REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED BOUDIAF - M'SILA

FACULTE : Des Mathématiques Et De L'informatique

DEPARTEMENT : D'INFORMATIQUE

N° :……..

DOMAINE : Mathématique et Informatique

FILIERE : Informatique

OPTION : Technologie de l'Information et de Communication

Mémoire présenté pour l'obtention
Du diplôme de Master Académique

Par: ABDELOUAHAB Khalid

Intitulé

Algorithme génétique appliqué au problème de *k* plus courts chemins dans un graphe : Etude & évaluation

Soutenu devant le jury composé de :

Dr. LINDA Belabdelouahab Fernini	Université de M'sila	Président
Dr. LAMICHE Chaabane	Université de M'sila	Rapporteur
Mr. BARKET Abdelbasset	Université de M'sila	Examinateur

Année universitaire : 2016 /2017


بسم الله الرحمن الرحيم

Dédicace

Je dédie ce travail à mes parents qui m'ont toujours offert le bonheur.

Je le dédie à ma sœur, mes frères, les petits enfants de la famille, et à ma grande famille, A Tous mes amis et Mes collègues.

A la promo 2016/2017 d'informatique.

Enfin, à toutes celles et tous ceux qui ont contribué de près ou de loin à l'accomplissement de ce travail.

khaled

Remerciements

Je tiens avant tout à remercier Dieu le tout puissant de m'avoir donné la force et la volonté pour achever ce modeste travail.

Je remercie Dr. LAMICHE Chaabane mon encadreur d'avoir bien dirigé ce travail, avec ses judicieux conseils dont il a fait preuve durant l'élaboration de notre étude.

Je souhaite remercier toutes les personnes qui m'ont aidé d'une façon directe ou indirecte à la réalisation de ce mémoire.

Table des Matières

Liste des figures ……I
Liste des tableaux ……...III
Introduction générale……...01

CHAPITRE 1 : Les GRAPHES : CONCEPTS FONDAMENTAUX

1. Introduction ……03
2. Application de la théorie des graphes ……03
3. Définition d'un graphe et ses différentes représentations ……03
   - 3.1.Graphe orienté ……..04
   - 3.2.Graphe non orientés ……..04
   - 3.3.Terminologie ……05
   - 3.4.Quelques types des graphes ……..05
   - 3.5.Nombre d'un graphe ……07
     - 3.5.1. Ordre d'un graphe ……07
     - 3.5.2. Notion de degré ……07
4. Représentation d'un graphe ……07
   - 4.1.Matrice d'adjacence ……..07
   - 4.2.Liste d'adjacence ……..08
5. Notions de sous-graphe et graphe partiel ……08
   - 5.1.Sous graphe ……..08
   - 5.2.Graphe partiel ……...08
   - 5.3.Sous graphe partiel ……...09
6. Cheminement dans un graphe ……09
   - 6.1.Chaines et cycles ……...09
     - 6.1.1. Chaine ……...09
     - 6.1.2. Cycle ……..……09
   - 6.2.Chemins et circuits ……..….10
     - 6.2.1. Chemins 10
     - 6.2.2. Circuits ……10
7. Notions de connexité ……..……10

7.1.Connexité simple ……..10
7.2.Connexité forte ……11
8. Graphes particulaires ……..……11
8.1.Graphe Eulérien ……11
8.1.1. Chaine Eulérienne ……11
8.1.2. Chemin Eulérien ……...12
8.2.Graphe hamiltonien ……..12
8.2.1. Chaine hamiltonienne ……...……12
8.2.2. Chemin hamiltonien ……..……12
9. Problème de cheminement dans un graphe ……12
9.1.Introduction ……..12
9.2.Introduction au problème du plus court chemin ….....13
9.3.Résolution de problème de type P1 ……...13
9.3.1. Algorithme Dijkstra ……..……13
9.3.2. Algorithme de Bellman-Ford ……14
10. Conclusion ……..……15

CHAPITRE 2 : REVUE SUR LES METAHEURISTIQUES D'OPTIMISATION

1. Introduction …..17
2. Problème d'optimisation combinatoire ….....17
3. Complexités des algorithmes ……..18
3.1.La classe NP ……18
3.2.Les problèmes NP-difficile ……..…….18
4. Méthodes de résolution ……...19
5. Algorithmes approchés d'optimisation ……...20
5.1.Heuristiques ……..20
5.2.Metaheuristiques ……..20
6. Présentation des principales Métaheuristique ……20
6.1.Méthode de trajectoire ……..20
6.1.1. Méthode de Descente ……....21
6.1.2. Recuit Simulé ……...21
6.1.3. Recherche Tabou ……..….22
6.2.Méthode basées sur les populations de solution ……23
6.2.1. Algorithme génétique ……...23

6.2.1.1.Codage (représentation) d'un individu ……...24
6.2.1.2.Génération de la population initiale ……25
6.2.1.3.La fonction d'évaluation ……25
6.2.1.4.Operateurs de sélection ……...25
6.2.1.4.1. Sélection par roulette ……..25
6.2.1.4.2. Sélection par range ……26
6.2.1.4.3. Sélection par tournoi ……...26
6.2.1.4.4. Sélection uniforme ……..27
6.2.1.5.Opérateurs de croisement ……...27
6.2.1.5.1. Croisement à un point ……27
6.2.1.5.2. Croisement à deux point ……28
6.2.1.6.Opération de mutation ……28
6.2.2. L'algorithme d'optimisation par essaim de particules ……..29
6.2.2.1.Principe de fonctionnement ……29
7. Conclusion ……..31

CHAPITRE 3 : APPLICATION DE L'ALGORITHME GENETIQUE AU PROBLEME *k*-PPC

1. Introduction ……33
2. Présentation du problème *K*-PPC ……33
2.1.Domaines applications de problème (*K*-PPC) ……...34
3. Adaptation de l'algorithme GA au problème *K*-PPC ……..34
3.1.Représentation du chromosome ……34
3.2.Population initiale ……35
3.3.La fonction fitness ……36
3.4.Les stratégies de sélection ……36
3.5.Operateur de croissement ……36
3.6.Operateur de mutation ……..38
3.7.Remplacement ……..38
4. Conclusion ……..40

CHAPITRE 4 : REALISATION ET EXPERIMENTATIONS

1. Introduction ……42
2. Environnement de développement ……42

2.1.Environnement matériel ……...42
2.2.Environnement logiciel ……42
2.2.1. Le Microsoft Visual Studio 2012 ……..42
2.2.2. Le langage de programmation C++ ……...43
3. Paramètres de l'algorithme ……43
4. Résultats obtenus ……44
4.1.Premier Exemple ……..44
4.1.1. Tests et résultats ……44
4.2.Deuxième exemple ……...47
4.2.1. Tests et résultats ……48
4.3.Effets des paramètres de l'algorithme génétique……...50
4.3.1. Effets du nombre de génération…… 50
4.3.2. Effets de la probabilité de mutation……...52
4.3.3. Effets de la probabilité de croisement……....53
4.4.Troisième exemple ……...54
4.4.1. Tests et résultats ……55
4.5.Effets des paramètres de l'algorithme génétique……...57
4.5.1. Effets du nombre de génération…… 57
4.5.2. Effets de la probabilité de mutation……...58
4.5.3. Effets de la probabilité de croisement……....59
5. Conclusion ……..60
Conclusion générale ……...61
Bibliographie ……..62

Liste des figures

Figure 1.1 : Un graphe orienté……..04

Figure 1.2 : Un graphe non orienté……...04

Figure 1.3 : Un graphe régulier……....06

Figure 1.4 : Un graphe complet……....06

Figure 1.5 : Un graphe biparti……...06

Figure 1.6 : Matrice d'adjacence d'un graphe orienté……...08

Figure 1.7 : Matrice d'adjacence d'un graphe non orienté……08

Figure 1.8 : Un graphe non orienté……...09

Figure 1.9 : Un graphe orienté……....10

Figure 1.10 : Les ponts de Königsberg……11

Figure 1.11 : présentation d'un graphe hamiltonien……12

Figure 2.1 : Classification de méthode de résolution le problème d'optimisation……19

Figure 2.2 : Principe générale d'un GA simple……....24

Figure 2.3 : Schéma d'une roulette de sélection……26

Figure 2.4 : Représentation schématique du croisement en un point……27

Figure 2.5 : Représentation schématique du croisement en deux points……...28

Figure 2.6 : Représentation schématique de la mutation……...29

Figure 2.7 : Déplacement d'une particule……30

Figure 3.1 : Un graphe orienté……33

Figure 3.2 : Représentation du chromosome……....35

Figure 3.3 : Exemple d'un graphe……....37

Figure 3.4 : Croisement a un point……37

Figure 3.5 : Opérateur de mutation……...38

Figure 4.1 : Visuel studio 2012……42

Figure 4.2 : Représentation d'un graphe à 8 sommets……...44

Figure 4.3 : Représentation d'un graphe avec 20 sommets……...47

Figure 4.4 : Les résultats donnés dans le tableau 4.6……51

Figure 4.5 : Les résultats donnés dans le tableau 4.7……52

Figure 4.6 : Les résultats donnés dans le tableau 4.8……53

Figure 4.7 : Représentation d'un graphe avec 30 sommets……54

Figure 4.8 : Les résultats donnés dans le tableau 4.10…….....57

Figure 4.9 : Les résultats donnés dans le tableau 4.11……...58

Figure 4.10 : Les résultats donnés dans le tableau 4.12……59

Liste des tableaux

Table 2.1 : Exemple de sélection par range pour 6 chromosomes…….....26

Table 4.1 : Paramètres de l'algorithme……43

Table 4.2 : Résultats obtenus pour (*pop_size* = 25, *B* = 8, *nbr_itération* = 600)……45

Table 4.3 : Résultats obtenus pour (*pop_size* = 25, *B* = 10, *nbr_itérations* = 600)……46

Table 4.4 : Nombre de chemins trouvés……46

Table 4.5 : Résultats obtenus pour (*pop_size* = 25, *B* = 9, *nbr_itération* = 6000)…….....48

Table 4.6 : Résultats obtenus pour différents N génération……...51

Table 4.7 : Résultats obtenus pour différents Pm (*pop_size* = 25, *nb_itération* = 7000, *Pc* = 0.9)…….52

Table 4.8 : Résultats obtenus pour différents Pc (*pop_size* = 25, *B* = 10, *nb_iteration* = 20000, *Pm*=0.1)…….53

Table 4.9 : Résultats obtenus pour (*pop_size* = 25, *B* = 11, *nbr_itération* = 200000)……55

Table 4.10 : Résultats obtenus pour différents N génération…….....57

Table 4.11 : Résultats obtenus pour différents Pm (*pop_size* = 25, *nb_itération* = 9000, *Pc* = 0.9)…....58

Table 4.12 : Résultats obtenus pour différents Pc (*pop_size* = 25, *B* = 10, *nb_iteration*=500000, *pm* = 0.1)……59

INTRODUCTION GENERALE

Introduction générale

Les problèmes d'optimisation occupent actuellement une place importante dans la communauté des ingénieurs, scientifiques. En effet, ce type de problèmes intervient dans plusieurs domaines, telles que la conception des systèmes mécaniques, traitement d'images, l'électronique et la recherche opérationnelle. L'optimisation combinatoire est une branche d'optimisation, qui est appliquée pour résoudre les problèmes complexes et difficiles dans le but de trouver une meilleure solution parmi toutes les solutions possibles. Elle couvre des domaines variés dans la vie réelle, comme les finances, le transport.

Le problème *k* plus courts chemins (*k*-PCC) est un problème d'optimisation combinatoire, il utilisé dans divers domaines comme le problème d'optimisation de réseau et le problème d'alignement de séquence. Le *k*-PCC est un problème classique plus connu qui appartient à la classe des problèmes NP-complet. Le but est de chercher l'ensemble des *k* plus courts chemins du nœud source vers les nœuds de destinations avec contrainte de bande passante par l'utilisation d'algorithme génétique. Notre objectif est de trouver la meilleure solution au problème *k*-PCC quand le nombre de sommets est considérable.

L'objectif de ce mémoire est d'étudier le problème du *k*-PCC. Nous avons appliqué un algorithme génétique pour résoudre ce problème dans l'espoir d'améliorer la qualité des solutions produites. De plus, les performances de notre approche ont été testées sur un ensemble de graphe bien connu. Notre mémoire est organisé comme suit :

Dans le premier chapitre, nous présenterons les aspects fondamentaux de la théorie des graphes pertinents pour notre étude. Dans Le deuxième chapitre, on va définir les problèmes d'optimisation combinatoire, les heuristiques et les métaheuristiques, et la définition de la notion de complexité des algorithmes et des problèmes (classe NP, classe NP-complet, classe NP-difficile). On a présentera ensuite, quelques méthodes du la résolution exactes et approchées. Le troisième chapitre, concernera la résolution du problème sujet étude. Tout d'abord nous avons défini le problème de *k*-PCC. On a appliqué l'algorithme génétique pour résoudre notre problème. Dans le quatrième chapitre, nous donnerons plusieurs résultats obtenus. Enfin, on termine par une conclusion générale et quelques perspectives.

CHAPITRE 1

Les GRAPHES : CONCEPTS FONDAMENTAUX

1. Introduction

Les graphes représentent de manière simple et naturelle des relations entre les objets. Les outils mathématiques et les algorithmes mis au point en théorie des graphes permettent de résoudre une multitude de problèmes, tels que les problèmes de cheminement, d'ordonnancement, d'affectation. Ce chapitre présente les aspects fondamentaux de la théorie des graphes.

2. Application de la théorie des graphes

La théorie des graphes a ses applications dans divers domaines de l'ingénierie :

- Génie électrique : Les concepts de la théorie des graphes soit largement utilisés dans la conception des connexions de circuit. Les types ou organisation de connexions sont nommés comme topologies. Quelques exemples pour topologies sont étoile, pont, série et topologies parallèles.
- Informatique : Théorie des graphes est utilisée pour l'étude des algorithmes. Par exemple,
- Algorithme de Kruskal.
- Algorithme de Prim.
- Algorithme de Dijkstra.
- Réseaux informatique : Les relations entre les ordinateurs interconnectés dans le réseau suivent les principes de la théorie des graphes.
- Sciences : La structure moléculaire et de la structure chimique d'une substance, la structure de ADN d'un organisme, sont représentés par les graphiques.
- Linguistique : L'arbre d'analyse syntaxique d'une langue et la grammaire d'une langue utilise des graphiques.
- Généralités : Routes entre les villes peuvent être représentées à l'aide de graphiques. Décrivant l'information hiérarchique ordonnée, comme arbre peut être utilisé comme un type spécial de graphique appelé arbre [5].

3. Définition d'un graphe et ses différentes représentations

Un graphe est une structure comportant un ensemble non vide V d'éléments appelés sommets et un ensemble E d'éléments appelés arêtes. Il existe en fait deux type de graphes : les graphes orientés et les graphes non orientes.

3.1. Graphe orienté

En donnant un sens aux arêtes d'un graphe, on obtient un digraphe (ou graphe orienté). Un digraphe fini $G = (V, U)$ est défini par l'ensemble fini $V = \{v_1, v_2,..., v_n\}$ dont les éléments sont appelés sommets, et par l'ensemble fini $U = \{u_1, u_2,..., u_m\}$ dont les éléments sont appelés arcs.

Chaque élément u_i de l'ensemble U est défini par une paire ordonnée de sommets. Lorsque $u_i = (x, y)$, on dit que l'arc u va de x à y. On dit aussi que x est l'extrémité initiale et y l'extrémité terminale de u_i [1].

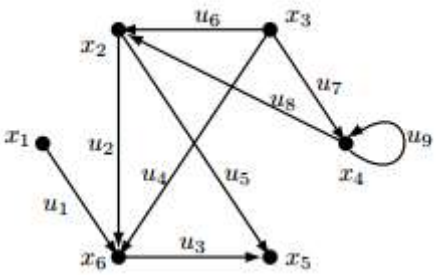

Figure 1.1 : Un graphe orienté [3].

3.2. Graphe non orienté

Un graphe fini $G = (V, E)$ est défini par l'ensemble fini $V = \{v_1, v_2,..., v_n\}$ dont les éléments sont appelés sommets, et par l'ensemble fini $E = \{e_1, e_2,..., e_m\}$ dont les éléments sont appelés arêtes.

Chaque élément e_i de l'ensemble E est définie par une paire non ordonnée de sommets, appelés les extrémités de e_i. Si l'arête e relie les sommets a et b, on dira que ces sommets sont adjacents, ou incidents avec e_i [1].

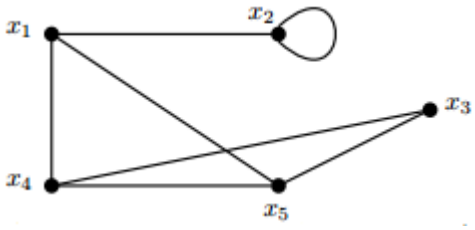

Figure 1.2 : Un graphe non orienté [3].

3.3. Terminologie

La terminologie suivante s'applique à tout graphe *G :*

- Les deux sommets *u* et *v* sont appelés extrémités de l'arête (*u, v*). s'il s'agit d'un arc, *u* et *v* désignent respectivement l'extrémité initiale et terminale de l'arc. Le sommet *u* est prédécesseur de *v* et *v* est successeur de *u*.
- Les arcs ayant même extrémités sont appelés arcs parallèles.
- Un arc de la forme (*v, v*) est une boucle.
- Un graphe simple s'il n'a ni arcs parallèles ni boucle.
- Un graphe sans arcs est un graphe vide ($G = (V, E) / E = \emptyset$).
- Un graphe sans sommets est un graphe nul ($G = (V, E) / V = \emptyset$).
- Un graphe ayant un seul sommet est un graphe travail ($G = (V, E) / Card(V) = 1$).
- Des arcs sont adjacents s'ils ont un sommet en commun à l'une des extrémités.
- Deux sommets *u* et *v* sont voisins ou adjacents s'ils sont connectes par un arc.
- Le voisinage d'un sommet désigne l'ensemble de ses sommets voisins.
- Un graphe *G* est dit complet si et seulement si $\forall x, y \in V$ si $(x, y)\ E \Rightarrow (y, x) \in E$. c'est-à-dire que tous les couples de sommets qu'il est possible de choisir (chacun étant pris une seule fois) sont présents dans *E* [4].

3.4. Quelques types de graphes

Il existe différents types de graphes en fonction du nombre de sommets, le nombre de arc, l'inter connectivité, et leur structure globale. Nous discuterons seulement quelques types importants de graphe :

1- Graphe simple : un graphe est dit simple s'il est sans boucle, il n'y a jamais plus d'une arête entre deux sommets quelconques.

2- Graphe multi-graphes : un multi-graphe est un graphe pour lequel il peut exister plusieurs arêtes entre deux sommets donnes.

3- Graphe régulier : un graphe G est dit k-réguliers si $\forall x$ sommets de G. on a $d_G(x) = k$. en d'autre termes, $\delta(G) = \Delta(G) = k$.

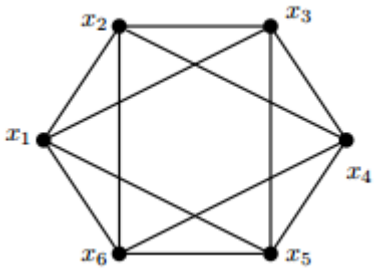

Figure 1.3 : Un graphe régulier [3].

4- Graphe complet : Un graphe complet est un graphe où chaque sommet est relié à tous les autres. Le graphe complet d'ordre n est noté *Kn.*

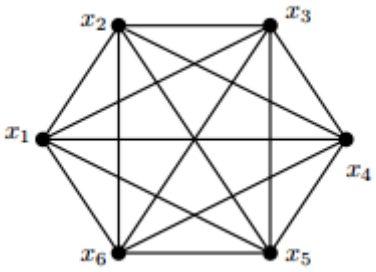

Figure 1.4 : Un graphe complet [3].

5- Graphe biparti : si ses sommets peuvent être divisés en deux ensembles *X* et *Y*, de sorte que toutes les arêtes du graphe relient un sommet dans *X* à un sommet dans *Y* [1].

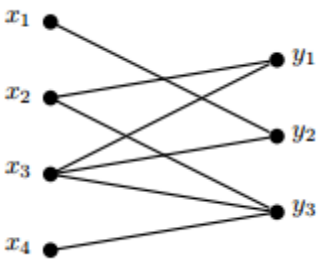

Figure 1.5 : Un graphe biparti [3].

3.5. Nombre d'un graphe

Les deux nombres d'un graphe sont l'ordre et le degré [4].

3.5.1. Ordre d'un graphe

Dans la définition ci-dessus d'un graphe, le nombre n représente l'ordre du graphe G, il s'agit du nombre de ses sommets [4].

3.5.2. Notion de degré

On appelle degré du sommet *v*, et on note *d*(*v*), le nombre d'arêtes incidentes à ce sommet. Dans un graphe simple, on peut aussi définir le degré d'un sommet comme étant le nombre de ses voisins (la taille de son voisinage).

Dans le cas d'un graphe orienté, le degré d'un sommet est calculé comme étant la somme des demi degrés extérieur et intérieur d'un sommet tel que :

- Demi-degré extérieur : $d^+(v)$ est le nombre d'arcs ayant *v* comme l'extérieur initial.
- Demi-degré intérieur : $d^-(v)$ est le nombre d'arcs ayant *v* comme l'intérieur terminal [1].

4. Représentation d'un graphe

Un certain nombre de représentations existent pour décrire un graphe. On distingue principalement la représentation par matrice d'adjacence et par liste d'adjacence.

4.1. Matrice d'adjacence

A tout graphe d'ordre *n* on associe une matrice *M* de *n* lignes et *n* colonnes dont les éléments sont notés M_{ij}. Pour un graphe non orienté $G = (V, E)$,

$$M(i,j)\begin{cases} 1 \; si \; (i,j) \in E \\ 0 \; sinon \end{cases} \qquad (1.1)$$

Remarque

- La matrice d'adjacence *M* d'un graphe non orienté est toujours symétrique.
- La somme d'une ligne *k* = la somme d'une colonne *k* = $d_G(k)$. La boucle est comptée deux fois.

Pour un graphe orienté $G = (V, U)$, M_{ij} représente le nombre d'arcs ayant *i* comme extrémité initiale et *j* comme extrémité terminale [3].

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 1.6 : Matrice d'adjacence du graphe orienté de la figure 1.1 [3].

$$M = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Figure 1.7 : Matrice d'adjacence du graphe non orienté de la figure 1.2 [3].

4.2. Liste d'adjacence

Pour un graphe $G = (V, E)$, la liste d'adjacence associe à ce graphe permet de représenter pour chaque sommet, l'ensemble de ses successeurs. Tous les arcs émanant d'un même sommet, sont liés entre eux dans une liste. A chaque arc sont donc associes l'extrémité terminale et le pointeur au prochain sommet de la liste [3].

5. Notion de sous-graphe et graphe partiel

Soit $G = (V, U)$ un graphe orienté (ou $G = (V, E)$ un graphe non orienté). Soit $A \in V$ un sous ensemble de sommets et $V \in U$ (ou $V \in E$).

5.1. Sous graphe

Un sous graphe de G engendré par l'ensemble de sommets A est le graphe :

$G_A = (A, U_A)$ où $UA = \{u \in U/I(u) \in A$ et $T(u) \in A\}$ dans le cas orienté.

$G_A = (A, E_A)$ où $EA = \{e = \{x, y\} \in E/x \in A$ et $y \in A\}$ dans le cas non orienté [3].

5.2. Graphe partiel

Un graphe partiel de G engendré par l'ensemble d'arcs (ou d'arêtes) R est le graphe $G_R = (V, R)$.

5.3. Sous graphe partiel

Un sous graphe partiel de G engendré par l'ensemble de sommets A et l'ensemble d'arcs (ou d'arêtes) V est le graphe $G_{A, V} = (A, V_A)$. V_A est l'ensemble d'arcs (ou arêtes) qui one leurs deux extrémités dans le sous ensemble V [3].

6. Cheminement dans un graphe

Plusieurs problématiques peuvent être considérées lorsque l'on s'intéresse au cheminement entre deux points d'un graphe. Cette section a pour objectif de définir les concepts de cheminement dans un graphe.

6.1. Chaines et cycles

Les concepts de chaine et cycle peuvent aussi bien s'appliquer à un graphe orienté qu'a un graphe non orienté [2].

6.1.1. Chaine

On appelle chaîne dans un graphe non orienté (ou orienté) $G = (V, E)$ (ou $G = (V, U)$), une suite alternée de sommets et d'arêtes (ou d'arcs) : $\mu = v_0, e_1, v_1 \ldots v_{k-1}, e_k, v_k$ (ou $\mu = v_0, u_1, v_1 \ldots v_{k-1}, u_k, v_k$) Tel que pour tout $1 \leq i \leq k$, v_i et v_{i+1} sont extrémités de l'arête e_i (ou de l'arc u_i). On dit que μ est une chaîne joignant les sommets v_0 et v_k de longueur k [2].

Le graphe ci-dessous $C1 = (x_1, e_2, x_4, e_5, x_3, e_6, x_5)$ est une chaine de longueur 3.

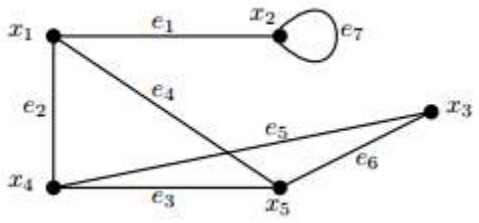

Figure 1.8 : Un graphe non orienté [3].

6.1.2. Cycle

On appelle cycle dans un graphe non orienté (ou orienté) $G = (V, E)$ (ou $G = (V, U)$), toute chaîne fermée simple : $\mu = v_0, e_1, v_1 \ldots v_{k-1}, e_k, v_k$ (ou $\mu = v_0, u_1, v_1 \ldots v_{k-1}, u_k, v_k$) Tel que k_0, et $v_0 = v_k$. On dit que μ est un cycle longueur k [2].

6.2. Chemins et circuits

Les concepts de chemin et circuit sont relatifs aux graphes orientes puisque le sens d'orientation des arcs est essentiel pour ces définitions.

6.2.1. Chemin

On appelle chemin dans un graphe orienté $G = (V, U)$, une suite alternée de sommets et d'arcs : $\gamma = v_0, u_1, v_1 \ldots v_{k-1}, u_k, v_k$ Tel que pour tout $1 \leq i \leq k$, le sommet v_i est extrémité initiale de l'arc u_i et le sommet v_{i+1} est son extrémité terminale.

Le graphe ci-dessous. On dit que γ est un chemin de v_0 vers v_k de longueur k. C2 = (v_3, u_6, v_2, u_5, v_5, u_3, v_6) est un chemin de longueur 3, C3 = (v_3, u_7, v_4, u_9, v_4, u_8, v_2, u_2, v_6) est un chemin de longueur 4, C4 = (v_1, u_1, v_6, u_2, v_2, u_6, v_3, u_7, v_4) est une chaine de longueur 4 [2].

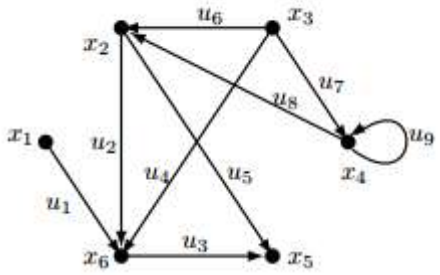

Figure 1.9 : Un graphe orienté [3].

6.2.2. Circuit

On appelle circuit dans un graphe orienté $G = (V, U)$, tout chemin fermé simple : $\gamma = v_0, u_1, v_1 \ldots v_{k-1}, u_k, v_k$ tel que $k > 0$, et $v_0 = v_k$. On dit que γ est un circuit de longueur k.

7. Notion de connexité

La connexité est une propriété essentielle des graphes surtout pour les problèmes relatifs aux réseaux. Dans ce qui suit, sont étudiées les notions de connexité simple et forte.

7.1. Connexité simple

Un graphe est dit connexe si et seulement si : [6] :

$$\forall\, i, j \in V \begin{cases} soit\ i = j\ ou \\ il\ existe\ une\ chaine\ reliant\ i\ et\ j \end{cases}$$

7.2. Connexité forte

Un graphe oriente est dit fortement connexe si seulement si :

$$\forall\, i,j \in V \begin{cases} i = j \; ou \\ il\ existe\ un\ chemin\ reliant\ i\ et\ j\ et\ un\ chemin\ reliant\ j\ et\ i \end{cases}$$

Un graphe est fortement connexe si et seulement s'il n'a qu'une seule composante fortement connexe [6].

8. Graphes particulaires

Les graphes ont parfois des caractéristiques qui les rendent particuliers. Dans ce qui suit, nous définissons certains types de graphe particuliers, comme le graphe eulériens et hamiltonien.

8.1. Graphe Eulérien

La ville de Königsberg est construite autour de deux îles situées sur le PREGEL et reliées entre elles par un pont. Six autres ponts relient les rives de la rivière à l'une ou l'autre des deux îles, comme représentés sur le plan ci-dessous. Le problème consiste à déterminer s'il existe ou non une promenade dans les rues de Königsberg passant une et une seule fois par chaque pont de la ville [7].

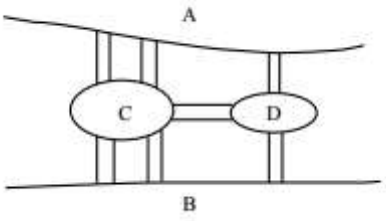

Figure 1.10 : Les ponts de Königsberg [3].

8.1.1. Chaine Eulérienne

Une chaine Eulérienne est une chaine qui passe par toutes les arêtes du graphe exactement une fois. Si la chaine est fermée, il s'agit d'un cycle Eulérien [7].

8.1.2. Chemin Eulérien

Un chemin Eulérien est un chemin qui passe par tous les arcs du graphe exactement une fois. Si le chemin est ferme, il s'agit d'un circuit Eulérien [7].

8.2. Graphe hamiltonien

Le problème hamiltonien, invente par Hamiltonien, revient à déterminer pour un ensemble de villes un itinéraire permettant de passer une et une seul fois par chaque ville et revenir au point de départ [7].

8.2.1. Chaine hamiltonienne

Une chaine hamiltonienne est une chaine qui visite chaque sommet du graphe exactement une fois. Un cycle Hamiltonien est donc une chaine Hamiltonienne fermée [7].

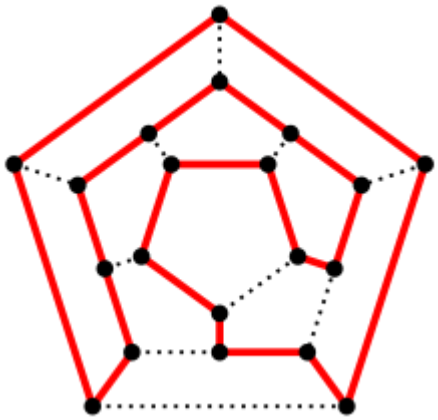

Figure 1.11 : présentation d'un graphe hamiltonien [7].

8.2.2. Chemin hamiltonien

Un chemin Hamiltonien est un chemin qui visite chaque sommet du graphe exactement une fois. Un circuit Hamiltonien est donc un chemin hamiltonien fermé [7].

9. Problème de cheminement dans un graphe

9.1. Introduction

Les problèmes de cheminement dans les graphes comptent parmi les plus anciens problèmes de la théorie des graphes. Il existe de nombreux problèmes spécifiques aux chemins. On en cite quelques-uns :

1) Soient i et $j \in V$:

- Existe-t-il au moins un chemin de i à j ?
- Existe-t-il au moins un chemin de longueur p entre i et j ?
- Quel est le nombre de chemins de longueur inférieure ou égale à p entre i et j ?

2) Trouver le ou les chemins qui relient le sommet a tous les autres sommets et qui satisfont une propriété P.
3) Trouver les chemins qui relient tout couple de sommets et qui satisfont une propriété P [4].

9.2. Introduction au problème du plus court chemin

Etant donne un graphe $G = (V, E)$ oriente tel que chaque arc (i, j) possède un nombre a (i, j) appelé longueur, de l'arc (i, j). Le problème du plus court chemin consiste à déterminer un chemin élémentaire C_h joignant les sommets i et j et qui soit de longueur extrémale (minimale ou maximale selon l'objectif). La longueur du chemin optimal est égale à la somme de longueurs des arcs du chemin.

On considère d'une manière plus générale deux types de problèmes :

- Problème P1 : il s'agit de trouver le plus court chemin reliant un sommet à tous les autres.
- Problème P2 : il s'agit de trouver tous les plus courts chemins entre tous les couples de sommets [4].

9.3. Résolution de problème de type P1

Dans cette section sont présentés les algorithmes suivants : l'algorithme de Dijkstra et l'algorithme de Bellman-Ford.

9.4.1. Algorithme Dijkstra

Le problème est le même que celui traité par Bellman, sauf qu'ici le réseau $R = (V, U, p)$ peut être avec ou sans circuits mais à condition que les poids soient tous positifs ou nuls. Le sommet de départ r n'est pas nécessairement une source [3].

- **Principe**

L'algorithme calcule le chemin de poids minimal en partant du sommet r et en prolongeant le chemin à chaque itération. Cette méthode s'appelle, calcul de la plus courte distance de proche en proche. A chaque étape, pour un sommet donné v, on ajuste les valeurs $\pi[y]$ pour tout sommet y successeur de v [3].

Le pseudo-code ci-dessous :

Algorithme Djikstra

```
Début
      s ← r ; k ← 1 ; f[k] ← r // r sommets
      Pour tout v ∈ V faire π[x] ← +∞ fait
      π[r] ← 0
     Tant que (k < n) et (π[x] < +∞)
     Faire
        Pour tout u ∈ U/ (I(u) = f[k]) et (T(u) ∈/ S)
        Faire
        x = T(u)
        Si π(x) > π [f[k]] + p[u]
          Alors π(x) ← π [f[k]] + p[u] ; P ré(x) ← f[k]
        Fait
     x ← y/y ∈ X − S et π[y] minimal
     k ← k + 1
    f[k] ← x
    S ← S \ ∪x
    Fait
Fin.
```

9.4.2. Algorithme de Bellman-Ford

A chaque sommet *v* ∈ *V*, on veut associer un chemin de poids optimal joignant la source du graphe *r* ∈ *V* à *v* dans le réseau *R* = (*V, U, p*) [3].

❖ **Principe**

L'algorithme consiste à affecter à chaque sommet v ∈ V d'un réseau R = (V, U, p) sans circuits, une valeur π(x) (appelée potentiel de v) qui représente le poids du chemin optimal reliant r à v. Il se base sur la liste des prédécesseurs de chaque sommet [3].

Le pseudo-code ci-dessous :

Algorithme de Bellman Ford

```
Début
S ← {r}
      Pour tout x ∈ X ; π(x) ← +∞; P ré(x) ← NULL; π(r) ← 0
        Pour tout (x ∈ S) et (∀u ∈ U avec T(u) = x on a I(u) ∈ S)
          Faire Pour tout ((x, y) ∈ U) Faire
            Si π(x) > π(y) + p(u) Alors π(x) ← π(y) + p(u) ; P ré(x) ← y
          Fait
S ← S \ x
Fait
Fin
```

10.Conclusion

Nous avons discuté dans ce chapitre quelques concepts de la théorie des graphes. Cela devrait nous permettre de passer au prochain chapitre avec un arrière-plan acceptable sur les graphes en général.

CHAPITRE 2

REVUE SUR LES METAHEURISTIQUES D'OPTIMISATION

1. Introduction

L'optimisation est une discipline en plein essor qui entre en jeu dans beaucoup de domaines, comme dans la conception de circuits électroniques, la recherche opérationnelle, la biologie, mais aussi pour répondre aux besoins croissants des secteurs économiques et industriels (maximisation des performances, minimisation des coûts).

Dans ce chapitre on va présenter les métaheuristiques pour la résolution de problèmes d'optimisation difficile. En premier temps, on va passer en revue la définition d'un problème d'optimisation et ainsi la complexité de ces types de problèmes. En deuxième temps, on va détailler les principes des différents algorithmes metaheuristiques.

2. Problème d'optimisation combinatoire

Un problème d'optimisation combinatoire est défini par un ensemble d'instances. A chaque instance du problème est associé un ensemble discret de solutions *S*, un sous-ensemble *X* de *S* représentant les solutions admissibles (réalisables) et une fonction de coût *f* (ou fonction objectif) qui assigne à chaque solution $s \in X$ le nombre réel (ou entier) *f(s)*. Résoudre un tel problème (plus précisément une telle instance du problème) consiste à trouver une solution $s^* \in X$ optimisant la valeur de la fonction de coût f. Une telle solution s^* s'appelle une solution optimale ou un optimum global. Nous avons donc la définition suivante :

Définition : Une instance I d'un problème de minimisation est un couple (X, f) où $X \subseteq S$ est un ensemble fini de solutions admissibles, et f une fonction de coût (ou objectif) à minimiser $f : X \rightarrow R$. Le problème est de trouver $s^* \in X$ tel que $f(s^*) \leq f(s)$ pour tout élément $s \in X$. [8]

Notons que d'une manière similaire, on peut également définir les problèmes de maximisation en remplaçant simplement $\leq$ par $\geq$. L'optimisation combinatoire trouve des applications dans des domaines aussi variés que la gestion, l'ingénierie, la conception, la production, les télécommunications, les transports, l'énergie, les sciences sociales et l'informatique elle-même. [9].

3. Complexité des algorithmes

Un même problème peut généralement être résolu par plusieurs algorithmes, donc il faut comparer entre ces algorithmes, cette comparaison se base sur le temps de calcul et sur l'espace mémoire requis par l'algorithme. On appelle complexité en temps d'un algorithme dans le pire cas, la fonction *F(n)* qui donne une borne supérieure du nombre d'opérations élémentaires effectuées par l'algorithme lorsque la taille de l'entrée est *n*.

Un algorithme dont le nombre d'opérations élémentaire nécessaires pour résoudre un exemple de taille *n* est une fonction polynomiale en *n*. Un algorithme est efficace (qu'il est <bon>) si le nombre des opérations nécessaires pour résoudre un problème est borné par une fonction polynomiale d'un paramètre caractérisant la taille de ce problème. Il faut noter qu'un problème décision, est le problème pour lequel une solution est soit « oui » soit « non », alors qu'un problème d'optimisation, est le problème pour lequel on doit chercher à déterminer une solution qui optimise un critère. A chaque problème d'optimisation on peut associer un problème de décision [10].

Réduction polynomiale

Soient Π et Π' deux problèmes de décision. Π' se réduit polynomialement a Π et on note Π' α Π, si Π' est polynomial ou s'il existe un algorithme polynomial qui construit, à partir d'une donnée *D'* de Π', une donnée D de Π telle que, la réponse pour *D'* soit oui si est seulement si la réponse pour *D* est oui [10].

3.1. La classe NP

La classe NP est la classe des problèmes de décision qui peuvent être résolus par une machine de Turing non déterministe en temps polynomial. On distingue trois sous classes : La classe P qui contient les problèmes les plus faciles de NP, ils ont un algorithme polynomial pour les résoudre La classe des problèmes NP-complets contient les problèmes les plus difficiles de NP. Alors que la classe des problèmes ouverts englobe les problèmes de statut indéterminé [10].

3.2. Les problèmes NP-difficile

Un problème d'optimisation est dit NP-difficile sil le problème de décision associe est NP-complet [10].

4. Méthodes de résolution

Il existe deux grandes catégories des méthodes de résolution des problèmes d'optimisation combinatoire : les méthodes exactes et les méthodes approchées. Les méthodes exactes se basent sur l'énumération de l'ensemble des solutions potentielles et permettent d'obtenir la solution optimale. On a par exemple l'algorithme de séparation et évolution (branche and bound, programmation dynamique,…), la taille du problème influe rationnellement sur l'efficacité (temps d'exécution) des méthodes de cette approche. Par contre les méthodes approchées, encore appelées heuristiques, permettent d'obtenir une bonne solution, qui n'est pas toujours la solution optimale, mais avec le temps raisonnable.

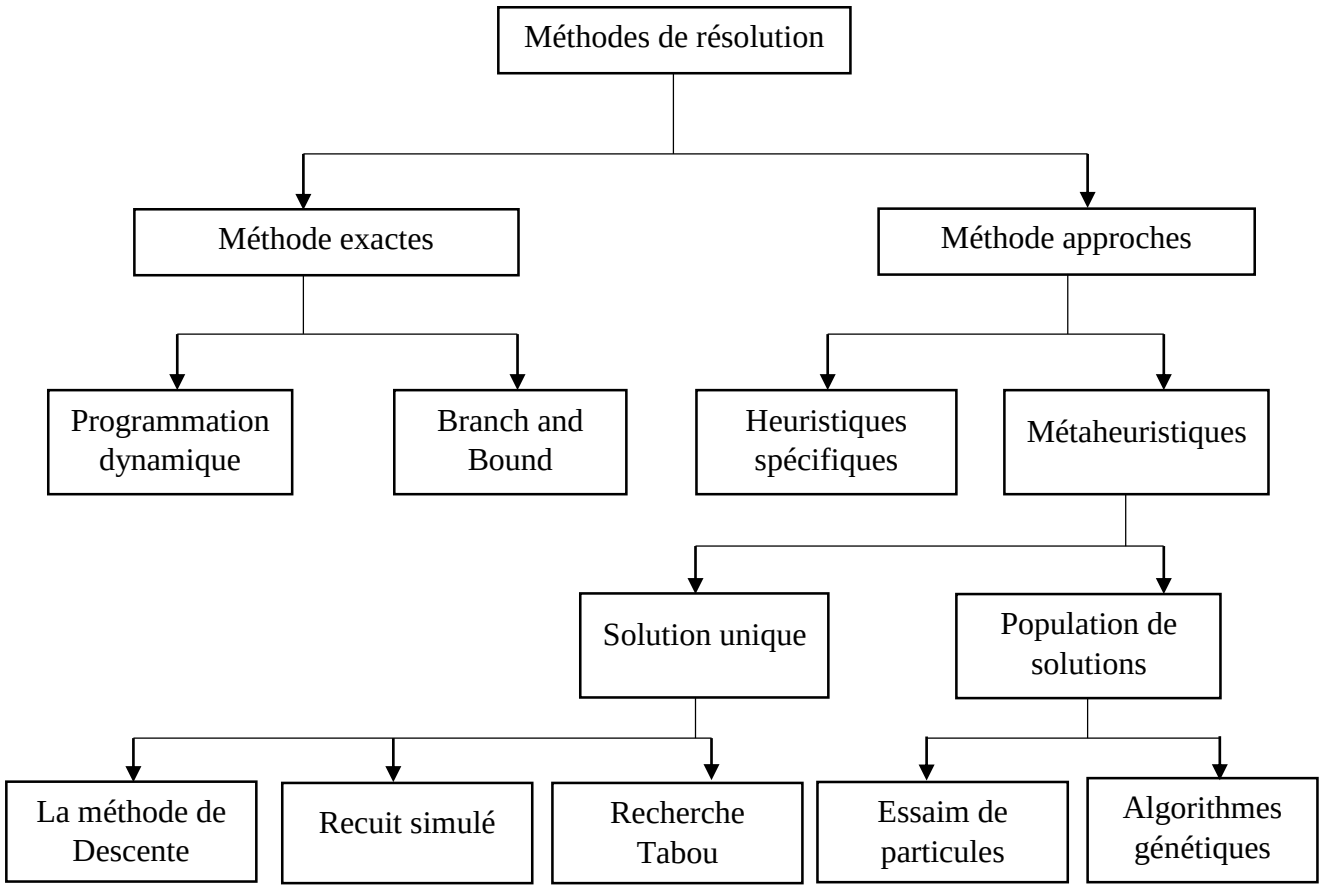

Figure 2.1 : Classification des méthodes de résolution le problème d'optimisation [11].

5. Algorithmes approchés d'optimisation

5.1. Heuristiques

L'utilisation de méthodes exactes n'est pas toujours possible pour un problème donné à cause d'un certain nombre de contraintes, tel que le temps de calcul souvent important ou bien la difficulté, voire l'impossibilité dans certains cas. Une heuristique est un algorithme qui fournit rapidement (en un temps polynomial) une solution approchée et réalisable, pas nécessairement optimale, pour un problème d'optimisation difficile. Cette méthode approximative est le contraire d'un algorithme exact qui donne une solution optimale pour un problème donné [12].

5.2. Métaheuristiques

Le terme métaheuristique a été inventé par Fred Glover en 1986, lors de la conception de la recherche Tabou [13]. Une métaheuristique est un processus itératif qui subordonne et qui guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace de recherche. Des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou presque-optimales [14]. Les metaheuristique sont utilisables pour résoudre différents problèmes d'optimisation, et ne nécessitant que peu de modifications pour qu'elle puisse s'adapter à un problème particulier. Elles ont donc pour objectif de pouvoir être programmées et testées rapidement sur un problème [15].

6. Présentation des principales Métaheuristique

Il existe un grand nombre de métaheuristiques d'optimisations. Nous allons nous appuyer sur la classification qui distingue les méthodes de trajectoires (qui exploitent séquentiellement un seul voisinage) des méthodes basées sur des populations de solutions (qui exploitent plusieurs à la fois).

6.1. Méthode de trajectoire

Les méthodes à solution unique, plus connues sous le nom de méthodes de recherche locale ou encore méthodes de trajectoire, sont basées sur l'évolution d'une seule solution dans l'espace de recherche. Nous présentons les méthodes les plus utilisées, notamment la méthode de Descente, le Recuit Simule et la recherche Tabou.

6.1.1. Méthode de Descente (Hill Climbing)

Les Méthodes de Descente sont assez anciennes. Leur succès revient à leur simplicité et à leur rapidité [16]**.** Le principe consiste à choisir une solution *s'* dans le voisinage d'une solution *s* en améliorant la recherche tel que *f (s') < f(s)* [17].

On peut décider soit d'examiner toutes les solutions de voisinage et prendre la meilleure de toutes (ou prendre la meilleur trouvée), soit d'examiner un sous ensemble de voisinage. La méthode de Descente peut être décrite comme suit :

Algorithme : Méthode de Descente
Procédure Descente Simple (solution initiale s) Répéter : Choisir s' dans N(s) Si f (s') < f (s) alors s→ s' Jusqu'à ce que f (s') >= f(s), ∀ s' ∈ S

6.1.2. Recuit Simulé (Simulated Annealing)

Cette méthode est issue d'une analogie entre le phénomène physique de refroidissement lent d'un corps en fusion, qui le conduit à un état solide, de basé énergie.

Il faut baisser lentement la température, en marquant des paliers suffisamment longs pour que le corps atteigne l'équilibre thermodynamique à chaque palier de température [18]. Pour les matériaux, cette basse énergie se manifeste par l'obtention d'une structure régulière, comme dans les cristaux et l'acier.

L'analogie exploitée par le recuit simulé consiste à considérer une fonction *f* à minimiser comme fonction d'énergie, et une solution *s* peut être considérée comme un état donné de la matière dont *f(s)* est l'énergie. Le recuit simulé se démarre par une solution initiale s qui peut être prise d'une manière aléatoire dans l'espace de recherche. A cette solution correspond une énergie initiale *f(s)*. Une température initiale *T* élevée est également choisie. A chaque itération de l'algorithme une modification de la solution est effectuée. Cette modification entraine une variation Δ de l'énergie du système. Si cette variation est négative (c'est-à-dire qu'elle fait baisser l'énergie du système), elle est appliquée à la solution courante. Sinon, elle est acceptée avec une probabilité $e^{\Delta/T}$. Ce choix de l'exponentielle pour la probabilité s'appelle règle de

Métropolis. On itère ensuite selon ce procédé en gardant la température constante [19]. L'algorithme Recuit Simulé est présenté ci-dessous :

Algorithme : Recuit Simulé
Choisir solution initiale $s \in S$ et température initiale T ; **Répéter** Choisir aléatoirement $s' \in V(s)$, $\Delta = f(s') - f(s)$ **Si** ($\Delta < 0$) **alors** $s := s'$ **Sinon** Choisir u un nombre aléatoire de [0,1] ; **Si** ($u < e^{\Delta/T}$) alors $s := s'$ **Fin si** **Fin si** **Jusqu'à** condition d'arête satisfaite

6.1.3. Recherche Tabou (tabou search)

La méthode Tabou est une technique de recherche dont les principes ont été proposés pour la première fois par Fred Glover dans les années 80 [13]. A l'inverse du recuit simulé qui génère de manière aléatoire une seule solution voisine $s' \in N(s)$ à chaque itération, tabou examine un échantillonnage de solutions de $N(s)$ et retient la meilleure s' même si s' est plus mauvaise que s. La recherche Tabou ne s'arrête donc pas au premier optimum trouvé. Cependant, cette stratégie peut entraîner des cycles, par exemple un cycle de longueur 2 : $s \rightarrow s' \rightarrow s \rightarrow s'$... Pour empêcher ce type de cycle, on mémorise les k dernières configurations visitées dans une mémoire à court terme et on interdit tout mouvement qui conduit à une de ces configurations. Cette mémoire est appelée la liste tabou, une des composantes essentielles de cette méthode. Elle permet d'éviter tous les cycles de longueur inférieure ou égale à k. La valeur de k dépend du problème à résoudre et peut éventuellement évoluer au cours de la recherche [20].

Algorithme : Recherche Tabou
1 : initialisation s_0 une solution initiale $s \leftarrow s_0$, $s^* \leftarrow$s0, $f^* \leftarrow f(s_0)$ T = ∅ 2: générer un sous-ensemble de solutions au voisinage de s $s' \in V(s)$ tel que $\forall x \in V(s)$, $f(x) \geq f(s')$ et s' T Si $f(s') < f^*$ alors $s^* \leftarrow s'$ et $f^* \leftarrow f(s')$ Mise-à-jour de T 3 : Si la condition d'arrêt n'est pas satisfaite retour à l'étape 2.

6.2. Méthode basées sur les populations de solution

Les métaheuristiques basées sur la manipulation d'une population des solutions essayent d'augmenter la qualité moyenne d'un ensemble des solutions via un mécanisme défini.

On distingue deux approches utilisées par les méthodes basées sur une population de solutions : la première est l'utilisation des mécanismes d'évolution naturels, cette approche est représentée par les algorithmes génétiques et autres. La deuxième est l'utilisation de l'intelligence collective, cette approche est représentée par des colonies fourmis, essaimes particulaires et autres.

6.2.1. Algorithme génétique (Genetic Algorithm)

Les algorithmes génétiques (AG) sont des techniques de recherche et d'optimisation stochastique dérivées de la génétique et des mécanismes de la sélection naturelle et de l'évolution. Un algorithme génétique est un algorithme itératif de recherche globale dont le but est d'optimiser une fonction définie par l'utilisateur. Pour cela, les AGs fonctionnent avec une population regroupant un ensemble d'individus appelés chromosomes distribués sur l'entièreté de l'espace de recherche. Chaque chromosome est constitué d'un ensemble de gènes. Pour chaque individu on attribue une valeur calculée par la fonction d'adaptation (fitness). En pratique, à partir d'une population, des chromosomes sont générés d'une façon aléatoire lors de l'initialisation, dans chaque cycle d'opérations génétiques, une nouvelle population appelée génération est créée à partir des chromosomes de la population courante. Pour cela certains chromosomes appelés 'parents' sont sélectionnés afin d'élaborer les opérations génétiques. Les gènes de ces parents sont mixés et recombinés pour la production d'autres chromosomes appelés (enfants) constituant la nouvelle génération possédant, dans son ensemble, de meilleures

solutions que la précédente. La figure 2.2 illustre les différentes étapes d'un algorithme génétique simple [21], [22] :

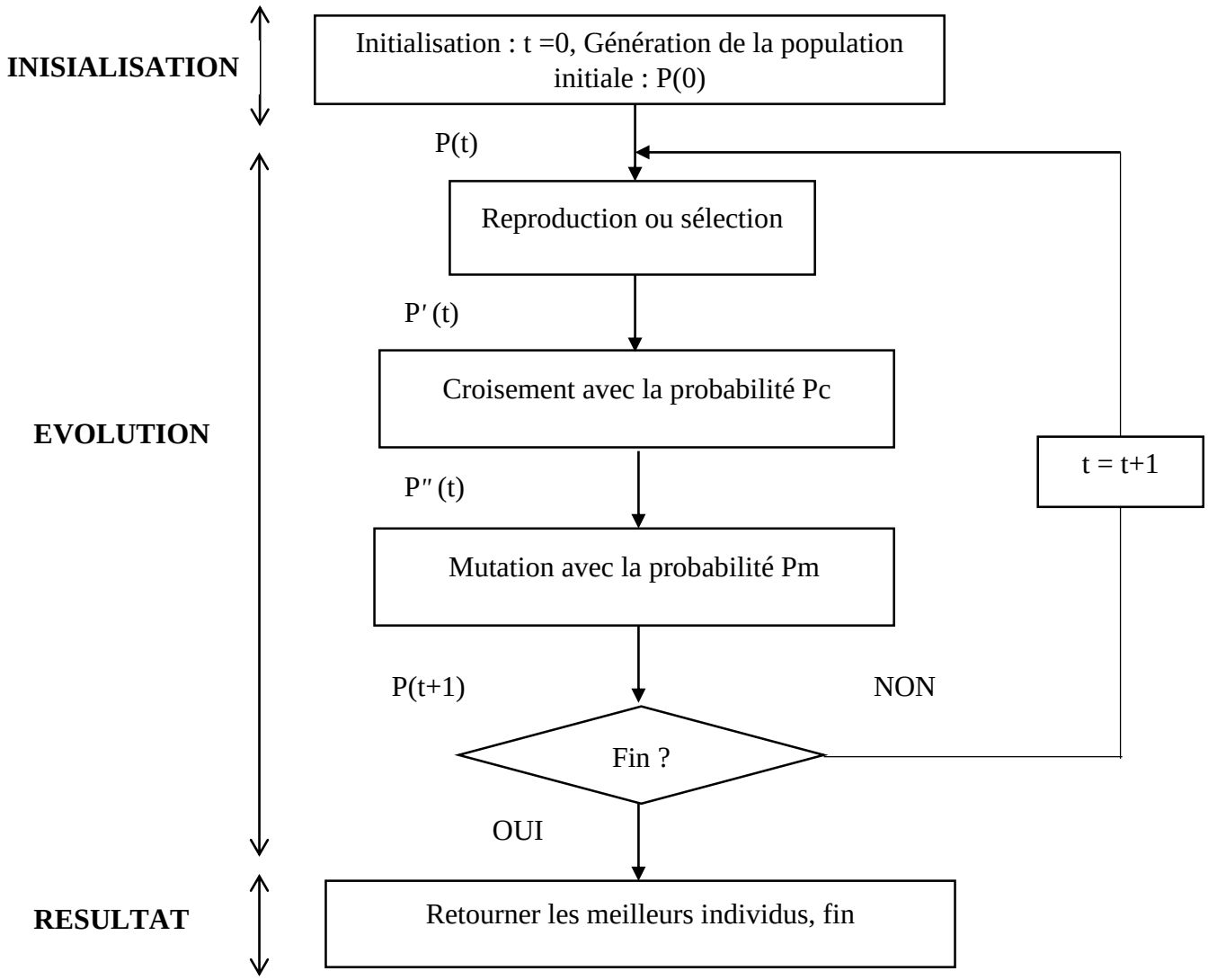

Figure 2.2 : Principe général d'un AG simple.

6.2.1.1. Codage (représentation) d'un individu

Premièrement, il faut représenter les différents états possibles de la variable dont on cherche la valeur optimale sous une forme utilisable par un AE, c'est-à-dire par le codage approprié des solutions sous forme de chromosomes ou génotypes. Cela permet d'établir une connexion entre les valeurs de la variable et les individus de la population, de manière à imiter la connexion qui existe en biologie entre le génotype et le phénotype [23]. Il existe principalement deux types de codage : le codage binaire qui est une représentation sous forme de chaîne binaire et le codage réel qui est une représentation directe des valeurs réelles de la variable.

6.2.1.2. Génération de la population initiale

L'AG démarre avec une population composée de *N* individus dans le codage retenu. Le choix des individus conditionne fortement la rapidité de l'algorithme. Si la position de l'optimum dans l'espace de recherche est totalement inconnue, il est intéressant que la population soit répartie sur tout l'espace de recherche. Si par contre des informations à priori sur le problème sont disponibles, il paraît évident de générer les individus dans un espace particulier afin d'accélérer la convergence. Disposant d'une population initiale souvent non homogène, la diversité de la population doit être entretenue aux cours des générations afin d'explorer le plus largement possible l'espace de recherche. C'est le rôle des opérateurs de croisement et de mutation [24].

6.2.1.3. La fonction d'évaluation

L'évaluation permet de s'assurer que les individus performants seront conservés, alors que les individus peu adaptés seront progressivement éliminés de la population. L'évaluation d'un individu ne dépendant pas de celle des autres individus, pour calculer le coût d'un point de l'espace de recherche, on utilise une fonction d'évaluation *f*, cette fonction est souvent une transformation g de la fonction objectif $f(x) = g(\Phi(x))$, appelée aussi la fonction fitness, le résultat fourni par cette fonction va permettre de sélectionner ou de refuser un individu pour ne garder que les individus ayant le meilleur coût en fonction de la population courante [34].

6.2.1.4. Operateurs de sélection

Pour sélectionner les meilleurs individus (chromosomes) dans une population, il existe plusieurs types de sélection, chacune étant adaptée à un type particulier de problèmes. On en cite les plus classiques qui sont: la sélection *roulette biaisée* (*Roulette wheel selection*), la sélection *rang* (*selection by rank*), la sélection *par tournoi* (*tournament selection*) et la sélection *uniforme* (*uniform selection*).

- **Sélection par roulette**

La sélection par roulette peut être vue comme une sorte de roulette de casino sur laquelle sont mis tous les chromosomes de la population. La place accordée à chaque chromosome étant en relation avec sa valeur d'adaptation. Un exemple de roulette de sélection est représenté par la (figure 2.3).

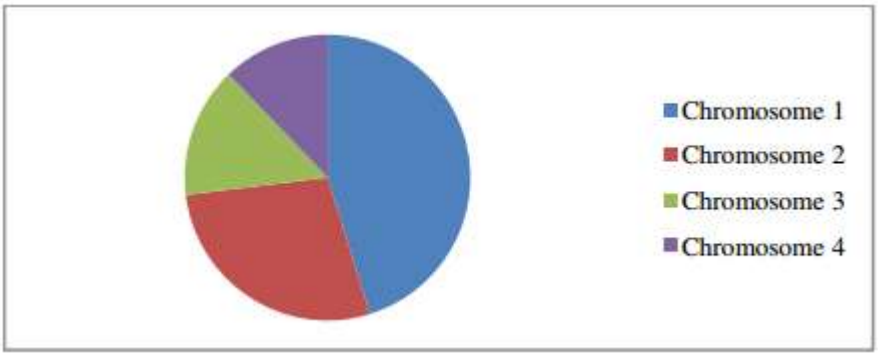

Figure 2.3 : Schéma d'une roulette de sélection [25].

❖ **Sélection par rang**

La sélection par rang effectue d'abord un tri sur la population par rapport à la fitness. Ensuite, à chaque chromosome est associe un rang en fonction de sa position. En conséquence, pour une population de *N* chromosomes, le plus mauvais aura le rang 1, le suivant 2, et ainsi de suite jusqu'au meilleur chromosome qui aura le rang *N*. la sélection par rang et la sélection par roulette sont presque identiques, sauf que, avec la sélection par rang les populations sont en rapport avec le rang plutôt qu'avec la valeur de l'évolution. Ainsi, les probabilités finales seront calculées avec la formule : Rang / ∑ (des rangs) [25].

Chromosomes	1	2	3	4	5	6	**Total**
Probabilités initiales	89%	5%	1%	4%	3%	2%	100%
Rang	6	5	1	4	3	2	21
Probabilités finales	29%	24%	5%	19%	14%	9%	100%

Table 2.1 : Exemple de sélection par range pour 6 chromosomes.

❖ **Sélection par tournoi**

On forme m paires de chromosomes dans une population de m chromosomes. Dans les paramètres de l'AG, on décide d'une probabilité de victoire du plus fort. Cette probabilité représente la chance que possède le meilleur chromosome de chaque paire d'être sélectionné. Cette probabilité doit être grande (entre 70% et 100%). A partir des m paires, on détermine ainsi m individus pour la reproduction [25].

❖ **Sélection uniforme**

C'est une technique très simple qui consiste à sélectionner un individu *Ci* aléatoirement de la population *P*. La probabilité pi pour qu'un individu soit sélectionné est définie par [26]:

$$Pi = 1 / taille\text{-}pop$$

6.2.1.5. Opérateurs de croisement

Le croisement est l'opérateur de recherche essentiel d'un AG. Il a lieu après la sélection. Il permet l'échange d'informations génétiques entre les individus après sélection, de façon aléatoire en une (ou plusieurs) position(s) de césure (locus). Dans ce cas, deux des individus retenus de la sélection (parents) échangent une ou plusieurs parties de leurs génotypes, selon une probabilité *Pc* (souvent supérieure à 60%). Pour former deux individus différents de ceux d'origine. Il existe différentes manières pour croiser deux génotypes, les trois les plus connues : le croisement "à un point" (one-point), "à deux points" (two-points) [27], [28].

❖ **Croisement a « 1 point »**

Le croisement "à un point" peut être illustré comme dans la figure 2.4. Il consiste à sélectionner aléatoirement un emplacement (une position) de césure et de permuter les parties droites des deux parents.

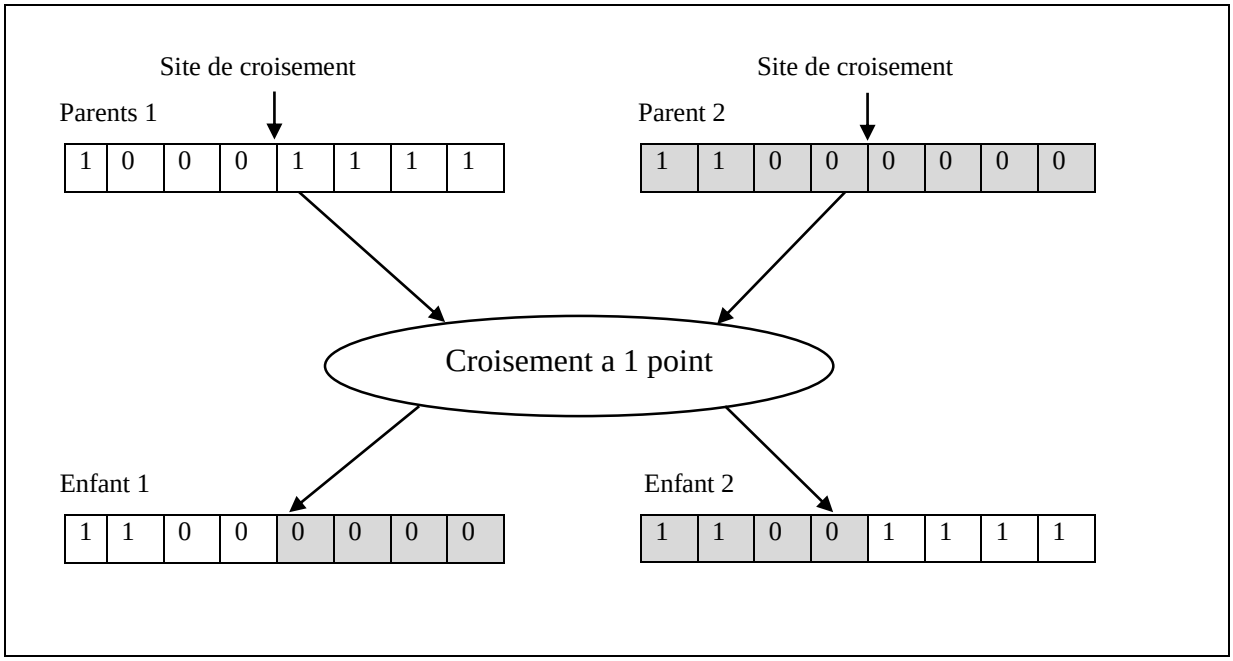

Figure 2.4 : Représentation schématique du croisement en un point.

❖ **Croisement a « 2 point »**

Ce croisement diffère du premier, tout simplement, du fait qu'il s'effectue en deux points de coupure, comme le montre la figure 2.5 [29].

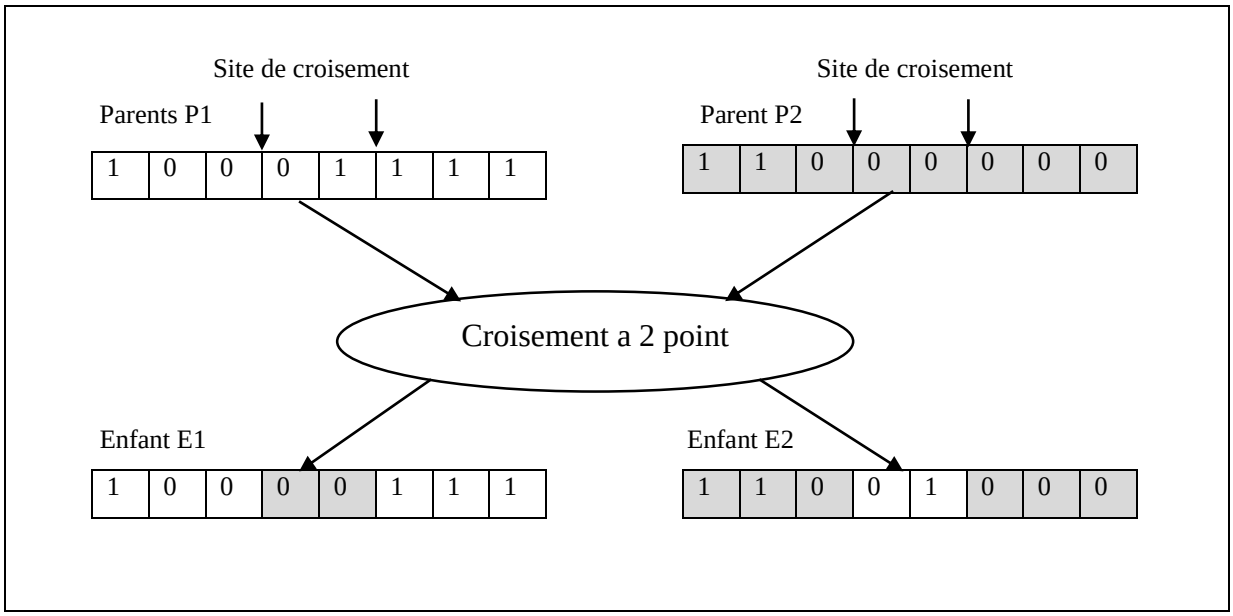

Figure 2.5 : Représentation schématique du croisement en deux points.

6.2.1.6. Opération de mutation

La mutation est un changement aléatoire selon une certaine règle probabiliste qui doit faire sur les génotypes, avec une faible probabilité *Pm* (fixée par l'utilisateur) de la valeur d'un ou plusieurs allèles d'un chromosome. En général, la mutation ne permet pas l'obtention de meilleures solutions, mais elle permet de garder une diversité dans l'évolution des individus et d'éviter les optimums locaux, et se protège contre une perte irrécouvrable dans les caractéristiques des individus. La mutation classique consiste à transformer dans un chromosome binaire un 1 en un 0 ou le contraire. (Figure 2.6) [30].

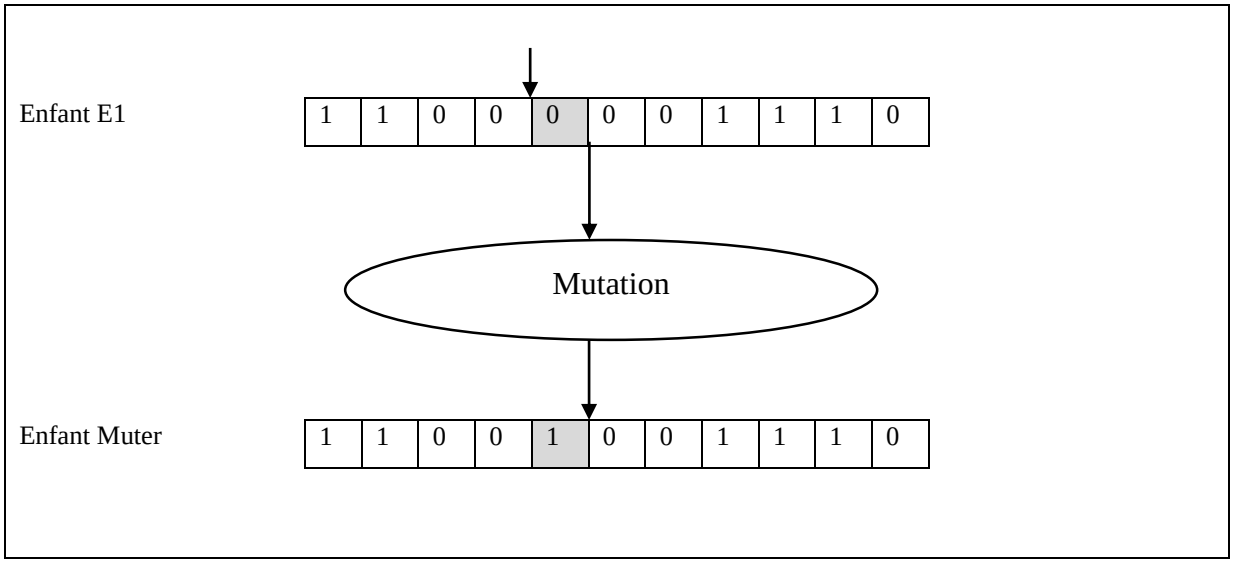

Figure 2.6 : Représentation schématique de la mutation.

6.2.2. L'algorithme d'optimisation par essaim de particules (PSO)

L'optimisation par Essaim de particul1e (PSO) est une technique utilisée pour explorer l'espace de recherche d'un problème quelconque pour trouver l'ensemble des paramètres qui maximise/minimise un objectif particulier. Kennedy et Eberhart [31] ont cherché à simuler la capacité des oiseaux à voler de façon synchrone, et leur aptitude à changer brusquement de direction tout en restant en une formation optimale.

6.2.2.1. Principe de fonctionnement

Dans un système PSO, un essaim d'individus (particules) volent dans l'espace de recherche. Chaque particule représente une solution potentielle au problème d'optimisation. La position d'une particule est influencée par la meilleure position visitée par elle-même (c.-à-d. ses propres expériences) et la position de la meilleure particule dans son voisinage (c.-à-d. l'expérience des particules voisines). Quand le voisinage d'une particule est l'essaim entier, la meilleure position dans le voisinage exprime la meilleure particule globale, et l'algorithme résultant désigné sous le nom d'un PSO gbest. Si le voisinage est petit l'algorithme est généralement connu sous le nom d'un PSO lbest.

La performance de chaque particule (c.-à-d. la convergence de la particule vers l'optimum global) est mesurée en utilisant une fonction de forme physique qui change selon le problème d'optimisation. Chaque particule dans l'essaim est représentée par les caractéristiques suivantes :

x_i : La position actuelle de la particule i.

v_i : La vitesse courante de la particule i.

y_i : La meilleure position personnelle de la particule i.

$\hat{y}_i$: La meilleure position de voisinage de la particule i.

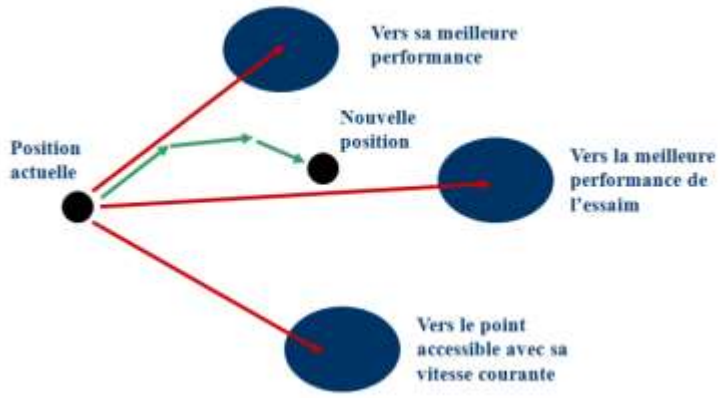

Figure 2.7 : Déplacement d'une particule.

La meilleure position personnelle de la particule i est la meilleure position (c- à- d, celle ayant pour résultat la meilleure valeur de forme physique) visitée par la particule i jusqu'ici. La mise à jour de la meilleure position de la particule est comme suit :

$$yi(t+1)\begin{cases} yi\ (t) si\ f\big(xi(t+1)\big) > f(yi(t) \\ xi(t+1)\ si\ f\big(xi(t+1)\big) < f(yi(t) \end{cases} \quad (2.1)$$

Pour le modèle gbest, la meilleure particule est déterminée à partir de l'essaim entier par le choix de la meilleure position personnelle. Si la position de la meilleure particule est notée par le vecteur $\hat{y}_i$ alors :

$$\hat{y}i\ (t)\ \in \{y0, y1, \dots, ys\} = \min\{f\big(y0(t)\big), f\big(y1(t)\big), \dots\dots\dots, f(ys(t))\} \quad (2.2)$$

Où s indique la taille de l'essaim.

L'étape de la mise à jour de vitesse est indiquée pour chaque dimension $j \in 1,\dots, N_d$ Par conséquent, $v_{i,j}$ représente le $j^{ème}$ élément du vecteur de vitesse de la $i^{ème}$ particule.

Ainsi la vitesse de la particule i est mise à jour en utilisent l'équation suivante :

Ou :

w est le facteur d'inertie. c_1 et c_2 sont les constantes d'accélérations. r_1 et r_2 : suivant une loi uniforme.

La position de la particule *i*, *xi* est mise à jours par l'équation suivante :

$$xi(t+1) = xi(t) + vi(t+1) \qquad (2.3)$$

C'est la vectrice vitesse qui dirige le processus de recherche et reflète la "sociabilité" des particules. Si on considère *N* particules et que chaque particule compare sa nouvelle position à sa meilleure position obtenue, c'est-à-dire qu'on ne prend pas compte du voisinage puisqu'on utilise uniquement l'amélioration obtenue sur la particule elle-même, cela donne l'algorithme [32], [33].

7. Conclusion

Dans ce chapitre nous avons passé en revue les problèmes d'optimisation et ainsi leurs complexités. Ensuite, nous avons présenté les principales métaheuristiques couramment utilisées pour la résolution des problèmes d'optimisation avec une significative concentration sur les algorithmes génétiques que nous allons utiliser dans notre travail.

CHAPITRE 3

APPLICATION DE L'ALGORITHME GENETIQUE AU PROBLEME *k*-PCC

1. Introduction

Dans ce chapitre, nous allons présenter le problème de *k* plus courts chemins (*k*-PCC). Ensuite on va détailler l'adaptation de l'algorithme génétique pour résoudre ce problème. Une grande partie sera consacrée aux différents composants de l'algorithme ainsi que son principe de fonctionnement.

2. Présentation du problème *k*-PCC

Un réseau est généralement représenté comme un graphe pondéré G = (*N*, *E*), ou $N = \{1,\ldots, n\}$, désigne l'ensemble de nœuds et $E = \{e_1,\ldots, e_m\}$ désigne l'ensemble de liens de communication reliant les nœuds. Soit $M = \{s, u_1, u_2,\ldots, u_m\} \in N$ être un ensemble de source de formulaire aux nœuds de destinations, ou *S* est le nœud source et $D = \{u_1, u_2,\ldots u_m\}$ désigne un ensemble de nœuds de destination. P (s, u_i) est un chemin du nœud source *S* au nœud de destination u_i. Le chemin *P* est le plus court chemin (PCC) si la bande passante de ce chemin est égale à la valeur constante *B* (cette valeur est déterminée de l'utilisateur ou est une valeur nécessaire de la bande passante). La bande passante de *P (Band(P))* est la valeur minimale de la bande passante du lien (*Band*(*e*)) en *P*.

$$Band\ (P) = min\ (Band\ (e), e \in Ep) \qquad (3.1)$$

Par conséquent, Le problème de la bande passante a contraint le *k* plus court chemin doit trouver tous les chemins du nœud source à chaque nœud de destination qui satisfait [36]:

$$Band\ (P) \geq B. \qquad (3.2)$$

Exemple :

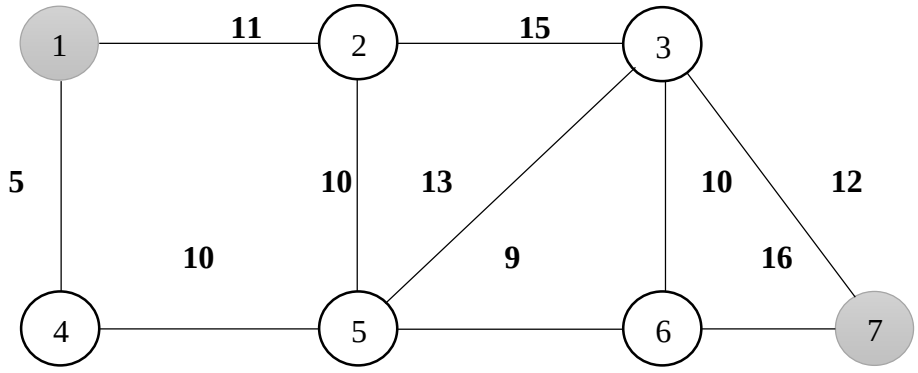

Figure 3.1 : Un graphe orienté.

Dans cette exemple, les chemins valides entre le nœud source (s = 1) et le nœud destination (d = 7) pour le contrainte de valeur B =10 est : P1 = (1, 2, 3, 7), P2 = (1, 2, 5, 3, 7), P3 = (1, 2, 3, 6, 7).

2.1. Domaines applications de problème *K*-PCC

Le problème *K*-PCC consiste à trouver les *k* plus courts chemins dans un graphe pondéré sous une contrainte, qui peut être par exemple la vérification d'une condition sur un seuil pour une bande passante. Les récents exemples de domaine d'application pour *K*-PCC problème incluent la logistique, l'analyse financière, la planification, l'optimisation les a routage réseau, le suivi d'objets multiples, l'alignement de séquences de gènes, le problème d'optimisation de réseau, le routage dynamique et bien d'autres domaines dans lesquels des problèmes d'optimisation doivent être résolus [35].

3. Adaptation de l'algorithme AG au problème *k*-PCC

Dans cette partie on va détailler les différents composants de l'algorithme génétique adapté à la résolution du problème *k*-PCC afin de trouver une solution approchée sous forme d'un ensemble de plus courts chemin d'une source *i* à une destination *j* dans un graphe donné.

3.1. Représentation du chromosome

Un chromosome de l'AG consiste en des séquences d'entiers positifs qui représentent les ID des nœuds par lesquels passe un chemin. Chaque locus du chromosome représente un ordre d'un nœud (indiqué par le gène du locus) dans un chemin. Le gène du premier locus est toujours réservé au nœud source. La longueur du chromosome est variable, mais elle ne doit pas dépasser la longueur maximale N, où N est le nombre total de nœuds dans un graphe. Donc, on va coder chaque solution (chromosome) par une chaine d'entiers qui représenté un chemin valide entre le nœud source (s) et le nœud destination (d), le chromosome contient n nœuds. La figure ci-dessous représente chromosome valide.

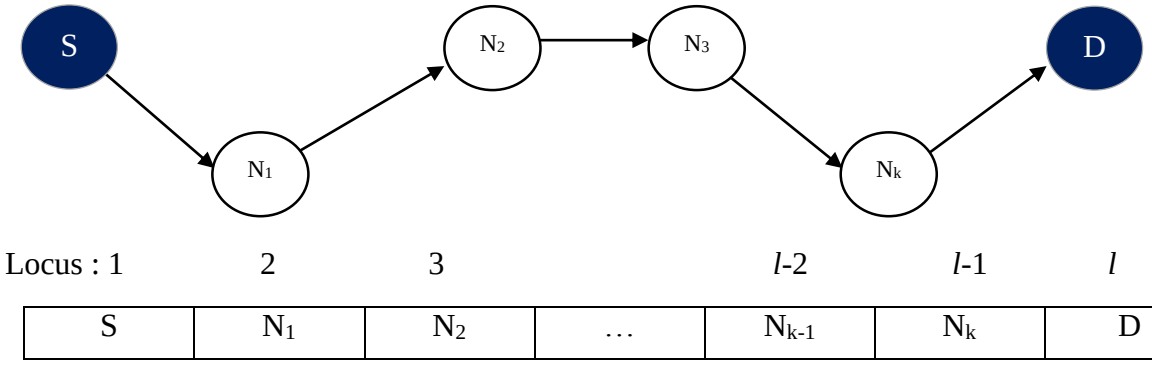

Figure 3.2 : Représentation du chromosome.

On montre un exemple de codage chromosome (chemin) du nœud *S* (source) au nœud *D* (destination) est représenté sur la Fig. 3.2. Le chromosome est essentiellement une liste de nœuds le long du chemin construit ($S \rightarrow N1 \rightarrow N2 \rightarrow N3 \rightarrow ... \rightarrow Nk\text{-}1 \rightarrow Nk \rightarrow D$). Dans la Fig.3.1. *l* représente le nombre total de nœuds formant un chemin.

Le gène du premier locus code le nœud source et le gène du second locus est choisi de façon aléatoire ou heuristique à partir des nœuds connectés au nœud source représenté par l'allèle du gène avant. Un nœud choisi est supprimé de la base de données d'informations pour empêcher que le nœud ne soit sélectionné deux fois, évitant ainsi les boucles sur le chemin. Ce processus se poursuit jusqu'à ce que le nœud de destination soit atteint [37].

3.2. Population initiale

La population initiale est un ensemble des solutions, chaque solution parmi ces solutions satisfait les contraintes du notre problème. Pour créer la population initiale, nous avons adapté la stratégie suivante :

- **Etape 1 :** Prendre aléatoire une chaine d'entier de *S* (source) à *D* (destination).
- **Etape 2 :** Si deux nœuds ne sont pas connectés par un lien alors éliminer ce chromosome et aller a étape 1.
- **Etape 3 :** Répéter 1 et 2 jusqu'à ce que génération d'un nombre *pop_size* de chromosome soit atteint.

3.3. La fonction fitness

Chaque chromosome apporte une solution potentielle au problème. Néanmoins, ces solutions n'ont pas toutes le même degré de pertinence. C'est à la fonction de performance de mesurer cette efficacité, pour permettre à l'AG de faire évoluer la population dans un sens bénéfique pour la recherche de la meilleure solution [38].

La fonction fitness dans notre problème c'est la plus petite valeur sur le chemin qui doit être $\geq B$, B est donné.

$$Fitness\ (P) = min\ [Band\ (e), \in Ep \geq B] \qquad (3.3)$$

3.4. Les stratégies de sélection

La sélection permet d'identifier statistiquement les meilleurs individus d'une population, de leur assurer une descendance, c'est-à-dire de favoriser leur présence dans la génération suivante, et d'éliminer les mauvais [39]. A chaque itération, on sélectionne un ensemble d'individus pour la préparation à la reproduction, afin de reproduire une nouvelle génération. Dans notre problème sujet d'étude on a choisie d'utiliser la sélection par roulette.

3.5. Operateur de croissement

On applique cet opérateur avec une probabilité *Pc* qui appartient à [0,1]. Dans notre problème nous avons utilisé le croisement un à point, qui consiste à choisir deux chromosomes parent (deux chemins), et on choisit un nœud de croisement entre [2, *n*-1] et qui doit être commun pour ces deux chemins pour former deux nouveaux fils.

La figure suivante (figure 3.3) présente un graphe a 9 sommets qu'il contient un nœud source $s = 1$, et nœud de destination $d = 9$.

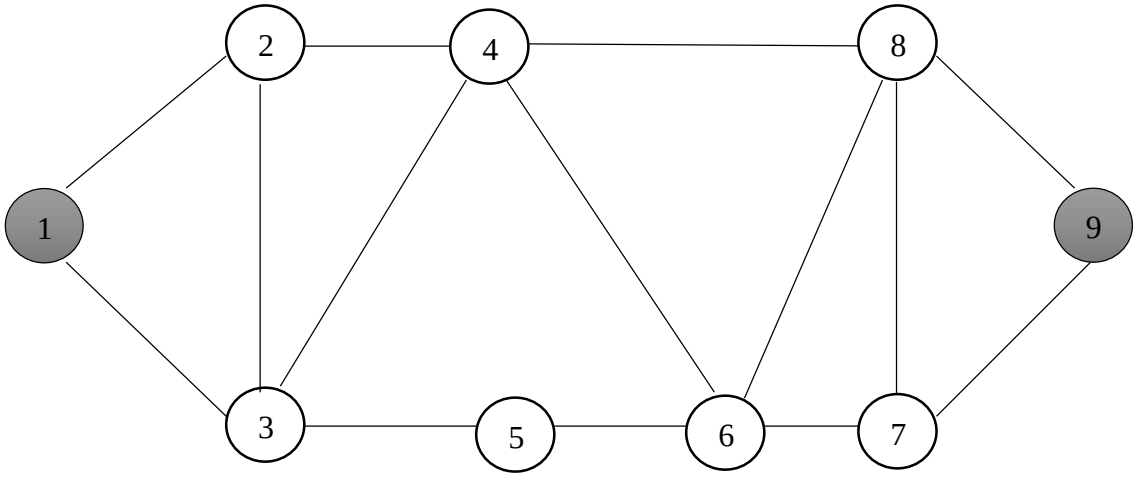

Figure 3.3 : Exemple d'un graphe**.**

Les figures ci-dessous illustrent le fonctionnement de ce croisement, les descendants gardent chacun la première partie de chemin avant le point de croisement de l'un des parents. Le reste du chemin est échangé avec la deuxième partie de l'autre parent. Dans l'exemple suivant, l'enfant 1 hérite le bout de chemin (1, 2, 4, 6) du parent 1. Puis, il hérite du reste de chemin (6, 7, 9) du parent 2.

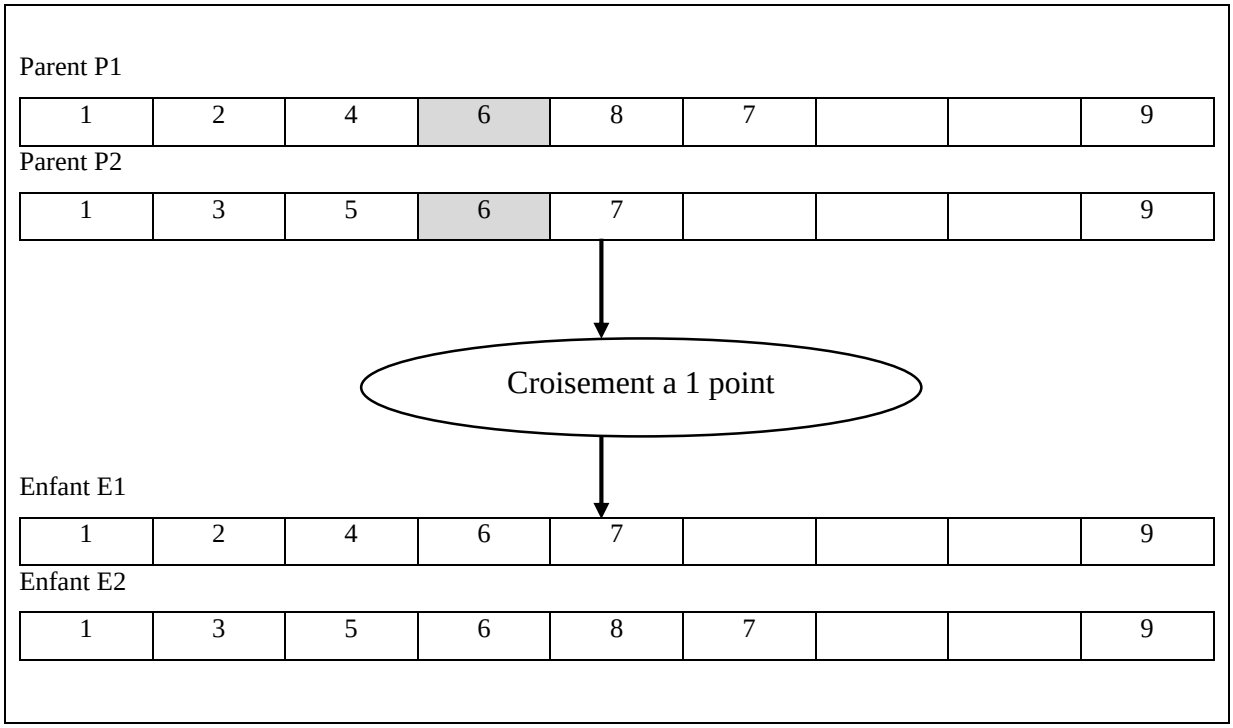

Figure 3.4 : Croisement a un point.

Les croisements sont à la base des algorithmes génétiques. Quoi qu'il en soit, il se peut que l'action conjointe de la reproduction et du croisement soit insuffisante pour assurer la réussite de l'AG.

3.6. Operateur de mutation

La mutation des individus correspond à des changements aléatoires de passage par un nœud d'une solution. Cependant, ces changements doivent respecter la réalisabilité du chemin.

On applique cet opérateur avec une probabilité Pm qui appartient à [0.001, 0.01]. La technique de mutation retenue pour notre étude consiste à choisir aléatoirement un point de mutation qui correspond à un nœud dans le graphe, puis démarrer en suivant un autre sous-chemin de ce point jusqu'à la destination D.

La figure suivante montre un exemple de mutation appliqué sur l'enfant 1 du dernier croisement (figure 3.4) :

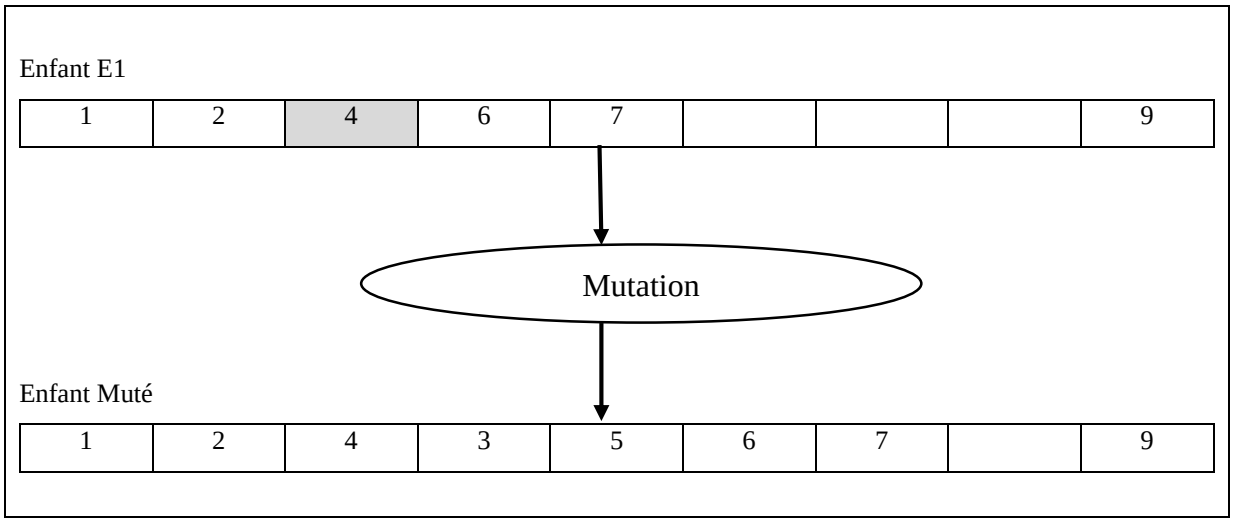

Figure 3.5 : Opérateur de mutation.

3.7. Remplacement

C'est la phase pour construire la nouvelle génération. Pour assurer cette étape, on va garder les nœuds qui ont fitness(Chromosome) ≥ B, ce qui reflète la contrainte définie dans problème.

3.8. Le fonctionnement de sélection par roulette dans notre système

Dans cette partie, nous allons présenter la méthode de sélectionne par roulette pour appliquer des dans notre système.

La sélection : la méthode choisie est la sélection par roulette, cette méthode est basée sur :

La premier étape : calculer la probabilité de chaque individu qui sous la forme suivante : $Pi = \frac{fi}{\sum_{j=0}^{N} fi}$ ou fi est fitness

La deuxième étape : Pour tous les individus de la population, A chaque fois en sélectionne deux individus au hasard, puis on calcule la probabilité de chaque individu.

Le pseudo-code de l'algorithme développe est donné dans la table 3.1 ci-dessous :

Table 3.1 : Algorithme génétique développé

Début
Entrées :
Graphe G value et sans cycles,
Initialiser : B // La valeur su seuil B
Pop_size // Taille de la population
Pc, Pm // probabilités de croisement et de mutation
Nb_itération // Nombre d'itérations
S, D // Nœud source et nœud destination
Générer la population initiale par des chemins valides
Répéter
ng = 1**;**
Tant que (*ng* ≤ *Nb_itération***) faire**
i=1 ;
Tant que (i ≤ *Pop_size*) **faire**
Sélectionner deux parents (P1, P2) par la méthode de roulette;
Faire le croisement de ces deux parents sélectionnés selon une probabilité *Pc*.
Muter les fils selon une probabilité *Pm*.
Calculer bande passante pour chaque chromosome.
Si (Band(i) ≥ *B*) **alors**
Sauvegarder les nouveaux enfants comme solution.
i++ ;
Fin si
Fin tant que
jusqu'un (ng == *Nb_itération*)
Fin tant que
Sorties : Imprimer toutes les solutions obtenues.

4. Conclusion

Nous avons vu dans ce chapitre le problème de (*k*-PCC), l'examen de son utilité dans les applications réelles, ensuite nous avons décrit les composants essentiels de l'algorithme génétique choisi pour la résolution du problème posé. Le chapitre suivant, on va présenter l'ensemble de résultats obtenus sur quelques graphes.

CHAPITRE 4

REALISATION ET EXPERIMENTATIONS

1. Introduction

L'implémentation d'un logiciel vient après un enchainement de plusieurs étapes dans le processus de développement, son but principal est de réaliser un produit capable de résoudre les problèmes posés on utilisant des outils et des algorithmes.

Dans ce dernier chapitre, nous présentons les différents résultats de simulations que nous avons réalisés.

2. Environnement de développement

Pour la réalisation de ce travail, nous avons eu recours aux environnements suivants :

2.1. Environnement matériel

Pour développer l'application, nous avons utilisé comme environnement matériel :

Système

Processeur :	Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz 2.50 GHz
Mémoire installée (RAM) :	4,00 Go
Type du système :	Système d'exploitation 64 bits, processeur x64
Stylet et fonction tactile :	La fonctionnalité de saisie tactile ou avec un stylet n'est pas disponible sur cet écran.

2.2. Environnement logiciel

- Windows 8.1 comme système d'exploitation
- Le Microsoft Visual studio 2012 comme Environnement de développement facile a utilisé.
- Langage C++ comme langage de programmation.

2.2.1. Le Microsoft Visual Studio 2012



Figure 4.1 : Visuel studio 2012.

Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications Web ASP.NET, des services Web XML, des applications bureautiques et des applications mobiles. Visual Basic, Visual C# et Visual C++ utilisent tous le même environnement de développement intégré (IDE), qui permet le partage d'outils et facilite la création de solutions à plusieurs langages. Par ailleurs, ces langages utilisent les fonctionnalités du .NET Framework, qui fournit un accès à des technologies clés simplifiant le développement d'applications Web ASP et de services Web XML [40].

2.2.2. Le langage de programmation C++

C++ est un langage de programmation compilé, permettant la programmation sous de multiples paradigmes comme la programmation procédurale, la programmation orientée objet et la programmation générique. Le langage C++ n'appartient à personne et par conséquent n'importe qui peut l'utiliser sans besoin d'une autorisation ou obligation de payer pour avoir le droit d'utilisation. C++ est l'un des langages de programmation les plus populaires, avec une grande variété de plates-formes matérielles et de systèmes d'exploitation [41].

3. Paramètres de l'algorithme

Les différents paramètres de l'algorithme génétique choisi sont résumés dans le tableau ci-dessous :

Paramètre	**Description**	**Nature**
Pop_size	La taille de population	Entrée
Nbr_itération	Le Nombre de génération	Entrée
Pc	La probabilité de croisement	Nombre aléatoire entre [0, 1]
Pm	La probabilité de mutation	Nombre aléatoire entre [0, 1]
S	Le Nœud source	Entrée
D	Le Nœud destination	Entrée
B	Valeur de la contrainte	Entrée
K	Le nombre de solution possible	Sortie
U	Le nombre de destination	Entrée

Table 4.1 : Paramètres de l'algorithme.

4. Résultats obtenus

Dans cette partie, nous allons représenter un ensemble des résultats obtenus par les tests sur notre problème.

4.1. Premier exemple

Dans le premier exemple, nous considérons un graphe avec 8 sommets, chaque lien correspond à une bande passante (Figure 4.2).

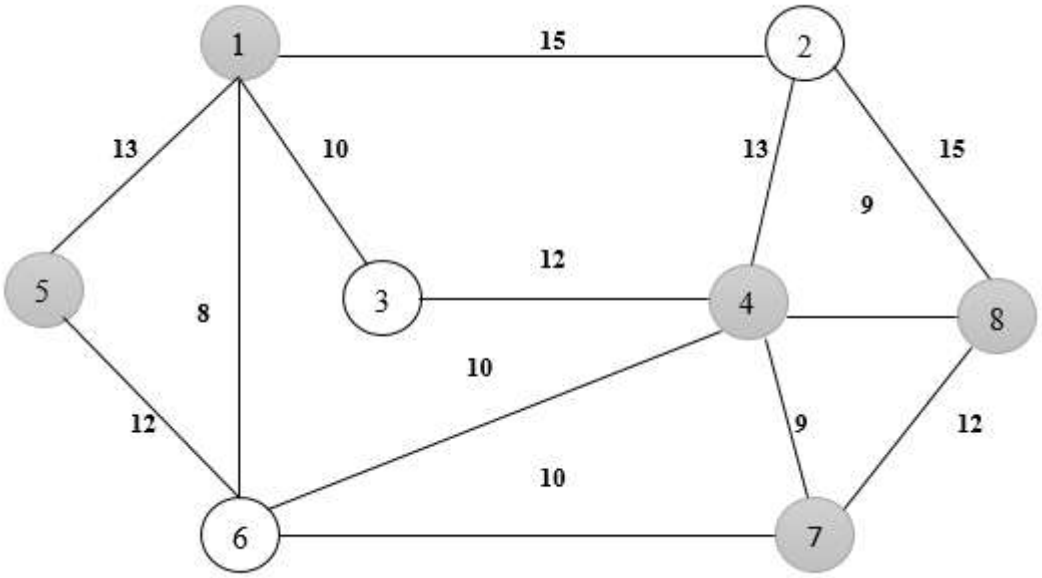

Figure 4.2 : Représentation d'un graphe à 8 sommets.

4.1.1. Tests et résultats

Nous avons testé l'algorithme génétique développé pour chercher l'ensemble des *k* plus courts chemins du nœud source vers les nœuds de destinations. Dans notre exemple, nous utilisons les paramètres suivants: *pop_size* = 25, *Pm* = 0.2, *Pc* = 0.9, *nbr_itération* = 600, le nœud source *S* = 1, et les nœuds de destinations sont *D* = {4, 5, 7, 8}. Le tableau 4.2 présente les résultats obtenus avec la valeur de *B* = 8. Et le tableau 4.3 présente les résultats obtenus avec la valeur de *B* = 10. Les tableaux 4.4 présente la performance de l'algorithme génétique implémenté.

Table 4.2 : Résultats obtenus pour (*pop_size* = 25, *B* = 8, *nbr_itération* = 600)

Nœud Source	Nœud Destination	Les plus courts chemins	Valeur de B	K
1	4	1- 2- 8- 7- 4 1- 5- 6- 7- 4 1- 6- 7- 4 1- 6- 7- 8- 4 1- 5- 6- 4 1- 6- 7- 8- 2- 4 1- 2- 8- 4 1- 6- 4 1- 2- 8- 7- 6- 4 1- 2- 4 1- 3- 4	9 9 8 8 10 8 9 8 10 13 10	11
1	5	1- 3- 4- 6- 5 1- 2- 4- 6- 5 1- 2- 8- 4- 6- 5 1- 3- 4- 7- 6- 5 1- 2- 8- 7- 6- 5 1- 6- 5 1- 2- 8- 4- 7- 6- 5 1- 2- 4- 7- 6- 5	10 10 9 9 10 8 9 9	8
1	7	1- 2- 4- 6- 7 1- 2- 4- 7 1- 3- 4- 8- 7 1- 3- 4- 6- 7 1- 2- 8- 4- 7 1- 5- 6- 7 1- 5- 6- 4- 8- 7 1- 5- 6- 4- 7 1- 3- 4- 7 1- 6- 4- 7 1- 2- 4- 8- 7 1- 6- 7 1- 2- 8- 4- 6- 7 1- 2- 8- 7 1- 6- 4- 8- 7 1- 6- 4- 2- 8- 7	10 9 9 10 9 10 9 9 9 8 9 8 9 12 8 8	16
1	8	1- 6- 7- 4- 8 1- 3- 4- 7- 8 1- 5- 6- 7- 8 1- 6- 7- 4- 2 – 8 1- 3- 4- 2- 8 1- 5- 6- 4- 7- 8 1- 2- 4- 8 1- 6- 4- 7- 8 1- 2- 4- 7- 8 1- 6- 7- 8 1- 3- 4- 8 1- 6- 4- 8 1- 6- 4- 2- 8 1- 2- 8 1- 2- 4- 6- 7- 8	8 9 10 8 10 9 9 8 9 8 9 8 8 15 10	16

		1- 3- 4- 6- 7- 8	10	

Table 4.3 : Résultats obtenus pour (*pop_size* = 25, *B* = 10, *nbr_itérations* = 600)

Nœud Source	**Nœud Destination**	**Les plus courts chemins**	**Valeur de B**	**K**
1	4	1- 5- 6- 4 1- 2- 8- 7- 6- 4 1- 5- 6- 7- 8- 2- 4 1- 3- 4 1- 2- 4	10 10 10 10 13	5
1	5	1- 2- 4- 6- 5 1- 3- 4- 6- 5 1- 2- 8- 7- 6- 5 1- 5	10 10 10 13	4
1	7	1- 2- 8- 7 1- 3- 4- 2- 8- 7 1- 5- 6- 7 1- 3- 4- 6- 7 1- 2- 4- 6- 7	12 10 10 10 10	5
1	8	1- 5- 6- 7- 8 1- 3- 4- 2- 8 1- 2- 8 1- 2- 4- 6- 7- 8 1- 3- 4- 6- 7- 8 1- 5- 6- 4- 2- 8	10 10 15 10 10 10	6

Le nombre de plus courts chemins donné par l'énumération complète et notre algorithme génétique est résumé dans le tableau 4.4 ci-dessous :

Table 4.4 : Nombre de chemins trouvés.

	Nœud source = 1, B = 8		
Nœuds destinations	Enumération complète	Algorithme génétique	Performance
4	13	12	92 %
5	13	12	92 %
7	17	16	94 %
8	17	16	94 %
	Nœud source = 1, B = 10		
4	05	05	100 %
5	04	04	100 %
7	06	05	83 %
8	06	06	100 %

Les résultats présentés dans les tableaux 4.2, 4.3 et 4.4 montrent clairement que l'algorithme génétique implémenté a réussi de trouver dans un temps raisonnable un nombre important des *k*-plus courts chemins du nœud source vers les nœuds destinations même dans le cas où la valeur de *B* est changée. Ce nombre est acceptable par rapport au nombre de chemins possibles par énumération complète.

4.2. Deuxième exemple

Dans le deuxième exemple, nous considérons un graphe avec 20 sommets, chaque lien correspond à une bonde passante (figure 4.3)

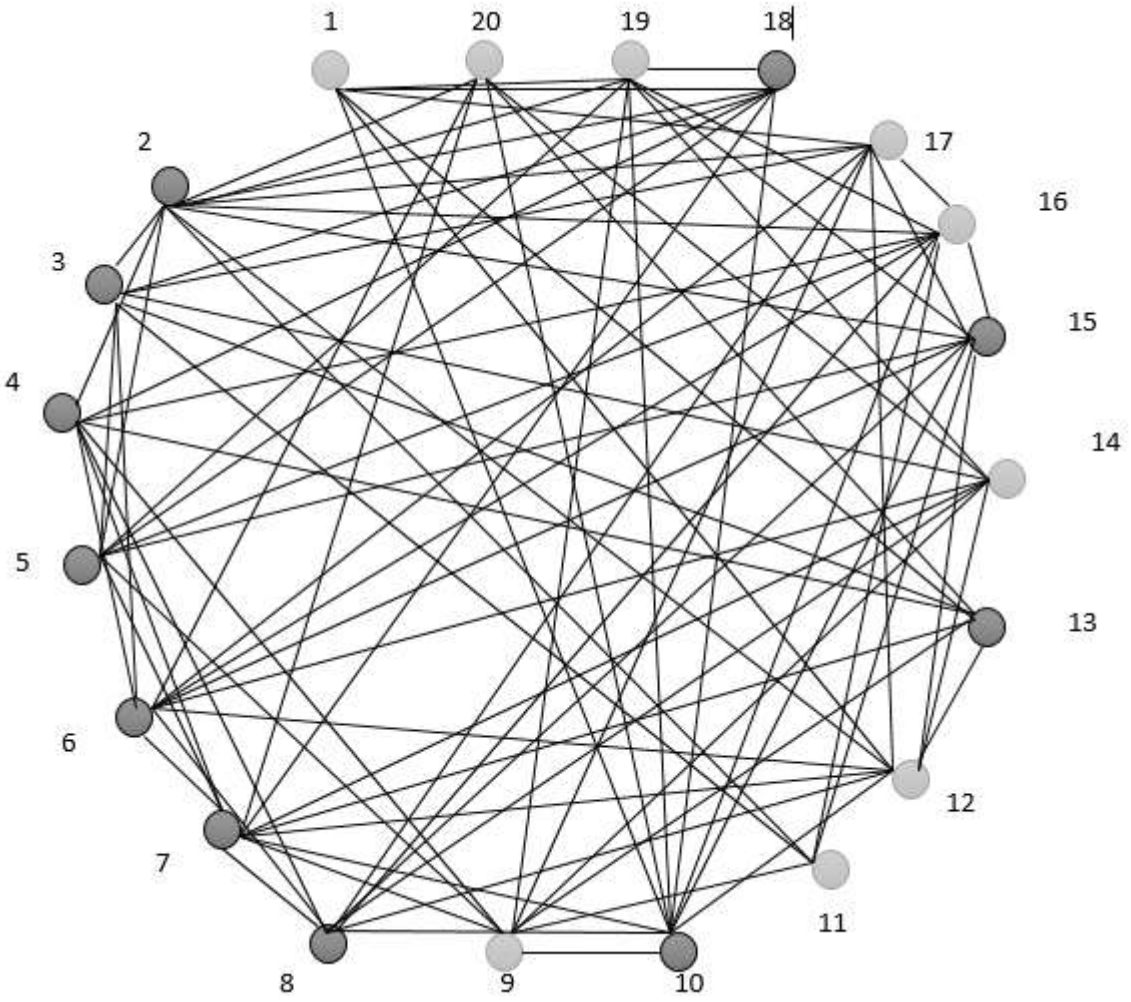

Figure 4.3 : Représentation d'un graphe avec 20 sommets.

4.2.1. Tests et résultats

Nous avons testé l'algorithme génétique développé pour chercher l'ensemble des *k* plus courts chemins du nœud source vers les nœuds de destinations. Dans notre exemple, nous utilisons les paramètres suivantes: *pop_size* = 25, $Pm \geq 0.1$, *Pc* = 0.9, *nbr_itération* = 6000, le nœud source *S* = 1, les nœuds de destinations sont *D* = {9, 11, 12, 14, 16, 17, 19, 20}. Le tableau 4.5 présente les résultats obtenus avec la valeur de *B* = 9, par contre le tableau 4.6 présente les résultats obtenus pour chaque nœud de destination à *N* générations. Le tableau 4.7 présente les résultats de l'effet de la variation de la probabilité de mutation.

Table 4.5 : Résultats obtenus pour (*pop_size* = 25, *B* = 9, *nbr_itération* = 6000)

Nœud Source	**Nœud Destination**	**Les plus courts chemins**	**Valeur de B**	**K**
1	9	1- 17- 3- 13- 7- 9 1- 17- 3- 13- 7- 4- 9 1- 17- 2- 16- 13- 7- 9 1- 17- 2- 16- 13- 7- 4- 9 1- 17- 2- 5- 16- 13- 7- 4- 9 1- 17- 2- 16- 4- 7- 9 1- 17- 8- 6- 3- 13- 7- 9 1- 17- 8- 6- 3- 13- 7- 4- 9 1- 17- 8- 6- 12- 13- 7- 9 1- 10- 12- 13- 7- 9	9 9 9 9 9 9 9 9 11 11	10
1	11	1- 12- 6- 8- 4- 16- 11 1- 17- 8- 4- 16- 11 1- 17- 2- 4- 16- 11 1- 17- 2- 16- 11 1- 17- 2- 5- 15- 11 1- 17- 2- 16- 5- 15- 11 1- 13- 16- 11 1- 13- 16- 5- 15- 11 1- 17- 2- 5- 16- 11 1- 13- 7- 8- 17- 2- 16- 11 1- 13- 12- 6- 4- 2- 16- 11 1- 13- 3- 17- 2- 16- 11 1- 13- 12- 6- 4- 16- 11 1- 12- 6- 3- 17- 2- 5- 15- 11 1- 12- 13- 16- 11 1- 12- 13- 7- 4- 16- 2- 5- 15 1- 12- 13- 7- 4- 2- 5- 15- 11 1- 12- 13- 7- 4- 2- 16- 11 1- 12- 13- 7- 4- 16- 11	9 12 9 9 9 9 12 9 9 9 11 9 11 9 9 9 9 9 9	19
1	12	1- 12 1- 13- 3- 14- 8- 6- 12 1- 17- 9- 14- 8- 6- 12 1- 17- 9- 7- 8- 6- 12	9 9 10 10	7

		1- 17- 9- 7- 8- 10- 12 1- 17- 9- 4- 6- 12 1- 10- 7- 8- 6- 12	9 10 10	
1	14	1- 13- 7- 8- 17- 9- 19- 14 1- 12- 6- 8- 17- 9- 19- 14 1- 10- 7- 13- 16- 2- 5- 9- 19- 14 1- 12- 10- 8- 6- 14 1- 12- 6- 8- 14 1- 12- 6- 8- 7- 13- 3- 14 1- 17- 8- 4- 7- 13- 3- 14 1- 10- 8- 4- 7- 13- 3- 14 1- 10- 8- 7- 4- 6- 3- 14 1- 10- 8- 4- 6- 3- 14 1- 10- 8- 4- 6- 14	9 9 9 9 9 9 9 9 9 9 9	11
1	16	1- 17- 2- 5- 16 1- 17- 2- 5- 15- 11- 16 1- 17- 9- 19- 14- 6- 8- 4- 16 1- 17- 9- 19- 14- 6- 8- 7- 4- 16 1- 17- 3- 14- 8- 4- 16 1- 17- 3- 14- 8- 6- 4- 16 1- 17- 3- 14- 6- 8- 4- 16 1- 17- 3- 14- 8- 7- 4- 16 1- 17- 3- 14- 6- 4- 16 1- 13- 3- 14- 6- 4- 16 1- 13- 3- 14- 6- 8- 4- 16 1- 17- 3- 6- 4- 16 1- 17- 9- 4- 2- 5- 15- 11- 16	9 9 9 9 10 10 9 10 9 9 9 9 9	13
1	17	1- 17 1- 10- 7- 8- 17 1- 10- 8- 17 1- 10- 7- 4- 16- 2- 17 1- 13- 7- 8- 14- 6- 3- 17 1- 13- 7- 8- 6- 3- 17 1- 7- 8- 6- 14- 3- 17 1- 13- 7- 8- 14- 3- 17 1- 13- 12- 6- 3- 17 1- 10- 8- 6- 3- 17 1- 10- 8- 6- 14- 3- 17 1- 10- 8- 14- 6- 3- 17 1- 10- 8- 4- 6- 3- 17 1- 10- 12- 6- 3- 17 1- 10- 7- 8- 6- 3- 17 1- 10- 7- 8- 14- 3- 17 1- 10- 8- 14- 3- 17 1- 12- 6- 8- 17 1- 12- 6- 3- 17	12 10 9 9 9 9 9 10 9 9 9 9 9 9 9 10 9 9 9	19
1	19	1- 13- 7- 8- 17- 9- 19 1- 10- 7- 8- 17- 2- 5- 9- 19 1- 10- 8- 17- 2- 5- 9- 19 1- 10- 8- 17- 9- 19 1- 10- 12- 6- 4- 2- 5- 9- 19 1- 10- 12- 6- 4- 2- 16- 5- 9- 19	9 9 9 9 9 9	10

		1- 10- 12- 6- 4- 9- 19 1- 12- 6- 4- 9- 19 1- 10- 8- 6- 4- 9- 19 1- 10- 12- 6- 14- 19	9 9 9 9	
1	20	1- 12- 6- 14- 20 1- 12- 10- 8- 14- 20 1- 12- 10- 20 1- 13- 16- 4- 6- 3- 14- 20 1- 13- 16- 4- 7- 8- 10- 20 1- 10- 12- 13- 7- 8- 14- 20	9 9 9 9 9 10	6

4.3. Effets des paramètres de l'algorithme génétique

4.3.1. Effet du nombre de générations

Dans cette section, on va étudier l'effet de l'ensemble des paramètres de l'algorithme génétique sur les résultats retournés par cet algorithme. Le nombre de k plus courts chemins trouvés après l'augmentation de nombre de générations est résumé dans le tableau 4.6 ci-dessous :

Table 4.6 : Résultats obtenus pour différents N génération.

N génération	k plus courts chemins pour chaque nœud de destination							
	9	**11**	**12**	**14**	**16**	**17**	**19**	**20**
1000	3	0	4	0	2	2	0	2
10000	15	8	21	7	7	11	1	5
20000	19	11	29	9	23	18	4	14
30000	26	13	32	17	26	14	3	13
80000	39	23	38	32	34	30	12	26
90000	42	23	48	37	42	34	10	31
120000	52	28	59	34	59	25	9	41
140000	56	33	64	41	60	29	9	44
180000	60	38	69	46	68	26	15	45
200000	62	40	72	60	54	37	17	52

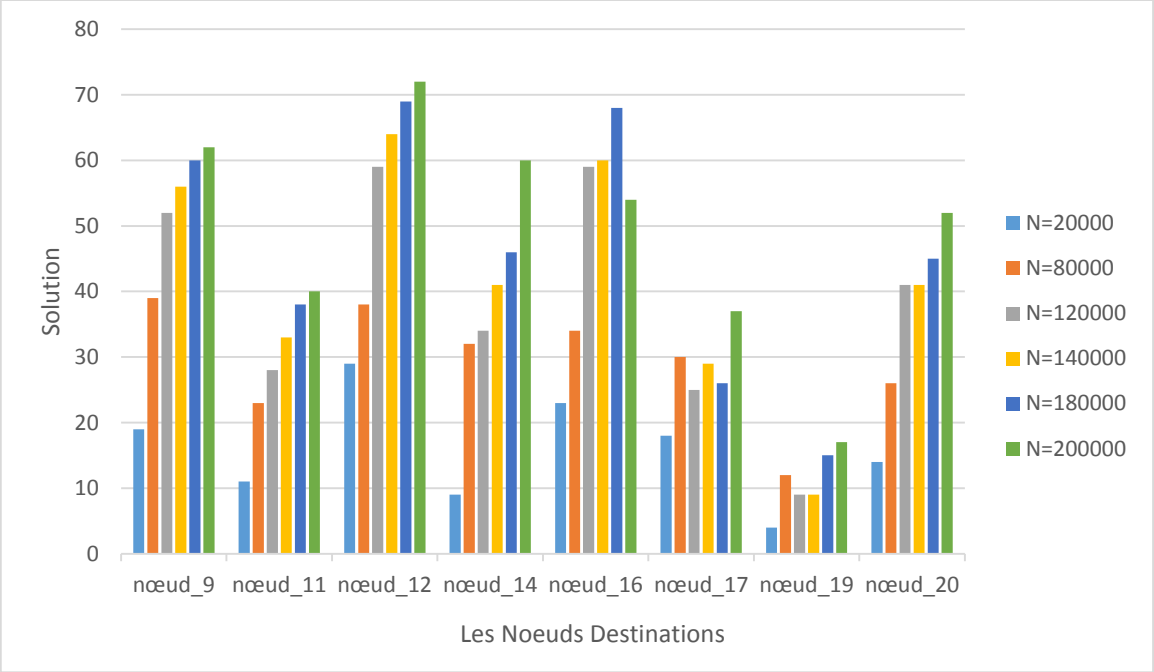

Figure 4.4 : Les résultats donnés dans le tableau 4.6.

A partir des résultats du tableau 4.6 et la figure 4.4 on peut remarquer que l'algorithme génétique est très sensible du nombre de génération choisi. En effet, quand le nombre de génération augmente, l'algorithme a donné des bons résultats.

4.3.2. Effet de la probabilité de mutation

Le nombre de k plus courts chemins trouvés après l'augmentation du taux de mutation est résumé dans le tableau 4.7 ci-dessous :

Table 4.7 : Résultats obtenus pour différents Pm (*pop_size* = 25, *nb_itération* = 7000, *Pc* = 0.9).

Pm	k plus courts chemins pour chaque nœud de destination							
	9	**11**	**12**	**14**	**16**	**17**	**19**	**20**
0.9	39	35	41	36	40	27	8	38
0.7	37	34	36	31	35	18	4	30
0.5	28	28	29	20	26	18	4	23
0.3	21	25	14	9	23	12	3	11
0.1	7	12	7	9	16	5	2	10

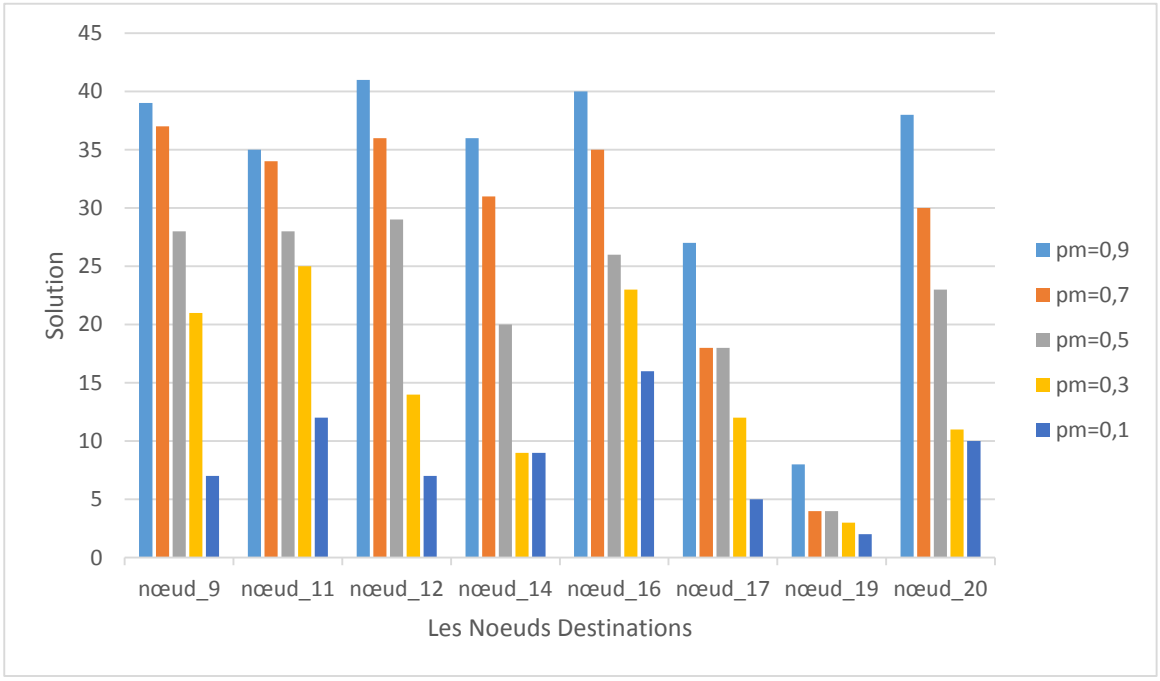

Figure 4.5 : Les résultats donnés dans le tableau 4.7.

A partir des résultats du tableau 4.7 et la figure 4.5 on peut remarquer que l'algorithme génétique est très sensible au taux de mutation choisi. En effet, quand le taux de mutation augmente, l'algorithme a donné de bons résultats.

4.3.3. Effet de la probabilité de croisement

Le nombre de k plus courts chemins trouvés en fonction de la variation du taux de croisement est présenté le tableau 4.8 ci-dessous :

Table 4.8 : Résultats obtenus pour différents Pc (*pop_size* = 25, *B* = 10, *nb_iteration* = 20000, *Pm*=0.1)

Pc	k plus courts chemins pour chaque nœud de destination							
	9	**11**	**12**	**14**	**16**	**17**	**19**	**20**
0.9	25	8	29	13	21	16	5	21
0.7	14	5	15	7	15	12	2	15
0.5	17	5	15	6	8	4	0	6
0.3	17	4	16	11	7	4	0	3

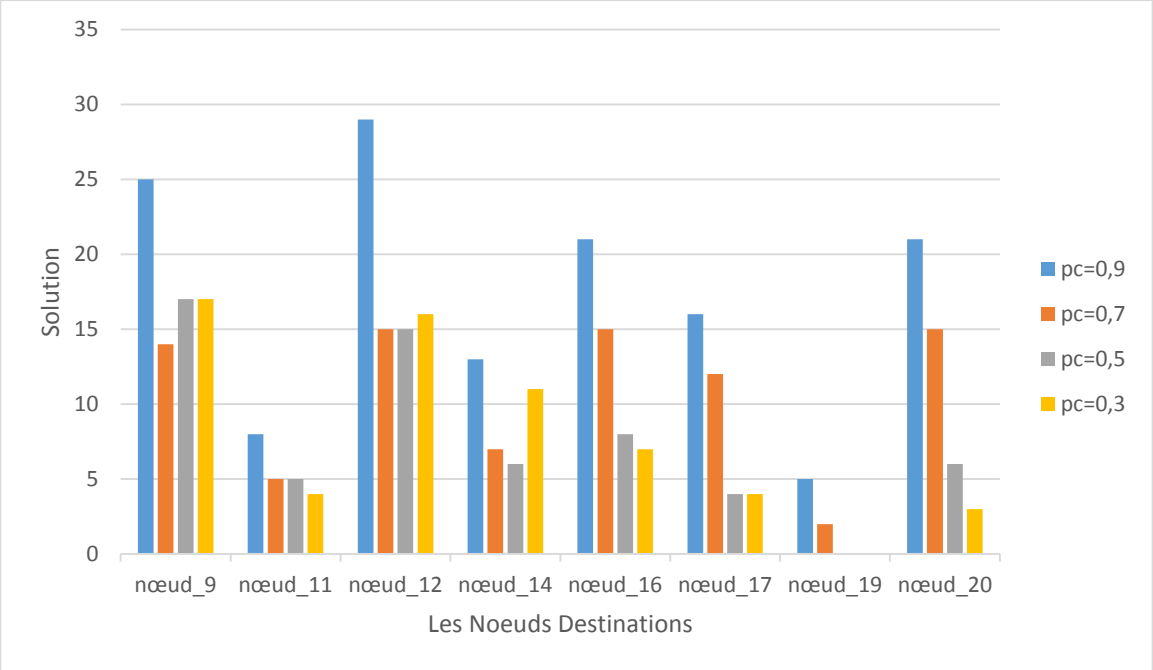

Figure 4.6 : Les résultats donnés dans le tableau 4.8.

A partir des résultats du tableau 4.8 et la figure 4.6 on peut remarquer que l'algorithme génétique est très sensible au taux de croisement choisi. En effet, quand le taux de croisement diminue, l'algorithme a donné des mauvais résultats.

4.4. Troisième exemple

Dans le troisième exemple, nous considérons un graphe avec 30 sommets. Chaque lien correspond à une bande passante.

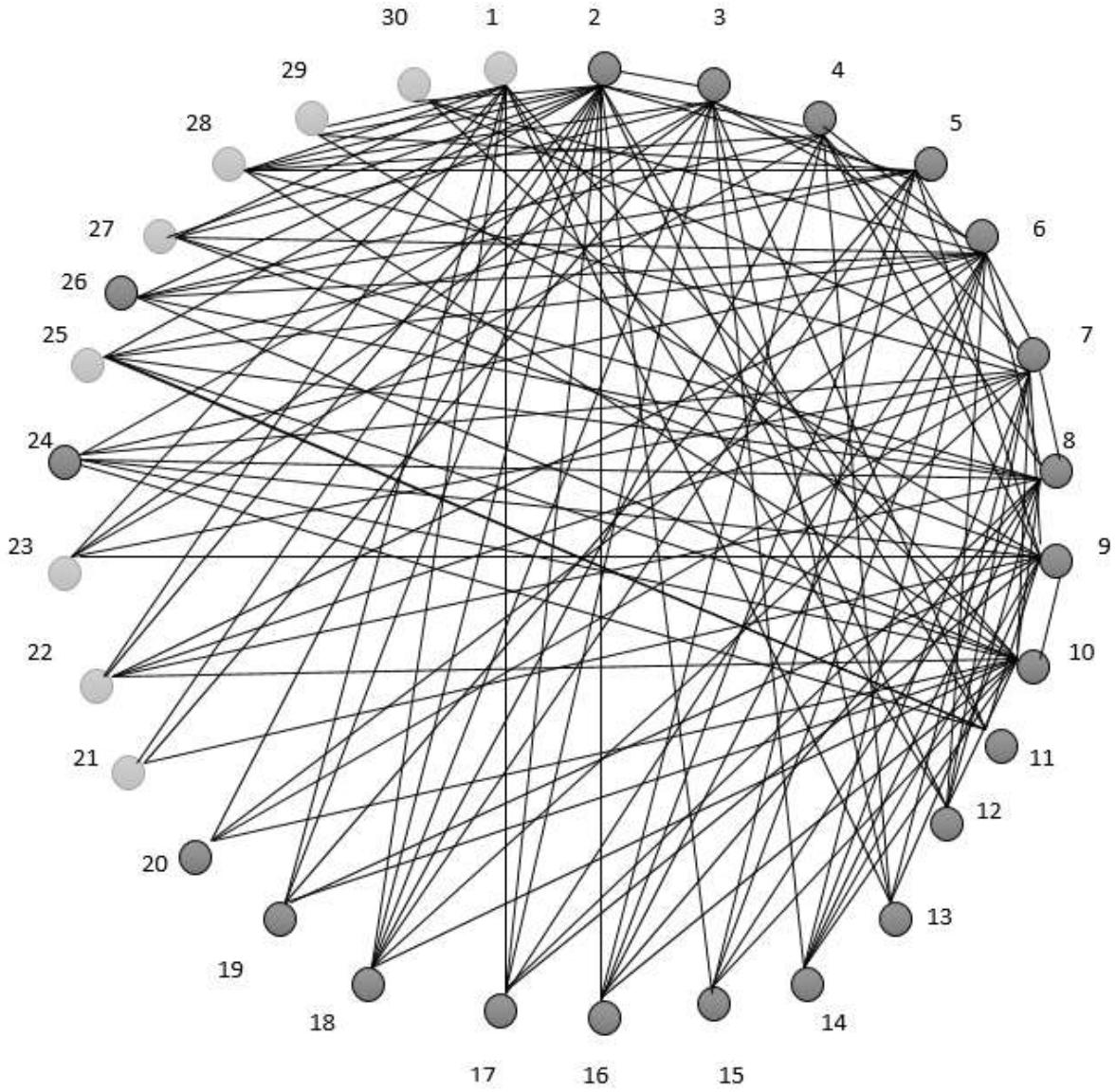

Figure 4.7 : Représentation d'un graphe avec 30 sommets.

4.4.1. Tests et résultats

Nous avons testé l'algorithme génétique développé pour chercher l'ensemble des k plus courts chemins du nœud source vers les nœuds de destinations. Dans notre exemple, nous utilisons les paramètres suivantes: *pop_size* = 25, $Pm \geq 0.1$, $Pc = 0.9$, *nbr_itération* = 200000, le nœud source $S = 1$, les nœuds de destinations sont $D = \{20, 21, 22, 24, 26, 28, 29, 30\}$. Le tableau 4.9 présente les résultats obtenus avec la valeur de $B = 11$. Le tableau 4.10 présente les résultats obtenus pour chaque nœud de destination à *N* générations. Le tableau 4.11 montré l'effet de la variation de la probabilité de mutation.

Table 4.9 : Résultats obtenus pour (*pop_size* = 25, $B = 11$, *nbr_itération* = 200000)

Nœud Source	**Nœud Destination**	**Les plus courts chemins**	**Valeur de B**	**K**
1	20	1- 23- 4- 9- 27- 8- 24- 19- 14- 20 1- 23- 3- 27- 8- 24- 19- 14- 20 1- 23- 3- 17- 8- 24- 19- 14- 20 1- 17- 3- 27- 8- 24- 19- 14- 20 1- 27- 3- 17- 9- 7- 10- 20 1- 27- 8- 4- 7- 9- 17- 3- 14- 20 1- 13- 16- 2- 9- 7- 10- 30- 14- 20	10 10 10 10 10 10 10	7
1	21	1- 23- 25- 11- 15- 21 1- 13- 7- 8- 24- 30- 23- 16- 21 1- 13- 7- 10- 12- 23- 16- 21 1- 13- 12- 10- 7- 4- 23- 16- 21 1- 13- 12- 10- 7- 4- 23- 3- 27- 21 1- 13- 12- 10- 7- 4- 16- 21 1- 17- 8- 4- 9- 7- 13- 12- 23- 16- 21 1- 13- 7- 8- 6- 12- 23- 16- 21 1- 27- 3- 14- 8- 7- 28- 21 1- 23- 3- 14- 19- 24- 8- 27- 21	10 10 11 10 10 11 10 10 10 10	10
1	22	1- 10- 30- 23 7- 13- 22 1- 17- 9- 7- 4- 6- 22 1- 10- 7- 13- 16- 2- 4- 6- 22 1- 10- 7- 23- 30- 14- 3- 27- 8- 6- 22 1- 13- 16- 2- 4- 8- 6- 22 1- 13- 16- 2- 4- 7- 8- 6- 22 1- 13- 16- 2- 4- 6- 22 1- 13- 26- 2- 4- 23- 12- 6- 22 1- 13- 16- 4- 23- 7- 10- 22 1- 13- 26- 4- 23- 7- 10- 22 1- 27- 8- 6- 12- 10- 22	11 11 11 11 13 10 13 10 10 10 11	11
1	24	1- 27- 8- 4- 7- 23- 30- 14- 19- 24 1- 27- 9- 4- 7- 23- 30- 14- 19- 24 1- 27- 8- 4- 23- 30- 14- 19- 24 1- 27- 8- 4- 7- 23- 3- 14- 19- 24 1- 27- 8- 7- 23- 30- 14- 19- 24	10 10 10 10 10	9

		1- 10- 7- 13- 12- 23- 3- 24 1- 10- 7- 13- 12- 23- 30- 24 1- 23- 16- 11- 25- 5- 9- 7- 10- 20- 24 1- 13- 16- 23- 4- 8- 14- 20- 24	10 11 11 10	
1	26	1- 13- 16- 11- 26 1- 10- 7- 4- 23- 16- 2- 26 1- 10- 7- 23- 16- 2- 26 1- 10- 7- 4- 16- 2- 26 1- 10- 7- 4- 2- 26 1- 13- 7- 4- 23- 16- 2- 26 1- 23- 7- 13- 16- 2- 26 1- 13- 22- 6- 4- 2- 26 1- 13- 22- 6- 8- 4- 2- 26 1- 13- 22- 6- 4- 2- 16- 11- 26 1- 17- 8- 6- 4- 26 1- 23- 12- 13- 16- 2- 26 1- 17- 9- 27- 21- 16- 2- 26 1- 17- 9- 27- 21- 15- 11- 26	12 10 11 11 11 10 13 12 12 12 12 11 10 10	14
1	28	1- 27- 9- 17- 8- 7- 28 1- 13- 7- 9- 5- 15- 21- 28 1- 23- 7- 9- 5- 15- 21- 28 1- 10- 12- 6- 4- 26- 2- 16- 21- 28 1- 13- 26- 4- 23- 7- 28 1- 13- 26- 4- 9- 7- 28 1- 23- 30- 10- 7- 28 1- 23- 16- 4- 7- 28	10 11 11 11 10 10 11 11	8
1	29	1- 10- 12- 23- 30 -14- 29	11	1
1	30	1- 13- 16- 4- 7- 10- 30 1- 23- 16- 4- 7- 10- 30 1- 13- 16- 4- 8- 6- 22- 10- 30 1- 13- 7- 10- 20- 14- 30 1- 13- 7- 8- 14- 30 1- 10- 7- 4- 6- 8- 14- 30 1- 10- 7- 8- 14- 30 1- 23- 16- 21- 27- 8- 14- 30 1- 23- 4- 16- 21- 27- 8- 14- 30 1- 23- 16- 4- 8- 14- 30 1- 13- 12- 10- 22- 6- 8- 14- 30 1- 13- 12- 10- 22- 6- 4- 8- 14 – 30 1- 13- 26- 2- 16- 23- 30 1- 17- 8- 7- 10- 30 1- 27- 8- 7- 10- 30 1- 17- 8- 4- 7- 9- 27- 3- 24- 30	11 11 12 10 10 11 10 10 10 11 11 11 11 10 10 11	16

4.5. Effets des paramètres de l'algorithme génétique

4.5.1. Effet du nombre de générations

Dans cette section on va étudier l'effet de l'ensemble des paramètres de l'algorithme génétique sur les résultats retournés par cet algorithme. Le nombre de k plus courts chemins trouvés après l'augmentation de nombre de générations est résumé dans le tableau 4.10 ci-dessous :

Table 4.10 : Résultats obtenus pour différents N génération.

N génération	k plus courts chemins pour chaque nœud de destination							
	20	**21**	**22**	**24**	**26**	**28**	**29**	**30**
200000	7	10	11	9	14	8	1	16
400000	18	14	18	20	12	10	6	7
600000	32	25	12	40	35	15	2	26
800000	47	33	22	31	42	26	3	45

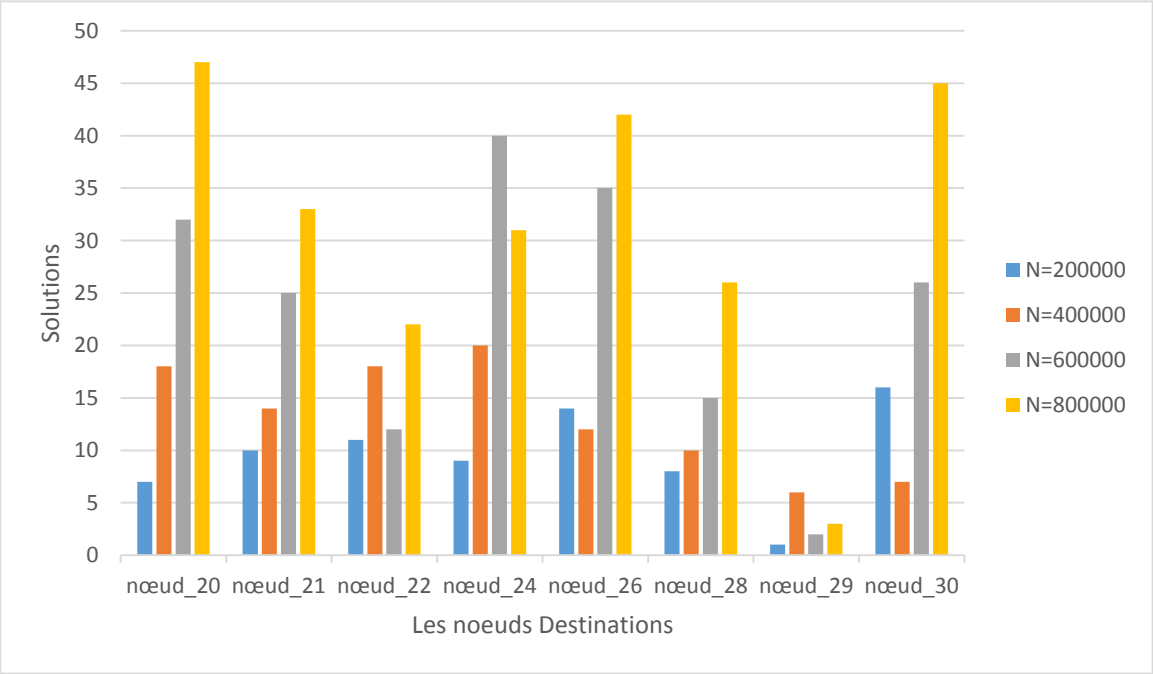

Figure 4.8 : Les résultats donnés dans le tableau 4.10.

A partir des résultats du tableau 4.10 et la figure 4.8 on peut remarquer que l'algorithme génétique est très sensible du nombre de génération choisi. En effet, quand le nombre de génération augmente, l'algorithme a donné des bons résultats.

4.5.2. Effet de la probabilité de mutation

Le nombre de k plus courts chemins trouvés après l'augmentation du taux de mutation est résumé dans le tableau 4.11 ci-dessous :

Table 4.11 : Résultats obtenus pour différents Pm (*pop_size* = 25, *nb_itération* = 9000, *Pc* = 0.9).

Pm	k plus courts chemins pour chaque nœud de destination							
	20	**21**	**22**	**24**	**26**	**28**	**29**	**30**
0.9	23	14	12	18	30	10	3	34
0.7	28	12	10	10	29	8	3	33
0.5	27	11	4	19	28	5	4	17
0.3	15	5	6	5	8	1	3	8
0.1	8	3	5	8	1	1	0	8

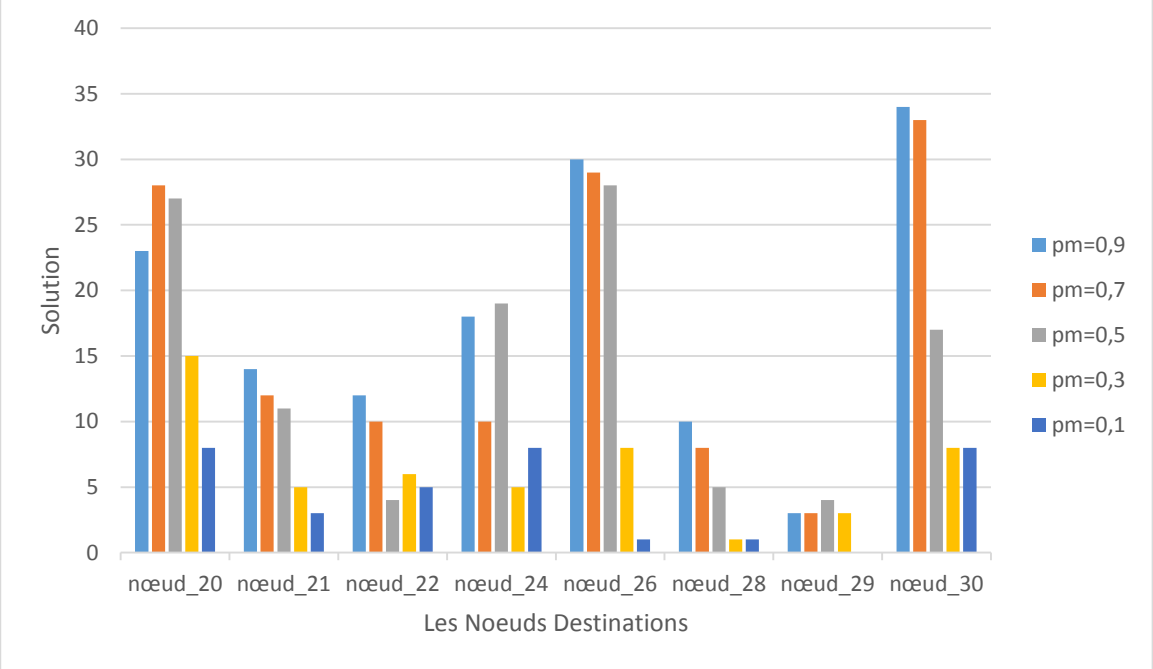

Figure 4.9 : Les résultats donnés dans le tableau 4.11.

A partir des résultats du tableau 4.11 et la figure 4.9 on peut remarquer que l'algorithme génétique est très sensible au taux de mutation choisi. En effet, quand le taux de mutation augmente, l'algorithme a donné des bons résultats.

4.5.3. Effet de la probabilité de croisement

Le nombre de k plus courts chemins trouvés après l'augmentation de probabilité de croisement est résumé dans le tableau 4.12 ci-dessous :

Table 4.12 : Résultats obtenus pour différents Pc (*pop_size* = 25, *B* = 10, *nb_iteration*=500000, *pm* = 0.1)

Pc	k plus courts chemins pour chaque nœud de destination							
	20	**21**	**22**	**24**	**26**	**28**	**29**	**30**
0.9	41	29	32	36	11	11	3	34
0.7	43	11	16	19	8	14	1	26
0.5	19	11	12	28	25	9	0	19
0.3	15	18	7	18	4	2	2	25

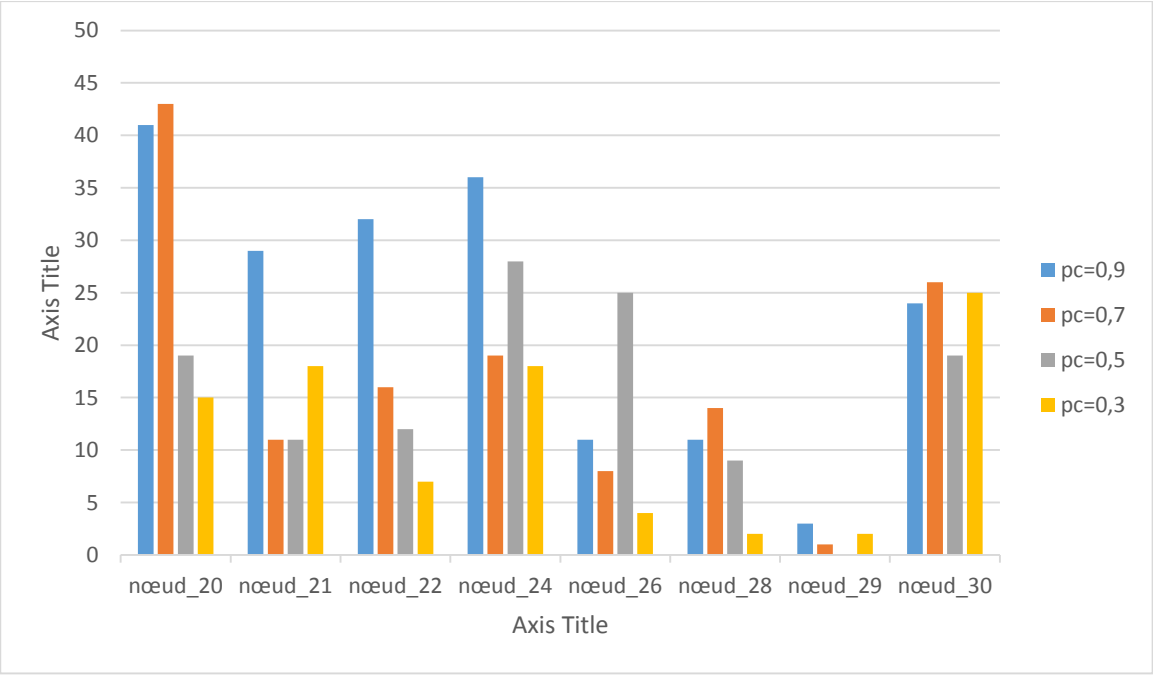

Figure 4.10 : Les résultats donnés dans le tableau 4.12.

A partir des résultats du tableau 4.12 et la figure 4.10 on peut remarquer que l'algorithme génétique est très sensible au taux de croisement choisi. En effet, quand le taux de croisement diminue, l'algorithme a donné des mauvais résultats.

5. Conclusion

Dans ce chapitre, en premier temps nous avons présenté les environnements matériel et logiciel et les données utilisés, ensuite de test les résultats obtenues sur des exemples de graphes, et en fin nous avons présenté et étudier l'effet de l'ensemble des paramètres de l'algorithme génétique sur les résultats retournés par cet algorithme.

CONCLUSION GENERALE

CONCLUSION GENERALE

Le besoin d'utiliser l'optimisation combinatoire est très grand. Surtout au présent d'un nombre d'application qui peut être formulé sous la forme d'un problème d'optimisation combinatoire dans est différents domaines de la vie réelle. Le problème de *k* plus courts chemins (*k*-PCC) l'un des problèmes les plus complexes de type NP-complet qui a ses applications réelles partout.

Dans ce travail, nous avons étudié un problème du *k*-PCC. Au cours de ce mémoire, nous avons commencé par la définition générale sur la théorie des graphes. En deuxième temps, nous avons passé en revue les métaheuristiques d'optimisation et la définition de la notion de complexité des algorithmes et des problèmes. Ensuite nous avons défini le problème de *k*-PCC. Nous avons appliqué l'algorithme génétique pour résoudre notre problème. Enfin, nous avons étudié l'effet de l'ensemble des paramètres de l'algorithme génétique sur les résultats retournés par cet algorithme. Cette méthode est très efficace est basée sur les mécanismes de la sélection naturelle. L'objectif est de trouver des k plus courts chemins du nœud source vers les nœuds de destinations avec contrainte de bande passante.

Nous avons choisi le langage C++ qui est très forte et rapides. Nous l'avons mis en œuvre dans différents graphes. Nous avons utilisé la matrice d'adjacence et la matrice de cout pour obtenir la bande passante.

Comme perspective à ce travail, nous voudrions essayer de travailler avec la méthode de croisement et mutation en parallèle pour améliorer le temps d'exécution, l'hybridation de l'algorithme génétique développé avec d'autres méthodes de recherche local comme recuit simulé ou la recherche Tabou ou la méthode de descente.

BIBLIOGRAPHIE

[1] Didier Müller, Introduction à la théorie des graphes, CAHIER NO 6, COMMISSION ROMANDE DE MATHÉMATIQUE, France, Décembre 2011.

[2] A. Bretto, A. Faisant, F. Hennecart. Eléments de théorie des graphes, Editeur SPRINGER, collection Iris, Année 2012.

[3] site web : http://perso.usthb.dz/~mboukala/CoursTG.pdf, consulté le 25/02/2017,15:45.

[4] Dr. Chahinez BACHTARZI, Théorie des graphes, Support de cours du module TG, Université CONSTANTINE 2, Année 2014/2015.

[5] Site web: http://www.w3ii.com/fr/graph_theory/graph_theory_introduction.html, consulte le 18/02/2017, 17.02.

[6] M. Gondran, M. Minoux, Graphes et algorithmes. Collection EDF R&D.4eme édition revu et augmentée, ISBN : 978-2-7430-1035-5. 2009.

[7] Site web : http://www.mathraining.be/chapters/42?type=10 , consulte le 18/02/2017, 15.00.

[8] C.H. PAPADIMITRIOU, K. STEIGLITZ, Combinatorial optimization – algorithms and complexity, Prentice Hall, 1982.

[9] Jin-Kao Hao, Philippe Galinier, Michel Habib, Méthaheuristique pour l'optimisation combinatoire et l'affectation sous contraintes, Revue d'Intelligence Artificielle Vol : No. 1999.

[10] carfier J. et Chertienne P.1988. Problèmes d'ordonnancement. Modélisation/ complexité/ algorithme.1988.

[11] Amira Gherboudj. Méthodes de résolution de problèmes difficiles académiques. Thèse Pour l'obtention du diplôme de Doctorat.2013

[12] Abbas EL DOR. Perfectionnement des algorithmes d'Optimisation par Essaim Particulaire. Applications en segmentation d'images et en électronique. THÈSE DE DOCTORAT EN INFORMATIQUE.UNIVERSITÉ PARIS-EST. 2012

[13] F. Glover. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, Vol. 13, pp. 533–549, 1986.

[14] I. H. Osman ET G. Laporte: Metaheuristics: A bibliography. Annals of Operations Research. 63:513.623, 1996.

[15] A. BAPTISTE, Les métaheuristiques en optimisation combinatoire, conservatoire national des arts et métiers paris. 2006.

[16] C. H. Papadimitriou: The complexity of combinatorial optimization problems. PhD, 1976.

[17] Alaoui Abdiya. Application des techniques des métaheuristiques pour l'optimisation de la tâche de la classification de la fouille de données. Magister en informatique. UNIVERSITE D'ORAN Mohamed Boudiaf. 2012.

[18] S.kirkpatrick, C.D.Gelatt, and M.P.Vecchi. Optimization by simulated annealing. Science, New Series, 220(4598):671-680, 1983.

[19] Mehdi Rouan Serik. Implémentation de méthodes de recherches locales sur les architectures multi et many cœurs. Diplôme de magister. Université d'Oran.2012.

[20] R. Battiti & G. Tecchiolli. The reactive tabu search. ORSA Journal on Computing, 6(2): 126–140, 1994.

[21] D. E. Goldberg, Genetic Algorithm in search, optimization and machine learning, Addison-Welsy, 1989.

[22] Man K. F., Tang K. S., Kwong S., Genetic algorithms: Concepts and designs, Springer, 2000.

[23] T. Vallée, M. Yildizoglu, Présentation des algorithmes génétiques et de leurs applications en économie.

[24] Mokeddemm Diab, Contrôle Flou des Processus Biotechnologiques à Base d'Algorithmes Génétiques, Présentée pour l'Obtention du Diplôme de Doctorat En Sciences en Electronique. Université FERHAT ABBAS DE SETIF, 2010.

[25] Fatiha SOUAM-AIT EL HADJ, Approche de détection de communautés chevauchantes dans des réseaux bipartis, Thèse de doctorat, Université MOULOUD MAMMERI DE TIZI OUZOU. 2013.

[26] LOTFI Amiar, Un système hybride AG/PMC pour la reconnaissance de la parole arabe, Diplôme de MAGISTER, Université BADJI MOKHTAR ANNABA, 2005.

[27] D. E. Goldberg, Genetic Algorithm in search, optimization and machine learning, Addison Welsy, 1989.

[28] Eiben A.E., Hinterding R., & Michalewicz Z, Parameter Control in Evolutionary Algorithms, Evolutionary Computation, IEEE Transactions on, 3(2), pp. 124-141, 1999.

[29] BOURAZZA, S, Variante d'algorithmes génétiques appliqués aux problèmes d'ordonnancement, Doctorat de l'université du Havre, Spécialité Mathématiques Appliquées et Informatique, pp. 5-8 ,2006.

[30] Ryan KAMAARTI, approches évolutionnistes pour la résolution du 1-PDPTW statique et dynamique, thèse doctorat, doctorat délivre conjointement par l'école centrale de LILLE et l'université des sciences et technologies de LILLE, 2006.

[31] Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. In Proceedings of IEEE International Conference on Neural Network, vol. IV, pp. 1942–1948.

[32] Clerc M. and Kennedy J., The Particle Swarm: Explosion, Stability and Convergence in a Multi- Dimensional complex Space, IEEE Transactions on Evolutionary Computation, vol. 6, 2001.

[33] Dréo J. & Siarry P., Métaheuristiques pour l'optimisation difficile, Edition Eyrolles, 2003.

[34] site web : http://sis.univ-tln.fr/~tollari/TER/AlgoGen1/node5.html, consulte le 20/05/2017, 14 :53.

[35] GUISONG LIU, ZHAO QIU, WENYU CHEN. An iterative Algorithm for single-pair K shortest paths computation. WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS Guisong Liu, Zhao Qiu, Wenyu Chen.2006.

[36] Ahmed Younes Hamed, A genetic algorithm for finding the *k* shortest paths in a network, Egyptian Informatics Journal, Volume 11, Issue 2, December 2010, Pages 75–79.

[37] Chang Wook Ahn and R. S. Ramakrishna, A Genetic Algorithm for Shortest Path Routing Problem and the Sizing of Populations, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 6, NO. 6, DECEMBER 2002.

[38] T. Vallée, M. Yildizoglu, Présentation des algorithmes génétiques et de leurs applications en économie, Décembre 2003, v. 4.2.

[39] C. K. Bounsaythip, Algorithmes heuristiques et évolutionnaires application à la résolution du problème de placement de formes irrégulières, Thèse de doctorat en Productique, université des sciences et technologies de Lille, 9 octobre 1998.

[40] Site web: http://www.01net.com/telecharger/windows/Programmation/creation/fiches/121467.html, consulte le 14/05/2017, 9.45.

[41] Site web : https://www.microsoft.com/fr-fr/download/details.aspx?id=30679, consulte le 14/05/2017, 9.47.

ملخص

الهدف من هذه المذكرة هو دراسة مشكلة ك اقصر المسارات التي لديها العديد من التطبيقات الهامة و المتنوعة في مختلف المجالات يعتبر هدا المشكل معروف جدا ولا يوجد لحد الان خوارزمية لحله بوقت معقول.

لقد طبقنا الخوارزمية الجينية للعثور على مجموعة ك اقصر المسارات من عقدة المصدر الى وجهات العقد مع قيود عرض النطاق الترددي. وقد تم اختبار فعالية هذه الخوارزمية على عدد قليل من الامثلة.

كلمات المفتاحية : اقصر مسارات, NP-كامل, خوارزمية الجينية, النطاق الترددي.

Abstract

The work of this thesis concerns the study of the k-shortest path problem (*k*-PCC) which has many important and diverse applications in various fields. It is a well-known NP-complete problem and hence no polynomial-time algorithm has been found yet for solving it to optimality.

We applied a genetic algorithm to look for the set of k shortest paths from the source node to the destination nodes with bandwidth constraints. The effectiveness of this algorithm has been tested on a few examples.

Keywords: shortest paths, NP-complete, genetic algorithms, bandwidth

---

Résumé

Le travail de ce mémoire concerne l'étude du problème du *k* plus courts chemins (*k*-PCC) qui a de nombreuses applications importantes et diverses dans différents domaines. Il est un problème de type NP-complet bien connu et donc aucun algorithme polynomial n'a encore été trouvé pour le résoudre à l'optimalité.

Nous avons appliqué un algorithme génétique pour chercher l'ensemble des *k* plus courts chemins du nœud source vers les nœuds de destinations avec des contraintes de bande passante. L'efficacité de cet algorithme a été testée sur quelques exemples.

Mots clés : plus courts chemins, NP-complet, algorithme génétique, bande passante.