN**	**Complément du code d'ADN**
Batna	ATCGTCGA	TAGCAGCT
Constantine	GTACACTA	CATGTGAT
Sétif	TCAGACGA	AGTCTGCT

Mila	CGATCGAT	GCTAGCTA

Voyage	**code d'ADN**
Batna – Constantine	TCGAGTAC
Batna – Sétif	TCGATCAG
Constantine – Sétif	ACTATCAG
Mila – Sétif	CGATTCAG
Constantine – Mila	ACTACGAT

- **Génération de tous les chemins possibles d'une façon aléatoire**

Pour notre graphe au-dessus, les chemins suivants sont quelques chemins qui sont générés dans l'étape 1 :

Batna à Sétif

5'TAGCAGCTCATGTGATAGTCTGCT3'
3'TCGAGTACACTATCAGA5'

Batna à Constantine à Sétif

5'TAGCAGCTCATGTGATGCTAGCTAAGTCTGCT3'
3'TCGAGTACACTACGATCGATTCAG5'

Constantine à Mila

5'CATGTGATGCTAGCTA3'
3'ACTACGAT5'

Batna à Constantine à Mila

5'TAGCAGCTCATGTGATGCTAGCTA3'
3'TCGAGTACACTACGAT5'

Mila à Sétif

5'GCTAGCTAAGTCTGCT3'
3'CGATTCAG5'

Batna à Constantine à Batna à Constantine à Sétif

5'TAGCAGCTCATGTGATTAGCAGCTCATGTGATAGTCTGCT3'
3'TCGAGTACACTAATCGTCGAGTACACTATCAG5'

Batna à Constantine à Mila à Sétif

5'TAGCAGCTCATGTGATGCTAGCTAAGTCTGCT3'
3'TCGAGTACACTACGATCGATTCAG5'

Etape2 : PCR :

On va utiliser le principe de la technique de PCR pour éliminer tous les chemins qui ont des mauvais départs et arrivées, et garder seulement les chemins qui commencent par Batna et se terminent par Sétif.

Après l'exécution de PCR, ce qui suit représente les chemins qui commencent par Batna et finissent par Sétif.

Batna à Sétif

5'TAGCAGCTCATGTGATAGTCTGCT3'
3'TCGAGTACACTATCAGA5'

Batna à Constantine à Sétif

5'TAGCAGCTCATGTGATGCTAGCTAAGTCTGCT3'
3'TCGAGTACACTACGATCGATTCAG5'

Batna à Constantine à Batna à Constantine à Sétif

5'TAGCAGCTCATGTGATTAGCAGCTCATGTGATAGTCTGCT3'
3'TCGAGTACACTAATCGTCGAGTACACTATCAG5'

Batna à Constantine à Mila à Sétif

5'TAGCAGCTCATGTGATGCTAGCTAAGTCTGCT3'
3'TCGAGTACACTACGATCGATTCAG5'

Etape3 : Gel-Electrophorèse

Maintenant, on va utiliser l'électrophorèse pour identifier les molécules (les chemins) qui ont la longueur correcte (dans notre exemple une longueur de 24 nucléotides). Toutes les autres molécules ou chemins seront éliminées. Le résultat, après l'achèvement de cette opération d'électrophorèse, sera le chemin suivant qui a une longueur de 24 base d'ADN.

Batna à Constantine à Mila à Sétif

TCGAGTACACTACGATCGATTCAG

Etape4 : Séparation d'affinité de Watson-Crick :

Dans cette étape, on va garder seulement les chemins qui ont l'apparition de chaque nœud exactement une seule fois. Pour notre exemple, le chemin résultant de l'étape 3 vérifie cette condition, il a exactement 4 nœuds qui sont différents (code-Batna ≠ code-Constantine ≠ code-Mila ≠ code-Sétif). Le résultat de cette étape est le même de l'étape précédente et qui est :

Batna à Constantine à Mila à Sétif

TCGAGTACACTACGATCGATTCAG

Etape5 : Séquençage

Après l'exécution de toutes les étapes précédentes, il nous reste qu'un seul chemin (qu'on peut le lire avec la technique de séquençage mentionnée dans le chapitre précédent), alors on peut dire que ce chemin représente le chemin hamiltonien, et qui est :

Batna à Constantine à Mila à Sétif

TCGAGTACACTACGATCGATTCAG

Résumé :

A travers le schéma présenté dans la figure 3.7 (dans la page suivante), et à travers notre exemple de 4 villes, on peut résumer l'algorithme d'Adleman d'une façon simple et compréhensible. Sachons que : Batna est codée par la séquence d'ADN ATCGTCGA, Constantine par GTACACTA, Sétif par TCAGACGA et enfin Mila par CGATCGAT

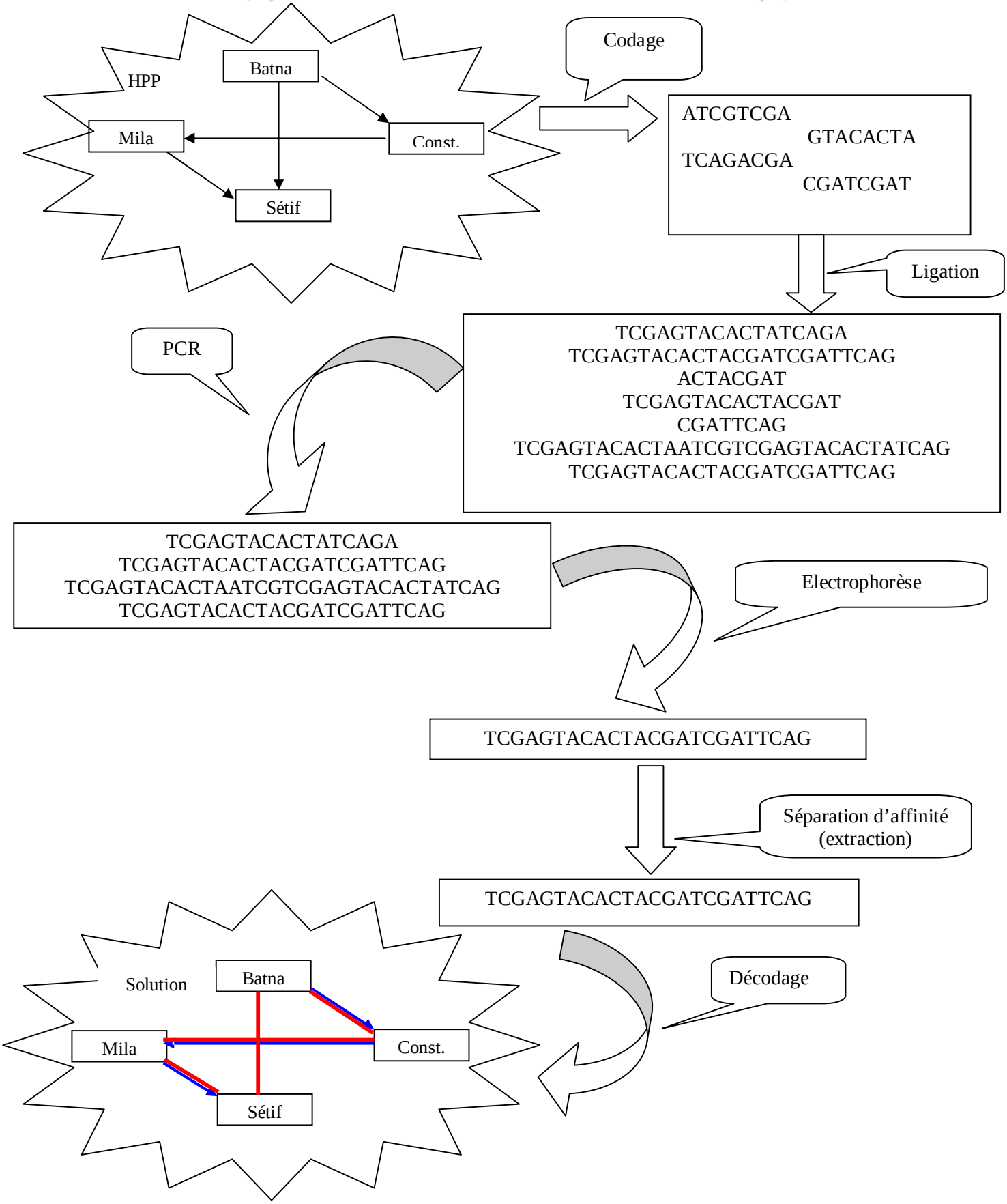

Figure 3.7 : Schéma résumant les opérations effectuées dans l'algorithme d'Adleman pour résoudre le problème HPP.

3.4.2 Les travaux qui suivent le travail d'Adleman

3.4.2.1 Les travaux qui suivent le travail d'Adleman pour la résolution des problèmes NP-complets :

Depuis l'expérience originale d'Adleman, des inondations d'idées ont été proposées par les chercheurs. Il y a des chercheurs qui ont réussi de résoudre quelques problèmes difficiles [112] [113] [114] et d'autres qui ont développé plusieurs modèles différents pour résoudre d'autres problèmes du calcul par le calcul moléculaire [96] [97] [98] [99] [100] [101] [102] [103] [104] [105] [106] [107] [108] [109] [110] [115]. Le Tableau 1 résume aussi quelques problèmes classiques qui ont été résolus par le calcul moléculaire.

Problème	L'année	Référence
Hamiltonian path problem (HPP)	1994	[84]
Boolean satisfiability	1994	[85]
3-colouring	1995	[86]
Quantified Boolean formulae	1995	[87]
Independent set	1996	[88]
Knapsack	1996	[89]
Subgraph isomorphism	1996	[90]
Maximum clique	1996	[90]
MAX-CNF satisfiability	1996	[91]
Circuit satisfiability	1996	[91][92]
(3_2)-system	1997	[93]
Shortest common superstring	1998	[94]
Bounded Post correspondence	2000	[95]

Tableau 3.2 : Quelques problèmes et leurs premières solutions en utilisant le calcul moléculaire [111]

3.4.2.2 Les travaux qui suivent le travail d'Adleman pour la résolution du HPP (ou TSP)

Quatre ans plus tard, après le travail d'Adleman , en 1998, Narayanan et Zorbalas [117] ont proposé deux algorithmes (ces deux algorithme seront détaillés dans le chapitre suivant) pour résoudre le problème de voyageur de commerce (TSP) en insérant des distances entre les villes. Ces deux algorithmes, cependant, n'ont pas été réalisés par aucune expérience d'aucun laboratoire [121], à cause de leurs inconvénients (qui seront mentionnés dans le chapitre suivant).

En 2002,Yamamoto *et al.* [119] a proposé une idée, pour résoudre le problème de TSP avec le calcul d'ADN, de concentration-contrôlé (concentration-controlled DNA computing) en effectuant une recherche locale pour trouver le chemin le plus cours. Cette méthode ne peut pas garantir que la bande la plus intensive contient l'ADN qui représente le chemin le plus court dans un graphe donné. De plus, il est techniquement difficile d'extraire une solution simple optimale de la bande la plus intensive [121].

Dans la même période (en 2002), Lee *et al.* [118] a proposé une approche pour la résolution de ce problème de TSP en basant sur l'idée de la température fondant l'ADN. La réaction en chaîne de Polymérase de Gradient de Température de Dénaturation (DTG-PCR) a été présentée, où l'ADN en double brins de solutions correctes sera dénaturé et amplifié par l'opération PCR. L'augmentation de température de dénaturation, permet aussi à d'autres

brins d'ADN à être amplifiés par la suite. Cependant, l'ensemble de solutions correctes sera aussi exponentiellement augmenté et qui affecte la solution finale.

En 2004**,** Ibrahim Zuwairie [121] de l'institut d'application du calcul d'ADN de l'université de Meiji''Institute of Applied DNA Computing, Meiji University'' a publier un article sous le titre de ''Towards Solving Weighted Graph Problems by Direct-Proportional Length-Based DNA Computing'', dont lequel il a montré comment on peut résoudre le problème de TSP avec le calcul moléculaire en éliminant tous les inconvénients et tous les problèmes des recherches précédemment mentionnés.

En 2005, dans la deuxième conférence mondiale IEEE de DNA computing en Chine, des centaines (probablement plus d'un mille) participants qui ont proposés des excellents idées pour la résolution des différents problèmes NP-complets notamment le TSP.

3.4.3 Les limites actuelles des ordinateurs à l'ADN

Jusqu'ici, nous avons peint un tableau assez encourageant des ordinateurs à l'ADN. En effet, nous avons montré comment résoudre un problème mathématique concret (le graphe hamiltonien) à l'aide des chaînes d'ADN.

En fait, tous les problèmes qui s'expriment avec des fonctions récursives sont résolubles (au moins en théorie) avec des algorithmes utilisant une séquence d'insertions et de délétions dans des chaînes d'ADN.

De plus, sa grande capacité de parallélisme permet de tester toutes les solutions possibles à un problème en même temps, au lieu de les essayer une à la fois comme dans un ordinateur conventionnel. Sous ces angles, l'ordinateur à l'ADN est un outil de calcul potentiellement très puissant. Cependant, il faut se rappeler que tous nos modèles théoriques font une présupposition importante : la nature est parfaite et il est possible pour nous de la manipuler autant que l'on veut. Or, c'est loin d'être le cas.

En effet, dans la nature, il arrive que des chaînes d'ADN en solution dans l'eau se brisent (s'hydrolisent) spontanément. Il arrive aussi qu'il y ait des erreurs quand une chaîne se lie à son complémentaire. Par exemple, si j'ai la chaîne AAGTACCA dont le complémentaire est TTCATGGT, il se pourrait qu'elle se lie à un "faux complémentaire" qui lui ressemble à une base prés. On pourrait donc se retrouver avec le double brin AAGTACCA TTTATGGT où le G est lié avec un T au lieu d'un C. On peut facilement comprendre que ce type d'erreur peut être fatal pour des algorithmes comme celui d'Adleman, qui dépendent de la complémentarité des bases. On pourrait imaginer contourner ce type d'erreur par un code correcteur d'erreurs. Il faut d'ailleurs savoir que l'expérience du graphe hamiltonien qui a été réussie en 1994 par Adleman a été répétée (sans succès !) en 1995 par Kaplan, Cecci et Libchaber... [82]

Et ce n'est même pas au niveau de la complémentarité des bases qu'est survenu le plus grand problème ! En effet, leur électrophorèse a été complètement ratée. A l'endroit où les solutions de la bonne longueur (celles passant par 6 arêtes) auraient dû se trouver, il y avait beaucoup de contaminants (solutions passant par plus ou moins de 6 arêtes). En effet, le gel utilisé pour l'électrophorèse avait beaucoup d'imperfections, et de plus les molécules d'ADN étaient parfois trop repliées sur elles-mêmes pour que leur vitesse de migration soit proportionnelle à leur longueur. D'ailleurs, Adleman a avoué qu'il a dû lui-même répéter l'électrophorèse plusieurs fois pour que ça fonctionne.

De plus, dans l'expérience d'Adleman, il y a toujours le risque que la solution ne soit pas générée. En effet, si on se rappelle de son algorithme, tous les chemins possibles dans le graphe étaient générés au départ par des associations aléatoires entre les chaînes représentant les sommets et celles représentant les arêtes. Or, rien ne peut nous garantir à 100% que tous les chemins possibles seront présents dans l'éprouvette et, en effet, il se peut bien que la

solution ne s'y trouve pas, même si elle existe. La seule parade contre ceci est de mettre un nombre tellement grand de chaînes représentant les sommets et les arêtes dans l'éprouvette, que la probabilité de ne pas générer tous les chemins possibles soient si petites qu'on peut la négliger.

Dans ce type de calculs, si on trouve une solution on peut conclure avec certitude. Mais, si on ne trouve pas de solution, on ne peut pas affirmer hors de tout doute que la solution n'existe pas avec cet algorithme. Tout ce qu'on peut dire, c'est qu'il est très probable qu'il n'y ait pas de solution. C'est donc un algorithme probabiliste.

3.5 Conclusion

A travers ce chapitre, on peut conclure:

- L'expérience d'Adleman était une démonstration de principe : oui on peut programmer l'ADN pour effectuer des calculs mathématiques parallèles ! Sa force est dans sa capacité à générer en un seul coup $\sim 10^{15}$ chemins lors de l'étape 1 (cela dépend bien sûr de nombre de mol d'ADN qu'on a mis dans la solution) et de trier ce très grand nombre également en un seul coup lors des étapes 2 et 3.
- Tous les modèles de calcul d'ADN appliquent une séquence spécifique d'opérations biologiques (voir chapitre précédent ‘‘les outils de la biologie moléculaire’’) à un ensemble de brins d'ADN. Ces opérations sont généralement employées par les biologistes moléculaires.
- Les opérations sur l'ADN sont massivement parallèles : un tube de test (éprouvette) d'ADN peut contenir des trillions de brins. Chaque opération est effectuée sur tous les brins, dans un tube de test, en parallèle.
- Dans un ordinateur à l'ADN, l'information n'est plus transportée par des électrons ni traitée par des transistors, mais par des molécules d'ADN qui mémorisent des quantités énormes de données en séquences de quatre bases (A -T - G - C). Des problèmes complexes, tels que le problème de chemin hamiltonien ou problème du commis voyageur, ont pu être résolu avec ce type d'ordinateur.
- D'après l'efficacité du calcul d'ADN, nous pouvons dire que le calcul moléculaire va jouer un rôle important pour résoudre des problèmes de calcul un peu particuliers en utilisant entièrement de nouveaux algorithmes inattendus.
- Finalement, le calcul d'ADN, théoriquement, a donné des résultats surprenants pour des problèmes extrêmement complexes. Mais, pratiquement, puisqu' un ordinateur d'ADN est un ordinateur bio-chimique, il se peut y avoir des problèmes biologiques soit dans les opérations biologiques (l'hybridation, la ligation, la dénaturation….) ou dans les molécules d'ADN eux même.

On a vu dans ce chapitre comment résoudre un problème de chemin hamiltonien qui est un parmis plusieurs problèmes NP-complets les plus connus. Le chapitre suivant sera entièrement sur la complexité et les problèmes complexes dans lequel on va voir comment résoudre d'autres problèmes NP-complets avec le calcul d'ADN.

4.1 Introduction

La recherche de résolution, en un temps de calcul raisonnable, des problèmes qui ne connaissent pas à l'heure actuelle de résolution en temps polynomial (problèmes NP-complets ou NP-difficiles) est une discipline carrefour de la recherche opérationnelle.

Un problème est dit *polynomial* s'il existe un algorithme permettant de trouver une solution optimale pour toutes ses instances en un temps polynomial par rapport à la taille de l'instance. Un tel algorithme est dit *efficace* pour le problème en question. C'est notamment le cas de certains problèmes de plus court chemin dans un graphe valué, du recouvrement d'un graphe valué par un arbre de poids minimum, des problèmes classiques de flots, ainsi que pour les problèmes Horn-SAT et 2-SAT (notons cependant que MAX-2-SAT reste NP-difficile). Cependant, pour la majorité des problèmes d'optimisation combinatoire, aucun algorithme polynomial n'est connu actuellement [74].

La difficulté intrinsèque de ces problèmes est bien caractérisée par la théorie de la NP-complétude. De nombreux problèmes d'optimisation combinatoire (la plupart de ceux qui sont vraiment intéressants dans les applications !) ont été prouvés NP-difficiles [74].

Cette difficulté n'est pas seulement théorique et se confirme hélas dans la pratique. Il arrive que des algorithmes exacts de complexité exponentielle se comportent efficacement face à de très grosses instances - pour certains problèmes et certaines classes d'instances. Mais c'est très souvent l'inverse qui se produit ; pour de nombreux problèmes, les meilleures méthodes exactes peuvent être mises en échec par des instances de taille modeste, parfois à partir de quelques dizaines de variables seulement. Par exemple, on ne connaît aucune méthode exacte qui soit capable de colorier de façon optimale un graphe aléatoire de densité 1/2 lorsque le nombre de sommets dépasse 90 [74]. Or pour le problème d'affectation de fréquences dans le domaine des réseaux radio-mobiles qui est une extension du problème de coloration, nous avons eu à traiter des instances comportant plus d'un millier de variables [74]. Pour traiter les grosses instances de ce type de problèmes, on se contente de solutions approchées obtenues avec une méthode heuristique [74].

On trouve dans ce chapitre, des généralités sur la complexité des problèmes, tel que la notion de complexité et le classement des problèmes (P, NP,NP-complet), qui représentent les notions de base qu'on doit connaître avant de parler de la résolution des problèmes complexes avec le calcul d'ADN. La classe des problèmes qui nous intéresse dans ce chapitre est la classe des problèmes NP-complets. Cette dernière sera claire à travers quelques exemples notamment ceux de TSP et SAT qui seront résolus par la suite avec le calcul d'ADN. Enfin, deux nouveaux algorithmes, en utilisant le calcul d'ADN, seront proposés pour la résolution de ces deux problèmes de TSP et SAT, suivant par des discussion et des critiques pour chaque algorithme.

4.2 Généralités sur la complexité des problèmes

4.2.1 Notion de complexité : Définition

On se rend intuitivement compte que tous les problèmes pouvant être résolus par une machine de Turing ne sont pas de même difficulté... mais comment mesurer cette difficulté ?
La difficulté d'un problème peut être mesurée par **le temps d'exécution** d'un algorithme qui le résout, ou encore la **quantité de mémoire** requise lors de la mise en œuvre de cet algorithme. Cependant, les notions de temps de traitement et d'espace mémoire sont

dépendantes de la machine physique utilisée pour implémenter l'algorithme; on a donc besoin d'une mesure absolue, i.e. indépendante de toute implémentation. La notion de machine de Turing va nous permettre de définir une telle mesure absolue de la difficulté d'un problème; elle sera appelée la complexité (algorithmique) du problème.

Considérons un problème P formalisable et décidable, et une machine de Turing T résolvant P :

- Nous appèlerons complexité (temporelle) pire cas (i.e. dans le cas le plus défavorable) de T pour P, le nombre maximal de déplacements de la tête de lecture/écriture que devra effectuer la machine T pour résoudre une instance de P de taille maximale n (où la taille n de l'instance est la longueur de la séquence binaire codant cette instance comme entrée de la machine de Turing)
- Nous appèlerons complexité (spatiale) pire cas de T pour P, la longueur de bande nécessaire à la machine T pour résoudre une instance de P de taille maximale n.

4.2.2 Problèmes de décision et d'optimisation

- **Problèmes de décision :** Il s'agit d'une collection d'instances qui sont des ensembles de données et qui admettent exactement une des deux réponses «oui» ou «non» à une certaine question [70].
- **Problèmes d'optimisation :** Ils diffèrent des premiers par le type de réponse. En effet, tout type de réponse est accepté comme des entiers, un tableau ... [70]

Les deux types sont intimement liés. En effet, une instance d'un problème d'optimisation peut facilement se traiter grâce à un algorithme prévu pour le problème d'optimisation, à un algorithme en temps au plus polynomial près [70].

4.2.3 Classement des problèmes:

4.2.3.1 Problèmes « faciles », problèmes « difficiles »

- **les problèmes faciles** [125]**:** pouvant être résolus par une machine de Turing de complexité pire cas bornée par un polynôme en la taille des entrées. Ces problèmes seront aussi appelés ***problèmes polynomiaux***, et les algorithmes permettant de les résoudre avec une complexité polynomiale seront appelés des ***algorithmes efficaces***.
- **les problèmes difficiles** [125]: (i.e. les problèmes formalisables, décidables, qui ne sont pas faciles) pour lesquels la complexité pire cas n'est pas bornée par un polynôme...Ces problèmes seront aussi appelés problèmes non polynomiaux, et les algorithmes permettant de les résoudre seront appelés des algorithmes inefficaces.

4.2.3.2 Les classes P et NP

- **La classe P** est la classe des problèmes qui admettent un algorithme déterministe avec un temps d'exécution polynomial en fonction de la taille du ruban d'entrée [124]. La classe **P** contient tous les problèmes relativement faciles c'est-à-dire ceux pour lesquels on connaît des algorithmes efficaces. Plus formellement, ce sont les problèmes pour lesquels on peut construire une machine déterministe (e.g. une machine de Turing) dont le temps d'exécution est de complexité polynomiale (le sigle **P** signifie "**P**olynomial time") [71].

- **La classe NP** est la classe des problèmes qui admettent un algorithme non-déterministe avec un temps d'exécution polynomial en fonction de la taille du ruban d'entrée [124]. Les problèmes de la classe **NP** sont ceux pour lesquels on peut construire une machine de Turing non déterministe dont le temps d'exécution est de complexité polynomiale [71]. Notons bien que le sigle **NP** provient de "**N**ondeterministic **P**olynomial time" (et non de "Non Polynomial"). Contrairement aux machines déterministes qui exécutent une séquence d'instructions bien déterminée, les machines non déterministes ont la remarquable capacité de toujours choisir la meilleure séquence d'instructions qui mène à la bonne réponse lorsque celle-ci existe [71]. Il faut noter aussi que la classe **P** est incluse dans la classe **NP**, on écrit **P** $\subset$ **NP**, car si l'on peut construire une machine déterministe pour résoudre efficacement un problème, on peut certainement (au moins dans notre esprit) en construire une non déterministe qui résout aussi efficacement le même problème. Par ailleurs, il ne faut pas croire que ce concept de divination optimale qu'est la machine non déterministe permet de tout résoudre puisqu'il existe en informatique théorique d'autres types de problèmes n'appartenant pas à la classe **NP** (e.g. les problèmes indécidables) [71].

4.2.3.3 Classe NP-complet

Parmi l'ensemble des problèmes appartenant à **NP**, il en existe un sous-ensemble qui contient les problèmes les plus difficiles : on les appelle les problèmes **NP**-complets. Un problème **NP**-complet possède la propriété que tout problème dans **NP** peut être transformé en celui-ci en temps polynômial [71].

Pour transformer un problème P_1 en un problème P_2, on dit souvent réduire P_1 en P_2, et on écrit $P_1 \alpha P_2$ il s'agit simplement de définir une application *f* qui met en relation les instances de P_1 avec celles de P_2 [71]:

$$f : P_1 \rightarrow P_2$$

D'une autre définition, On dit que le problème de décision P_1 se réduit polynomialement à P_2 s'il existe une fonction $\varphi : \Sigma^* \rightarrow \Sigma^*$ calculable en temps polynomial, telle que tout x est instance positive de P_1 si et seulement si $\varphi(x)$ est une instance positive de P_2. On note $P_1 \alpha P_2$ (ou parfois $P_1 \leq_p P_2$) [70] voire figure 4.1.

Donc On essaie de transformer un problème dont on ne sait pas s'il est polynomial en un problème connu (2 - SAT?). On écrit alors $P_1 \leq P_2 \wedge P_2 \in P \Rightarrow P_1 \in P$ [73]

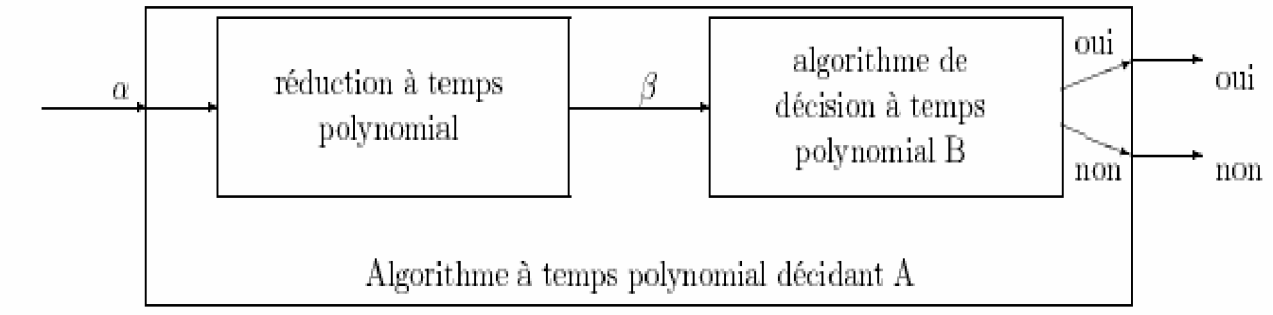

Figure : 4.1 : Réduction d'un problème de décision P_1 polynomialement à un problème P_2 [70].

Enfin, un problème P est NP-complet si tout problème de NP se réduit polynomialement à P [70].

$$\mathbf{NP\text{-}COMPLET} = \{\Pi \in \mathbf{NP}\ tq\ \forall P \in \mathbf{NP},\ P \propto \Pi\}$$

Une propriété intéressante de la classe NP-complet est : S'il existe un algorithme permettant de résoudre en temps polynomial un problème NP-complet, alors pour tout problème de NP, il existe un algorithme en temps polynomial pour le résoudre [70]. De plus, si on prouve qu'un problème de NP n'est pas soluble en temps polynomial, alors aucun problème de la classe NP-complet n'admet un algorithme en temps polynomial pour le résoudre [70].

Enfin, pour montrer qu'un problème est NP-complet, il faut vérifier qu'il est dans NP et que tout problème de NP lui est réductible. Cependant, par transitivité de la relation α , il suffit qu'un problème NP-complet se réduise à ce problème [70].

Notons aussi q'un Problème NP-difficile est un problème (pas forcément NP) tel que s'il était résolu, alors il permettrait de résoudre en temps polynomial un problème NP complet [69]. Encore, un problème est NP-difficile si tout problème de NP lui est polynomialement réductible. Un problème NP-difficile de NP s'appelle un problème NP-complet

Exemple d'une réduction polynomiale: Problème du circuit hamiltonien : étant donné un graphe, trouver un circuit simple qui comprend tous les sommets (une fois et une seule) voire la figure 4.2 .

Le problème du circuit hamiltonien est polynomialement réductible au problème du voyageur de commerce.

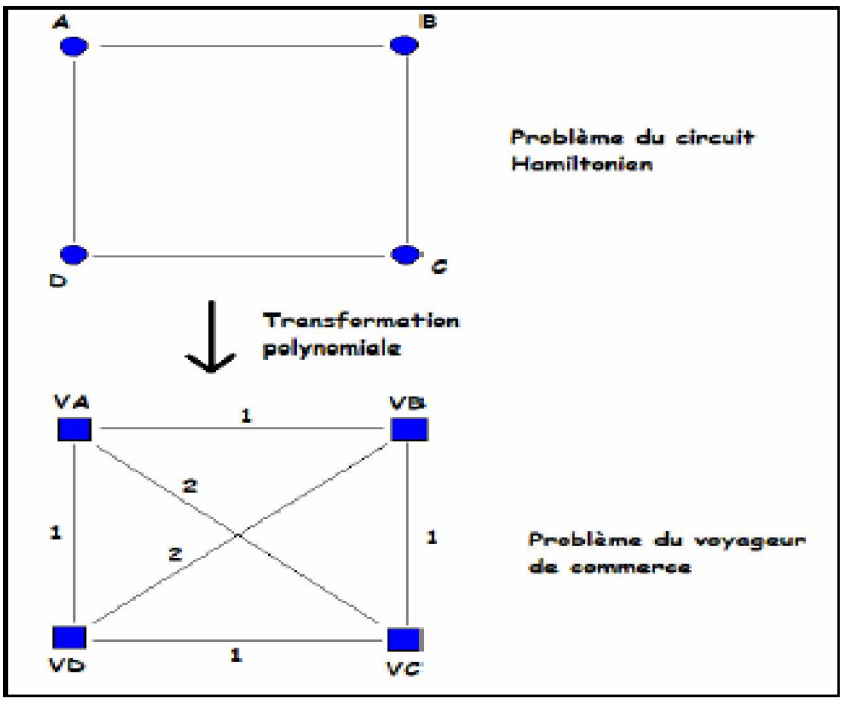

Figure 4.2 : transformation polynomiale d'un problème de circuit Hamiltonien vers un problème de PVC.

4.2.3.4 Relation entre les classes

La théorie la plus développée sur les liens entre les 3 classes est celle définit ci après : $P \in NP$ et l'inclusion est stricte [70].

Cette question de savoir si $P = NP$ est au coeur des recherches aujourd'hui et ses implications sont très importantes (*30 ans de recherche sans réponse!* [73]). Les chercheurs sont quelque peu pessimistes et estiment que l'inclusion est stricte. En effet, au vue du grand nombre de problèmes NP-complets connus à ce jour, on estime que s'il existait, pour chacun, un algorithme polynomial, alors c'est algorithme aurait déjà du être découvert. Néanmoins, un espoir subsiste de trouver un algorithme polynomial pour un problème NP-complet.

4.3 Les problèmes NP-complets les plus célèbres:

4.3.1 Problème de satisfiabilité (ou satisfaisabilité) :SAT

La satisfiabilité propositionnelle est un problème important de la théorie de la complexité et de l'intelligence artificielle. Il consiste à déterminer, pour une formule du calcul propositionnel, l'existence d'une instanciation de ses variables pour laquelle la formule est évaluée à vrai. SAT restreint ce problème aux formules en forme normale conjonctive (CNF) .Une formule CNF est une conjonction de clauses, elles-mêmes disjonctions de littéraux. Un littéral est une variable propositionnelle ou sa négation et ne peut prendre que la valeur Vrai ou Faux. Une interprétation d'une formule booléenne est une instanciation de ses variables. Un modèle est une interprétation qui satisfait la formule. SAT est voué, s'il existe, à trouver un modèle d'une formule CNF, ou le cas échéant, à prouver qu'il n'en existe pas.

On peut définir la satisfiabilité d'une formule de la manière suivante :

Soit une formule F du calcul propositionnel sous forme normale conjonctive utilisant les variables propositionnelles ($x_1 \dots x_n$).

Résoudre le problème de la satisfiabilité de F consiste à déterminer s'il existe une interprétation des x_i telle que F soit satisfaite (F vraie). Ce problème est appelé problème SAT.

La restriction de SAT aux cas où toutes les clauses sont de longueur k est notée k-SAT. Un intérêt particulier est porté aux valeurs 2 et 3 de k. En effet, 3 est la plus petite valeur de k pour laquelle k-SAT est NP-complet [126], et 2-SAT peut-être résolu en temps linéaire [129]. Horn-SAT représente les problèmes où chaque clause a au plus une variable sans négation, résolu également en temps linéaire [128].

Exemple :

- f=(A1∨A2∨A3) ∧ (B1∨B2) ∧ (C1∨C2∨C3∨C4) est une formule.
- Chaque variable propositionnelle peut être précédée de $\neg$, signifiant négation : $\neg A_1 \vee A_2 \vee \neg A_3$.
- $A_1 \vee A_2 \vee A_3$, $B_1 \vee B_2$, $C_1 \vee C_2 \vee C_3 \vee C_4$ sont respectivement une 3-clause (clause à 3 variables propositionnelles), 2-clauses et 4-clause, $\neg A_1 \vee A_2 \vee \neg A_3$ est une clause de Horn.
- $A_1, A_{2,}, A_3, B_1, B_2, C_1, C_2, C_3, C_4$ sont des variables propositionnelles, qui ne peuvent prendre que la valeur Vrai ou Faux.

Théorème : Théorème de Cook [126]: Le problème SAT est NP-complet.

De nombreux problèmes pratiques sont NP-complets et peuvent être transformés en SAT ; les algorithmes SAT performants peuvent donc avoir de considérables utilités. Parmi ces algorithmes de résolution, les plus classiques appartiennent à deux catégories :

- les algorithmes complets : La méthode la plus simple, appelée Generate & Test ou Brute Force, pour tester la satisfiabilité, est de générer toutes les interprétations possibles. Davis et Putnam [127] ont introduit une méthode énumérative dans laquelle les variables sont instanciées et éliminées une par une de la formule, dite ''Generate & Test''.

- les algorithmes incomplets : ils ne peuvent pas prouver la non satisfiabilité, mais ils sont souvent nettement plus efficaces à trouver une instanciation, quand elle existe, satisfaisant la formule. Le plus largement étudié est GSAT de Selman [67] et [68].

Tandis que SAT est un problème de décision, MAX-SAT (Maximum Satisfiability), une extension de SAT, est dédié à la recherche d'une solution partielle satisfaisant le plus grand nombre de clauses d'une formule logique en forme normale conjonctive. MAX-SAT est un problème NP-complet et contrairement à SAT il le reste même lorsque les clauses ne contiennent que deux littéraux [130].De plus, dans l'optique de la recherche d'une solution optimale (beaucoup trop coûteuse pour les algorithmes complets), les équipes de recherche, fortement développés depuis la fin des années 70 et le début des années 80, se sont décidées à résoudre ces problèmes par des méthodes heuristiques. Pour cela deux conditions sont nécessaires, la première nécessitant un algorithme d'approximation efficace, et la seconde, la valeur de la solution trouvée devant être très proche de la solution optimale.

4.3.2 Problème du voyageur de commerce

L'énoncé du problème du voyageur de commerce est le suivant : étant donné *n* points (des « villes ») et les distances séparant chaque point, trouver un chemin de longueur totale minimale qui passe exactement une fois par chaque point (et revienne au point de départ) (voir figure 4.3).

Ce problème est plus compliqué qu'il n'y paraît; on ne connaît pas de méthode de résolution permettant d'obtenir des solutions exactes en un temps raisonnable pour de grandes instances (grand nombre de villes) du problème. Pour ces grandes instances, on devra donc souvent se contenter de solutions *approchées*, car on se retrouve face à une explosion combinatoire : Le nombre de chemins possibles passant par 69 villes est déjà un nombre de 100 chiffres. Pour comparaison, un nombre de 80 chiffres permettrait déjà de représenter le nombre d'atomes dans tout l'univers connu!).

Plus généralement, divers problèmes de recherche opérationnelle se ramènent au voyageur de commerce. Un *voyageur de commerce* peu scrupuleux serait intéressé par le double problème du *chemin le plus court* (pour son trajet réel) et du *chemin le plus long* (pour sa note de frais).

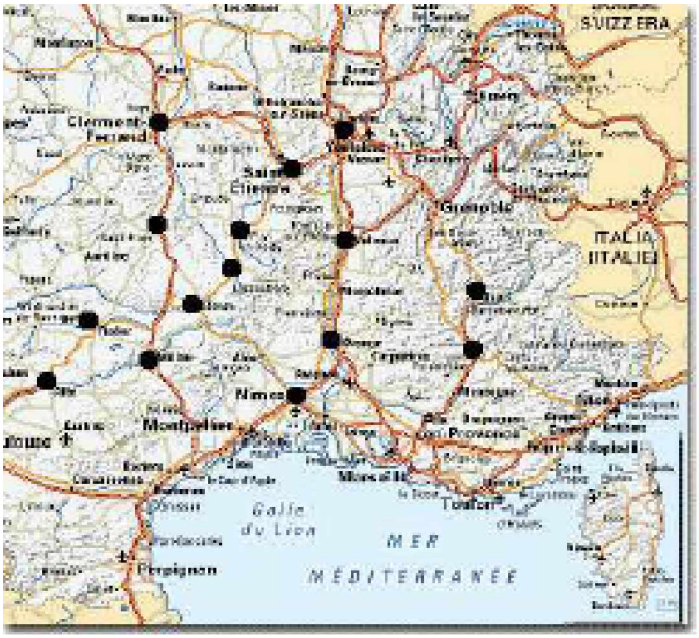

Figure 4.3 : Carte géographique d'un voyageur de commerce

4.3.3 Problème de sac à dos (Knapsack problem)

Le problème du sac à dos est un exemple typique classique de problème (NP-complet) pour lequel aucun algorithme efficace n'est connu et où il faut explorer toutes les possibilités pour obtenir la meilleure solution.

Soit E un ensemble d'objets e_i ayant chacun un certain poids $p(e_i)$ ($p(e_i) \in \boldsymbol{R}^+$). Soit M la charge maximum que l'on peut emporter dans le sac à dos ($M \in \boldsymbol{N}$). On veut trouver un ensemble d'objets dont la somme des poids soit la plus voisine possible de M tout en lui étant inférieure ou égale. Dans ce problème, la condition C portant sur le poids du sac à ne pas dépasser. Souvent la formulation du problème du sac à dos fait intervenir deux fonctions: le poids et la valeur, ou encore le volume et le poids. Il s'agit alors d'avoir la valeur maximale avec une borne supérieure sur le poids, ou de minimiser le poids avec une borne supérieure sur le volume.

4.3.4 problèmes de 8 reines

Gauss en 1850 a inventé le problème suivant: placer huit reines sur un échiquier sans qu'aucune d'entre elles ne soit en prise par une autre. On peut en fait définir ce problème pour n reines sur un échiquier de taille $n \times n$. Quand on essaie de le résoudre à la main, cela n'a rien d'évident. Deux solutions se trouvent sur la figure 4.4.

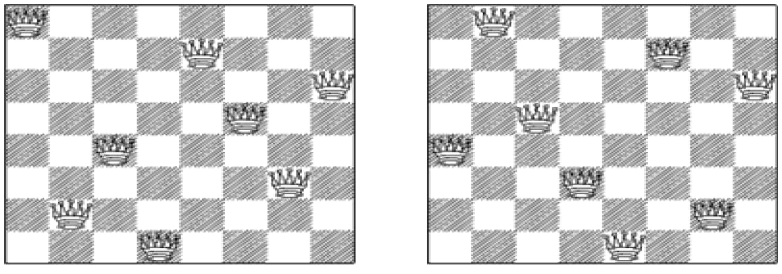

Figure 4.4 : Huit reines sur un échiquier.

4.4 Résolution de quelques problèmes NP-complets sen utilisant le calcul d'ADN

4.4.1 Le problème de la satisfaisabilité

Supposons qu'on ait un énoncé logique bâti uniquement de $\vee$ (OU), $\wedge$ (ET) et $\neg$ (NON), comme celui-ci par exemple : $\alpha = (x_1 \vee x_2) \wedge \neg x_3$. Ici, x_1, x_2 et x_3 sont des variables qui peuvent prendre la valeur VRAI ou FAUX et α représente la valeur de vérité totale de l'énoncé logique. Bien sûr, α sera VRAI ou FAUX dépendamment des valeurs assignées aux variables. Par exemple, si x_1, x_2 et x_3 ont la valeur VRAI, la valeur de α sera FAUX.

Peut-on assigner des valeurs de vérité à x_1, x_2 et x_3 telles que α ait la valeur VRAI ? Il est facile de voir que oui. En effet, nous pourrions par exemple assigner la valeur VRAI à x_1 et x_2 et la valeur FAUX à x_3. Nous disons qu'on peut satisfaire cette équation logique.

En général, le problème de la satisfaisabilité pour un énoncé logique consiste à décider s'il existe des valeurs de vérité qui satisfont une équation logique. L'exemple présenté ci-haut était facile à visualiser pour un être humain et facile à programmer, même avec un algorithme rudimentaire. En effet, pour un ordinateur il s'agit simplement de tester toutes les assignations de valeurs possibles (2^3 dans le cas de cet exemple, car il y a trois variables et chacune peut prendre soit la valeur VRAI, soit la valeur FAUX).

Par contre, le problème devient beaucoup plus compliqué lorsqu'on travaille avec un grand nombre de variables. Déjà, à 100 variables, l'ordinateur doit tester 2^{100} assignations de valeurs [82].

C'est pourquoi un algorithme par ADN a été proposé pour le résoudre. En effet, comme tous les problèmes de "fouille extensive ", qui demandent que l'ordinateur vérifie chacun des cas l'un après l'autre, celui-ci bénéficie grandement des capacités de calcul en parallèle de la machine à l'ADN, puisque avec l'ADN, tous les cas sont essayés en même temps. La difficulté devient alors d'extraire la bonne solution.

Au départ, il s'agit de trouver une façon de générer dans l'éprouvette toutes les assignations de valeurs possibles. Par exemple, si on a trois variables, il faudra trouver une façon de coder chacune des $2^3 = 8$ possibilités.

Ceci est possible à l'aide de la théorie des graphes. En effet, on modélise les valeurs d'assignations possibles par le graphe de la Figure 4.5 :

Ici, chaque chemin que l'on peut prendre dans le graphe représente une suite d'assignations de valeurs de vérité possibles. Les a_j^0 représentent la valeur 0 pour le sommet j et les a_j^1 , la valeur 1. Les v_i sont des séparateurs. Par exemple, le chemin $a_1^0 v_1 a_2^0 v_2 a_3^0 v_3$ représente l'assignation de la valeur de vérité 0 à chacune des trois variables. On peut facilement voir que, en suivant tous les trajets possibles dans le graphe, on passera par les huit assignations différentes de valeurs pour les trois variables [82].

Ceci est utile, car la première étape de l'algorithme par ADN sera de générer plusieurs exemplaires de chacun des huit chemins possibles pour pouvoir les tester ensuite. Pour ce faire, l'expérience acquise dans la résolution du problème de Adleman nous sera très importante : en effet, nous savons déjà comment coder les sommets et les arêtes dans un graphe orienté pour générer tous les chemins possibles. Nous commencerons par coder chacun des sommets du graphe par une séquence de *2m* bases azotées et chacune des arêtes par les compléments des *m* dernières bases azotées de leur sommet de départ suivies des compléments des *m* premières bases du sommet d'arrivée.

On met donc dans une éprouvette plusieurs exemplaires de chacune des chaînes d'ADN codantes et, au bout d'un certain temps, tous les chemins possibles sont formés. Il reste donc à performer des tests pour voir si l'énoncé logique peut être satisfait.

La première étape est de transformer l'énoncé en *forme conjonctive normale*, c'est-à-dire du genre $\alpha = C_1 \wedge C_2 \wedge C_3 \wedge ... \wedge C_n$ où tous les C_1 sont des propositions logiques n'utilisant que le $\vee$ et le $\neg$. Un théorème de logique assure que cette conversion est possible pour n'importe quel énoncé logique utilisant des ET, des OUI et des NON.

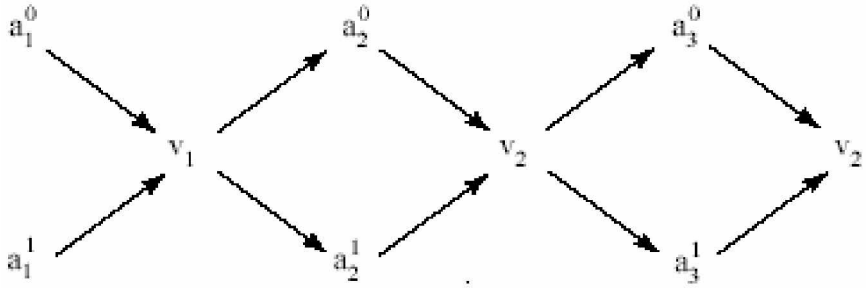

Figure 4.5 : Encodage du deuxième sommet[82]

Remarque : Notons que, bien que la conversion soit possible, elle ne se fait pas toujours facilement. En effet, les algorithmes connus pour ce type de conversion sont assez complexes et ne donnent pas nécessairement la forme conjonctive normale en un temps raisonnable [82]. Cependant, dans plusieurs problèmes, l'énoncé est déjà sous forme conjonctive normale et ne nécessite pas cette traduction laborieuse.

Cette première étape servira à guider le processus d'extraction de la solution à partir de l'éprouvette de départ. Notons que, dans le cas de l'exemple donné ci-haut, cette conversion n'est pas à faire, puisque l'énoncé est déjà sous la forme conjonctive normale.

On prend alors l'énoncé C_1 et on extrait de la solution toutes les chaînes qui correspondent à ses critères. Dans notre exemple, $C_1 = x_1 \vee x_2$. Ceci veut dire qu'on veut extraire toutes les chaînes où soit un des deux x_1 et x_2, soit les deux ont une valeur de 1.

Ceci peut se faire en premier lieu en extrayant toutes les chaînes où $x_1 = 1$ de la solution. Pour ce faire, on peut utiliser des techniques similaires à celles d'Adleman. En effet, on peut mettre dans la solution contenue dans l'éprouvette de départ (éprouvette A) des petites billes de fer auxquelles sont attachées des amorces constituées des chaînes complémentaires aux chaînes d'ADN représentant $a_1^1 v_1$ [82]. Les chaînes qui codent des chemins dans le graphe où $x_1 = 1$ sont alors attirées à leurs amorces complémentaires, tandis que les chaînes qui ne codent pas des solutions restent en suspension dans l'éprouvette.

On attire alors les billes de fer au rebord de l'éprouvette à l'aide d'un aimant, ce qui retient les billes de fer et les chaînes qui y sont attachées sur l'éprouvette, et on vide le reste de la solution dans une autre éprouvette (éprouvette B)[82]. Ensuite, on remet les chaînes accrochées aux billes de fer (celles qui codent des chemins où $x_1 = 1$) en solution dans l'éprouvette A [82].

On a donc isolé dans l'éprouvette A les chemins où x_1 ait une valeur de 1. Mais, pour que $x_1 \vee x_2$ ait une valeur de 1, il se peut aussi que x_2 ait une valeur de 1 tandis que x_1 a une valeur de 0. Il faut donc ajouter dans l'éprouvette A toutes les chaînes restées dans l'éprouvette B, mais dont la valeur de x_2 est de 1. Pour ce faire, on extrait ces chaînes d'ADN de l'éprouvette B en utilisant la méthode des billes de fer et on ajoute les chaînes ainsi isolées dans l'éprouvette A[82].

On a donc maintenant dans l'éprouvette A toutes les chaînes qui donnent une valeur de vérité 1 à la proposition C_1. L'idée sera alors d'extraire de l'éprouvette A toutes les solutions qui correspondent à la valeur de vérité 1 de C_2, car pour que $C_1 \wedge C_2$ soit vrai, il faut à la fois que C_1 soit vrai et que C_2 soit vrai : on ne peut donc trouver nos solutions que dans l'éprouvette A[82].

Si on reprend notre exemple, $C_2 = \neg x_3$. On doit donc extraire de l'éprouvette A toutes les chaînes où la valeur de x_3 est de 0.

4.4.2 Le voyageur de commerce (TSP : Traveling Salesman Problem)

Dans le chapitre précédent, le problème de voyageur de commence a été résolu avec le calcul d'ADN selon l'expérience d'Adleman, et sans introduire les distances entre les nœuds. Dans cette section, ce problème sera résolu, en introduisant les distances entre les nœuds, selon les deux algorithmes de Narayanan et Zorbalas.

Algorithme1 :

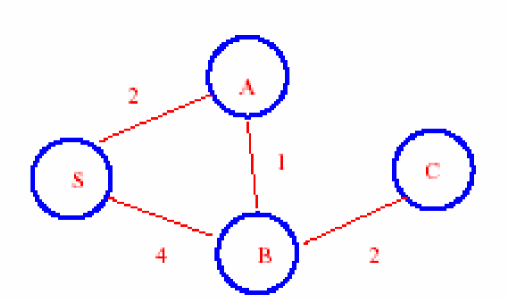

Figure 4.6 : graphe de 4 nœuds : le nœud de départ est S et celui d'arrivée est C [117].

Considérons le graphe de la figure 4.6, les étapes suivantes vont extraire le chemin le plus cours entre le nœud de départ S et celui d'arrivée C :

1- Associer à chaque ville, aléatoirement, une séquence d'ADN.
   Ex :
   S = AAAA A= CCCC B = GGGG C = TTTT
2- Générer tous les chemins binaires possibles : Soit X et Y deux nœuds et *n* la distance entre ces deux nœuds :
   a- pour le chemin binaire X à Y, relier *n* occurrences de séquences d'ADN pour X avec une seule occurrence de séquence d'ADN pour Y.
      Ex: S à A, *n=2 :*
      Le code d'ADN pour S à A sera : AAAAAAAACCCC (le code de S sera répété deux fois, car n=2).
   b- On va répéter la même opération (celle de a) pour le chemin binaire inverse Y à X .
      Ex: A à S, *n=2 :*
      Le code d'ADN pour A à S sera : CCCCCCCCAAAA (le code de A sera répété deux fois, car n=2).
   c- Former un brin complémentaire où la jointure a lieu pour assurer l'union du chemin binaire.
3- Générer des chemins longs à partir des chemins binaires précédemment construits (voire figure 4.7). Soit P1, P2 deux chemins, on peut attacher ces deux chemins si et seulement si le code de la ville finale de P1 égale à celui de la première ville du P2.
   Ex : S à A : AAAAAAAACCCC et A à B : CCCCGGGG

S à A à B sera : AAAAAAAACCCCGGGG, où la moitié de la ville de A du chemin S à A (CC) et la moitié de la ville A du chemin A à B (CC) seront éliminées.

4- Placer tous les chemins produits de l'étape 3 dans un test-tube (éprouvette).
5- Extraire tous les chemins qui commencent par le code d'ADN de la ville de départ et placer les dans un autre test-tube.
6- Extraire tous les chemins qui se terminent par le code d'ADN de la ville de destination désirée et placer les dans un autre test-tube.
7- Le brin d'ADN le plus petit représente le chemin le plus cours, en calculant sa distance à partir de cette équation : D = M/L-1 tel que D : distance du chemin, M : le nombre de nucléotides d'ADN du chemin, et L : le nombre de nucléotides de chaque ville (pour cet exemple L=4) .
Pour cet exemple de 4 villes le chemin le plus cours est S à A à B à C avec

D=24/4-1=5

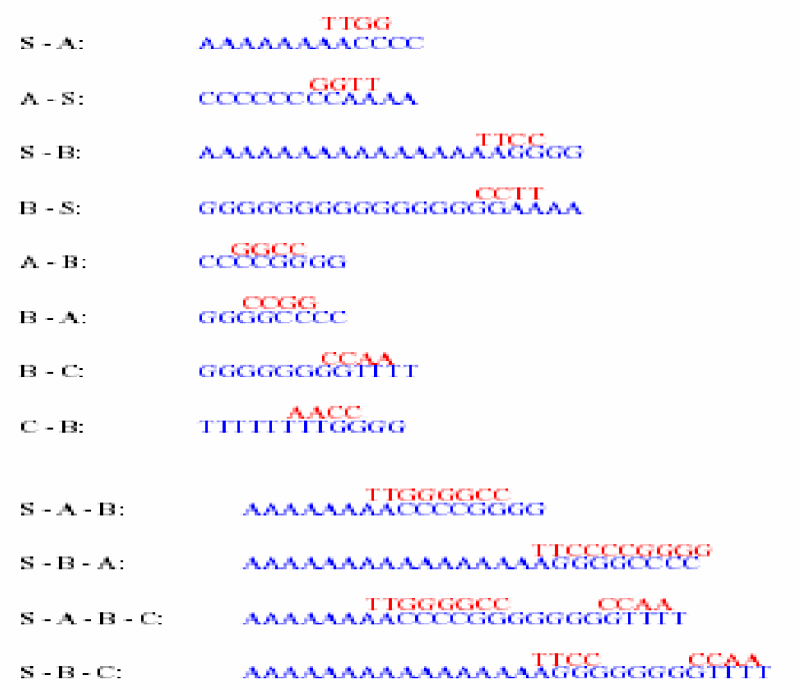

Figure 4.7 : les chemins possibles générés selon l'étape 2, 3 [117].

Algorithme2 : (voir la figure 4.8)

1- **V :** nombre total des noeuds du graphe G.
P : nombre total des chemins binaires dans le graphe G.
D : ensemble des chemins triés par distance, avec le premier est celui le plus petit.
2- Générer les séquences d'ADN suivantes :
a. O_i (i = 1,…,v) : longueur fixée des séquences aléatoires d'ADN correspondant aux villes.
b. $\overline{O_i}$ (i=1,..,v) : compléments du O_i .
c. O_d (d=1,.., p) : longueurs variables des séquences aléatoires correspondant à toutes les distances D dans le graphe.
d. $\overline{O_d}$ (d=1,.., p) : compléments du O_d .

i.e : la longueur L du O_d et $\overline{O_d}$ va être proportionnelle au location des distances correspondants dans D et peut être augmentée par un facteur K: $w \in \{1,...,P\} \Rightarrow L = d * K$.

Ex : Si K=3 et D= {2,5,9,10}, alors la distance 2 est représentée par un brin d'ADN d'une longueur de 3. La distance 5 sera représentée par un brin d'une longueur de 6 (3*2). La distance 9 sera représentée par une longueur de 9 (3*3), enfin 10 par 12 (3*4).

3- Générer tous les chemins binaire (entre deux nœuds)possibles :
i: nœud de début de chemin, d: distance de chemin, j: nœud de fin de chemin.

a- **si** i=1 **alors** créer brin : $O_{id \to} j$ as ALL O_i + ALL O_d + HL O_j ;

b- **si** i>1 **alors** créer brin : $O_{jd} \to j$ as HR O_i + ALL O_d + HL O_j ;

$O_{jd} \to i$ as HR O_j + ALL O_d + HL O_i

Où : 'ALL'(tous) : représente tout le brin d'ADN du nœud.
'HL' (left half) : représente la moitié du brin d'ADN du nœud prenant la partie gauche.
'HR'(right half) : représente la moitié du brin d'ADN du nœud prenant la partie droite.

4- Générer tous les chemins possibles, en insérant tous les chemins binaires précédemment construits et tous les $\overline{O}$ (i.e : $\overline{O_i}$, $\overline{O_j}$, $\overline{O_d}$) dans le test-tube. Insérant aussi l'enzyme de ligase pour les coller.

5- Garder seulement les chemins qui commencent par le nœud de départ spécifié dans le graphe ainsi que le nœud d'arrivée

6- Garder le chemin qui a la plus petite longueur. Ce chemin représente le chemin le plus cours.

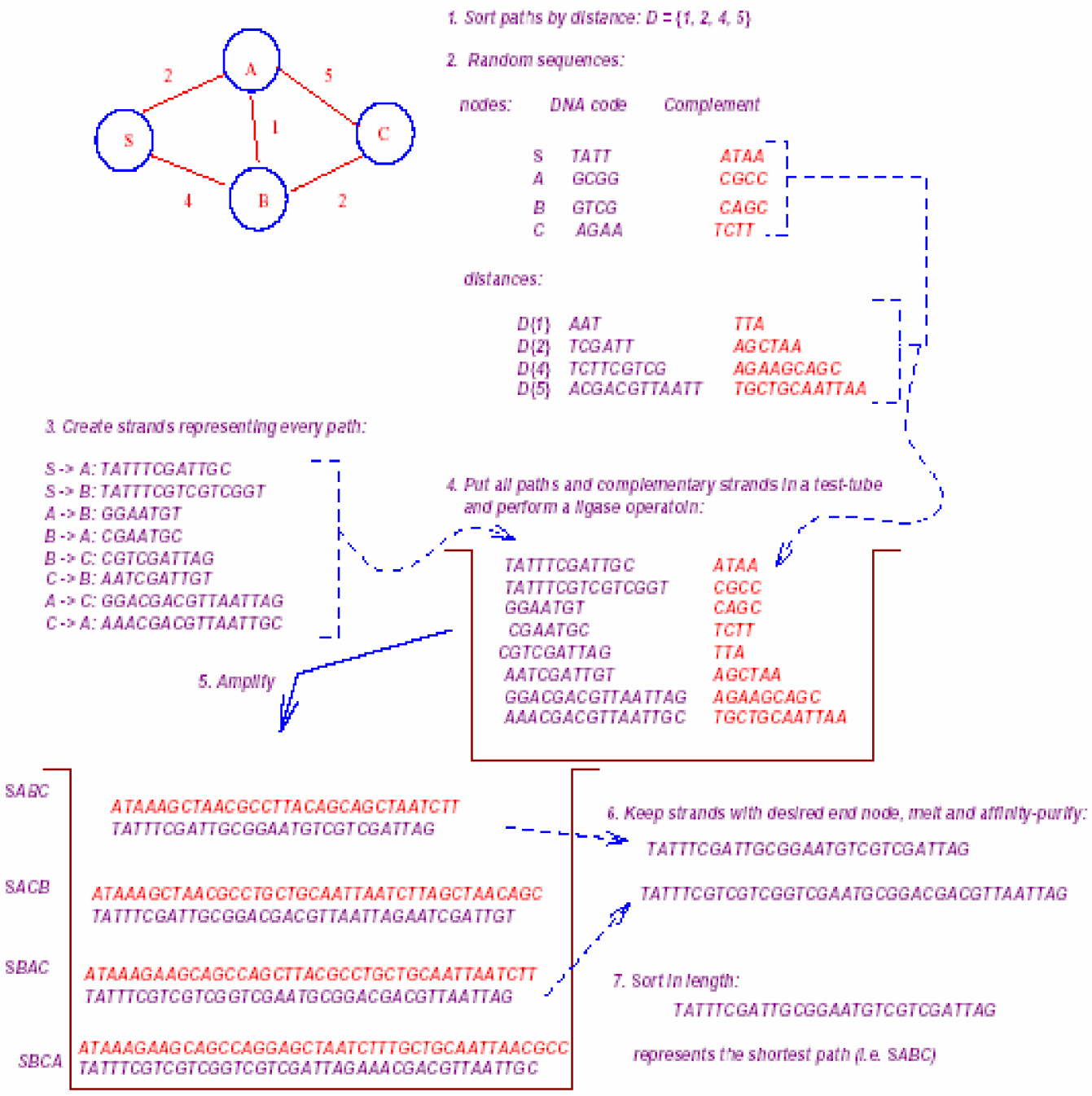

Figure 4.8 : Le deuxième algorithme de Nayanan et Zorbalas [117]

4.5 Deux algorithmes proposés pour la résolution de TSP et SAT par le calcul d'ADN

4.5.1 Nouvel algorithme proposé pour la résolution du TSP

On a proposé un algorithme pour la résolution du problème de voyageur de commerce avec des distances entre les villes.

4.5.1.1 Etapes de l'algorithme

L'algorithme proposé est décrit comme suit :

Entrée : graphe G, nœuds : V_{in} [1] et V_{out}

Etape 1 :

a. Assigner aléatoirement tous les nœuds du graphe avec des séquences d'ADN.

b. Assigner aléatoirement toutes les distances, du graphe G, avec des séquences d'ADN comme suit :
Si distance = 1 **alors Distance** = coder aléatoirement la distance par un des nucléotides d'ADN (A, T, G, C) => **Distance** = A ou **Distance** = T ou **Distance** = C ou **Distance** = G**.**
Sinon
Si distance = 2 **alors Distance** = coder aléatoirement la distance par deux nucléotides d'ADN (A, T, G, C) => **Distance = AT** ou **Distance = CT** ou…..
Si distance = 3 ……..

Donc

Chaque valeur de distance = nombre aléatoire de nucléotide d'ADN

Etape 2 :

Appliquant l'algorithme d'Adleman avec des modifications :

a. générer aléatoirement des chemins binaires (entre deux sommets) à partir du graphe G et garder seulement ceux qui commencent par $V_i \neq V_{out}$ et se terminent par $V_j \neq V_{in}$ (durant la génération, s'il y a une apparition de nœud V_{out} qui est le nœud de fin (ville d'arrivée) alors, STOP génération !) (Voir exemple).

b. générer aléatoirement des chemins comme suit :
Durant la génération : **Si** (il y a une répétition d'un nœud dans le chemin) ou (apparition du nœud V_{out} qui est le nœud d'arrivée et le nombre des nœuds dans le graphe est inférieur au nombre totale des nœuds dans le graphe G) **alors** STOP. (voir l'exemple)

La sortie des étapes a et b est : seulement des chemins qui commencent par V_{in} ***et se terminent par*** V_{out} **.**

- Soit l_i *la* longueur des chemins trouvés dans cette étape (i = de 1 au nombre de chemins trouvés dans cette étape)

Etape 3 :

La sortie de l'étape 2 (chemins trouvés) est l'entrée de cette étape.

a. insérer chaque séquence d'ADN représentant la distance (qui est équivalente à la distance qui se trouve entre les deux nœuds précisés) après chaque longueur de chaque

[1] Le V vient de mot anglais « vertex » qui veut dire sommet ou nœud. Vin est le sommet d'entrée et Vout est le sommet de sortie.

ville comme il a été codé dans le début (4 bases d'ADN dans notre exemple qui représentent le code de chaque chemin binaires (chemin entre deux nœuds)).

b. Après l'étape précédente, le chemin le plus cours sera trouvé en calculant sa distance à travers l'équation suivante : $D=n-m*s$ où :

D est la distance du chemin hamiltonien, n : est le nombre totale des nucléotides dans le chemin, s : est la longueur des bases d'ADN codant chaque ville (sommet), m : est le nombre des chemins binaires qui existe dans le chemin sachant que m= l_i / **s.** (*Li : longueur des chemins trouvés dans l' étape2*)

4.5.1.2 Exemple d'application

Pour appliquer cet algorithme, On a choisi d'utiliser un graphe de sept villes de l'Algérie (i.e ces 7 villes sont situées dans l'est de l'Algérie) avec des distances comme le montre la figure 4.9.

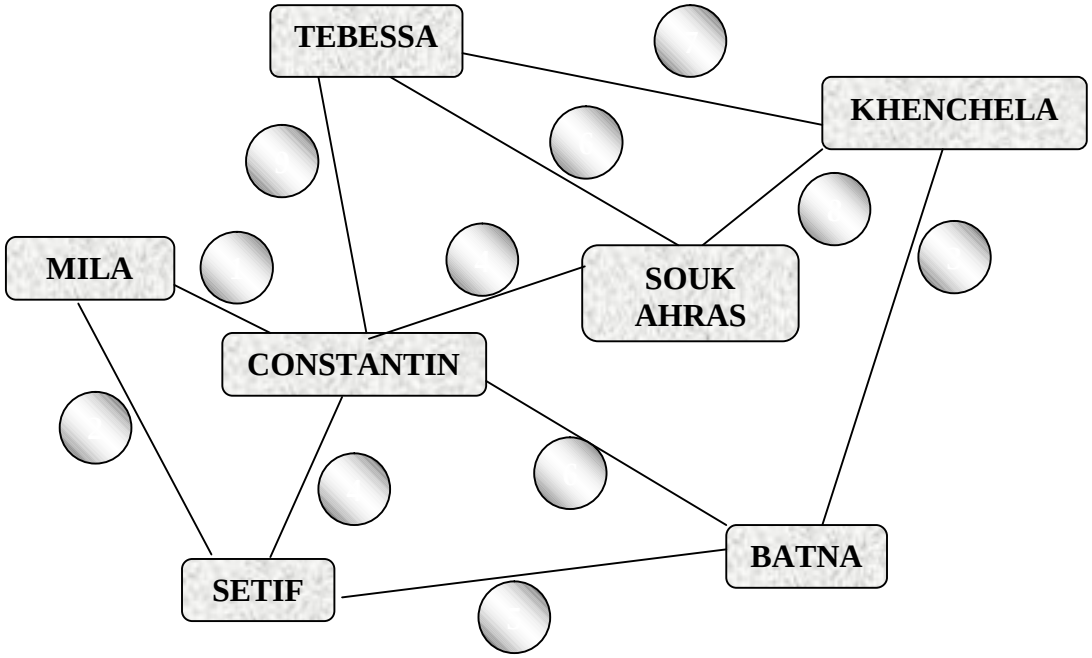

Figure 4.9 : Un graphe de 7 villes avec des distances entre eux.

*Entrée***:** graphe G (de la figure 4.9), Nœud : V_{in} est BATNA *et* V_{out} est MILA.

1. a et b: séquences aléatoires des nœuds et distances :

Nœuds :

	DNA code	*Complement*
BATNA	***TATT***	***ATAA***
CONST.	***GCGG***	***CGCC***
SETIF	***GTCG***	***CAGC***
MILA	***AGAA***	***TCTT***
TEBES.	***ACGT***	***TGCA***
KHEN.	***CGAT***	***GCTA***
SOUK	***TGTC***	***ACAG***

Distances*:*

Weights	*DNA code*
1	***A***
2	***TA***
3	***ACT***
4	***AGCT***
5	***GCGAT***
6	***GTACAG***
7	***CGTAGGT***
8	***ACTTCGAT***
9	***AATCGGATC***

2. L'algorithme d'Adleman' avec modifications:

a. Générer des brins d'ADN pour tous les chemins binaires possibles en éliminant ceux qui commencent par Mila et se terminent par Batna comme suit:

BATNA -> CONST.: TTGC √
BATNA -> SETIF:TTGT √
BATNA à ***KHEN.: TTCG*** √
CONST. -> SETIF:GGGT √
CONST. ->MILAC:GGAG √
CONST. ->BATNA:GGTA ×
CONST.-> TEBE. : GGAC √
CONST.-> SOUK: GGTG √
SETIF -> CONST.:CGGC √
SETIF ->MILA:CGAG √
SETIF -> BATNA:GTTA ×
KHEN.->TEBE. : ATAC √
KHEN.->BATNA : ATTA ×
KHEN.->SOUK. : ATTG √
MILA -> SETIF:AA STOP ×
MILA -> CONST.:AA STOP ×
TEBE-> CONST.:GTGC √
TEBE.->KHEN.: GTCG √
TEBE.->SOUK.: GTTG √
SOUK->CONST:. TCGC √
SOUK->KHEN.: TC CG √
SOUK->TEBE.: TCAC √

b. Générer aléatoirement tous les chemins possibles en utilisant les chemins binaires de l'étape précédente, comme suit: (ie : ici on va générer quelques chemins)

BATNA-CONST-SETIF-MILA

ATAACGCCCAGCTCTT
TTGCGGGTCGAG
B->C C->S S->M
! STOP apparition de noeud M qui est le noeud de fin (noud d'arrivée) et le nombre de nœud à ce moment est 4 (≠7)

BATNA-CONST.-SETIF-CONST.

ATAACGCCCAGC
TTGCGGGTCGGC
B->C C->S S->C
! STOP répétition de noeud C.

BATNA-SETIF-CONST.-MILA

BATNA-SETIF-MILA

ATAACAGCTCTT
TTGTCGAG
B->S S-M
! STOP apparition de noeud M qui est le noeud de fin et le nombre de nœuds à ce moment est 3 (≠7)

ATAACAGCCGCCTCTT
TTGTCGGCGGAG
B->S S->C C->M
! STOP apparition de noeud M qui est le noeud de fin (noeud d'arrivée) et le nombre de nœud à ce moment est
4 (≠7)

BATNA-CONST.-MILA

ATAACGCC
TTGCGGAG
B->C C->M
! STOP apparition de noeud M qui est le noeud de fin et le nombre de des noeuds =3 (≠7)

BATNA-SETIF-CONST. SETIF

ATAACAGCCGCC
TTGTCGGCGGGT
B->S S->C C->S
! STOP répétition de noeud S

BATNA-KHEN.-TEBE.-SOUK-CONST.SETIF-MILA

ATAAGCTATGCAACAGCGCCCAGCTCTT
TTCGATACGTTG TCGCGGGT CGAG
B->K K-T T-H H-C C-S S-M
SUCCES, le nombre des noeuds est 7

BATNA-KHEN.-SOUK- TEBE.-CONST.SETIF-MILA

ATAAGCTAACAGTGCACGCCCAGCTCTT
TTCGATTG TCAC GTGCGGGTCGAG
B->K K-H H-T T-C C-S S-M
SUCCES, le nombre des noeuds est 7

Remarque :
B : BATNA, C : CONST., S : SETIF, M: MILA, K: KHENCHELA, T: TEBESSA, H : SOUK AHRAS

Donc, on a deux chemins correctes, commençant par Batna, se terminant par Mila et passant une seule fois par chaque ville :

- Chemin1: ***BATNA-KHENCHELA.-TEBESSA.-SOUK AHRAS-CONSTANTINE-SETIF-MILA***
  - Codés par le brin d'ADN suivant : ***TTCGATACGTTG TCGCGGGTCGAG***
  - Avec une longueur l_1 =length (Chemin1)= 24
- Chemin2: ***BATNA-KHENCHELA-SOUK AHRAS- TEBESSA-CONSTANTINA-SETIF-MILA***
  - Codés par le brin d'ADN suivant : ***TTCGATTGTCACGTGCGGGTCGAG***
  - Avec une longueur l_2 = length (Chemin2)= 24

3- L'entrée de cette étape est deux chemins de l'étape précédente :
a- Affectation des distances (codées par des séquences d'ADN) aux chemins binaires:

Chemins Binaires trouvés dans l'étape précédente.	Distances codées en ADN
BATNA-KHENCHELA	***ACT (Distance=3)***
KHENCHELA.-TEBESSA	***CGTAGGT (Distance=7)***
TEBESSA.-SOUK AHRAS	***GTACAG (Distance=6)***
SOUK AHRAS-CONSTANTINE	***AGCT (Distance=4)***

CONSTANTINE-SETIF	***AGCT (Distance=4)***
SETIF-MILA	***TA (Distance=2)***
KHENCHELA-SOUK AHRAS	***ACTTCGAT (Distance=8)***
SOUK AHRAS- TEBESSA	***GTACAG (Distance=6)***
TEBESSA-CONSTANTINA	***AATCGGATC (Distance=9)***

b- Insérer les distances dans les deux chemins Hamiltoniens après chaque 4 nucléotides d'ADN dans l'ordre comme suit :

Chemin1: ***TTCGATACGTTGTCGCGGGTCGAG***

=> ***TTCG ACT ATAC CGTAGGT GTTG GTACAG TCGC AGCT GGGT AGCT CGAG TA***

Chemin2: ***TTCGATTGTCACGTGCGGGTCGAG***

=> ***TTCG ACT ATTG ACTTCGAT TCAC GTACAG GTGC AATCGGATC GGGT AGCT CGAG TA***

c- Calcul de distance***:*** D = n - s * m

Chemin1: n = **50**, s = **4**, m = l_1 / s => m=24/4=**6** => **D1=50-6*4=26**

Chemin2: n = **56**, s = **4**, m = l_2 / s => m= 24/4=**6** => **D2=56-6*4=32**

Le chemin Hamiltonien le plus cours est: ***BATNA-KHENCHELA.-TEBESSA.-SOUK AHRAS-CONSTANTINE-SETIF-MILA***
avec une distance de D=26

4.5.1.3 Discussion

On peut dire que cet algorithme est caractérisé par l'efficacité et l'optimisation dans le temps et même dans l'espace, surtout dans :

a- Etape 2, où la génération des chemins va être arrêtée si elle ne respecte pas ces deux règles:

1. Continuer la génération des chemins s'il n'y a pas une répétition des nœuds, sinon ERROR Stop!
2. Continuer la génération des chemins s'il n'y a pas une apparition du noeud de fin, sinon si le nombre total des nœuds trouvés dans ce chemin est inférieur à celui du graphe, alors ERROR Stop! Sinon SUCCESS.

b- l'insertion des distances dans le chemin ne sera pas faite au début de l'algorithme comme d'autres algorithmes (Ex: l'algorithme de Narayanan et Zorbalas [117]) au contraire, les distances sont insérées dans les chemins dans l'étape finale de l'algorithme.

On a vu les deux algorithmes de Narayanan et Zorbalas, on peut dire exactement ce que dit Zuwairie Ibrahim *et al*. dans son article 'Towards Solving Weighted Graph Problems by Direct-Proportional Length-Based DNA Computing'[121] : "A constant increase of DNA strands is encoded according to the actual length of the distance. A drawback of this method is that, there is a possibility of an occurrence of concatenated DNA strands of two distances which could be longer than the DNA strand of the longest distance that has been encoded". This may lead to errors in computing the shortest path [120]. Pour cette raison, on a choisi un

nouveau algorithme avec une nouvelle idée pour résoudre ce problème de voyageur de commerce où :

-Les distances sont codées aléatoirement avec des séquences d'ADN, dont le nombre de nucléotides égale à celui de la distance réelle (ex: A pour 1, GT pour 2, etc.) comme on a vu précédemment. Par contre le deuxième algorithme de Narayanan et Zorbalas [117] code les distances par incrémentation à travers un facteur constant K; par exemple, si K=3 et distances = {2,5,9,10}, alors 2 est représenté par un brin d'ADN de longueur 3, 5 par un brin de longueur 6, 9 par un brin de 9, et 10 par un brin de 12.

- Les séquences d'ADN des distances sont insérées dans la dernière étape de cet algorithme, comme il est bien expliqué précédemment et cela signifie qu'on a gagné plus de temps et même d'espace pour donner le chemin correct et qui est le plus cour comme solution pour ce type de problème de voyageur de commerce. Narayanan et Zorbalas ont choisi d'insérer les distances dans la première étape de leurs deux algorithmes, et cela va prendre beaucoup de temps et, surtout, beaucoup d'espace pour donner le chemin le plus cours.

On a appliqué cet algorithme en langage de Matlab avec succès et cela avec des graphes de 4, 7, 8, 10 villes (nœuds).

L'algorithme a réussi de donner le chemin le plus cours avec sa distance.

Cet algorithme de TSP a été présenté dans une conférence de Grèce : International Conference of Computational Methods in Sciences and Engineering 2005 (ICCMSE 2005) (i.e : la lettre d'acceptation est à la fin du ce mémoire). Il a été lu et accepté par des lecteurs experts. Il est aussi accepté dans la conférence de Alger COSI'06.

4.5.2 Nouveaux algorithmes proposés pour la résolution de K-SAT

Plusieurs algorithmes ont été proposés pour la résolution du problème de satisfaisabilité. Jusqu'à maintenant il n' y a pas un algorithme efficace pour la résolution du ''K-SAT problem'', la plus part des algorithmes proposés ont été adressés pour la résolution du problème de 3-SAT. En utilisant le calcul d'ADN, il y a des chercheurs qui ont réussi de résoudre le 3-SAT comme Wenbin Liu *et al* qui a proposé, dans cette année 2005, une nouvelle idée en utilisant le « Plasmids DNA » et appliquant l'algorithme de Schöning [133] dans son article « A Random Walk DNA Algorithm for the 3-SAT Problem » [132], qui est une idée très efficace et très utile pour résoudre ce genre de problème.

Dans cette section, deux nouveaux algorithmes pour la résolution de K-SAT problem seront présentés, expliqués selon des exemples et discutés par la suite en donnant ses avantages et les difficultés rencontrés.

4.5.2.1 Algorithme1:

4.5.2.1.1 Les étapes de l'algorithme1:

Entrée: la formule *F* avec *n* variables ou la négation de la formule F

1. Assigner chaque variable des *n* variables de la formule F, par une séquence d'ADN, ainsi que sa négation : coder aléatoirement chaque variable des n variables simultanément. La négation de chaque variable est codée par le complément d'ADN de la séquence d'ADN de la variable qui représente sa négation (voir l'exemple).

2. A l'aide de l'enzyme de ligase, générer tout les brins simples d'ADN possibles (i.e: éviter de baisser la température pour ne pas créer des brins complémentaires).
3. A l'aide de l'opération de PCR, extraire les séquences d'ADN qui commencent par des nucléotides codant la première variable des clauses de la formule (sans négation) et se terminent par des nucléotides codant la dernière variable des clauses de la formule (la variable et sa négation), et les mettre dans une autre éprouvette.
4. A l'aide de l'opération de l'électrophorèse, et à partir de cet ensemble de séquences d'ADN, extraire seulement les séquences d'ADN qui ont une longueur égale à la longueur d'une séquence codant une clause de la formule.
5. Extraire à partir de cet ensemble, les séquences qui ont l'apparition des nucléotides codant les variables dans l'ordre de leur apparition dans la clause (prendre en considération la $4^{ième}$ variable absente dans chaque clause).
6. A basse température (autours de 60°), créer des brins d'ADN complémentaires aux séquences d'ADN qui se trouvent dans l'éprouvette.
7. A haute température (autours de 90°), séparer les fragments de doubles brins à des brins simples (à l'aide de l'opération de dénaturation).
8. Les séquences d'ADN et leurs compléments trouvant dans l'éprouvette représentent les solutions possibles (2^n : n: nombre de variables de F) pour que la formule *F* soit toujours vraie où :
   Solution = sélectionner aléatoirement un brin simple d'ADN à partir de l'ensemble des séquences d'ADN et leurs compléments existent dans l'éprouvette.

4.5.2.1.2 Exemple d'application

On peut résumer et appliquer cet algorithme théoriquement pour résoudre un problème de 3-SAT comme suit:

Entrée: la formule *F* de 3-SAT:

$$F = (w \vee x \vee \neg y) \wedge (w \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg w \vee \neg x \vee \neg z)$$

1. Coder chaque variable comme suit:

w = TGC	w'= ACG	(w' est $\neg w$)
x = TAG	x'=ATC	(x' est $\neg x$)
y = GCT	y'=CGA	(y' est $\neg y$)
z = AAC	z'= TTG	(z' est $\neg z$)

2. Générer tout les brins simples d'ADN possibles comme suit:

Remarque1 : si on a une formule de 4 variables, le cas de notre exemple, et chaque clause a 3 variables, il faut prendre en considération l'état de la quatrième variable qui est en absence. Exemple : $C = (\neg w \vee x \vee y)$, on remarque que z est en absence, alors il faut insérer les deux états de (z et $\neg z$). Dans ce cas, on obtient 2 clauses, une contient la variable z : $C = (\neg w \vee x \vee y \vee z)$, et l'autre contient la négation de la variable z: $C = (\neg w \vee x \vee y \vee \neg z)$.

Remarque2: Il faut prendre en considération qu'il y a une quantité très élevée des nucléotides codant des trillions de brin d'ADN. Dans cet exemple on va générer quelques séquences d'ADN.

Clause	*DNA code*
wxyz	TGCTAGGCTAAC
wxyz'	TGCTAGGCTTTG
wxy'z	TGCTAGCGAAAC
wxy'z'	TGCTAGCGATTG
wxxyy'z	TGCTAGTAGGCTCGA
wyyyxzz'	TGCGCTGCTGCTTAGAACTTG
wx'yz	TGCATCGCTAAC
wx'yz'	TGCATCGCTTTG
wx'y'z	TGCATCCGAAAC
wx'y'z'	TGCATCCGATTG
zx'yy'ww'	AACATCGCTCGATGCACG
w'xyz	ACGTAGGCTAAC
w'x'y'z'	ACGATCCGATTG
wz'wxy'x'z'xzy'yw'z	TGCTTGTGCTAGCGAATCTTGTAGAACCGAGCTACGAAC
…etc.	

3. Extraire les séquences d'ADN qui commencent par w et finissent par z ou z':

wxyz ⟶ TGCTAGGCTAAC
wxyz' ⟶ TGCTAGGCTTTG
wxy'z' ⟶ TGCTAGCGATTG
wx'yz' ⟶ TGCATCGCTTTG
wx'y'z' ⟶ TGCATCCGATTG
wxxyy'z ⟶ TGCTAGTAGGCTCGA
wxy'z ⟶ TGCTAGCGAAAC
wx'yz ⟶ TGCATCGCTAAC
wx'y'z ⟶ TGCATCCGAAAC
wz'wxy'x'z'xzy'yw'z ⟶ TGCTTGTGCTAGCGAATCTTGTAGAACCGAGCTACGAAC
wyyyxzz' ⟶ TGCGCTGCTGCTTAGAACTTG

4. A l'aide de l'électrophorèse sélectionner les séquences qui ont une longueur correcte (ici la longueur totale d'une séquence d'ADN codant une clause est de 12 nucléotides):

wxyz' ⟶ TGCTAGGCTTTG
wxy'z' ⟶ TGCTAGCGATTG
wx'yz' ⟶ TGCATCGCTTTG
wx'y'z' ⟶ TGCATCCGATTG
wxy'z ⟶ TGCTAGCGAAAC
wx'yz ⟶ TGCATCGCTAAC
wx'y'z ⟶ TGCATCCGAAAC
wxyz ⟶ TGCTAGGCTAAC

5. Sélectionner les séquences qui ont l'apparition des variables dans l'ordre w,(x/x'),(y/y'),(z/z'):

wxyz' ⟶ TGCTAGGCTTTG
wxy'z' ⟶ TGCTAGCGATTG
wx'yz' ⟶ TGCATCGCTTTG
wx'y'z' ⟶ TGCATCCGATTG
wxy'z ⟶ TGCTAGCGAAAC
wx'yz ⟶ TGCATCGCTAAC
wx'y'z ⟶ TGCATCCGAAAC
wxyz ⟶ TGCTAGGCTAAC

6. Créer des séquences complémentaires pour les différentes séquences de l'étape précédente comme suit :

Clause	*DNA code*	*Complement*	*clause complémentaire*
wxyz'	TGCTAGGCTTTG	ACGATCCGAAAC	w'x'y'z
wxy'z	TGCTAGCGAAAC	ACGATCGCTTTG	w'x'yz'
wxy'z'	TGCTAGCGATTG	ACGATCGCTAAC	w'x'yz
wx'yz	TGCATCGCTAAC	ACGTAGCGATTG	w'xy'z'
wx'yz'	TGCATCGCTTTG	ACGTAGCGAAAC	w'xy'z
wx'y'z	TGCATCCGAAAC	ACGTAGGCTTTG	w'xyz'
wx'y'z'	TGCATCCGATTG	ACGTAGGCAAAC	w'xyz
wxyz	TGCTAGGCTAAC	ACGATCCGATTG	w'x'y'z'

7. Les séquences d'ADN et leurs compléments de l'étape précédente représentent les solutions possibles pour que la formule F soit vraie:

Les solutions sont: 2^n (n: nombre de variables) $\Rightarrow$ 2^4 =16 solutions possibles qui sont les 16 séquences d'ADN: {TGCTAGGCTTTG ACGATCCGAAAC TGCTAGCGAAAC ACGATCGCTTTG TGCTAGCGATTG ACGATCGCTAAC TGCATCGCTAAC ACGTAGCGATTG TGCATCGCTTTG ACGTAGCGAAAC TGCATCCGAAAC ACGTAGGCTTTG TGCATCCGATTG ACGTAGGCAAAC TGCTAGGCTAAC ACGATCCGATTG}.

Par exemple:

- On va sélectionner aléatoirement: ACGATCCGATTG
- Tester la formule F par cette séquence d'ADN en remplaçant chaque 3 nucléotides par sa variable équivalente comme elle était représentée dans l'étape 1:
  ACGATCCGATTG (qui représente w'x'y'z') $\Rightarrow$ $\neg w \vee \neg x \vee \neg y \vee \neg z$
  On a la formule F est:

$$F = (w \vee x \vee \neg y) \wedge (w \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg w \vee \neg x \vee \neg z)$$

La séquence d'ADN ACGATCCGATTG met la première clause, la deuxième, la troisième et la quatrième clause vraie $\Rightarrow$ F est vraie.

4.5.2.1.3 Discussion:

Pour ce problème de K-SAT, qui est un problème NP-complet, on a proposé un chemin différent et une autre méthode pour donner toutes les solutions possibles à ce problème en utilisant le calcul de l'ADN. Le nombre de solution de problème de K-SAT est 2^n (n: nombre de variables), si le nombre de variables est très élevé, ainsi que le K de satisfiabilité, le problème sera très difficile à résoudre avec nos ordinateurs conventionnels.

Théoriquement l'algorithme est simple et peut être très efficace pour donner toutes les solutions possibles (2^n). Nous avons appliqué cet algorithme en langage de Matlab sur nos ordinateurs conventionnels avec succès où le nombre de variable ne doit pas être très élevé. Nous avons exécuté cet algorithme pour K=3, n=4 et K=4, n=5 plusieurs fois, et nous avons conclus que la probabilité d'obtenir 2^n solutions dépend toujours de nombre de séquence

d'ADN utilisé dans l'étape2 qui doit être très élevé (biologiquement le problème ne se pose pas). L'exécution de l'exemple d'application de 3-SAT précédent en Matlab a donné des bons résultats. Le nombre de solution exacte de 3-SAT avec 4 variables est 16 solutions. L'exécution de ce problème en donnant toutes les solutions possibles nécessite une quantité énorme de séquences d'ADN. Voila quelques exemples de résultats obtenus en exécutant cet algorithme (en Matlab, avec une mémoire d'une capacité de 128 M∅ et un processeur de 2.1Gh) jouant sur le nombre de séquences utilisé:

Nombre d'exécution	Nombre de séquences	Nombre de solution	Temps pour chaque exécution
10	10	2	Moins de 1s
10	50	2	Moins de 1s
10	100	2	Moins de 1s
20	1000	[2-10]	Moins de 1mn
15	10000	[10-16](16 apparu plusieurs fois)	2 mn
3	100000	16	2 heurs

Imaginons si on utilise biologiquement (expérience réelle) dans l'étape 2 un nombre de trillion de séquences d'ADN pour chercher toutes les solutions possibles, la probabilité d'obtenir le nombre de solutions exacte sera très élevée, autrement dit: le nombre de solution dépend du nombre de séquence d'ADN utilisé, où plus le nombre de séquences d'ADN augmente plus le nombre de solutions augmente et vice versa. Vraiment c'est cela qui traduit la puissance des ordinateurs à l'ADN: la capacité de stockage et la puissance d'exécution.

Pratiquement ou biologiquement, la difficulté d'application dépend toujours de type d'opération utilisé dans l'expérience. Pour notre algorithme l'opération de PCR dans l'étape 3 et celle de l'électrophorèse dans l'étape 4 rendent l'algorithme un peu difficile où on peut avoir des trillions de séquences d'ADN qu'il faut les filtrer à travers ces deux opérations.

On a aussi participé avec cet algorithme de satisfaisabilité dans la même conférence du Grèce : International Conference of Computational Methods in Sciences and Engineering 2005 (ICCMSE 2005) (i.e : la lettre d'acceptation est à la fin de ce mémoire), et dans la conférence de Saida CIIA06.

4.5.2.2 Algorithme 2:

4.5.2.2.1 Les étapes de l'algorithme 2:

Nous avons proposé un autre algorithme basé sur le principe de l'algorithme qu'on a proposé pour la résolution de problème de TSP dont les étapes sont les suivantes:

Entrée : graphe G, nœuds : V_{in1} , V_{in2} et V_{out1} , V_{out2}

Etape 1 : Assigner aléatoirement tous les nœuds du graphe avec des séquences d'ADN (coder les variables et leurs négations avec des séquences d'ADN différentes).

Etape 2 : Générer aléatoirement des chemins binaires (entre deux sommets) à partir du graphe G et garder seulement ceux qui commencent par $V_i \neq V_{out1}$ et $V_i \neq V_{out2}$ et se terminent par $V_j \neq V_{in1}$ et $V_j \neq V_{in2}$ (durant la génération, s'il y a une apparition de nœud

V_{out1} ou V_{out2} qui sont les nœuds de fin (villes d'arrivée) alors, STOP génération !) (Voir exemple).

Etape3: Générer aléatoirement des chemins comme suit :

Durant la génération : **Si** (il y a une répétition d'un nœud dans le chemin) ou (apparition du nœud V_{out1} ou V_{out2} qui sont les nœuds d'arrivée et le nombre des nœuds dans le graphe est inférieur ou supérieur au nombre totale des nœuds dans le graphe G) **alors** STOP.

Etape4 : Les chemins sortants de l'étape 2 sont les solutions possibles pour que la formule F soit toujours vraie ou satisfaite.

4.5.2.2.2 Exemple d'application:

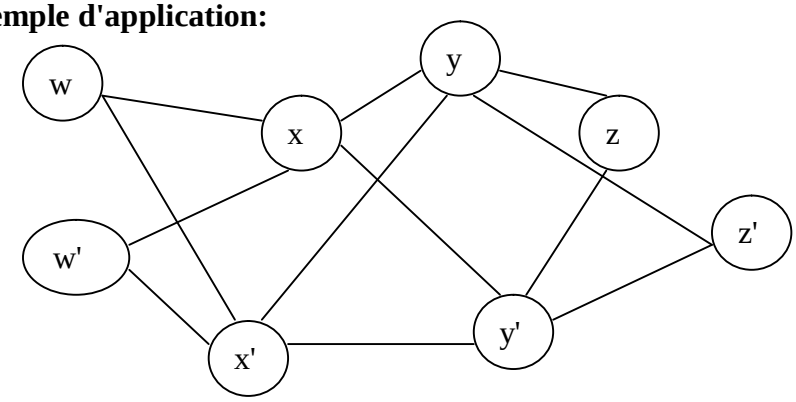

Figure 4.10 : Un graphe de 8 villes.

*Entrée***:** Formule sous forme d'un graphe: Nœuds d'entée : V_{in1} : w, V_{in2} : w' et nœuds de sortie V_{out1} : z , V_{out2} : z' la formule *F* de 3-SAT:

$$F = (w \vee x \vee \neg y) \wedge (w \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg w \vee \neg x \vee \neg z)$$

Etape1:

Dans cette étape on va coder les variables de la formule ainsi que leurs négations par des séquences d'ADN aléatoirement comme suit:

Nœuds :

	DNA code	*Complement*
w	***TATT***	***ATAA***
w'	***GCGG***	***CGCC***
x	***GTGC***	***CACG***
x'	***AGAA***	***TCTT***
y	***ACGT***	***TGCA***
y'	***CGAT***	***GCTA***
z	***TGTC***	***ACAG***
z'	***AACG***	***TTGC***

Etape2:

On va générer quelques chemins binaires (plus de détail voir l'exemple d'application de l'algorithme de TSP) comme suit:

w à ***x.: TTGT*** √
x à ***z:GCTG*** √
w' à ***x':GGAG*** √
x' à ***w:AATA*** ×
y à ***w':GTGC*** ×
z à ***x:TC STOP*** ×
z' à ***x' : CG STOP*** √
x à ***y: GCAC*** √

x à ***y':GCCG***
y' à ***x:ATGT***
y à ***x':GTAG***
x' à ***z':AAAA***
y' à ***z:ATTG***
w à ***x':TTAG***
x' à ***y:AAAC***
y à ***z:GTTG***

x' à ***y':AACG***

Etape3:

Générer aléatoirement tous les chemins possibles en utilisant les chemins binaires de l'étape précédente, respectant les conditions de génération mentionnées dans l'algorithme, comme suit: (i.e : ici on va générer quelques chemins)

ATAACACGTGCATCTT
TTGTCGACGTAGAAAA
w->x x->y y->x' x'->z'
! STOP apparition de noeud z' qui est le noeud de fin (noud d'arrivée) et le nombre de nœud à ce moment est 5 (≠4: supérieur au nombre de variable de SAT)

BATNA-CONST.-SETIF-CONST.

ATAACGCCCAGC
TTGCGGGTCGGC
w'->x x->y y->z
! STOP répétition de noeud C.

ATAACACG
TTGTGCTG
w->x x->z
! STOP apparition de noeud z qui est le noeud de fin et le nombre de nœuds (variables) à ce moment est 3 (≠4)

ATAACACGGCTA
TTGTGCCGATGT
w->x x->y' y'->x
! STOP répétition de noeud x

ATAACACGGCTAACAG
TTGTGCCGATTG
w->x x->y' y'->z
SUCCES, le nombre des nœuds (variable) est 4

ATAATCTTTGCAACAG
TTAGAAACGTTG
w->x' x'->y y->z
SUCCES, le nombre des nœuds (variables) est 4

CGCCTCTTGCTAACAG
GGAGAACGATTG
w'->x' x'->y' y'->z
SUCCES, le nombre des noeuds (variables) est 4

Etape4:

Les solutions possibles sont 2^n (n:nombre de variables) $\Rightarrow$ $2^4 = 16$ solutions possibles qui se traduisent dans les 16 chemins Hamiltoniens possibles sortant de l'étape précédente:

Les trois chemins de l'étape précédente représentent 3 solutions parmi les 16 solutions qu'on doit avoir à la fin de l'expérience.

Ch1: TTGTGCCGATTG w->x x->y' y'->z

Ch2 : TTAGAAACGTTG w->x' x'->y y->z

Ch3:GGAGAACGATTG w'->x' x'->y' y'->z

4.5.2.2.3 Discussion

Le problème de SAT peut être représenté sous forme d'un graphe avec deux nouds d'entrée (la première variable et sa négation) et deux nœuds de sortie (la dernière variable et sa négation) et un ensemble des nœuds représentant le reste des variables ainsi que leurs négation.

Nous avons présenté une discussion de cet algorithme, ci-dessus, pour la résolution de TSP. De la même manière le problème de SAT peut être résolu mais cette fois sans distances entre les nœuds (cas de résolution de problème de chemin Hamiltonien). Cet algorithme va nous permettre de trouver l'ensemble de solutions possibles pour que la formule soi satisfaite (2^n solution. n: nombre de variable de la formule). Cet algorithme a été appliqué en langage de Matlab avec succès.

4.6 Conclusion

D'après ce qu'on a vu et ce qu'on a lu, on peut conclure que :

- En théorie de la complexité, un problème est formalisé de la manière suivante : un ensemble de données en entrée, et une question sur ces données (pouvant demander éventuellement un calcul). La théorie de la complexité ne traite que des problèmes de décision binaire, c'est-à-dire posant une question dont la réponse soit oui ou non.
- Nous avons vu que certains problèmes sont insolubles au sens pratique du terme. Il existe des solutions à ces problèmes, mais ces solutions ne sont pas applicables dans des temps raisonnables. Nous avons donc vu une classe de problèmes qui sont de ce type que nous avons appelé NP-complets et que ce type de problèmes sont insolubles en temps raisonnable (polynomial).
- Personne n'a réussi à trouver un algorithme polynomial pour résoudre un des problèmes NP-complets. Si un tel algorithme était trouvé, il pourrait d'office s'appliquer à tous les autres problèmes NP-complets. Une personne qui ne trouve pas de solution polynomiale à un problème NP-complet a donc une bonne excuse, personne d'autre n'a réussi. Par ailleurs, personne n'a pu prouver, à l'inverse, qu'une telle solution n'existait pas.
- Le calcul d'ADN ou le calcul moléculaire a réussi de résoudre les problèmes combinatoires tel que le problème de TSP (PVC:Problème de Voyageur de commerce), de SAT,…etc. où toutes les molécules d'ADN peuvent travailler en parallèles en effectuant des trillions de calculs simultanément.

Conclusion générale

Les êtres vivants sont constitués de véritables machines moléculaires (ADN, ARN, ribosomes, etc.), qui fonctionnent à l'échelle atomique, et agencent de façon extrêmement précise les atomes et les molécules qui constituent les êtres vivants... avec finalement beaucoup de succès!

Nous avons vu dans ce mémoire que certains problèmes sont insolubles au sens pratique du terme. Il existe des solutions à ces problèmes, mais ces solutions ne sont pas applicables dans des temps raisonnables. Nous avons donc vu une classe de problème qui sont de ce type et qui sont appelés NP-complet.

Pour résoudre ce type de problème complexe (NP-complet), la biologie moléculaire a joué un rôle très important. En 1994, Leonard Adleman fit une avancée spectaculaire dans le domaine des ordinateurs chimiques ou biochimiques. Il utilisa des fragments d'ADN pour résoudre informatiquement un problème complexe de théorie graphique. Tel un ordinateur comportant plusieurs processeurs, ce type d'ordinateur à ADN est capable de considérer simultanément plusieurs solutions à un problème. De plus, les brins d'ADN employés pour un tel calcul sont beaucoup plus nombreux et compacts que les processeurs électroniques des superordinateurs actuels. C'est le premier exemple de calcul effectué grâce à un ordinateur non-électronique.

Si on veut faire une comparaison entre les ordinateurs conventionnels et ceux d'ADN, on peut dire que la différence principale entre ces deux types d'ordinateur réside dans la capacité de leur mémoire. Les ordinateurs habituels (au transistor) fonctionnent en utilisant le système binaire, c'est-à-dire: des "oui" et des "non", autrement dit des "1" et des "0", par contre ceux à l'ADN en a quatre (A, C, G et T): **A** (adénine), **C** (cytosine), **G** (guanine) et **T** (thymine).

Un programme dans un ordinateur d'ADN est exécuté comme une série d'opérations biochimiques. Les entrées et les sorties comportent deux paires de brins d'ADN, dont les séquences génétiques sont considérées capable d'encoder certaines informations, c'est-à-dire synthétiser, extraire, modifier et cloner les brins d'ADN.

Les ordinateurs actuels opèrent de façon séquentielle, ils manipulent un bloc après l'autre contre le système parallèle de l'ADN. En une seule opération biochimique l'ADN peut être arrangée de façon à affecter des trillions de brins. Ce qui veut dire que ces propriétés peuvent résoudre des problèmes traditionnellement considérés comme insolubles.

L'avantage évident entre les deux types d'ordinateur réside dans la capacité de leur mémoire, où la capacité de stockage d'information dans ces molécules vivantes est énorme. Aussi, toutes les molécules d'ADN peuvent travailler en parallèles en effectuant dix trillions de calculs simultanément. Donc, Les ordinateurs à ADN peuvent envisager simultanément plusieurs solutions à un problème complexe, ils sont très légers, n'utilisent quasiment aucune énergie et résolvent des problèmes complexes très rapidement

Mais les PC à l'ADN, eux aussi, souffrent de certains désavantages. Les opérations sont souvent lentes et sujettes à des erreurs. Ce qui fait qu'on est encore au stade des tests de

la précision des opérations ce qui signifie que des développements ultérieurs sont nécessaires pour clarifier les potentiels effectifs de ces ordinateurs.

De plus, ils ne peuvent résoudre que des problèmes combinatoires (pas de possibilité de traitement de texte ou de jeu sur de tels ordinateurs), ils peuvent être très lents dans la résolution de problèmes simples pour des ordinateurs classiques. Les réponses qu'ils fournissent peuvent être extrêmement compliquées. D'autre part la fiabilité de ces ordinateurs peut être remise en cause du fait de la capacité de mutation de l'ADN.

Un autre défaut du calcul avec l'ADN est son ‘‘humidité’’ ! Ce sont des manipulations physiques qui peuvent accumuler beaucoup d'artefacts, et quand ils sont implantés, ils perdent beaucoup de leurs charmes. Adleman lui même n'a pas été très rigoureux dans son travail : pas de test par exemple pour vérifier que si il n'y a pas de chemin, ses manipulations donnent effectivement une réponse négative.

Des nouveaux algorithmes ont été proposés pour la résolution des problèmes NP-complets. Le premier est celui de voyageur de commerce (TSP). Théoriquement les étapes a suivre pour trouver la solution ou le chemin le plus cour est très logique et applicable sur nos ordinateurs de silicium. Mais, pratiquement ou biologiquement on ne peut pas garantir que l’algorithme sera bien exécuté en donnant la solution désirée, cela dépend de l’expérience biologique concrète.

Les deux autres algorithmes sont adressés pour résoudre le problème de la satisfiabilité (SAT). Théoriquement les deux algorithmes sont simples et peuvent être très efficaces pour donner toutes les solutions possibles (2^n ,n: nombre de variables de la formule à résoudre). Pratiquement ou biologiquement, la difficulté d'application dépend toujours de type d'opération utilisé dans l'expérience. Pour nos algorithmes l'opération de PCR et celle de l'électrophorèse rendent les deux algorithmes un peu difficiles à réaliser où on peut avoir des trillions de séquences d'ADN qu'il faut les filtrer à travers ces deux opérations.

Ces deux algorithmes pour la résolution de SAT peuvent être utilisés aussi pour résoudre le problème de chevalier dans les jeux d'échecs (en anglais Knight problem)

On peut résoudre ces deux problèmes -TSP et SAT- et d’autres problèmes NP-complets (comme le jeux d’échecs ‘Chess Problem’, le Tic Tac Toe, le sac à dos, coloration de graphe,….etc.) en introduisant d’autres principes biologiques tel que la méthode de construction de protéine où on peut rencontrer un autre type d’acide nucléique, c’est l’ARN (RNA). Cela signifie que la biologie est très riche avec ses idées surprenantes et elle va nous aider pour chercher ce qui est meilleur pour nos besoins.

Finalement, on espère que ce modeste travail a donnée un aperçu sur ce domaine de recherche très actif, intéressant, et défiant.

Bibliographie

[1] Centre de Documentation et d'Information sur l'Enseignement supérieur – CEDIES, Biologie - Biochimie – Biotechnologies, Ministère de la Culture, de l'Enseignement Supérieur et de la Recherche, édition 2004, 29/10/**2004.**

URL : http://www.cedies.lu

[2] Aurélie Deléglise , Quand l'ADN remplace l'électronique 03/12/**2001.**

URL : http://www. cybersciences.com

[3] Peter Habermehl, Cours Intelligence Artificielle **2004-2005.**

[4] Microsoft Encarta, **2005.**

[5] Hassiba Talbi, thèse de magister, Constantine **2004.**

[6] URL : http://www.sintef.no/

[7] la vie artificielle : URL :http://www.vieartificielle.com

[8] Reza GhasemAghaei & Fazel Keshtkar, Biologically Inspired Intelligent Systems, ELG5121 Term Project Fall **2004.**

[9] Bonabeau E., Dorigo M., et Theraulaz G. Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press. ISBN 0-19-513159-2, **1999**.

[10]Jean-Philippe Rennard, Auto-organisation chez les insectes sociaux, Ants Viewer, J.-Ph. Rennard 02/**2003.**

URL http://www.rennard.org/alife -- alife@rennard.org

[11] C. Bourjot, V. Chevrier, A. Bernard, and B. Krafft. Coordination par le biais de l'environnement: une approche biologique. In M. P. Gleizes and P. Marcenac, editors, Ingénierie des systèmes multi-agents: JFIADSMA'99, pages 237–250, Saint Denis de la Réunion, France, **1999**. Hermès.

[12] C. Bourjot and V. Chevrier. De la simulation de construction collective à la détection de régions dans les images à niveau de gris: l'inspiration des araignées sociales. In A. E. F. Seghrouchni and L. Magnin, editors, Fondements des systèmes multi-agents: JFIADSMA'01, pages 253–263, Montréal, Canada, **2001**. Hermès/Lavoisier.

[13] Moyson F. et Manderick B, « The Collective Behaviour of Ants: an Example of Self-Organisation in Massive Parallelism », Proceedings of the AAAI Spring Symposium on Parallel Models of Intelligence. Stanford, California, **1988**.

[14] Deneubourg J.-L., Pasteels J.-M. et Verhaeghe J.-C. « Probabilistic Behaviour in Ants: a Strategy of Errors ? », Journal of Theoretical Biology, 105, **1983**.

[15] Colorni A., Dorigo M. et Maniezzo V. « Distributed Optimization by Ant Colonies », Proceedings of the First European Conference on Artificial Life, MIT Press/Bradford Book, Paris, **1991**.

[16] Dorigo M, Optimization, Learning and Natural Algorithms, PhD thesis, Dipartimento di Electronica, Politecnico di Milano, IT, **1992**.

[17] Yann Semet, Pierre Collet, Application de l'optimisation par colonies de fourmis à la structuration automatique de parcours pédagogiques, Association EPI3e trimestre **2003**.

[18] J. P. Rennard, La Vie Artificielle, Avril **2003**.
URL: http://www.rennard.org/alife.

[19] J. Dréo, P. Siarry, Diverses techniques d'optimisation inspirées de la théorie de l'auto-organisation dans les systèmes biologiques, Séminaire OEP, 24 Octobre **2003**.

[20] Stützle T. et Dorigo M. « ACO Algorithms for the Travelling Salesman Problem », in M Makela, K Miettinen, P Neittaanmaki, J Periaux (Eds), Proceedings of the EUROGEN conference, John Wiley & Sons, ISBN: 0471999024, **1999**.

[21] Stutzle T. et Hoos H. « The Max-Min ant system and local search for the Travelling Salesman Problem », T. Baeck, Z. Mickalewicz and X. Yao, editors, Proceedings of IEEE-ICEC-EPS'97 International Conference on Evolutionary Computation and Evolutionary Programming, IEEE Press, p. 309-314., **1997**.

[22] MacGill, J. Smart Pattern Hunting Flocks, School of Geography, University of Leeds, **2000**.
URL: http://www.geog.leeds.ac.uk/people/j.macgill/alife.html 31 March **2002**.

[23] Reynolds, C. W. Flocks, Herds, And Schools: A Distributed Behavioural Model, in: Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings), pp 25-34, **1987.**
URL: http://www.red3d.com/cwr/papers/1987/boids.html 13 March **2002**.

[24] D. Phan, Les systèmes X agents : Boids – Reynolds, ENST Bretagne.
URL: http://digemer.enst-bretagne.fr/~phan/complexe/intro00.htm

[25] C. W. Reynolds, Boids : Background and update, Juin **1995**.
URL: http://www.red.com/cwr/boids.html

[26] C. W. Reynolds, Flocks, Herds, and Schools: A Distributed Behavioural Model, in the proceeding of Computer Graphics (SIGGRAPH'87), 21(4), pp.25-34, July **1987**.

[25] URL : http://www.red3d.com/cwr/boids/.

[26] Jean-Philippe Rennard, La Vie Artificielle, Avril **2003**.
URL : http://www.rennard.org/alife -- alife@rennard.org

[27] Hopfield, J. J., Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. USA, 79, 2554—2558, **1982**.

[28] Rosen, R., Essays on Life Itself, Columbia University Press, New York, **2000**.

[29] Rashevsky, N., Mathematical Biophysics: Physico-Mathematical Foundations of Biology,3rdedition,Vols.1and2,Dover,NewYork,**1960**.

[30] McCulloch, W. S. and W. Pitts, A logical calculus of the ideas of immanent in nervous activity. Bull. Math. Biophys., 5, 115—133, **1943**.

[31] Churchland, P. M., Cognitive activity in artificial neural networks, in An In-vitation to Cognitive Science, Volume 3: Thinking (D. N. Osherson and E. E. Smith, Eds.), pp. 199—227, MIT Press, Cambridge, MA, and London, **1990**.

[32] Chester, M., Neural Networks: A Tutorial, PTR Prentice-Hall, Englewood Cli .s, NJ, **1993**.

[33] Fu, L., Neural Networks in Computer Intelligence,McGraw-Hill,NewYork, St. Louis, San Francisco, Auckland, Bogotá, Caracas, Lisbon, London, Madrid, Mexico City, Milan, Montreal, New Delhi, San Juan, Singapore, Sydney, Tokyo and Toronto, **1994**.

[34] Gurney, K., An Introduction to Neural Networks, University College London Press, London, **1997**.

[35] Perlovsky, L. I., Neural Networks and Intellect: Using Model-Based Concepts, Oxford University Press, New York and Oxford, **2001**.

[36] Hu, Y. H. and J.-N. Hwang, Handbook of Neural Network Signal Processing, CRC Press, Boca Raton, London, New York and Washington, DC, **2002**.

[37] Les réseaux neuronaux artificiels.

[38] C. Touzet, Les réseaux de neurones artificiels, Juillet **1992**.

[39] De Jong, K., An analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis, University of Michigan, **1975**.

[40] Anne Spalanzani, Algorithmes évolutionnaires pour l'étude de la robustesse des systèmes de reconnaissance automatique de la parole,thèse de doctorat, 28 Octobre **1999**

[41] De Jong K.A., Learning with Genetic algorithm *:* An Overview. Machine Learning, vol. 3, pp. 121-138, **1988**.

[42] Eiben A.E., van der Hauw J.K. et van Hemert J.I., *Graph **Coloring** with Adaptive Evolutionary Algorithms*. Journal of Heuristics, vol. 4:1, pp. 25-46, 1998.

[43] Tao, G. and Michalewicz Z., *Inver-over Operator for the TSP*, Proceedings of the 5th Parallel Problem Solving from Nature, Springer-Verlag, Lecture Notes in Computer Science, Amsterdam, Septembre **1998**.

[44]Watson J.P., Ross C., Eisele V., Denton J, Bins J., Guerra C., Whitley D. et Howe A., *The Traveling Salesrep Problem, Edge Assembly Crossover, and 2-opt*. 5th Conference on Parallel Problem Solving from Nature, Springer Verlag, Amsterdam, Pays-Bas, **1998**.

[45]Paechter B., Rankin R.C., Cumming A. et Fogarty T.C., *Timetabling the Classes of an Entire University with an Evolutionary Algorithm*. 5th Conference on Parallel Problem Solving from Nature, Springer Verlag, Amsterdam, Pays-Bas, **1998**.

[46] Ahuactzin J-M., Mazer E. et Bessière P., *Fondements mathématiques de l'algorithme "fil d'Ariane"*, Revue d'Intelligence Artificielle, **1995**.

[47] Schultz A.C. et Grefenstette J.J., *Using a Genetic Algorithm to Learn Behaviors for Autonomous Vehicles*. AIAA Guidance, Navigation and Control Conference, Hilton Head, SC, **1992**.

[48] Heap T. et Samaria F., *Real-Time Hand Tracking and Gesture Recognition Using Smart Snakes*. Technical Report 95.1 AT&T Laboratories Cambridge, 24a Trumpington Street, Cambridge CB2 1QA, England, **1995**.

[49] Baluja S., *Finding Regions of Uncertainty in Learned Models : An Application to Face Detection.* 5th Conference on Parallel Problem Solving from Nature, Springer Verlag, pp. 663-671, Amsterdam, Pays-Bas, **1998**.

[50] Liu C. et Wechsler H., *Face Recognition Using Evolutionary Pursuit*. Fifth European Conference on Computer Vision, ECCV'98, Université de Freiburg, Allemagne, juin **1998**.

[51] Brouard T., Slimane M., Venturini G. et Asselin De Bauville J-P., *Apprentissage génétique hybride de chaînes de Markov cachées*. Apprentissage : des principes naturels aux méthodes artificielles, ed. Ritschard G., Berchtold A., Duc F.et Zighed D., éditions Hermès, pp.241-256, **1998**.

[52] Roux C. et Jacq J.J., Registration of successive DSA images using a simple genetic algorithm with a stochastic performance function. Proceeding of 19th IEEE Northeast Bioengineering Conf., Newark NJ, 223-224, **1993**.

[53] Iwata M., Kitani I., Yamada H., I ba H. et Higuchi T., *A Pattern Recognition System Using Evolvable Hardware*. 1st Conference on Parallel Problem Solving from Nature, Berlin, Allemagne, **1990**.

[54] Kane C. et Schoenauer M., *Optimisation Topologique de Formes parAlgorithmes Génétiques*. Revue Française de Mécanique 4, pp 237-246, **1997**.

[55] Geyer H., Ulbig P. et Schulz S., *Encapsulated Evolution Strategies for the Determination of Group Contribution Model Parameters in Order to Predict Thermodynamic Properties*. 5th Conference on Parallel Problem Solving from Nature, Springer Verlag, pp. 663-671, Amsterdam, Pays-Bas, **1998**.

[56] Fontanili F. et Vincent A., *Comment optimiser le fonctionnement d'un atelier avec la simulation de flux ?* Revue Française de Gestion Industrielle, vol. 16, n. 3, **1997**.

[57] Oussedik S. et Delahaye D., *Reduction of Air Traffic Congestion by Genetic Algorithms*. 5th Conference on Parallel Problem Solving from Nature, Springer Verlag, Amsterdam, Pays-Bas, **1998**.

[58] Belew B., McInerney J. et Schraudolph N., *Evolving Networks : Using the Genetic Algorithms with Connectionist Learning*. CSE Technical Report CS90-174, Computer Science, UCSD, **1990**.

[59] Whitley D., *Genetic Algorithms and Neural Networks*. Genetic Algorithms in Engineering and Computer Science. Ed. J. Periaux et G. Winter, **1995**.

[60] Goldberg D.E., Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Massachusetts, **1989**.

[61] Algorithmes évolutionnaires et problèmes inverses
URL:
http://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/poly/cours009.html

[62] Nathanael Paul, Randomness in Cellular Automata, CS851 – Biological Computing, February 6, **2003**

[63] J. P. Rennard, Automates cellulaires, Décembre **2000**.

[64] Vie Artificielle, **URL**: http://www.math-info.univ-paris5.fr/~latc/va/va.html

[65] http://www.geocities.com/tperz/L4Home.htm.

[66] joël de rosnay, de la biologie moléculaire à la biotique: l'essor des bio-, info- et anotechnologies, **2001**.

[67] Bart Selman,Henry Kautz et Bram Cohen. Local search Strategies for Satisfiability testing.Cliques,Coloring and Satisfiability :Second DIMACS Implemenrtation Challenge .Vol.26 p.521-531,**1996**.

[68] Bart Selman, Hector Levesque, et David Mitchell.A new method for solving hard satisfiability problems. Proceedings,AAAI-92,San Jose,CA,**1992**.

[69] JP Delahaye, Complexité, information**, 1999.**

[70] N. DOUZIECH, La complexité en informatique, juin **2004.**

[71] Marc Parizeau, Introduction à la NP-complétude, Hiver **1998.**

[72] Mange, D., D. Madon, A. Stau.er and G. Tempesti, Von Neumann revisited: a Turing machine with self-repair and self-production properties. Robot. Autonom. Syst., 22, 35—58, **1997**.

[73] Charles Lesire, Introduction à la complexité, **2004 – 2005.**

[74] Jin-Kao Hao, Philippe Galinier, Michel Habib, Méthaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. Revue d'Intelligence Artificielle, **1999.**

[75] M.E. Arhab et al., le nouveau dans les science : le système immunitaire **1997**.

[76] Bahram Houchmandzadeh, Ballade autour de quelques thèmes de biologie, 13 septembre **2004.**

[77] Emmanuelle HENRY, Molécules du vivant: ADNg, Octobre **2000.**

[78] Harun Yahyia, Le Miracle de la Création dans l'ADN, Harun Yahya International **2005.**
URL : http://www.harunyahya.com

[79] http://ead.univ-angers.fr/~jalouzot/genetique/courshtm/chap3/chap3-1.htm.

[80] J.D.Watson, F.H.C.Crick: Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid, Nature (London), 171,737-738,**1953.**

[81] Timothée Lionnet et Vincent Croquette, Introduction à la Biologie Moléculaire, 25 mars **2005.**
URL : http://www.phys.ens.fr/~biolps/

[82] Hélène Antaya, Isabelle Ascah-Coallier, L'ordinateur à l'ADN.

[83] L. Kari, G. Thierrin, Contextual insertions/deletions and computability, Information and computation, **131** numéro 1, pp.47-61, **1996**.

[84] L. M. Adleman, Molecular computation of solutions to combinatorial problems, Science 266, 1021-1024, **1994**.
URL ftp://ftp.krl.caltech.edu/pub/users/brown/adleman.ps.gz

[85] R. J. Lipton, Speeding up computations via molecular biology, Unpublished manuscript, **1994**.
URL ftp://ftp.cs.princeton.edu/pub/people/rjl/bio.ps

[86] L. M. Adleman, On constructing a molecular computer, Vol. 27 of DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, **1996**.
URL http://citeseer.nj.nec.com/adleman95constructing.html

[87] D. Rooss, K. W. Wagner, On the power of DNA-computers, Tech. Rep. 103, University of W. urzburg **1995**.
URL http://citeseer.nj.nec.com/55405.html

[88] E. Bach, A. Condon, E. Glaser, C. Tanguay, DNA models and algorithms for NP-complete problems, in: Proceedings, Eleventh Annual IEEE Conference on Computational Complexity, IEEE Computer Society Press, Philadelphia, Pennsylvania, pp. 290-300, **1996**.

[89] E. B. Baum, D. Boneh, Running dynamic programming algorithms on a DNA computer, in: Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, **1996**.

URL http://citeseer.nj.nec.com/baum96running.html

[90] M. Amos, A. Gibbons, D. Hodgson, Error-resistant implementation of DNA computations, in: Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, **1996**.
URL http://citeseer.nj.nec.com/26034.html

[91] D. Boneh, C. Dunworth, J. Sgall, On the computational power of DNA, Discrete Applied Mathematics 71 (1-3) 79-94, **1996**.
URL ftp.cs.princeton.edu/pub/people/dabo/biocircuit.ps.Z

[92] M. Ogihara, A. Ray, Simulating boolean circuits on a DNA computer, Tech. Rep. TR631, University of Rochester **1996**.
URL http://citeseer.nj.nec.com/ogihara96simulating.html

[93] R. Beigel, B. Fu, Molecular computing, bounded nondeterminism, and e_cient recursion, in: Proceedings of the 24th International Colloquium on Automata, Languages, and Programming, no. 1256 in Lecture Notes in Computer Science, **1997**, pp. 816-826.
URL http://citeseer.nj.nec.com/beigel96molecular.html

[94] G. Gloor, L. Kari, M. Gaasenbeek, S. Yu, Towards a DNA solution to the Shortest Common Superstring Problem, in: Proceedings of IEEE'98 International Joint Symposia on Intelligence and Systems, pp. 111-113, **1998**.
URL http://citeseer.nj.nec.com/498.html

[95] L. Kari, G. Gloor, S. Yu, Using DNA to solve the Bounded Post Correspondence Problem, Theoretical Computer Science 231 (2) 193-203, **2000**.

[96] Joh-Thomes Amenyo. Mesoscopic computer engineering: Automating dna-based molecular computing via traditional practices of parallel computer architecture design. In Second Annual Meeting on DNA Based Computers, pages 217-235, June **1996**.

[97] Martyn Amos. DNA Computation. PhD thesis, University of Warwick, UK, September **1997**.

[98] Andrew J. Blumberg. Parallel computation on a DNA substrate. In 3rd DIMACS Workshop on DNA Based Computers, pages 275-289, June **1997**.

[99] Michael Conrad and Klaus-Peter Zauner. Design for a DNA conformational processor. In 3rd DIMACS Workshop on DNA Based Computers, pages 290-295, June **1997**.

[100] Y. Gao, M. Garzon, R.C. Murphy, J.A. Rose, R. Deaton, D.R. Franceschetti, and S.E. Stevens Jr. DNA implementattion of nondeterminism. In 3rd DIMACS Workshop on DNA Based Computers, pages 204-211, June **1997**.

[101] Gre Gloor, Lila Kari, Michelle Gaasenbeek, and Sheng Yu. Towards a DNA solution to the shortest common superstring problem. In Fourth Internation Meeting on DNA Based Computers, pages 111-116, June **1998**.

[102] Vineet Gupta, Srinivasan Parthasarathy, and Mohammed J. Zaki. Arithmetic and logic operation with dna. In 3rd DIMACS Workshop on DNA Based Computers, pages 212-222, June **1997**.

[103] Masami Hagiya and Masanori Arita. Towards parallel evaluation and learning of boolean mu-formulas with molecules. In 3rd DIMACS Workshop on DNA Based Computers, pages 105-114, June **1997**.

[104] Peter Kaplan, David Thaler, and Albert Libchaber. Paralle overlap assembly of paths through a directed graph. In 3rd DIMACS Workshop on DNA Based Computers, pages 127-141, June **1997**.

[105] Thomas H. Leete, Matthew D. Schwartz, Robert M. Williams, David H. Wood, Jerome S. Salem, and Harvey Rubin. Massively parallel DNA computation: Expansion of symbolic determinants. In Second Annual Meeting on DNA Based Computers, pages 49-66, June **1996**.

[106] Richard Lipton. Using DNA to solve SAT. Unpulished Draft, **1995**.

[107] Nobuhiko Morimoto and Masanori Arita Akira Suyama. Solid phase DNA solution to the hamiltonian path problem. In 3rd DIMACS Workshop on DNA Based Computers, pages 83-92, June **1997**.

[108] Mitsunori Ogihara and Animesh Ray. DNA-based parallel computation by 'counting'. In 3rd DIMACS Workshop on DNA Based Computers, pages 265-274, June **1997**.

[109] John S. Oliver. Computation with DNA-matrix multiplication. In Second Annual Meeting on DNA Based Computers, pages 236-248, June **1996**.

[110] Z. Frank Qiu and Mi Lu. Arithmetic and logic operations for DNA computers. In Parallel and Distributed Computing and Networks (PDCN'98), pages 481-486. IASTED, December **1998.**

[111] Istvάn Katsányi,Solutions of some Classical Problems in Various Theoretical DNA Computing Models, Eötvös Loránd University, Department of Algorithms and applications H-1117 Budapest, Pázmány Péter sétány 1/C, Hungary, Preprint submitted to Theoretical Computer Science9-12-**2003.**

[112] Braich, R., Chelyapov, N., Johnson, C., Rothemund, P., and Adleman, L., Solution of a 20 - variable 3 - sat problem on a DNA computer. Science, vol.296, pp.499–502, **2002**.

[113] Chang, W. and Guo, M., Solving the set cover problem and the problem of exact cover by 3-sets in the adleman-lipton model. BioSystems, vol.72, pp.263–275, **2003**.

[114] Chang, W., Ho, M., and Guo, M., Molecular solutions for the subset-sum problem on DNA-based supercomputing. BioSystems, vol.73, pp.117–130, **2004**.

[115] H. Ahrabian, A. Nowzari-Dalini, DNA Simulation of Nand Boolean Circuits, Department of Mathematics and Computer Science, Faculty of Science, University of Tehran, Tehran, Iran, AMO - Advanced Modeling and Optimization, Volume 6, Number 2, **2004**.

[116] Alex Chediak, Karen Scott, Peng Zhang, the future of computing, University of California, Berkeley, TICS 6, Prof. Sands, MSE 225, May 13, **2002**

[117] Narayanan A. and Zorbalas S, DNA algorithms for computing shortest paths. Proceedings of Genetic Programming, pp. 718-723, **1998**.

[118] Lee J.Y., Shin S.Y., Augh S.J., Park T.H. and Zhang B.T, Temperature gradient-based DNA computing for graph problems with weighted edges. Lecture Notes in Computer Science, Vol. 2568, pp. 73-84, **2003**.

[119] Yamamoto M., Kameda A., Matsuura N., Shiba T., Kawazoe Y. and Ahochi A., A separation method for DNA computing based on concentration control. New Generation Computing, Vol. 20, pp. 251-262, **2002**.

[120] Lee J.Y., Shin S.Y., Augh S.J., Park T.H. and Zhang B. T., Temper ature gradient-based DNA computing for graph problems with weighted edges. Preliminary Proceedings of the Eighth International Meeting on DNA Based Computers, pp. 41-50, **2002.**

[121] Zuwairie Ibrahim, Higashi-mita, Tama-ku, Kawasaki- shi, Kanagawa- ken, Towards Solving Weighted Graph Problems by Direct-Proportional Length-Based DNA Computing, Institute of Applied DNA Computing, Research Report, IEEE Computational Intelligence Society (CIS) Walter J Karplus Summer Research Grant, **2004**.
URL: http://www.isc.meiji.ac.jp/~i3erabc/IADC.htm

[122] Calcul moléculaire, IFT 6299, A2004, Miklós Csũrös, Université de Montréal, **2004**.

[123] Eric RANNAUD, Informatique et Evolution, 26 juin **2002.**

[124] Philippe Muller, Représentation et résolution de problèmes : Intelligence artificielle II February 9, **2004.**

[125] Ecole polytechnique fédérale de lausan, Informatique II : Algorithmique.

[126] Stephen Cook.The Complexity of theorem proving procedures.*In Proc. 3rd Ann. ACM Symp. On Theory of Computing* , pages 151-158 , New York , **1971**.Association for Computing Machinery.

[127] M.Davis, M.Logemann, D. Loveland.A machine program for Theorem Proving.Communications of the ACM, 5:394-397,**1962**.

[128] W.F.Dowling et J.H.Gallier Linear-time algorithms for testing the satisfiability of propositional Horn formulas .*Journal of Logic Programming* , 1(3) :267-284,**1984**.

[129] S.Even,A.Itai , et A. Shamir.On the complexity of timetable and multi-commodity flow problems.*SIAM Journal OnComputing* , 5(4),**1976**.

[130] *″Crossing Number is NP-complete ″* , SIAM Journal on algebraic and Discrete Methods, **1983**.

[131] Danielle Desmarais, l'expertise d'ADN en droit criminel, **2000.**

[132] Wenbin Liu, Lin Gao, Qiang Zhang, Guandong Xu, Xiangou Zhu, Xiangrong Liu and Jin Xu, A Random Walk DNA Algorithm for the 3-SAT Problem, Current Nanoscience, 1, 85-90, **2005.**

[133] Hofmeister, T.; Schöning, U.; Schuler R.; Watanabe, O. Annual Symposium on Theoretical Aspects of Computer Science (STACS), 192-202, **2002**.