REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU
FACULTE DU GENIE ELECTRIQUE ET D'INFORMATIQUE
DEPARTEMENT INFORMATIQUE

MEMOIRE

De fin d'études pour l'obtention d'un
Master en Informatique
Spécialité: Systèmes informatiques

Thème

Détection de communautés dans les réseaux complexes en utilisant les algorithmes génétiques

Réalisé et présenté par :

M^elle^. AIT YAKOUB Zina
M^elle^. BELHADI Rabia

Dirigé et proposé par :

M^me^. AÏT EL HADJ Fatiha

Devant le jury d'examen composé de :

-
-
-
-

Promotion 2010/2011

SOMMAIRE

Résumé

Nous nous situons dans le contexte des algorithmes évolutionnaire, et nous nous intéressons particulièrement aux algorithmes génétiques. Le travail qui nous a été proposé d est dans un premier temps d'implémenter une méthode de détection de communautés disjointes sur un graphe monoparti en utilisant les algorithmes génétiques. Dans un deuxième temps de mettre en œuvre une approche de détection de communautés chevauchantes sur un graphe biparti en utilisant l'algorithme de détection de communautés disjointes.

Introduction générale

De récentes avancées dans le domaine des systèmes complexes ont fait ressortir le rôle central que jouent les graphes dans de nombreux phénomènes. Ces grands graphes, permettent de modéliser les interactions entre les différents acteurs de ces phénomènes complexes, qui interviennent dans de très nombreux domaines : sociologie, biologie, linguistique, physique, informatique, épidémiologie, etc. La complexité de ces graphes rend leurs utilisations plus difficile à gérer, pour cela les chercheurs ont pensé à découper les réseaux en un ensemble de parties, c'est ce qu'on appelle « détection de communauté » (appeler aussi partitionnement de graphe en théorie des graphes).

La détection de communauté a pour but de découper un graphe en différentes parties qui satisfont certaines contraintes et qui optimisent une certaine fonction objective. Elle possède de nombreuses applications comme la conception de circuits intégrés électroniques, la répartition de charge pour les machines parallèles ou la segmentation d'images. Cependant, la détection de communauté est un problème complexe (NP-complet), dont la solution ne peut pas être trouvée au moyen d'une méthode de résolution exacte, à cet effet des méthodes métaheuristiques sont proposées et les plus populaires sont les algorithmes évolutionnaires.

Les algorithmes évolutionnaires ont étés proposés à la fin des années 60, pour résoudre des problèmes d'optimisation difficiles. Ils sont basés sur des notions de biologie et permettent d'obtenir des solutions optimisées (mais pas forcément optimales) en ce basant sur les principes du darwinisme (variation, sélection, production). Parmi ces algorithmes les algorithmes génétiques, les stratégies d'évolution ou la programmation génétique.

Nous nous situons dans le contexte des algorithmes évolutionnaire, et nous nous intéressons particulièrement aux algorithmes génétiques. Le travail qui nous a été proposé dans le cadre de ce mémoire est dans un premier temps d'implémenter une méthode de détection de communautés disjointes sur un graphe monoparti en utilisant les algorithmes génétiques. Dans un deuxième temps de mettre en œuvre une approche de détection de communautés chevauchantes sur un graphe biparti en utilisant l'algorithme de détection de communautés disjointes.

Pour cela, nous avons subdivisé ce rapport en plusieurs volets comme suit : Outre ce volet introductif, le chapitre 1 présente les graphes complexes et leurs applications dans les différents domaines concernés. Nous introduirons ensuite la problématique de détection de communautés et les différentes méthodes utilisées. Le chapitre 2 donne une vue générale sur les algorithmes évolutionnaires et en particulier les algorithmes génétiques. Le chapitre 3 présente la méthode génétique que nous adoptons dans la partie implémentation. Le chapitre 4 décrit les différents algorithmes et structures de données utilisés pour implémenter la méthode génétique. Le chapitre 5 est dédié à la mise en œuvre et l'expérimentation du logiciel, nous terminons notre rapport sur une conclusion.

Chapitre1

Détection de communautés

---

1. Introduction

Le partitionnement de graphe est une technique très connue en informatique, elle est utilisé pour résoudre un grand nombre de problèmes d'ingénierie: la conception de circuits intégrés électroniques, la répartition de charge pour les machines parallèles, la dynamique des fluides, le calcul matriciel, la segmentation d'images ou la classification d'objets.

Dans ce chapitre nous allons d'abord présenter les grands graphes complexes et leurs applications dans les différents domaines concernés. Nous introduirons ensuite la problématique de détection de communautés. Les différentes méthodes utilisées seront aussi proposées.

2. Les graphes complexes

2.1. Définition

Un graphe G = (V,E) permet de modéliser des interactions entre des objets (représentés par un ensemble V de n sommets) reliés par des liens (représentés par un ensemble $E \subset V \times V$ de m paires de sommets nommées arêtes). Ainsi des graphes complexes permettent de modéliser de nombreux phénomènes réels (le terme complexe fait référence au nombre important de sommets de ces graphes). Ces graphes peuvent être, selon le contexte, orientés ou non-orientés, pondérés ou non-pondérés. Nous définirons formellement ces objets mathématiques justes après.

2.2. Quelles que notions sur les graphes

❖ **Définition 1 (Graphe) :** Soient V un ensemble de $n_v \in N^*$ éléments et A un ensemble de $n_E \in N$ couples d'éléments de V. On appelle graphe G le couple (V, E). Les éléments de V sont appelés sommets du graphe et ceux de E, les arcs ou arêtes du graphe, suivant qu'ils sont orientés ou non.

❖ **Définition 2 (Graphe valué ou pondéré) :** Soit un graphe G = (V,E). On dit que le graphe est valué (ou pondéré) si à chaque élément a de V est associé une valeur entière appelée poids(a).

Par extension, on considère qu'un couple de sommets (v, v′) ∈ V^2 tel que (v, v′) ∉ E possède un poids nul : poids(v, v′) = 0. Le poids d'un sous-ensemble X d'éléments de E, est la somme des poids des éléments de X : **poids(X) =** $\sum_{a \in X}$ **poids(a).**

❖ **Définition 3 (degré d'un sommet) :** Dans le cas des graphes valués, on redéfinit le degré d'un sommet v ∈ V comme étant la somme des poids des arêtes adjacentes à ce sommet. Ainsi: **deg(v) =** $\sum_{(v,v') \in E}$ $\boldsymbol{poids}(\boldsymbol{v}, \boldsymbol{v}')$**.**

Pour un graphe quelconque, on peut définir une unique fonction poids : E → N qui à toute arête de ce graphe associe son poids. Cette fonction poids peut être représentée sous la forme d'une matrice. On définit la matrice d'adjacence d'un graphe G comme étant la matrice associant à chaque arête de G son poids.

❖ **Définition 4 (Adjacence) :** Soient un graphe G = (V,E) et une arête a = (v, v′) ∈ E. On dit que les sommets v et v′ sont des sommets adjacents. De même, on dit que a est l'arête adjacente aux sommets v et v′.

❖ **Définition 5 (Matrices d'adjacence) :** Soit un graphe simple G = (V,E).
La matrice M_{Adj} de dimensions $n_v \times n_v$, telle que ∀ (i, j) ∈ $\{1, \ldots, n_v\}^2$:

$$(M_{Adj})_{ij} = \begin{cases} 0 & \text{si } i = j \\ p(i,j) & \text{sinon} \end{cases}$$

❖ **Définition 6 (densité)** : La densité d'un graphe G=(V, E) est le rapport $|E| / |V|^2$.

-Un graphe peu dense contient peu d'arêtes (arcs)

-Un graphe dense contient beaucoup d'arêtes (arcs)

❖ **Définition7 (Graphe biparti) :** un graphe biparti est un graphe qui contient deux familles (types) de sommets disjointes, telles qu'il n'existe aucun lien reliant deux sommets de la même famille.

Autrement dit : G=(S,A) est biparti si et seulement s'il existe une partition S1∪S2 de S tel que ∀(u,v)∈A, u∈S1⇔v∈S2.

❖ **Définition8 (arête incidente à un sommet) :** une arête est dite incidente à un sommet v_i si v_i est une de ses extrémités.

❖ **Définition9 : (Graphe d'arête) :** Soit G un graphe simple, on appelle graphe-arête (ou graphe représentatif des arêtes) de G, le graphe G' obtenu de la façon suivante :

- les sommets de G' représentent les arêtes de G,
- deux sommets de G' sont reliés si et seulement si les arêtes correspondantes de G sont adjacentes.

2.3. De nombreux domaines concernés

Nous allons maintenant présenter quelques exemples dans lesquels des graphes sont utilisés comme outil de modélisation de phénomènes complexes. Ces exemples vont illustrer la diversité des domaines d'applications possibles [1] [2]. Pour chaque cas, nous identifierons les acteurs du phénomène, modélisés par les sommets du graphe, et les interactions entre eux, modélisées par des liens ou arêtes entre les sommets.

➢ **Les réseaux sociaux :** Ils constituent un champ d'application ancien et important dans lequel les acteurs sont des individus ou entités sociales (associations, entreprises, pays, etc.). Les liens entre eux peuvent être de différentes natures. Nous pouvons ainsi observer plusieurs types de réseaux : les réseaux de connaissance (deux individus sont reliés s'ils se connaissent), les réseaux de contact physique (deux individus sont reliés s'ils ont été physiquement en contact), les réseaux de collaboration (deux individus sont reliés s'ils ont travaillé ensemble, les réseaux d'appels téléphoniques (deux individus ou numéros de téléphones sont reliés s'il y a eu un appel entre eux), les réseaux d'échanges (deux entités sont reliées si elles ont échangé un fichier ou un courrier électronique par exemple), etc.

- **Les réseaux biologiques:** Il en existe plusieurs types parmi lesquels nous pouvons citer les réseaux métaboliques (les sommets sont des gènes ou des protéines qui sont liés par leurs interactions chimiques), les réseaux de neurones (chaque neurone est connecté à plusieurs autres neurones) ou les réseaux trophiques (les espèces d'un écosystème sont reliées pour représenter les chaînes alimentaires).

- **Les réseaux d'infrastructure :** Ils représentent des connections matérielles entre objets distribués dans un espace géographique. Nous pouvons citer les réseaux de transport (routes entre villes ou liaisons aériennes entre aéroports), les réseaux de distribution électrique (câbles entre les lieux de production et de consommation) ou encore le réseau physique de l'Internet (câbles entre ordinateurs).

- **Les réseaux d'information :** Ils représentent des liens abstraits de référencement entre des supports d'information. Parmi eux, les réseaux de citation d'articles ou les graphes du Web (les sommets sont des pages Web liées par des liens hypertextes).

- **Les réseaux linguistiques :** Ils relient les mots d'un langage donné et regroupent entre autres les réseaux des synonymies (deux mots sont reliés s'ils sont synonymes), les réseaux de cooccurrences (deux mots sont reliés s'ils apparaissent dans une même phrase d'un ouvrage) ou encore les réseaux de dictionnaires (deux mots sont liés si l'un est utilisé dans la définition de l'autre).

2.4. Problèmes liés aux graphes complexes

Avec la croissance des graphes et l'augmentation de ses entités, ceux ci les rendent de plus en plus complexes et difficile à gérer. De ce fait il été nécessaire de trouver un moyen qui permet de mieux comprendre la structure de ces graphes et qui peut être utilisé comme brique de base dans l'élaboration d'autres traitements plus complexes (parallélisassions, visualisation, compression etc.). Pour cela les chercheurs ont pensé à découper les réseaux en sous ensembles de communautés.

3. Détection de communautés

3.1. Définition

La détection de communautés est une problématique assez ancienne, qui tire ses origines du problème de partitionnement de graphe, qui est un problème classique en théorie des graphes, il consiste à diviser G en **k** partitionnement disjointes. Au point de vue mathématique, on peut partitionner les sommets ou bien les arêtes. Par contre, dans la plupart des applications, on ne s'intéresse qu'au partitionnement des sommets du graphe.

Soient un graphe **G = (V, E)** et un ensemble de k sous-ensembles de V, noté $P_k = \{V_1, V_2, \ldots, V_k\}$. On dit que P_k est une partition de G si :

1. Aucun élément de P n'est vide
   $\forall\ i \in \{1,\ldots,k\} \rightarrow V_i \neq \emptyset$
2. L'union des éléments de P est égale à V
   $\bigcup_{i=1}^{k} Vi = V$
3. Les éléments de P sont deux à deux disjoints.
   $\forall\ i, \in \{1,\ldots,k\}\ , \neq j \rightarrow Vi \cap Vj = \emptyset$

Notons que nous ne connaissons a priori ni le nombre ni la taille des communautés qui composent la partition recherchée.

Comme le montre cette définition, on n'a pas a priori de définition de la notion de communauté : tout repose sur la fonction de qualité Q (dit aussi fonction objective ou fitness en anglais) qui permettent de mesurer la qualité d'une partition en communautés qui joue donc un rôle clé. La fonction de qualité donne un score à toute partition qui mesure à quel point les communautés de cette partition vérifient des critères caractéristiques des communautés. En particulier, ces fonctions de qualité Q(P) tiennent généralement compte de rendre la densité plus forte à l'intérieur de chaque communauté (intra communauté) et plus faible à l'extérieur (inter communauté), la fonction la plus communément utilisée étant la modularité, elle sera présentée en détaille dans le chapitre 3 Les autres fonctions ne seront pas présentées dans ce travail. Toutefois, le lecteur intéressé est invité à consulter la référence suivante [2]

Notons enfin que cette définition impose que les communautés soient disjointes, or en, pratique rien n'empêche l'existence de communautés qui se chevauchent. La détection de telles communautés est un problème bien plus difficile que la détection de communautés disjointes. Ce problème a pour le moment été très peu traité et sera abordé au chapitre3 de cette étude.

Les méthodes de détection de communautés présentées dans la section 3 réservée à « l'état de l'art» et la méthode qu'on va développer au chapitre3 étant consacré au problème de détection de communautés disjointes.

3.2. Applications

Les méthodes de détection de communautés (partitionnement de graphes) sont encore aujourd'hui très étudiées et visent des domaines d'applications très larges tels que:

- **La conception de circuits intégrés électroniques** Le problème de conception de circuits intégrés électroniques est de diviser les composants aux sous-circuits tels que le nombre de connexion entre ceux-ci soit minimal. C'est une tâche de plus en plus complexe car les composants sont de plus en plus petits et les nombres de composants augment rapidement.

Un graphe peut être utilisé pour représenter un circuit électronique et ce problème devient le problème de partitionnement de graphe. Les sommets représentent les composants et les arrêtes représentent les connexions des composants.

- **La répartition de charge pour les machines parallèles** La répartition de charge pour les machines parallèles est pour but de répartir les charges de calcul entre les processeurs et de réduire la durée des communications entre les processeurs. Un graphe non-orienté permet de modéliser la répartition des charges pour les machines parallèles. Les sommets du graphe représentent les blocs de calcul et les arrêtes sont correspondants aux communications nécessaires à la programmation parallèles

- **La segmentation d'images** La segmentation d'images est pour but d'isoler les différents objets sur une image. Une image est représentée par une matrice des pixels. Il est possible de créer un graphe de l'image, les sommets représentent les pixels et les arrêtes sont correspondants la différence d'intensité lumineuse entre les pixels

➢ **La classification** Les outils de classification permettent de trier un ensemble des objets afin de regrouper des objets de même type. Il est donc possible de créer un graphe dont chaque objet est un sommet et chaque arrête exprime la similarité entre deux objets.

3.3. Complexité

La complexité d'un algorithme permet de quantifier mathématiquement le niveau de difficulté lors de la résolution de problèmes, selon des critères de temps d'exécution, de nombre d'itérations (ou d'opérations arithmétiques) ou encore d'espace mémoire nécessaire.

Ces grandeurs sont indiquées en fonction de la taille des données du problème à résoudre. La théorie de la complexité établit une classification des problèmes en fonction des algorithmes susceptibles de les résoudre, les machines de Turing constituant un étalon fréquemment utilisé. Sans entrer dans les détails, nous distinguons deux grandes classes2 P et NP des problèmes. Une description plus complète des différentes classes des problèmes d'optimisation combinatoire peut être trouvée dans [3] [4].

Un problème est dit de classe P s'il peut être résolu en un temps polynomial par une machine de Turing déterministe dont le temps d'exécution est de complexité polynomiale (P signifiant alors *Polynomial time).* Les problèmes de classe P sont considérés comme des problèmes faciles. C'est le cas notamment du problème de la connectivité d'un graphe.

La classe NP est une classe plus riche de problèmes pour lesquels on peut construire une machine de Turing non déterministe dont le temps d'exécution est de complexité polynomiale *(NP* signifiant *Nondeterministic Polynomial time).*

Un problème *NP* est dit *NP-complet* si tout problème de la classe NP peut lui être réduit : c'est le cas de nombreux problèmes considérés comme intraitables en pratique (pour lesquels les algorithmes déterministes connus sont exponentiels).

Le problème de partitionnement est un problème NP-complet [5, 6], il peut être résolu par :

- **Méthode exacte** : Dans le pire des cas, un tel algorithme aurait besoin d'énumérer toutes les solutions candidates de l'espace de recherche. Par conséquent, seule une très petite instance du problème peut être résolue à l'intérieur d'un temps raisonnable et les grandes instances sont impraticables.

- **Les métaheuristiques** : Elles ne garantissent pas de trouver de manière exacte l'optimal mais qui ont une bonne chance de trouver une bonne solution qui est proche de l'optimal en un temps raisonnable. Ces méthodes approchées constituent une alternative très intéressante pour traiter les problèmes NP-complet.

4. Les méthodes de détection de communautés

Le classement de méthodes de détection de communautés a été pris de différents points de vue.

Dans [7] les méthodes sont regroupées dans deux classes :

- Méthodes ***"globales"***: reposent sur l'exploitation de la globalité de l'information disponible sur le graphe en tentant de regrouper les sommets en prennent en compte leur position géographique.

- Méthodes ***"locales"***: reposent sur l'exploitation de déplacement de proche en proche dans le graphe en tentant de regrouper les sommets fortement connectés sans prendre en compte leur position géographique).

Dans [8] le regroupement en méthodes de partitionnement contraint (principale caractéristique est de rechercher des partitions de tailles égales) ou non contraint (la principale caractéristique est de rechercher des partitions de tailles différentes).

Dans [9] elles sont regroupées en classes d'algorithmes séparatives (partant d'une partition d'une seule communauté à n sommets qui sera divisée par la suite) ou d'algorithmes agglomératives (partant d'une partition de n communautés composées d'un seul sommet qui vont être regroupés d'une manière itérative)

Dans cette recherche, notre choix est porté sur la classification de différentes méthodes de partitionnements, selon une taxonomie constituée de deux classes : la classe des méthodes classiques et celle des méthodes métaheuristiques.

On précise que la liste des méthodes de partitionnement de graphe qui serra présentées dans ce qui suit est non exhaustive. Nous avons retenu les méthodes qui ont reçu le plus d'attention de la part de la communauté scientifique [9].

4.1. Méthodes classiques

4.1.1. Greedy Graph Growing Algorithm (GGGP)

Le *Greedy Graph Growing Algorithm* (GGGP) a été introduit dans [10]. Algorithme GGGP est pour résoudre le problème de partitionnement de graphe avec k = 2 (bissection).Elle consiste à créer de manière itérative un ensemble E. Cet ensemble est initialisé par un sommet par hasard. A chaque itération, un des sommets adjacents aux sommets de l'ensemble E est ajouté à E. Le sommet sélectionné est le sommet qui peut diminuer le plus le coût de coupe entre E et le reste de G. Autrement dit, c'est le sommet qui a le gain de la fonction objectif le plus grande. Le processus s'arrête si le poids de l'ensemble E est égal la moitié du poids total des sommets de graphe. Cette méthode est très simple et la qualité de la partition obtenue dépend principalement du sommet de l'initialisation. Pour obtenir de meilleur résultat, on peut lancer le programme avec des sommets de l'initialisation différents.

Procédure GGGP (G = (V, E))

$Prendre\ au\ hasard\ un\ sommet\ s0 \in R$

$P \leftarrow \{V_0\}$

$F \leftarrow \{V \in R\ tel\ que\ (v, v_0) \in E\}$

$Calculer\ les\ gains\ de\ F$

Tant que $poids(P) < ½\ poids(V)\ faire$

$Prendre\ le\ sommet\ V_i \in F\ de\ gain\ maximal$

$Déplacer\ V_i\ de\ F\ vers\ P$

Pour toute $arête\ (v, v_i)\ dans\ E\ faire$

si $v \in F\ alors$

$mettre\ à\ jour\ le\ gain\ de\ V$

sinon $si\ V \in E\ alors$

ajouter V à F
calculer le gain de V
fin si
fin pour
fin tant que
retourner P
fin Procédure

4.1.2. Méthode spectrale

L'utilisation de la méthode spectrale pour résoudre les problèmes de partitionnement de graphe est très ancienne. Les premiers articles sont proposés depuis les années 70s par W.Donath et A.Hoffman [11]. Cette méthode est basée au théorème spectral de l'algèbre linéaire. Ce théorème permet d'affirmer la diagonalisation des matrices réelles. Il justifie également la décomposition des matrices symétriques réelles en valeurs propres dans une base ortho normale de vecteurs propres. Le problème de partitionnement de graphe peut être ramené à la résolution d'un système numérique Mx = λx. Résoudre ce système numérique consiste à trouver une base orthogonale de vecteurs propres de la matrice M.

4.1.3. Algorithme de Kernighan-Lin

Kernighan et Lin ont proposé en 1970 une des premières méthodes de partitionnement de graphes [12]. Spécialement conçue pour le problème de bipartition de graphes, cette méthode utilise une partition initiale, et tente de l'améliorer tout en conservant le même nombre de sommets dans chacun des sous-graphes. Cette technique est souvent employée lors de phases dites de *raffinement* dans d'autres algorithmes de recherche. L'idée est la suivante:

Soit une bissection $\{V_1, V_2\}$ d'un graphe G = {V, E}, le traitement par l'algorithme de K-L consiste à trouver deux sous-ensembles V'_1 et V'_2 (avec $|V'_1| = |V'_2|$) contenus respectivement dans V_1 et V_2 puis de les permuter entre les partitions de façon à réduire au maximum la taille de la coupe (Figure 1.1). Ces permutations se font jusqu'à ce que l'on ne puisse plus améliorer la qualité de la partition. L'identification des ensembles V'_1

et V'_2 est basée sur un algorithme glouton recherchant de manière itérative une paire de sommets (chacun appartenant à des sous-graphes différents), puis à les permuter si cela diminue la taille de la coupe. La première paire sélectionnée est celle qui apporte la plus forte amélioration et ainsi de suite, sachant qu'un sommet ne peut être déplacé qu'une seule fois durant une même procédure. Lorsque toutes les paires ont été trouvées, une nouvelle procédure peut alors commencer.

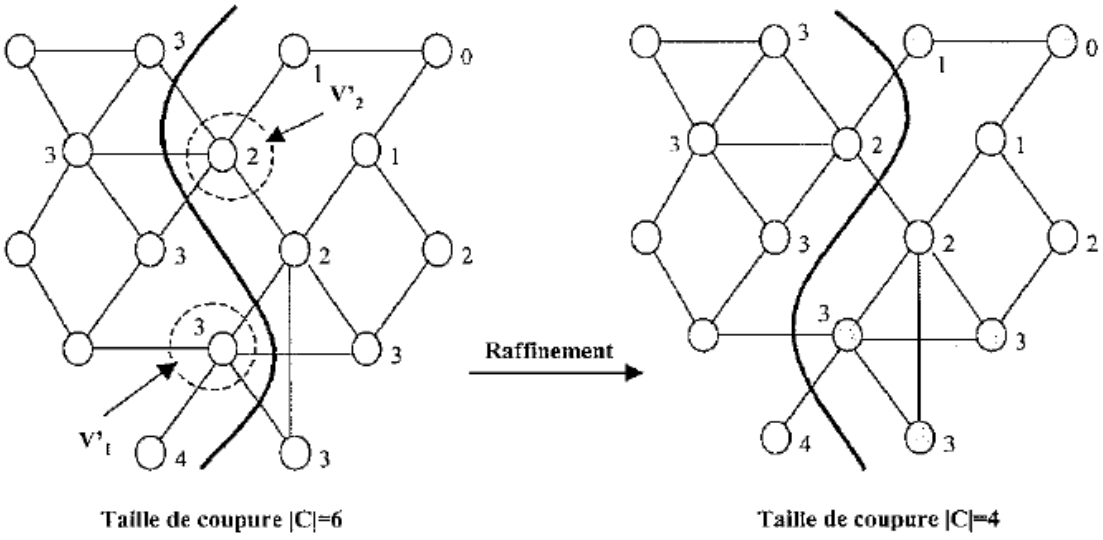


Figure1.1. Algorithme de Kernighan-Lin

4.1.4. Méthode Multi-niveaux

La méthode multi-niveaux a été parallèlement utilisée pour la première fois par Stephen Barnard et Horst Simon [13] et par Thang Bui et Curt Jones [14] en 1993.

L'idée principale de cette méthode est de regrouper les sommets ensemble pour traiter de groupes de sommets plutôt que de sommets indépendants. La méthode multi-niveaux se décompose en trois phases :

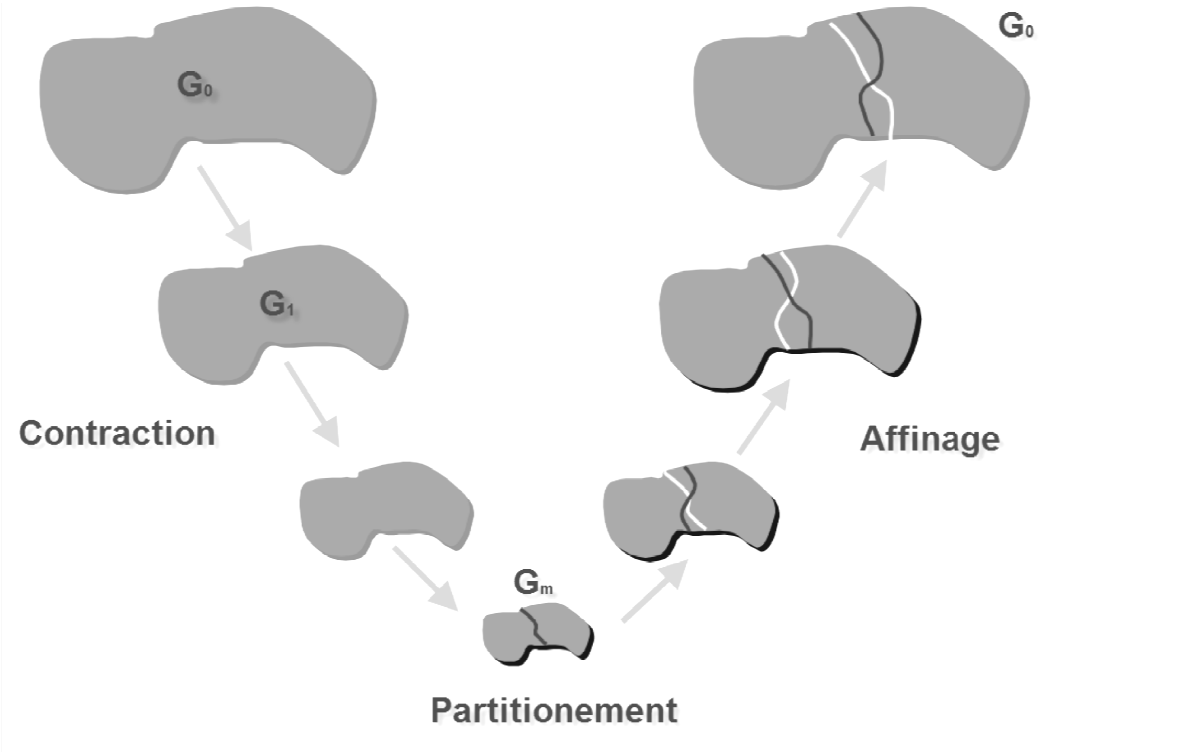


Figure1.2.Méthode multi-niveaux

▪ **Contraction**: Le graphe G0 est transformé en une suite de petits graphiques G1, G2, ..Gm pour que |V0|>|V1|>...>|Vm|

▪ **Partitionnement** : Création d'une partition Pm de graphe Gm

▪ **Affinage** : Les partitions Pi est projeté sur les graphes Gi. Chaque partition Pi est affinée en utilisant un algorithme local d'affinage. Cette méthode est générale et son efficacité dépend fortement des méthodes utilisées par les fonctions « contraction », « partitionnement » et « affinage ». De nombreuses méthodes sont utilisées pour réaliser ces trois fonctions. Parce que la méthode de multi-niveaux est très performante pour résoudre les problèmes de partitionnement de graphe, elle a progressivement remplacé l'utilisation des méthodes spectrales.

4.2. Méthodes de métaheuristiques

4.2.1. Le recuit simulé

Le recuit simulé est une métaheuristique inspirée d'un processus utilisé en métallurgie. Elle a été présentée pour la première fois par S.Kirkpatrick, C.Gelatt et M.Vecchi dans [15]. Ce processus alterne des cycles de refroidissement lent et de réchauffage (recuit) qui tendent à minimiser l'énergie du matériau. Son principe est simple: à partir d'un état initial, un nouvel état est créé basé sur l'état précédent par une modification élémentaire de cet état. Il est accepté si son énergie est plus faible, sinon il est accepté avec une certaine probabilité. Cette probabilité est basée sur la température, paramètre essentiel de contrôle. Au début, la température est haute et l'algorithme accepte plus des mouvais état pour éviter la minima locaux. De temps en temps, la température est diminuée et l'équilibre est atteint.

4.2.2. La recherche Taboue

La méthode taboue est une autre technique élaborée par Glover [16,17]. Elle est basée sur la notion de mouvements interdits (ou tabou). Chaque itération consiste à trouver le mouvement qui nous donne la meilleure solution dans le voisinage de la solution courante, sachant que certains mouvements sont interdits. Parfois, on choisit une solution qui détériore légèrement la solution courante pour sortir des minima locaux. L'inverse du mouvement effectué à chaque itération est rajouté dans une liste, appelée liste tabou, qui contient les mouvements interdits. Initialement, la liste taboue est vide.

Rolland et al. [18] ont utilisé cette méthode pour le partitionnement de graphes et ont obtenu de bons résultats, en terme de qualité de la solution et de temps d'exécution, par rapport au recuit simulé.

4.2.3. Algorithme de colonies de fourmis

L'algorithme de colonies de fourmis simule la recherche de nourriture des fourmis. Cette idée a été présentée première fois par A.Colorni, M.Dorigo et V.Maniezzo [19]. Une fourmi pose de la phéromone quand il se déplace. Les fourmis se dirigent de manière probabiliste en comptant de la quantité de phéromone qui est autour d'elles. Plus la

quantité de phéromone indiquant un chemin est grand, plus des fourmis a tendance à suivre ce chemin. Comme la phéromone s'évapore progressivement, le choix probabiliste que prend une fourmi pour choisir son chemin évolue continuellement. Grâce à l'utilisation de phéromone, les fourmis peuvent collectivement trouver le plus court chemin entre deux points. L'algorithme de colonies de fourmis est bien utilisé à nombre de problèmes d'optimisation et elle est adaptée pour le problème de partitionnement de graphe dans les années récemment.

4.2.4. Algorithmes évolutionnaires

Ces algorithmes sont basés sur une analogie avec l'évolution des espèces [20]. Ils partent d'une population initiale (ensemble de solutions) à partir de laquelle ils effectuent des croisements pour engendrer de nouvelles configurations. Ils gardent les meilleures populations résultantes pour effectuer d'autres croisements. Afin d'éviter les minima locaux, de temps à autre, les espèces sont soumises à des mutations. Des travaux de recherche sur l'utilisation des algorithmes génétiques pour le partitionnement de graphe [6] ont permis de montrer l'intérêt d'une approche évolutive. Dans [6], Bui et al. Ont proposé un algorithme génétique qu'ils comparent avec l'algorithme de recuit simulé. Ils obtiennent expérimentalement des partitions de qualité comparable voire supérieure**.**

5. Conclusion

Dans ce chapitre nous avons présenté les graphes complexes et leurs applications dans différents domaines concernés, ensuite on a introduit la problématique de détection de communautés, suivi d'un état de l'art des méthodes les plus utilisées dans la résolution du problème de détection de communautés (partitionnement de graphe) et l'une de ces méthodes , la méthode évolutionnaire, a été retenue et sera l'objet du deuxième chapitre, pour plusieurs raisons:

- Les algorithmes évolutionnaires présentent des caractéristiques intéressantes les rendant "populaires". Leur pouvoir de représentation leur permet de s'adapter facilement au problème traité. Leurs points forts sont une certaine robustesse par rapport aux différentes instances du problème.

Chapitre 2

Les algorithmes évolutionnaires

1. Introduction

Les algorithmes évolutionnaires (AE) font parti du champ de la vie artificielle qui présente des comportements similaires aux systèmes vivants naturels. Elle complète l'approche traditionnelle de la biologie en essayant de synthétiser leurs comportements sur des supports artificiels.

Leur objectif ne s'agit pas de trouver une solution analytique exacte, ou une bonne approximation numérique, mais de trouver une solution optimale d'une manière aléatoire.

Ce chapitre sera divisé en deux parties: la première partie consiste à donner une vue générale sur les algorithmes évolutionnaires, et la deuxième partie va détailler l'une de ces classes principales qui est « l'algorithme génétique ».

2. Les algorithmes évolutionnaires

2.1. Définition

Les algorithmes évolutionnistes ou algorithmes évolutionnaires (evolutionary computation en anglais), sont une famille d'algorithmes s'inspirant de la théorie de l'évolution[1] pour résoudre des problèmes divers. Ils font évoluer plusieurs solutions pour un problème donné dans le but de trouver une solution optimale, il s'agit donc de résoudre des problèmes d'optimisation, elles sont pour cela des métaheuristiques. [21]

On peut même les considérer comme étant des algorithmes stochastiques [21], car ils utilisent des processus aléatoires de façon itérative.

2.2. Principe [22]

L'algorithme évolutionnaire s'inspire du principe de sélection naturelle de Charles Darwin. La théorie de la sélection naturelle s'applique aux individus d'une population et repose sur trois principes:

La variation : les individus diffèrent les uns des autres

L'adaptation : les individus les mieux adaptés à leur environnement vivent plus longtemps et procréent plus.

L'hérédité : les caractéristiques des individus sont héréditaires.

Le principe de la sélection naturelle est le suivant : Les meilleurs individus vont plus procréer, ses caractéristiques avantageuses seront héritées par les générations suivantes et avec le temps, deviendront dominantes dans la population.

[1] L'évolution désigne la transformation des espèces vivantes qui se manifeste par des changements de leurs caractères phénotypiques au cours des générations. Les changements successifs peuvent aboutir à la formation de nouvelles espèces.

La théorie de l'évolution est une explication scientifique de la diversification des formes de vie qui apparaissent dans la nature. (wikipédia)

2.3. Terminologies et notations

Avant de présenter le déroulement général des algorithmes évolutionnaires, on va d'abord jeter l'œil sur quelques concepts utilisé :

-une population représente un ensemble d'individus indépendants les uns des autres.

-Un individu est une réponse à un problème donné, qu'elle soit ou non une solution valide du problème. (L'individu représente donc une solution)

- points de l'espace de recherche est un individu;

-Une génération est une itération dans l'algorithme.

-Fonction d'adaptation= fonction d'optimisation= fonction objectif=fitness

2.4. Principe général du fonctionnement d'un AE

Tous les algorithmes évolutionnaires font évoluer une « population » de solutions (individus), auxquels on associe une qualité (fitness), et à chaque étape de l'algorithme, on applique un des opérateurs de manipulation des individus. Les différents opérateurs sont regroupés sous des termes génériques [22] :

-**opérateurs de sélection** pour la sélection et le remplacement,

-**opérateurs de variation** pour la mutation et le croisement.

Les grandes étapes de cette évolution sont les suivantes

1) création d'une population initiale de N individus d'une manière aléatoire, afin de couvrir une large partie de l'espace des solutions du problème.

2) évaluation des individus composant la population initiale par la fonction objective.

3) Faire évoluer la population en génération successive, jusqu'à ce que la condition d'arrêt soit satisfiable:

a- Sélection de k individus qui vont jouer le rôle des futurs parents, c'est-à-dire ceux qui vont se reproduire.

b- croisement des individus sélectionné.

c- mutation des enfants (individus issus de croisement),

d- évaluation des individus (enfants) résultant des croisements et des mutations.

e- sélection de N individus parmi tous les individus de la population (parents et enfants), qui seront présentés à la prochaine itération. Cette sélection est assurée par l'opérateur de remplacement.

f- après la sélection de la nouvelle population, il faut vérifier si le critère d'arrêt est atteint, si c'est le cas, l'algorithme s'arrête en sortant avec le meilleur individu, sinon on recommence le processus 3.

2.5. Principale famille

A l'origine, trois types d'algorithmes évolutionnaires ont été développés isolément et à peu prés simultanément, par différents scientifiques : la programmation évolutionniste (L. Fogel 1966), les Stratégies d'évolution (J. Rechenberg 1973) et les Algorithmes Génétiques (J. Holland 1975). Dans les années 90, ces trois champs ont commencé à sortir de leur isolement et ont été regroupés sous le terme des algorithmes évolutionnaires (Evolutionnary Computation). Deux ans après, Un autre type d'algorithme évolutionnaire à été apparu sous le nom : « programmation génétiques » (Koza 1992), dans le but de trouver le meilleur programme qui répond au mieux à un problème donnée. [21][28]

2.5.1. Programmation évolutionnaire

Programmation évolutionnaire (EP) a été imaginée par L.J. Fogel et ses coauteurs dans les années 60, et reprise par son fils D.B. Fogel dans les années 90, en Californie, USA.

Historiquement, ces algorithmes étaient conçus pour des problèmes d'apprentissage à partir d'automates à états finis et n'utilisaient que des opérateurs de mutation et de remplacement. EP a rapidement été généralisée à des espaces de recherche très variés, mais n'utilisent toujours pas d'opérateur de croisement. Ils diffèrent des stratégies d'évolution en ce qu'ils privilégient des opérateurs de remplacement stochastiques, dans lequel les individus les plus mauvais ont tout de même une (petite) chance de survie. [21,26]

2.5.2. Stratégies d'évolution

La stratégie d'évolution (SE) est proposée par Ingo Rencherberg, en 1973, à l'université technique de Berlin, en Allemagne. L'algorithme manipule itérativement un ensemble de vecteurs de variables réelles de dimension fixe (c.à.d. la dimension des individus est fixe), à l'aide d'opérateurs de mutation et de sélection. La sélection s'effectue par un choix déterministe des meilleurs individus, selon l'échelle de valeur de la fonction objectif. L'étape de mutation est classiquement effectuée par l'ajout d'une valeur aléatoire, tirée au sein d'une distribution gaussienne[2].

Les ES les plus performantes utilisent les mutations auto adaptatives, dans lesquelles chaque individu porte avec lui les paramètres de la mutation gaussienne qui lui sera appliquée : « paramètres eux-mêmes soumis à mutation ». [21,25]

2.5.3. Algorithmes Génétiques

Les algorithmes génétiques (GA) sont les plus populaires des algorithmes évolutionnaires, ils respectent leurs schéma générale (c.à.d. le principe général du fonctionnement d'un algorithme évolutionnaire) avec une particularité : chaque individu de la population est codé par un chromosome, en effet, Une population est un ensemble de chromosomes dont chacun est constitué d'un ensemble de gènes. [21]

Cet algorithme fait l'objet de notre travail, il sera détaillé dans la deuxième partie.

[2] La mutation gaussienne, consiste à rajouter une valeur aléatoire appelée pas au vecteur des variables. La forme la plus générale est alors X := X+ pas.

2.5.4. Programmation génétique

La programmation génétique **(GP)** est une méthode automatique, basée sur le même principe que les algorithmes génétiques, à la différence que celle-ci utilise le codage en arbre et elle a pour but de trouver par approximations successives des programmes répondant au mieux à une tâche donnée. (Elle s'intéresse spécifiquement à la construction automatique de programmes)

-La population initiale est créée d'une manière aléatoire à l'aide des fonctions, opérateurs et terminaux qui composent le problème.

-Chaque individu représente un programme, et il sera évolué en fonction de la façon dont il résout le problème.

-A l'aide des opérateurs génétique, il crée de nouveaux programmes.

La programmation génétique est coûteuse en temps de calcul machine, puisqu'elle met en concurrence de façon parallèle un grand nombre d'algorithmes voisins. [21, 24, 23]

3. Les algorithmes génétiques

3.1. Définition

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes, dans le but d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode exacte pour le résoudre en un temps raisonnable. Les algorithmes génétiques, basant sur la théorie de « l'évolution des espèces » formulée par le naturaliste Charles Darwin, sont le fruit des recherches de John Holland et de ses collègues et élèves de l'Université du Michigan qui ont, dès 1960, travaillé sur ce sujet. Le premier aboutissement de ces recherches a été la publication en 1975 de son livre : « Adaptation in Natural and Artificial System», dans lequel il a expliqué comment ajouter des opérateurs de variation (croisement et mutation) dans un programme informatique. [26]

Le premier but de Holland, dans le développement des AG, n'est pas de trouver une solution pour un problème précis, mais plutôt d'améliorer la compréhension des processus naturels d'adaptation, et de concevoir des systèmes artificiels possédant des propriétés similaires aux systèmes naturels. [26]

Pourquoi utiliser les algorithmes génétiques ?

-Les AG ne permettent pas d'obtenir certainement une solution optimale exacte, mais plutôt une solution de qualité en temps raisonnable.

-Espaces de recherche importants.

-Pas d'algorithmes d'déterministes adaptés.

3.2. Le déroulement général de cet algorithme

L'algorithme génétique débute par la création d'une population initiale, après le choix de codage adéquat aux solutions (c'est-à-dire les structures de données qui coderont les gènes de chromosome), ensuite il fait évaluer tous les individus pour sélectionner k individus qui seront participé dans les opérations de croisement et de mutation afin de créer de nouveaux individus, il ne reste donc qu'à sélectionner ceux qui vont continuer à participer à l'amélioration de notre population.

La méthode consiste à insérer les nouveaux individus dans la population, à trier cette population selon l'évaluation de ses membres, et à ne conserver que les N meilleurs individus, tel que N ne doit être ni trop petit, pour éviter d'avoir des populations de plus en plus petite au fil des générations: elle pourrait même disparaître. Ni trop grand, sinon le temps de traitement sera explosé puisque la population de chaque génération sera plus grande. Donc, il faut toujours garder la même taille de population d'une génération à l'autre. [27]

A la fin de chaque itération, une nouvelle population sera obtenue, le processus d'amélioration des individus continuera pour obtenir une nouvelle population et ainsi de suite, jusqu'à ce que la condition d'arrêt soit vérifiée.

Le déroulement d'un algorithme génétique est résumé brièvement dans le schéma suivant :

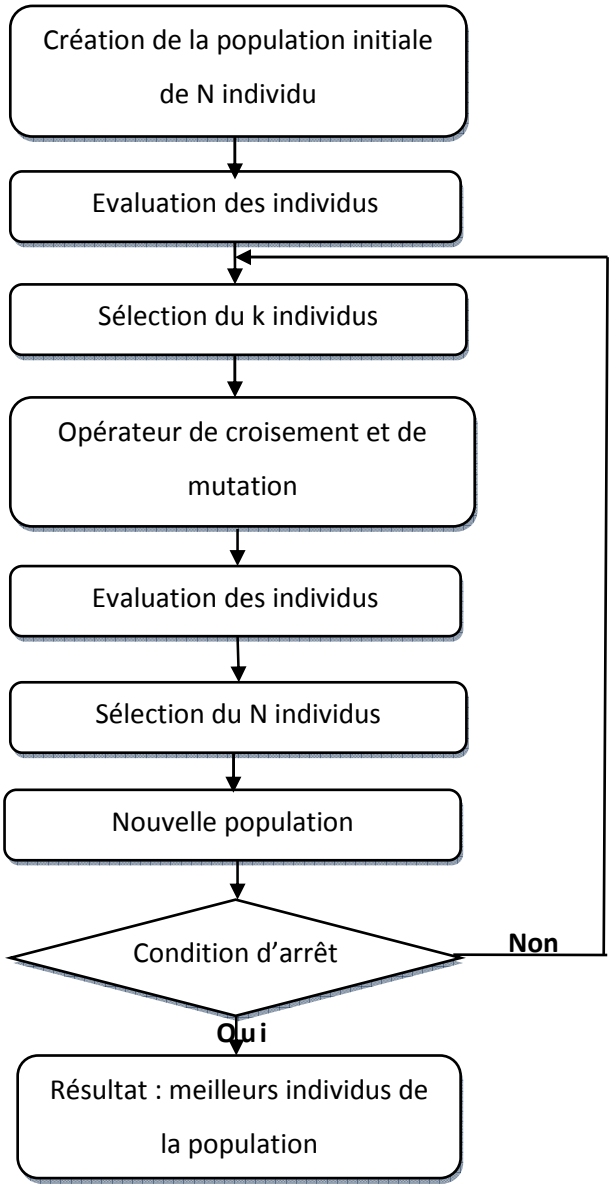


Schéma2.1. Déroulement générale d'un algorithme génétique

3.3. Description détaillée

3.3.1. Codage des données

Le premier pas dans l'implémentation des algorithmes génétiques est de choisir une structure de donnée qui codera les gènes de chromosome, ce type de codage va jouer un rôle essentiel dans l'efficacité de l'algorithme. Principalement, il existe 3 types de codage:

Le Codage binaire

Historiquement le codage utilisé par les algorithmes génétiques était représenté sous forme de chaînes de bits {0,1} contenant toute l'information nécessaire à la description d'un point dans l'espace d'état. Ce type de codage a pour intérêt de permettre de créer des opérateurs de croisement et de mutation assez simples.

Cependant, ce type de codage n'est pas toujours bon, puisqu'il manque de la signification.

Codage à caractères multiples

Pour plusieurs applications, il est plus naturel de coder les chromosomes par des types de base primitif tel que: les caractères, les nombres entiers, les nombres réels ou bien à l'aide d'une combinaison de ces 3 types, il dépend donc de problème à résoudre. Aujourd'hui, ce type de codage est le plus répondus.

Le Codage sous forme d'arbre

Ce codage utilise une structure de donnée complexe, il s'agit des structures de données arborescentes. Le Codage sous forme d'arbre est généralement utilisé dans le cas où les solutions n'ont pas une taille fixe, son problème est que les arbres résultants sont souvent difficiles à analyser et que l'on peut se retrouver avec des arbres « solutions » dont la taille sera importante. De plus, les performances de ce type de codage sont très élevées.

Le choix du type de codage ne peut pas être effectué de manière sûre dans l'état actuel des connaissances. Selon les chercheurs dans ce domaine, la méthode actuelle à

appliquer dans le choix du codage consiste à choisir celui qui semble le plus naturel en fonction du problème à traiter et développer ensuite l'algorithme de traitement.

3.3.2. Création de la population initiale

Ce mécanisme doit être capable de produire une population d'individus non homogène qui servira de base pour les générations futures.

La manière dont le programmeur va créer chacun des individus de cette population est entièrement libre. Il suffit que tous les individus créés fournissent une solution, même mauvaise, au problème donné. Par exemple, si vous cherchez des solutions pour un jeu d'échecs, il suffira que les individus créés possèdent des mouvements autorisés sur des pièces existantes.

Le principe suivit pour créer ces individus est le hasard, cette méthode amène un concept très utile dans les algorithmes génétiques qui est la diversité de solutions.

Les premiers individus créés sont sélectionnés d'une manière aléatoire, et l'objectif n'est pas de trouver une solution optimale, mais de quoi fabriquer les meilleures solutions possibles dans les futures générations.

3.3.3. L'évaluation des individus

Une fois que la population initiale a été créée, il faut évaluer chacune des solutions obtenue à l'aide d'une fonction objectif.

La fonction d'évaluation doit être choisie avec soin, vu que c'est elle qui détermine ce qui sera optimisé par l'algorithme génétique.

3.3.4. La sélection

La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les mauvais. On trouve dans la littérature un nombre important de principes de sélection plus ou moins adaptés aux problèmes qu'ils traitent, parmi elle on peut citer:

La roulette

La sélection des individus par le système de la roulette s'inspire des roues de loterie. A chacun des individus de la population est associé un secteur d'une roue. L'angle du secteur étant proportionnel à la qualité de l'individu qu'il représente. Vous tournez la roue et vous obtenez un individu. Les tirages des individus sont ainsi pondérés par leur qualité. Ainsi, les meilleurs individus ont plus de chance d'être croisés et de participer à l'amélioration de notre population.

ome1
ome2
ome3
ome4
ome5
ome6

La sélection par rang

La sélection par rang est une variante du système de roulette. Il s'agit également d'implémenter une roulette, mais cette fois ci les secteurs de la roue ne sont plus proportionnels à la qualité des individus, mais à leur rang dans la population triée en fonction de la qualité des individus.

D'une manière plus parlante, il faut trier la population en fonction de la qualité des individus puis leur attribuer à chacun un rang. Les individus de moins bonne qualité obtiennent un rang faible (à partir de 1). Et ainsi en itérant sur chaque individu on finit par attribuer le rang N au meilleur individu (où N est la taille de la population). La suite de la méthode consiste uniquement en l'implémentation d'une roulette basée sur les rangs des individus. L'angle de chaque secteur de la roue sera proportionnel au rang de l'individu qu'il représente.

La sélection par tournoi

Le principe de la sélection par tournoi est d'augmenter les chances pour les individus de piètre qualité de participer à l'amélioration de la population. Le principe est très rapide à implémenter. Un tournoi consiste en une rencontre entre plusieurs individus pris au hasard dans la population. Le vainqueur du tournoi est l'individu de meilleure qualité. Vous pouvez choisir de ne conserver que le vainqueur comme vous pouvez choisir de conserver les 2 meilleurs individus ou les 3 meilleurs. A vous de voir, selon que vous souhaitez créer beaucoup de tournois, ou bien créer des tournois avec beaucoup de participants ou bien mettre en avant ceux qui gagnent les tournois haut la main. Vous pouvez faire participer un même individu à plusieurs tournois. Une fois de plus, vous êtes totalement libre quant à la manière d'implémenter cette technique de sélection.

L'élitisme

Cette méthode de sélection permet de mettre en avant les meilleurs individus de la population. Ce sont donc les individus les plus prometteurs qui vont participer à l'amélioration de notre population. Cette méthode a l'avantage de permettre une convergence (plus) rapide des solutions, mais au détriment de la diversité des individus. On prend en effet le risque d'écarter des individus de piètre qualité, mais qui auraient pu apporter de quoi créer de très bonnes solutions dans les générations suivantes.

3.3.5. Croisement

Le croisement a pour but d'enrichir la diversité de la population en manipulant la structure des chromosomes. Classiquement, les croisements sont envisagés avec deux parents et génèrent deux individus enfants.

Pour développer un opérateur de croisement, trois étapes sont nécessaires :

-choix de deux chromosomes survivants, c'est à dire sélectionnés par la procédure de reproduction,

-choix au hasard d'un emplacement où couper ces deux chromosomes,

-recoller les portions de chromosomes en les croisant. Les deux chromosomes de départ ont ainsi échangé des segments de code génétique.

Lors de cette opération, deux chromosomes s'échangent des parties de leurs chaînes, pour donner de nouveaux chromosomes. Ces recombinaisons peuvent être simples ou multiples.

Dans le premier cas, les deux chromosomes se croisent et s'échangent des portions des gènes en un seul point.

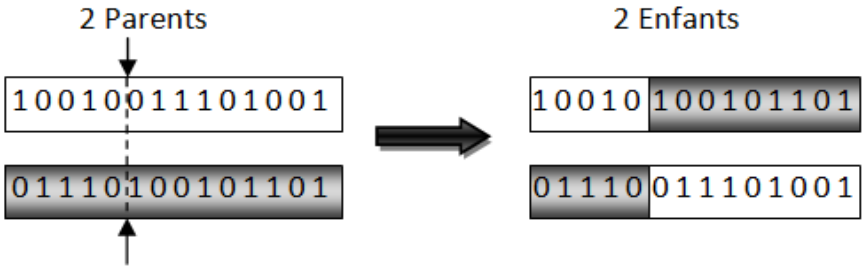


Schéma2.3. Croisement en un seul point

Dans le deuxième cas, il y a plusieurs points de croisement (2 ou 3 peuvent suffire), c'est le croisement multi-points. Cette opération est la plus prépondérante.

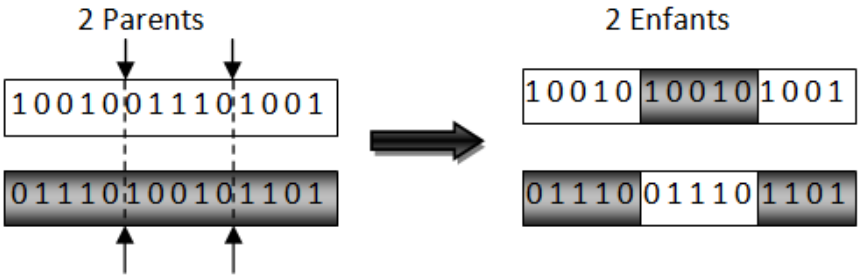


Schéma2.4. Croisement multi-points

Dans le cas où on sélectionne plusieurs endroits, les bits de la chaîne sont comparés entre les deux parents. Les bits sont échangés avec une probabilité fixe, en général 0.5. Ce croisement s'appelle « le croisement uniforme ».

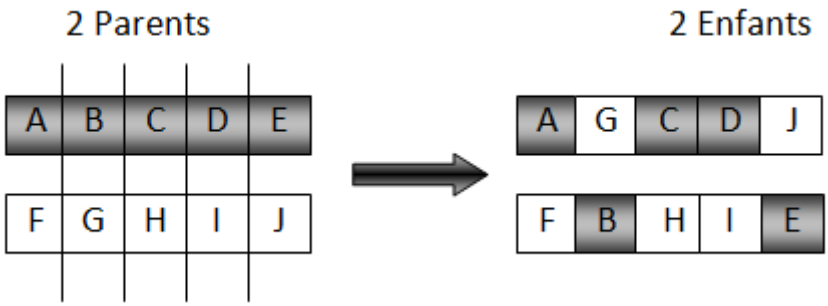


Schéma2.5. Croisement uniforme

3.3.6. Mutation

La mutation est un bagage génétique qui apporte une modification aléatoire au code génétique d'un individu. Pour l'appliquer, on sélectionne au hasard un ou plusieurs bits du chromosome (qui s'approche plus de la sortie que les autres) et modifie sa valeur, ça ne veut pas dire que l'individu muté sera meilleur ou moins bon, mais il apportera des possibilités supplémentaires qui pourraient bien être utiles pour la création des bonnes solutions.

La mutation ne doit pas être appliquée systématiquement, mais en fonction d'une probabilité (Un pourcentage de mutation). Il est nécessaire de choisir pour ce taux une valeur relativement faible de manière à ne pas tomber dans une recherche aléatoire et conserver le principe de sélection et d'évolution. On peut prendre par exemple une valeur = 1 / (longueur chromosome), c.à.d. plus que la taille de chromosome soit grande, plus on prendra des valeurs petites.

Schéma2.6. mutation

L'opérateur de mutation modifie donc de manière complètement aléatoire les caractéristiques d'une solution et de garantir que chaque point de l'espace de recherche puisse être atteint, ce qui permet d'introduire et de maintenir la diversité au sein de notre population de solutions. Elle sert aussi à limiter les risques d'une convergence

prématurée. En effet, dans le cas d'une convergence prématurée on se retrouve avec une population dont tous les individus sont identiques. Dans ce cas, le croisement ne changera rien à la situation, puisqu'il se retrouvera toujours entraîne d'échanger des informations entre des individus strictement identiques, d'où le blocage de l'évolution de la population. Et l'algorithme s'arrêtera sans atteindre la solution optimale.

3.3.7. Conditions d'arrêt

Il existe plusieurs critères d'arrêt dans l'algorithme génétiques. Les plus utilisé sont:

-Fixé le nombre de génération maximum à générer.

-Suivre le meilleur chromosome de la population : s'il ne s'est pas amélioré les x dernières générations, l'algorithme s'arrête.

-Ou bien de suivre la différence entre les deux meilleurs individus et si elle est inférieure à une certaine tolérance, il faut s'arrêter.

3.4. Les limites des algorithmes génétiques [26]

-Le temps de calcul : par apport à d'autres métaheuristiques, les algorithmes génétiques nécessitent de nombreux calculs, en particulier au niveau de la fonction d'évaluation.

-Ils sont le plus souvent difficiles à mettre en œuvre : des paramètres comme la taille de la population ou le taux de mutation sont parfois difficiles à déterminer, or le succès de l'évolution en dépend. Plusieurs essais sont donc nécessaires, ce qui limite encore l'efficacité de l'algorithme. En outre, choisir une bonne fonction d'évaluation est aussi critique. Celle-ci doit prendre en compte les bons paramètres du problème. Elle doit donc être choisie avec soin.

-Il faut aussi noter l'impossibilité d'être assuré, même après un nombre important de générations, que la solution trouvée soit la meilleure. On peut seulement être sûr que l'on s'est approché de la solution optimale (pour les paramètres et la fonction d'évaluation choisie), sans la certitude de l'avoir atteinte.

4. Conclusion

Dans ce chapitre on a débuté par une partie d'introduction aux algorithmes évolutionnaires, et on a vu que dans ces types d'algorithmes il n'existe pas de chef qui donne des ordres aux autres individus, ni des règles dictant un comportement global, mais l'évolution des individus se fait d'une manière totalement aléatoire.

Ensuite, on a détaillé dans la deuxième partie l'une de ces principales catégories, il s'agit des algorithmes génétiques, qui ont l'énorme avantage de pouvoir être appliqués dans un grand nombre de domaines de recherche de solutions, lorsqu'il n'est pas nécessaire d'avoir la solution optimale, qui prendrait par exemple trop de temps et de ressources pour être calculée (ou tout simplement si personne n'est capable de la trouver de manière théorique), elle offre aussi aux programmeurs la liberté de choisir la façon d'implémenter des différents traitements et la liberté de choisir les paramètres à utiliser et de les modifier ensuite si les solutions obtenus ne sont pas bonnes. On a noté aussi certaines limites attachées à l'emploiement de ces algorithmes, tel que le temps de calcule, le choix des paramètres, l'incertitude d'avoir atteint la solution optimale, etc.…

Chapitre3

L’algorithme génétique

1. Introduction

L’identification de la structure communautaire a été un sujet de recherche important dans les réseaux complexes, il ya eu de nombreux algorithmes proposés jusqu'à présent pour détecter les structures communautaires dans les réseaux complexes, où la plupart des algorithmes ne sont pas adaptés pour les très grands réseaux en raison de leur complexité temporelle.

La détection de communautés en utilisant les algorithmes génétiques a été l’objet de nombreux chercheurs et différentes approches ont été proposées [29,30].

Ce chapitre est devisé en deux sections :

Nous présentons dans la première section un algorithme génétique donné dans [31], qui est la référence de l’article publié en 2007 par « Mursel Tasgin » & « Haluk Bingol », pour détecter des communautés disjointes dans les graphes complexes monopartis, basée sur l’optimisation de la modularité de «Newman» [32]. Elle est adaptable à de très grands graphes et ne nécessite aucune connaissance à priori sur le nombre de communautés ou de toute valeur de seuil. Il a une complexité temporelle O (e) où e est le nombre d'arêtes dans le graphe.

Dans la deuxième section, nous présentons une approche qui consiste à adapter l’algorithme génétique présenté dans la première section dans le but de détecter des communautés chevauchantes dans les réseaux biparti.

2. Présentation de l'algorithme génétique

2.1. Principe de l'algorithme génétique

Les étapes de l'algorithme génétique ont été modifiées pour satisfaire les besoins de notre algorithme, tel que le changement du fonctionnement du cross-over et l'insertion de quelques étapes supplémentaires lors de la création initiale de la population.

Chaque solution du problème de détection de communautés est appelé chromosome (terme utilisé dans les algorithmes génétique). Chacun de ces chromosomes est évalué par une fonction objective qui est la modularité [32].

L'algorithme génétique débute par la création de la population initiale, suivie de l'exécution des opérations génétiques classiques, tel que le croisement et la mutation, pour plusieurs itérations. Pendant chaque itération, avant que l'algorithme évalue les valeurs de fitness de chaque chromosome, il réalise au préalable une autre opération appelée Clean-up afin d'améliorer la qualité de la fitness. Ensuite, les meilleurs chromosomes seront sélectionnés en fonction de leurs fitness pour l'itération suivante. Avec cette méthode, il est garanti qu'un bon chromosome de la population, ne sera jamais perdu.

2.2. Représentation d'un individu

Chaque chromosome est représenté par un tableau d'entier à N élément tel que N représente le nombre total de nœuds du graphe. Le tableau enregistre les identifiants de la communauté (***IDcomm***) pour chaque nœud. (Les indices du tableau désignent les numéros de nœuds. Le schéma ci-dessous montre la structure générale d'un chromosome.

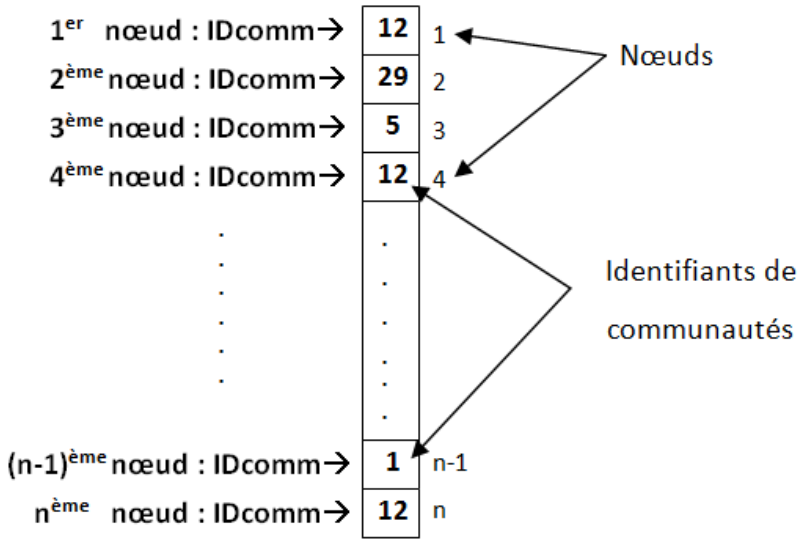


Schéma 3.1. Structure générale d'un chromosome

2.3. Fonction objective

Pour mesurer la qualité d'un découpage en communautés, une mesure a été introduite par NEWMAN en 2004 [32], c'est la modularité.

- Posons, pour un nœud v, c_v la communauté de v.
- Posons m le nombre total de lien du graphe G, tel que :

$$m = \frac{1}{2}\sum_{vw} A_{vw}$$

Où : A_{vw} vaut 1 s'il existe une arête entre v et w, 0 sinon.

L'expression de la modularité la plus facilement manipulable est :

$$\mathbf{Q} = \sum_{\boldsymbol{i}} \boldsymbol{e_{ii}} - \boldsymbol{a_i^2}$$

Dont:

$$e_{ii} = \frac{1}{2\mathrm{m}}\sum_{vw} A_{vw£(C_v,i)£(C_w,i)}$$

£(c_w,i) : Cette écriture veut dire que le nœud w appartient à la communauté i.

Ou de manière plus intuitive : e_{ii} est le poids (normalisé) des arêtes dans la communauté i.

$$a_i = \frac{1}{2m}\sum_v K_v £(C_v, i)$$

Où $k_v = \sum_w A_{vw}$ ou formulé autrement, k_v est le degré de v.

a_i est donc le poids total (normalisé) des arêtes connectées à des nœuds de la communauté i.

Cette modularité a été largement utilisée dans les études récentes. Elle représente la métrique de qualité pour l'évaluation du partitionnement d'un réseau en communautés.

Les algorithmes qui se basent sur cette modularité, ne nécessite aucune connaissance, ni sur la taille, ni sur le nombre des communautés. Il y'a lieu de chercher le « meilleur découpage en communautés » en maximisant la modularité du réseau. Par rapport aux autres méthodes, celle-ci nécessite moins de calcule, ce qui rend l'algorithme utile dans les réseaux réels de grande taille.

2.4. Population initiale

Une population initiale sera constituée d'un certain nombre de chromosome. Lors de l'initialisation d'un chromosome, chaque nœud se voit attribuer aléatoirement un identifiant de communauté. Cependant nous avons besoin d'un mécanisme pour donner un sens au placement initial des nœuds dans les communautés. Si deux nœuds doivent être dans la même communauté, ils doivent posséder une connectivité avec les autres. Dans le cas le plus simple, ils peuvent être voisins. De ce fait, après avoir attribué aléatoirement des IDcomm aux nœuds du graphe, nous sélectionnons, certains nœuds et nous attribuons leurs identifiants aux nœuds qui leurs sont adjacents. Ce processus de création d'un chromosome d'une manière aléatoire et l'application de cette procedure est répété autant de fois que la taille de population à former. Ce mécanisme dans la

création initiale de la population améliore la convergence de l'algorithme et élimine les itérations inutiles.

Suivant ce principe, pour créer chaque chromosome, un ensemble de nœud est sélectionné d'une manière aléatoire, puis leurs ***IDcomms*** sont affectés à tous leurs voisins. Ce mécanisme utilisé pour créer la population initiale, améliore la convergence de l'algorithme et élimine les itérations inutiles.

Le schéma suivant, montre un exemple d'initialisation d'un chromosome à partir du chromosome crée aléatoirement, où deux nœuds, nœud 4 avec ID*comms*=12 et nœud 8 avec ID*comms*=24, sont sélectionnés d'une manière aléatoire, ensuite ID*comms*=12 sera affecté à tous les voisins du nœud 4 et ID*comms*=24 sera affecté à tous les voisins du nœud 8.

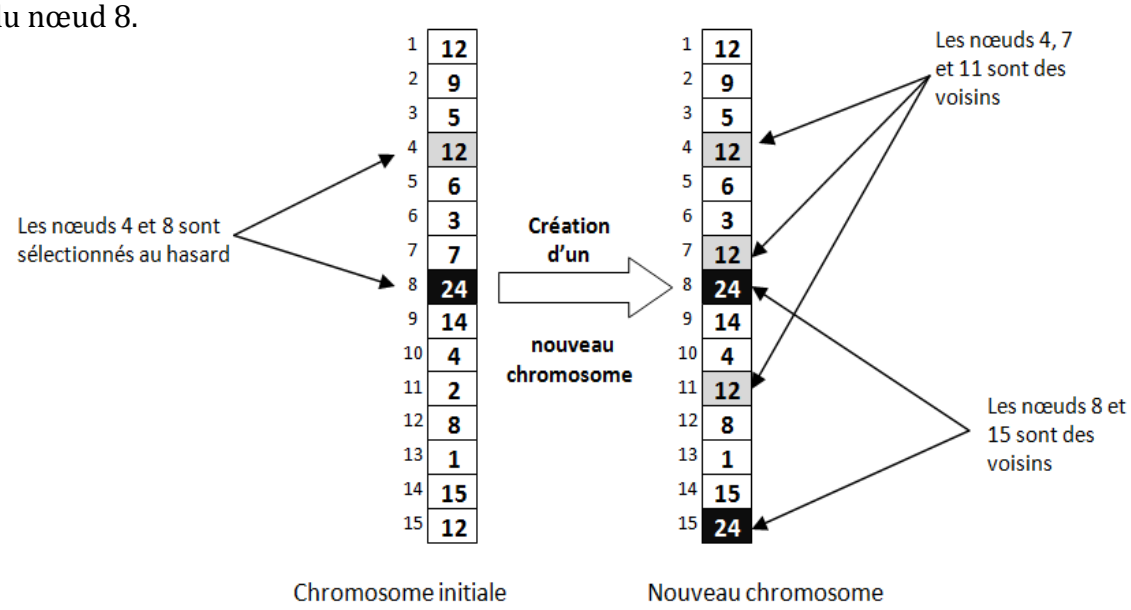


Schéma 3.2. Initialisation d'un chromosome

2.5. Les operateurs génétiques de l'algorithme

Dans cette section, on détaille le fonctionnement de différents opérateurs utilisés.

2.5.1. Croisement

Au lieu d'échanger tous les gènes après un point de croisement, sélectionné au hasard, entre deux chromosomes, une idée consiste à sélectionner une communauté du premier chromosome. Puis chercher, dans ce même chromosome, tous les nœuds qui

appartiennent à cette communauté, et enfin, affecter cette communauté aux mêmes nœuds du deuxième chromosome.

La modification du cross-over garantit un fonctionnement d'échange de communautés dans un seul sens, en effet, après chaque croisement un seul fils sera créé.

Pour le nombre de croisement à réaliser dans chaque génération :

1- *Sélectionner au hasard deux chromosomes en fonction de leurs fitness,*

2- *nommer chacun de ces chromosomes* :

le premier « source » et le deuxième « destination »

3- *sélectionner au hasard certaines communautés du chromosome source.*

4- *pour chaque communauté sélectionnée*:

−chercher itérativement les nœuds qui appartiennent à cette communauté.

*−transférer l'**IDcomm** de ces nœuds (du chromosome source) aux nœuds du chromosome destination.*

Le schéma suivant montre les détailles d'un simple croisement définit dans cet algorithme.

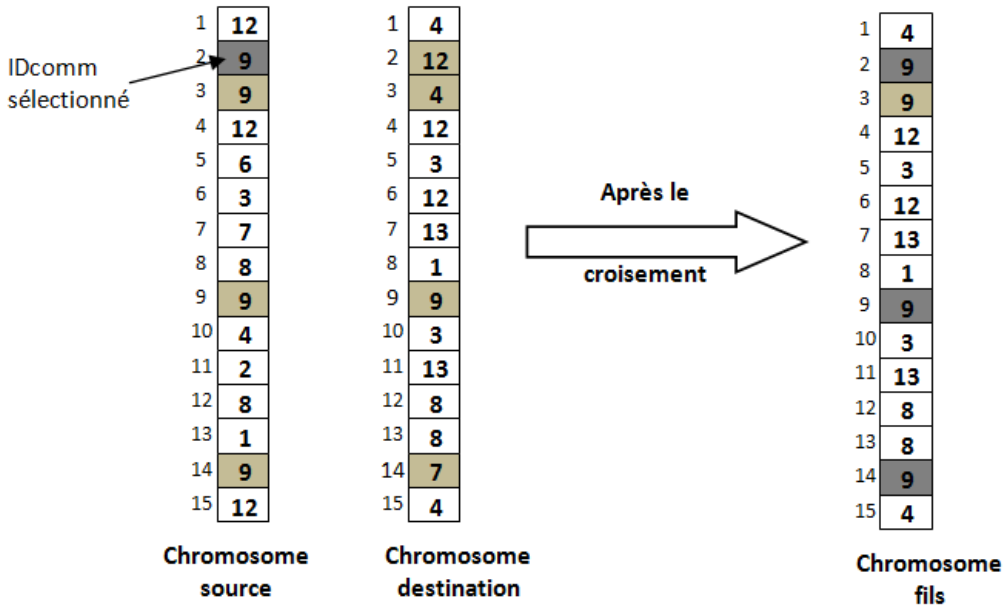


Schéma 3. 3. L'opérateur de croisement

2.5.2. Mutations

Après avoir effectué un certain nombre de croisement, l'algorithme passe à l'étape suivante pour réaliser la mutation. Le principe de la mutation utilisée dans cet algorithme est similaire à celui définit dans le chapitre précédant : Pour un certain nombre de chromosome choisis au hasard, un nœud est placé dans une communauté d'une manière aléatoire.

2.5.3. Le processus clean-up

Bien que la fitness totale soit bonne pour la détection de communautés, il peut y avoir un petit nombre de nœuds mal placés qui n'affectent pas la valeur de la fitness. Pour cela, un mécanisme d'amélioration de la qualité des communautés est nécessaire.

Le processus clean-up est basé sur une nouvelle variance de la communauté, qui vise à réduire le nombre de nœuds mal placé.

- **La variance de la communauté (Community Variance)**

Une communauté doit contenir plus de liens internes (entre les nœuds de la même communauté) que de liens externes (entre les communautés). Pour cette raison, les voisins d'un nœud doivent, la plupart du temps, être à l'intérieur de la même communauté.

La variance de communauté CV (i***)*** d'un nœud **i** est définit comme le nombre de communautés différentes parmi les voisins et le nœud lui-même. CV (i) doit être faible pour une bonne structure de la communauté.

$$CV(i) = \frac{\sum_{(i,j)\in E} f(i,j)}{\deg(i)}$$

Où :

$$f(xi,j) = \begin{cases} 1, & comm(i) \neq comm(j) \\ 0, & sinon \end{cases}$$

Tel que :

- $\deg(i)$ *est le degré du noeud* i
- E *est le nombre total des arrêt*
- $comm(i)$ *est la communauté du noeud* i

Le processus clean-up consiste à analyser La variance de la communauté CV(i) de certains nœuds choisis au hasard.

Si CV(i) de ce nœud est supérieur à la valeur seuil (paramètre en entrée), alors le nœud et tous ses voisins sont mis dans une même communauté. La nouvelle collectivité sera la communauté la plus répandue chez les voisins, la communauté qui contient le plus grand nombre de nœuds dans le voisinage du nœud sélectionné. Si la valeur seuil n'est pas dépassée, aucune opération n'est effectuée sur les identifiants de la communauté.

Ce processus améliore à la fois la qualité de la détection de communauté en supprimant les nœuds placés à tort, en raison du comportement aléatoires de l'algorithme, et fournit un meilleur mécanisme pour identifier les communautés des nœuds hub, qui résident entre les communautés.

A la fin des itérations, l'algorithme se termine en fournissant la meilleure partition trouvée du graphe.

2.6. Cas d'utilisation de l'algorithme

L'algorithme génétique présenté dans ce chapitre s'applique sur des graphes mono-partis, sans boucle, non orienté et non pondéré. On ne s'intéresse ni aux types des objets connectés, ni aux types de relation qui les relient, on se base uniquement sur la connectivité des objets dans le but de détecter des communautés disjointes. Mais dans les réseaux réels, de nombreuses collectivités peuvent se chevaucher comme est le cas dans les réseaux sociaux, où les individus peuvent appartenir à plusieurs groupes. Il existe d'ailleurs plusieurs façons de détecter la structure de communautés chevauchantes dans un réseau.

La plupart de ces méthodes sont dédiées pour des réseaux mono-partis. Toutefois, de nombreuses recherches sont consacrées, ces dernières années, afin de trouver des nouvelles approches pour la détection de communautés chevauchantes dans les réseaux bipartis et c'est l'une de ces approches qui fera l'objet de la suite de ce chapitre.

3. Détection de communautés chevauchantes dans des graphes bipartis

Un graphe biparti est une catégorie spéciale de graphe, où les nœuds peuvent être divisés en deux sous-ensembles disjoints, tels qu'il ne peut y avoir aucun lien qui relie deux nœuds du même sous ensemble. Les réseaux auteur-publication, les réseaux acteur-film, des réseaux produit-consommation, etc. sont des exemples concrets dans la catégorie des réseaux bipartis.

L'approche sur laquelle nous allons se baser pour détecter des communautés chevauchantes dans les réseaux bipartis, se base sur le même algorithme appliqué pour détecter des communautés disjointes. Toutefois, dans ce cas l'algorithme se base sur le partitionnement d'arrêts.

Le principe général de cette approche consiste à transformer d'abord le graphe biparti G, en un autre graphe de lien G'. Dans lequel, les sommets désignent les numéros des arêtes de G et les arêtes représentent les liens d'incidence entre les arêtes de G. Puis, en appliquant l'algorithme génétique présenté dans la première partie, pour partitionner le graphe de lien obtenu en sous ensembles. Et enfin, en se basant sur les communautés du graphe G' trouvé, on déduit la structure en communautés chevauchantes des sommets pour le graphe G.

3.1. Construction du graphe de lien

Pour construire le graphe G' dont les sommets sont les numéros des arêtes de G et dont les arêtes sont les liens d'incidence entre les arêtes de G, on suit les étapes suivantes :

-construire la matrice A[n, m] du graphe biparti G, dans laquelle, les indices de colonne représentent l'ensemble des nœuds de la première famille et les indices de ligne représentent l'ensemble des nœuds de la deuxième famille ,

D'où : m désigne le nombre total des nœuds de la 1ère famille et n le nombre total des nœuds de la 2ère famille.

$$A[i,j] = \begin{cases} 1 & \text{si } e(i,j) \in E \\ 0 & \text{sinon} \end{cases}$$

Tel que :

e(i, j): Étant l'arête qui relie les sommets i et j de G.

- calculer le nombre d'arêtes du G(V,E) → |E| qui représentent par la suite le nombre de nœuds du graphe G'(V',E') → |V'|, en appliquant cette formule :

$$|E| = |V'| = m_a = \sum_i \sum_j A[i,j]$$

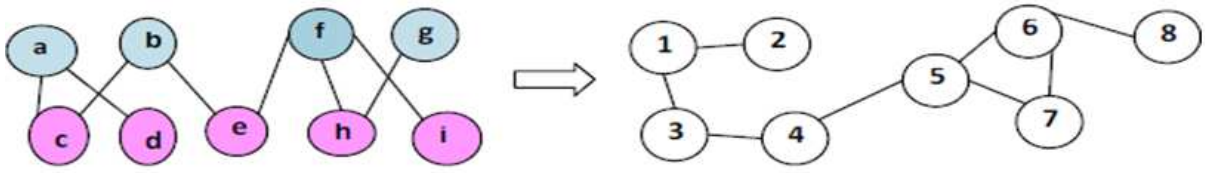
-Numéroter les arêtes du graphe G de k=1 ,… m_a dans la matrice A[n,m] de G :

$$A[i,j] = \begin{cases} k & \text{si } e(i,j) \in E \\ 0 & \text{sinon} \end{cases}$$

- Construire la matrice $A_E[m_a, m_a]$ qui est la matrice d'adjacence du graphe G´ telle que :

$$A_E[i,j] = \begin{cases} 1 & \text{si arete i est incidente à arete j} \\ 0 & \text{sinon} \end{cases}$$

La figure suivante montre un exemple de transformation d'un graphe de nœud au graphe de lien :



Graphe d'origine Graphe d'incidence des arêtes

Figure3.4. transformation d'un graphe de nœud au graphe de lien

3.2. Partitionnement du graphe de lien

Le partitionnement de E consiste à trouver un ensemble P de N_c clusters tel que $P= \{C_1, \dots C_{Nc}\}$ où les clusters Ci sont disjoints, c'est-à-dire, $(\forall i \neq j,\ C_i \cap C_j = \emptyset)$ et P recouvre E. Autrement dit on a : $\bigcup_i C_i = E$), pour i=1,… N_c.

On va donc appliquer directement l'algorithme génétique, pour partitionner le graphe G' (graphe de lien)

Dans ce cas le problème de chevauchement des arêtes ne se pose pas car une arête représente un lien, et ne peut par conséquent pas appartenir à plus d'une communauté de liens.

La figure3.5. montre un exemple de partitionnement de graphe de lien présenté dans la figure3.4. en deux communautés disjointes.

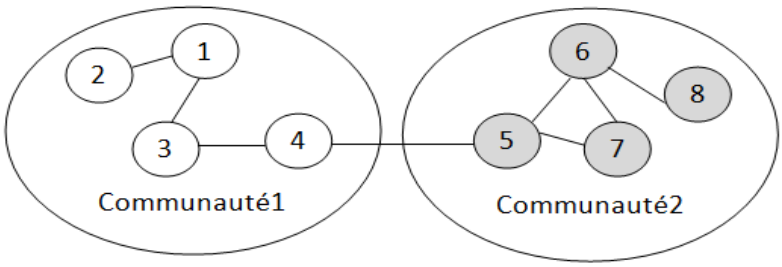


Figure3.5. exemple de partitionnement de graphe de lien en communautés disjointes

3.3. Déduction de communautés chevauchantes

Après avoir partitionné le graphe de lien G' en communauté disjointe, il ne reste qu'à déduire l'ensemble de communautés des nœuds pour le graphe G. La réalisation de ce processus se base sur les communautés trouvées en partitionnant l'ensemble E et sur la matrice adjacence de G après le nommage des liens.

Le nombre de communauté qu'on aura, sera égal au nombre de communautés de G', à la différence que dans ce cas les communautés obtenues sont des communautés chevauchantes.

Les communautés chevauchantes (pour le graphe de nœud), obtenant à partir des communautés disjointes (pour le graphe de lien), sont présenté dans la figure3.6.

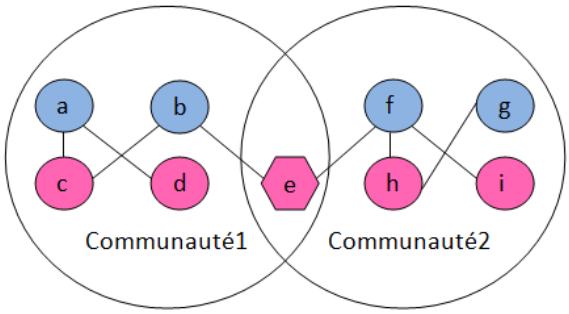


Figure3.6. communautés chevauchantes du graphe biparti (Présenté dans la figure3.4.)

Après avoir terminé de détecter les communautés chevauchantes il est indispensable de mesurer la qualité du partitionnement obtenu, en utilisant une autre fonction de qualité MQ_{Over}, utilisée pour mesurer la qualité du partitionnement de graphe en communautés chevauchantes. Cette fonction est destinée aux graphes mono-partis, mais dans notre travaille, nous allons l'adapter à un graphe biparti.

3.4. Mesure de qualité MQ_{Over}

3.4.1. Mesure de qualité MQ_{Over} pour une décomposition en groupes chevauchants

La mesure de qualité MQ a été généralisée par Bourqui et al [] pour une décomposition de l'ensemble des sommets d'un graphe en groupes chevauchants.

Considérons un graphe $G = (V, E)$ où V est l'ensemble des sommets et E l'ensemble des arêtes, une décomposition $Cm = \{Cm_1, Cm_2, \dots, Cm_k\}$ de l'ensemble des sommets V, tel que $\exists\ (i,j)\ tel\ que\ Cm_i \cap Cm_j \neq \emptyset\ \ et\ Cm = \cup\ \mathrm{Cm_i}\ (\mathrm{i} = 1, .. \mathrm{k})$.

La mesure de qualité MQ_{Over} est définie par :

$$\mathrm{MQ_{Over}} = \mathrm{MQ}^{+} - \mathrm{MQ}^{-}_{\mathrm{Over}}$$

Avec :

$$MQ^{+} = \frac{1}{k} \sum_{i=1}^{k} S(Cm_i, Cm_i)$$

Et :

$$\mathrm{MQ}^{-}_{\mathrm{Over}} = \frac{1}{\mathrm{k(k-1)}} \sum_{\mathrm{i}} \sum_{\mathrm{j} \neq \mathrm{i}} \mathrm{S}_{\mathrm{Over}} \left(\mathrm{Cm_i}, \mathrm{Cm_j}\right)$$

Où :

$$\mathrm{S}_{\mathrm{Over}}\left(\mathrm{Cm_i}, \mathrm{Cm_j}\right) = \frac{\mathrm{E}\left(\mathrm{Cm_i},\ \mathrm{Cm_j} \backslash (\mathrm{Cm_j} \cap \mathrm{Cm_i})\right)}{|\mathrm{Cm_i}| * |\mathrm{Cm_j}(\mathrm{Cm_j} \cap \mathrm{Cm_i})|}$$

$|\mathrm{Cm_j} \backslash (\mathrm{Cm_j} \cap \mathrm{Cm_i})|$ est le nombre de sommets du groupe $\mathrm{Cm_j}$ dépourvu des sommets qu'il a en commun avec $\mathrm{Cm_i}$.

$\mathrm{E}\left(\mathrm{Cm_i},\ \mathrm{Cm_j} \backslash (\mathrm{Cm_j} \cap \mathrm{Cm_i})\right)$, représente le nombre d'arêtes entre le groupe $\mathrm{Cm_i}$ et le groupe $\mathrm{Cm_j}$ diminué des nœuds qui sont en communs entre $\mathrm{Cm_i}$ et $\mathrm{Cm_j}$.

$$\mathrm{MQ}_{\mathrm{Over}} = \frac{1}{k} \sum_{i=1}^{k} \frac{E(C_i, C_i)}{|C_i| * |C_i|} \quad - \frac{1}{k(k-1)} \sum_{i} \sum_{j \neq i} \frac{\mathrm{E}\left(\mathrm{Cm_i}, \mathrm{Cm_j} \backslash (\mathrm{Cm_j} \cap \mathrm{Cm_i})\right)}{|\mathrm{Cm_i}| * |\mathrm{Cm_j} \backslash (\mathrm{Cm_j} \cap \mathrm{Cm_i})|}$$

3.4.2. Adaptation de la mesure $\mathrm{MQ}_{\mathrm{Over}}$ à un graphe bipartite

Soit G (Vr, Vb, E) un graphe bipartite non orienté, non pondéré. Vr et Vb représentent respectivement les ensembles de sommets rouges et bleus et E l'ensemble des arêtes.

Soient : Nr=|Vr|, Nb=|Vb|, Nr+Nb= n, |E|= m et A[n, n] la matrice d'adjacence du graphe, avec A[i,j]=1 si l'arête (i, j) Є E et 0 sinon.

La mesure de qualité d'une décomposition des sommets en groupes chevauchants dans un graphe biparti sera :

$$\mathrm{MQB}_{\mathrm{Over}} = \mathrm{MQB}^{+} - \mathrm{MBQ}^{-}_{\mathrm{Over}}$$

Soit une décomposition $Cm = \{Cm_1, Cm_2, \dots, Cm_k\}$ de l'ensemble des sommets Vr U Vb. On distinguera pour un groupe de sommets Cm_i, les deux sous ensembles des deux types de sommets par Cmr_i et Cmb_i avec $Cm_i = Cmr_i \cup Cmb_i$. Pour un groupe de sommets Cm_i le maximum d'arêtes qu'on peut avoir et $|Cmr_i| * |Cmb_i|$, c'est le cas d'un sous-graphe biparti complet, on a donc :

$$MQB^{+} = \frac{1}{k} \sum_{i=1}^{k} \frac{E(Cm_i,\ Cm_i)}{|Cmri\ | * |Cmbi|}$$

Pour MQ^{-}_{Over} on doit considérer le cas du sous-graphe complet contenant l'ensemble des sommets rouges et l'ensemble des sommets bleus des deux groupes de sommets Cm_i et Cm_j à la fois. Ce qui donne :

$$MQB^{-}_{Over} = \frac{1}{k(k-1)} \sum_i \sum_{j \neq i} \frac{E(Cm_i, Cm_j \backslash (Cm_j \cap Cm_i))}{|Cmr_i| * |Cmb_j \backslash (Cmb_j \cap Cmb_i)| + |Cmb_i| * |Cmr_j \backslash (Cmr_j \cap Cmr_i)|}$$

4. Conclusion

Apres avoir expliqué dans ce chapitre l'algorithme génétique proposé par « Mursel Tasgin » & « Haluk Bingol »afin de résoudre le problème de détection de communautés disjointes dans un graphe mono-parti, et son adaptation sur un graphe biparti, dans le but de résoudre le problème de détection de communautés chevauchantes. Nous avons aussi montré comment mesurer la qualité d'une décomposition en communautés chevauchantes. Nous présenterons dans le chapitre suivant la mise en œuvre de ces approches dans un environnement purement informatique afin de pouvoir tester leurs performances et leurs efficacités.

Chapitre4

Conception et implémentation

1. Introduction

Notre travail consiste à mettre en œuvre un algorithme génétique pour détecter des communautés disjointes dans un graphe monoparti, et l'utiliser par la suite pour détecter des communautés chevauchantes sur un graphe biparti.

Dans ce chapitre nous allons commencer par le codage des données à manipuler, à savoir la représentation du graphe, les structures de données employées et les paramètres d'entré de la l'algorithme. Une fois que ces données sont définies, nous présenterons les différents algorithmes implémentés pour réaliser notre logiciel.

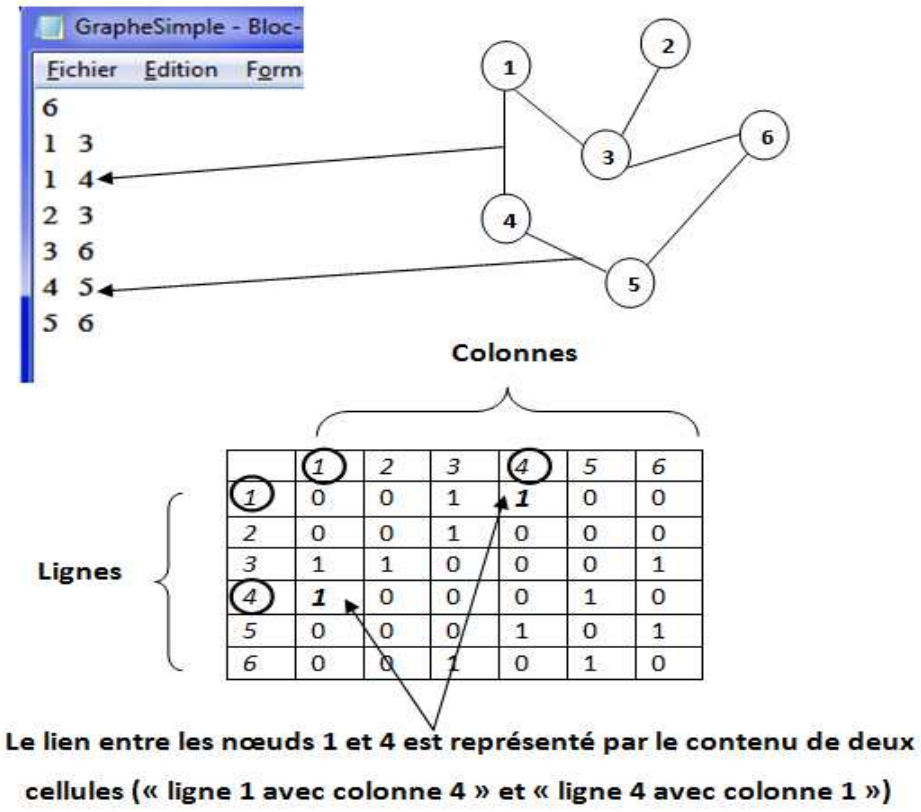
2. Codages des données

2.1. Format d'introduction des graphes

La structure du fichier que nous avons utilisé dans le cas du graphe monoparti est comme suite :

- La première ligne du fichier contient le nombre total de nœuds du graphe.
- les autres lignes contiennent les couples de nœuds qui se relient. C.à.d. qu'au lieu de mettre toute la matrice qui représente le graphe dans un fichier, on insère seulement le couple de nœuds qui sont reliés sur chaque linge. Le nombre total de couples représentés sur le fichier, indique le nombre de liens existant dans le graphe.

La figure4.1. Représente une capture d'écran d'un fichier sur lequel nous avons sauvegardé un graphe monoparti.



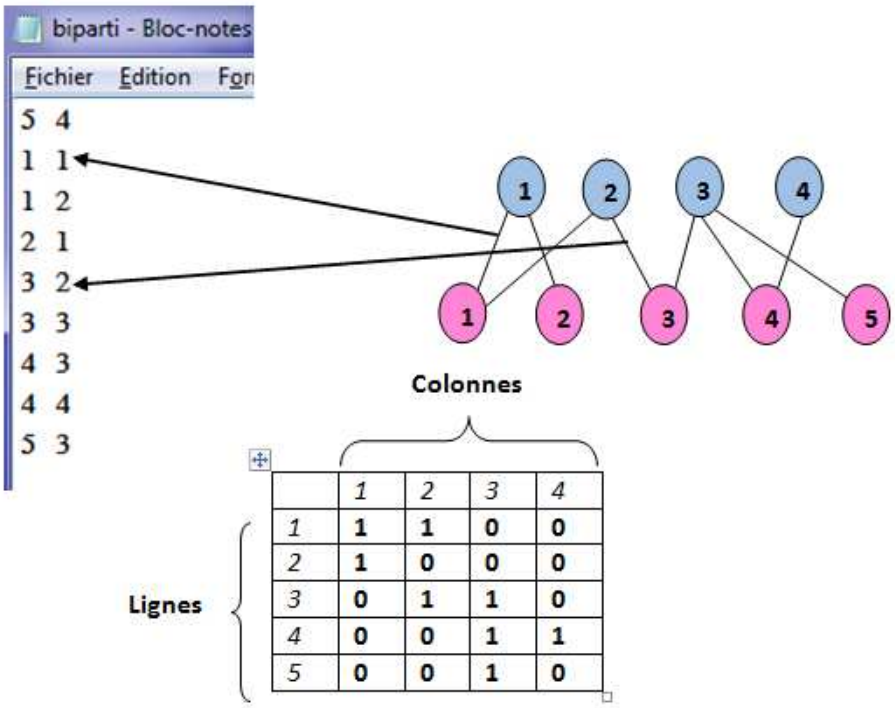
	1	2	3	4	5	6
1	0	0	1	**1**	0	0
2	0	0	1	0	0	0
3	1	1	0	0	0	1
4	**1**	0	0	0	1	0
5	0	0	0	1	0	1
6	0	0	1	0	1	0

Figure4.1 représentation d'un graphe dans un fichier

Dans le cas d'un graphe biparti, il est impossible d'avoir un lien entre deux nœuds du même type, à cet effet nous avons représenté les nœuds de la première famille par des indices colonnes, et les nœuds de la deuxième famille par des indices lignes. De même, les liens entre les nœuds sont représentés par le contenu des cellules. Cela évite de gaspiller un espace mémoire inutilement et facilite certaines opérations effectuées pour détecter les communautés chevauchantes et leurs évaluations.

La représentation de ce graphe sur le fichier est identique à celle présentée précédemment, à la différence que dans ce cas la première ligne du fichier contient deux valeurs : la première indique le nombre de nœuds de la première famille (nombre de ligne) et la deuxième indique le nombre de nœuds de la deuxième famille (nombre de colonne).

La figure suivante montre une représentation matricielle d'un graphe biparti.



Les colonnes représentent les nœuds bleus et les lignes les nœuds roses, le lien entre deux nœuds est représenté une seule fois.

Figure4.2.représentation d'un graphe biparti dans un fichier

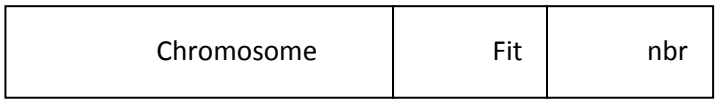
2.2. Les structures de données

Population : est constituée d'un ensemble d'individus manipulés par l'algorithme génétique, et comme on ne peut pas fixer, auparavant, la taille de la population, nous avons employé un tableau dynamique pour le codage, dans lequel on peut ajouter ou supprimer des éléments.

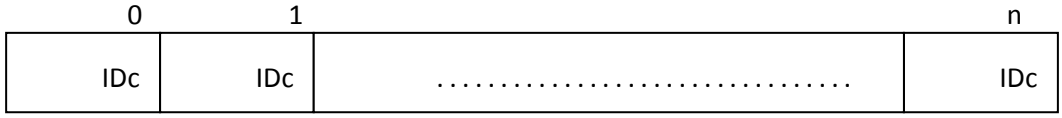
In	In	In	.....	Indi

Individu : afin de faciliter les différentes opérations effectuées par l'algorithme génétique et éviter de recalculer à chaque fois la fitness des chromosomes, nous avons pensé à sauvegarder la fitness et le nombre de communauté de chaque chromosome

créé, dans un objet nommé individu. On trouve alors, au sein de chaque individu : un chromosome, sa fitness et son nombre de communautés.

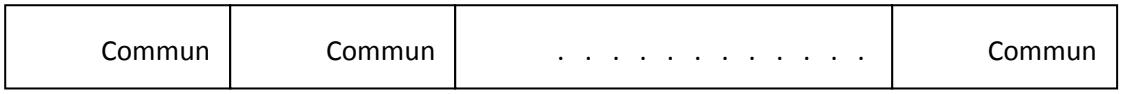


Chromosome : est un tableau d'entiers de taille fixe, dans lequel les indices désignent les nœuds du graphe et le contenu des cellules désignent l'identifiant de communauté (***IDcomm***) pour chaque nœud.



Groupe de Communautés: est un tableau dynamique employé pour structurer chaque chromosome sous forme de communautés.

Cette représentation en communautés facilite le calcule de la fitness, l'opération cleanUP et la génération du fichier qui permet de visualiser le graphe après le partitionnement.



Communauté : est un objet composé de deux éléments : le premier est un entier pour indiquer l'IDcomm pour chaque communauté et le deuxième est un tableau dynamique de type entier (ListNoeud) qui dispose des nœuds appartenant à cette communauté.

Chaque nœud apparait une seule fois dans un seul groupe de communauté, ça veut dire qu'on ne peut pas trouver le même nœud dans deux listNoeud différentes (car on détecte des communautés disjointes).

Groupe de communautés Chevauchantes : dans le cas du partitionnement de graphe biparti en communautés chevauchantes, on utilise le groupe de communautés pour

représenter les différentes communautés pour le partitionnement du graphe de lien, puis les communautés chevauchantes seront représentées par un autre tableau dynamique qui ressemble au premier à la différence que dans ce cas, certains nœuds peuvent apparaître dans plus d'une communauté, il s'agit des nœuds chevauchants.

2.3. Paramètres

Les paramètres sont des variables d'entrée du programme, l'efficacité de l'algorithme génétique en dépend. Le réglage de ces paramètres est très important afin d'avoir une solution qui s'approche de l'optimum.

Nomfichier : est une variable de type chaine de caractère utilisée pour désigner le nom du fichier (.txt) sur lequel le graphe est sauvegardé.

Nbrnoeud est un terme employé pour désigner le nombre total de nœuds du graphe

Tailpop: est un terme qui désigne la taille de la population (égale au nombre de chromosome)

Nbrcrois : variable entière utilisée pour indiquer le nombre de croisements à effectuer pour chaque génération.

Nbrpere : variable entière utilisée pour indiquer le nombre de chromosomes à sélectionner pour le croisement

Parmutation : variable réelle entre 0 et 1 qui désigne la probabilité de mutation

parmCV : variable réelle entre 0 et 1 qui désigne la probabilité de CleanUP

nbrselectCV : valeur entière qui désigne le nombre de nœuds à sélectionner pour effectuer l'opération CleanUP

nbrGeneselectCreat : nombre de nœuds à sélectionner lors de la création de chaque chromosome pour la population initial.

nbrIDcommselectCros : nombre de communautés à sélectionner lors du croisement

3. Les algorithmes

Après avoir présenté les structures de données employées afin de coder les variables indispensables qui sont employées dans l'algorithme, nous allons aborder dans cette partie les différents algorithmes implémentés pour la réalisation de notre logiciel.

Il faut juste signaler qu' à chaque fois qu'on crée un nouveau chromosome, ce dernier sera évalué directement, par la procédure qui calcule la fitness, avant de l'insérer dans la population. La fitness de chaque chromosome est donc calculée une seule fois.

Les procédures principales que nous avons implémentées et les interactions entre elles sont représentées dans l'organigramme ci-dessous.

L'organigramme est constitué de 3 parties :

(1) Préparation : lire les paramètres et générer les matrices.

(2) Exécution de l'algorithme génétique

(3) Affichage des résultats obtenus

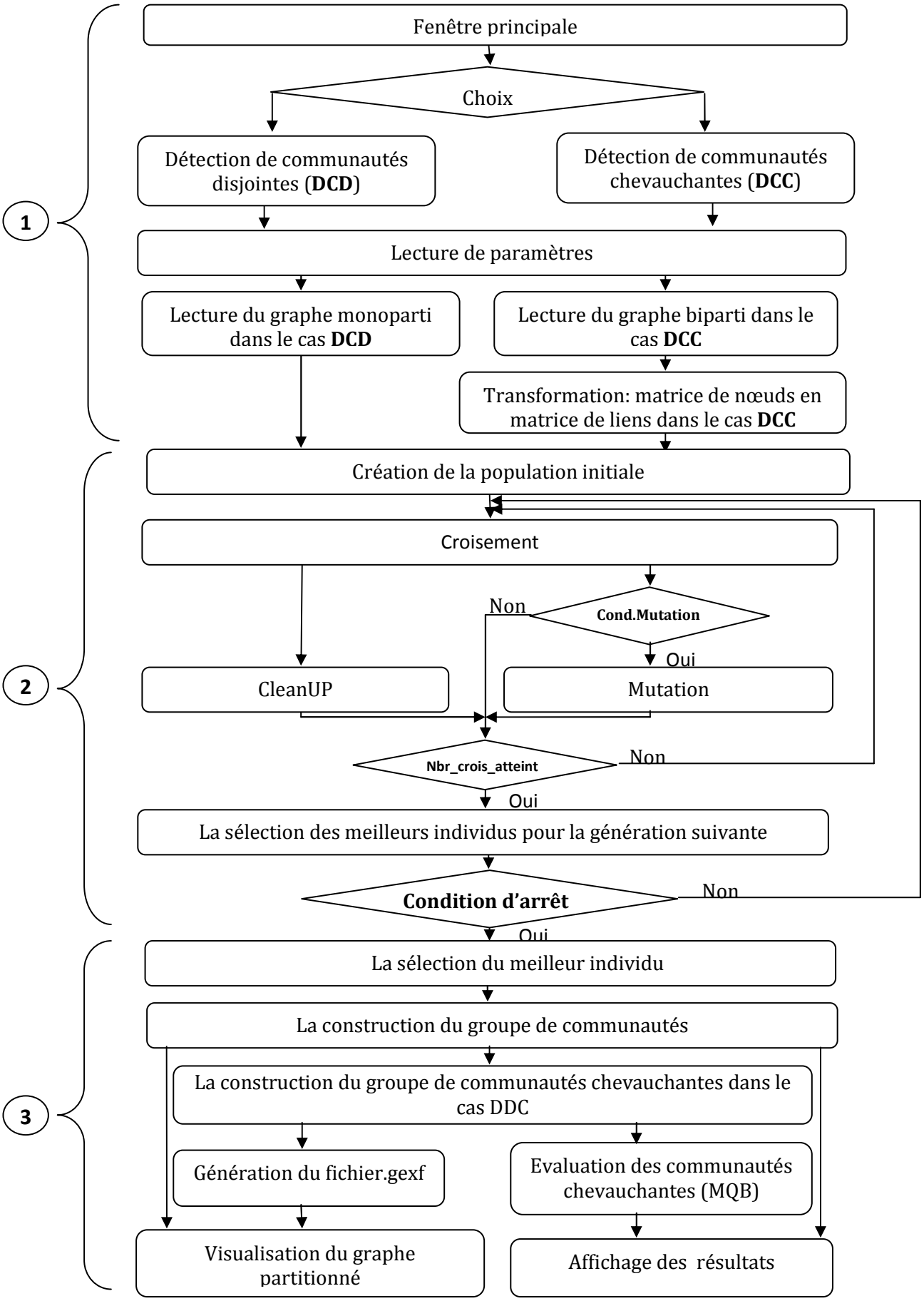


Figure4.3. Organigramme de l'application

3.1. Détection de communautés disjointes

C'est la procédure principale pour la détection de communautés disjointes, son rôle est de répéter le cycle génétique plusieurs fois jusqu'à ce que la condition d'arrêt soit satisfiable. Avant de décrire l'algorithme général, nous allons d'abord définir les variables employées :

Nbrgeneration : est une variable de type entier, initialisée à 0, elle permet de compter le nombre de générations globale.

Condition d'arrêt : le critère d'arrêt utilisé est constitué de deux conditions :

- Dépasser le nombre de génération fixé par l'utilisateur.
- Suivre le meilleur chromosome de la population : s'il ne s'est pas amélioré les x dernières générations, l'algorithme s'arrête.

Algorithme1

Début
LireParamétre()
LectureGraphe()
Population←*vide*
CreationPopulationInitial()
Tant que (*non*(*condition d'arrêt*)) faire
Début
Croisement()
Sélectionner les meilleurs individus pour la génération suivante
nbrgeneration + +
fin
construireGroupeCommunaute(*meilleur_individu*)
Afficher grouprcommunaute(*meilleur_individu*)
GénérationFichierDisjoint. *gexf* (*meilleur_individu*)
LancerGephiDisjoint
Fin

3.2. Détection de communautés chevauchantes

Cette procédure est considérée comme étant la procédure principale pour l'algorithme de détection de communautés chevauchantes. On prend en considération les mêmes variables définies auparavant.

Algorithme2

Début
LireParamétre()
LectureGraphe()
CreatMatriceLien()
Population←*vide*
CreationPopulationInitial()
Tant que (*non*(*condition d'arrêt*)) faire
Début
Croisement()
Sélectionner les meilleurs individus
nbrgeneration + +
fin
construireGroupeCommunaute (*meilleur_individu*)
Afficher grouprcommunaute (*meilleur_individu*)
GénérationFichierChevauchante. *gexf* (*meilleur_individu*)
LancerGephiChevauchante (*meilleur_individu*)
Fin

3.3. Lecture du graphe

La lecture du graphe à partir du fichier texte et la construction de la matrice correspondante, n'est pas le même dans les deux cas, nous avons alors employé deux procédures pour la lecture du graphe. L'une est appelée par la procédure de détection de communautés disjointes et l'autre par la procédure de détection de communautés chevauchantes.

❖ Procédure de la lecture du graphe pour la détection de communauté disjointe

Algorithme3

début

ouvrir en ecriture le fichier(nomfichier)

lire la première ligne

NbrNoeud ← la valeur de la ligne

création de la matrice[NbrNoeud][NbrNoeud]

tant que(ce n'estpas la fin de fichier)faire

début

lire la ligne suivante

ligne ← le premier élément de la ligne

colonne ← le deuxième élément de la ligne

matrice[ligne − 1][colonne − 1] ← 1

matrice[colonne − 1][ligne − 1] ← 1

fin

fin

❖ Procédure de la lecture du graphe pour la détection de communautés chevauchantes

Algorithme4

début

ouvrir en ecriture le fichier(nomfichier)

lire la première ligne

NbrLigne ←le premier élément de la ligne

NbrColonne ←le deuxième élément de la ligne

création de la matrice[NbrLigne][NbrColonne]

tant que(ce n'estpas la fin du fichier)faire

début

lire la ligne suivante

ligne ←le premier élément de la ligne

colonne ←le deuxième élément de la ligne

matrice[ligne − 1][colonne − 1] ←1

fin

fin

3.4. Transformation de la matrice de nœuds en matrice de liens

Après avoir crée la matrice de nœuds, dans le cas de détection de communauté chevauchante, il est indispensable de transformer cette matrice vers une matrice de liens qui sera employée pour partitionner les arêtes en communautés disjointes, et de déduire vers la fin les communautés chevauchantes du graphe biparti.

Algorithme5

Début
$pour(i\ allont\ de\ 0\ \ à\ \ \ (nbrLigne-1))faire$
$pour(\ j\ allont\ de\ \ 0\ \ à\ (nbrColonne-1))faire$
$si\ (matriceNoeud[i][j]=1)alors$
$début$
$matriceNoeud[i][j] \leftarrow numArete$
$numArete++$
fin
$nbrArete \leftarrow numArete-1$
$creer\ \ matriceLien[nbrArete][nbrArete]$

$pour(ligne\ allons\ de\ 0\ à\ (nbrLigne-1))faire$

$pour(colonne\ allons\ de\ 0\ à\ (nbrColonne-1))faire$

$début$

$numarc1 \leftarrow matriceNoeud[ligne][colonne]$

$si(numarc1 \neq 0)alors$

$début$

$pour(c\ allons\ de\ (colonne+1)à\ nbrColonne-1)faire$

$si(matriceNoeud[ligne][c] \neq 0)alors$

$début$

$numarc2 \leftarrow matriceNoeud[ligne][c]$

$matriceLien[numarc1-1][numarc2-1] \leftarrow 1$

$matrice\ matriceLien\ [numarc2-1][numarc1-1] \leftarrow 1$

fin

$pour(l\ allons\ de\ \ (ligne+1)à\ nbrLigne-1)faire$

$si(matriceNoeud\ [l][colonne] \neq 0)alors$

$début$

$numarc2 \leftarrow matriceNoeud[l][colonne]$

$matriceLien[numarc1-1][numarc2-1] \leftarrow 1$

$matriceLien[numarc2-1][numarc1-1] \leftarrow 1$

fin

fin

fin

fin

3.5. Evaluation des chromosomes (calcule de la fitness)

Cette procédure est appelée à chaque fois qu'un nouveau chromosome est crée, et avant de calculer la fitness, faire appel à la procédure de normalisation pour former les groupes de communautés du chromosome, qui seront employés par la suite dans la procédure.

Algorithme6

$début$

$normalisation(individut)$

$pour(i\ allont\ de\ 0\ \text{à}\ nbrnoeud-1)faire$

$pour(j\ allont\ de\ 0\ \text{à}\ nbrnoeud-1)faire$

$m \leftarrow m + matrice[i][j]$

$m = m/2$

$pour(i\ allont\ de\ 0\ \text{à}\ taille.groupcommunaute-1)faire$

$début$

$E \leftarrow 0$

$communaute1 \leftarrow creerNouvelCommunaute$

$communaute1 \leftarrow groupcommunaute[i]$

$pour(j\ allont\ de\ 0\ \text{à}\ (taille.communaute1.listnoeud-1))faire$

$pour(k\ allont\ de\ 0\ \text{à}\ (taille.communaute1.listnoeud-1)faire$

$E \leftarrow E$

$+ (matrice[communaute1.listnoeud[j]][communaute1.listnoeud[k]])$

$A \leftarrow 0$

$pour(\ j\ allont\ de\ 0\ \text{à}\ (taille.communaute1.listnoeud-1))faire$

$pour(k\ allont\ de\ 0\ \text{à}\ nbrnoeud-1)faire$

$A \leftarrow A + matrice[communaute1.listnoeud.[j]][k]$

$A \leftarrow A/(m*2)$

$E \leftarrow E/(m*2)$

$Q \leftarrow Q + (E - A*A)$

fin

fin

3.6. Normalisation (Former le groupe de communautés)

Cette procédure a pour objectif d'ordonner les nœuds d'un chromosome par communautés. A chaque fois qu'il prend un nœud du chromosome, il parcoure toutes les communautés du « groupe de communautés », s'il trouve une communauté qui a le même IDcomm que lui, le nœud sera ajouté à la liste de cette communauté et il sort de la boucle pour passer au nœud suivant.

Dans le cas où le nœud ne trouve pas la communauté qui lui correspond, une nouvelle communauté qui va contenir ce nœud, sera crée.

Algorithme7

$début$

$groupcommunaute \leftarrow vide$

$communaute \leftarrow nouvelCommunaute$

$pour(i\ allont\ de\ 0 \quad à \quad nbrnoeud - 1) faire$

$début$

$j \leftarrow 0$

$tant\ que(j < taille.groupcommunaute) faire$

$début$

$communaute \leftarrow groupcommunaute[j]$

$si(individu.chromosome[i] = communaute.idcommunaute) alors$

$début$

$Inserer\ le\ noeud\ i\ dans\ listNoeud\ de\ la\ Communaute$

$sortir\ de\ la\ boucle\ tant\ que$

fin

$j++$

fin

$si(j \geq taille.groupcommunaute) alors$

$début$

$communaute1 \leftarrow creerNouvelCommunaute$

$communaute1.idcommunaute \leftarrow individu.chromosome[i]$

$Inserer\ le\ noeud\ i\ dans\ listNoeud\ de\ la\ communaute1$

$Inserer\ la\ communauté1\ dans\ GroupCommunaute$

fin

fin

$individu.nbrcom \leftarrow taille.groupcommunaute$

fin

3.7. Création de la population initiale

Cette procédure est appelée une seule fois au début de l'exécution de l'algorithme génétique, afin de créer la population initiale des individus qui seront évolués par la suite grâce aux différents opérateurs génétiques. Les chromosomes créés seront insérés dans la population par rapport à leurs valeurs de fitness, ce qui permet d'avoir, à la fin de cette procédure, une population triée et facile à employer par la suite.

Algorithme8

$début$

$pour(nbr\ allont\ de\ 0 \quad à \quad taillepopulation - 1) faire$

$début$

$Individu \leftarrow creerNouveauIndividus$

$Individu.chromosome \leftarrow creerNouveauChromosome$

$pour(i \leftarrow 0 \quad à \quad nbrNœud) faire$

$individu.Chromosome[i] \leftarrow valeurAléatoir\ entre\ 0\ et\ nbrNoeud$

$pour(nbr\ allont\ de\ 0 \quad à \quad nbrGeneselectCreat - 1) faire$

$début$

$i \leftarrow valeurAléatoir\ entre\ 0\ et\ nbrNoeud$

$pour(j\ allont\ de\ 0 \quad à \quad nbrnoeud - 1)\ faire$

$si(matrice[i][j] = 1)\ alors$

$individu.chromosome[j] \leftarrow individu.chromosome[i]$

fin

$individu.fitness \leftarrow calculFitness$

$individu\ sera\ inseré\ dans\ la\ population$

fin

fin

3.8. Croisement

L'opérateur de croisement s'applique plusieurs fois sur les X meilleur chromosomes

A chaque fois qu'un nouveau fils est créé, il fait appel d'abord à la procédure CleanUP puis à la procédure de mutation si la fitness du fils est supérieur à la probabilité de mutation.

- Pourquoi CleanUP d'abord ?

L'opération CleanUP, comme le calcule de la fitness, ont besoin d'avoir le groupe de communautés du chromosome. Et comme la fitness du fils est calculé après sa création, ça implique que le groupe de communautés est déjà formée, et si on commence d'abord par la mutation ce groupe sera écrasé lors du calcule de la fitness du nouveau chromosome produit par la mutation. Donc ils vont mieux commencer par CleanUP pour éviter de rechercher le groupe de communautés afin de gagner du temps.

Algorithme9

$Début$

$Pour\ (r\ allont\ de\ 0\ \ à\ \ nbrcrois - 1) faire$

début

$Individusource \leftarrow individu\ sélectionné\ aléatoirement$

$Individudestination \leftarrow individu\ sélectionné\ aléatoirement$

$Individufils \leftarrow creerNouveauIndividu$

$Individufils.chromosome \leftarrow creerNouveauChromosome$

$pour(nbr\ allont\ de\ 0\ \ à\ \ nbrIDcommselectCros - 1) faire$

$début$

$numcom \leftarrow valeuraléatoire$
$pour(h \leftarrow 0 \quad à \quad nbrnoeud - 1) faire$
$si(Individusource.chromosome[h] = numcom) alors$
$Individufils.chromosome[h] \leftarrow source.chromosome[h]$
$sinon$
$Individufils.chromosome[h] \leftarrow destin.chromosome[h]$
fin
$Individufils.fitness \leftarrow calculfitness(Individufils)$
$Individufils\ sera\ inseré\ dans\ la\ population$
$cleanUp(Individufils)$
$si(fils.fitness > parmutation)$ alors
$mutation(Individufils)$
fin
fin

3.9. Mutation

De même, après la création d'un nouveau chromosome par l'operateur de mutation, il sera évalué et inséré dans la population.

Algorithme10

$début$
$IndividuMuté \leftarrow CréerNouveauIndividu$
$IndividuMuté.chromosome \leftarrow nouveauChromosome$
$pour(j\ allont\ de\ 0 \quad à \quad nbrnoeud - 1) faire$
$IndividuMuté.chromosome[j] \leftarrow individufils.chromosome[j]$
$nœud \leftarrow valeur\ aléatoire\ entre\ 0\ et\ nbrnoeud$
$IndividuMuté.chromosome[noeud] \leftarrow IDcomm\ séléctionné\ aléatoirement$
$IndividuMuté.fitness \leftarrow calculfitness(IndividuMuté)$
$IndividuMuté\ sera\ inseré\ dans\ la\ population$
fin

3.10. CleanUP

Algorithme11

$début$

$pour(h\ allont\ de 1\ \text{à}\ nbrselectCV - 1) faire$

$début$

$node \leftarrow un\ noeud\ sélectionné\ au\ hasard$

$nbrVoisin \leftarrow calculeNombrevoisinTotal(node)$

$nbrVoisinCom \leftarrow calculeNombrevoisinDansCommunaut(node)$

$si(nbrVoisin \neq 0) alors$

$CV \leftarrow (nbrVoisin - nbrVoisinCom)/nbrVoisin$

$si(CV > parmCV) alors$

$début$

$newcommunaute \leftarrow creerNouvelcommunaut$

$newcommunaute.idcommunaute \leftarrow nouvelIDcommn$

$newcommunaute.listNoeud \leftarrow creerNouvelListNoeud$

$le\ noeud\ et\ ses\ voisins\ seront\ insérés\ dans\ la\ liste\ de\ la\ nouvelle$
$communauté,\ en\ les\ supprimant\ de\ leurs\ communautés\ anciennes$

$la\ nouvelle\ communauté\ sera\ inséré\ \ dans\ GroupeCommunaute$

$un\ nouveau\ \ chromosome\ sera\ créé\ \text{à}\ partir\ de\ groupe\ de\ communauté$

$calculer\ la\ fitness\ de\ ce\ chromosome$

$le\ nouveau\ individus\ (chromosome + fitness$
$+ nbrcommunauté) sera\ inseré\ dans\ la\ population$

fin

fin

fin

3.11. Déduction du groupe de communautés chevauchantes

A la fin de la détection de communautés disjointes pour le graphe de lien, le programme va appeler cette procédure afin de déduire les communautés chevauchantes pour le graphe de nœuds de départ.

Algorithme12

```
Début
  pour(i allons de 0  à  taille.groupeCommunaute)faire
   début
     communaute←groupeCommunaute[i]
     pour(j allons de 0  jusqu'à  taille.communaute.listnoeud − 1)faire
      début
        objet←creerNouveauObjet
        objet.edge←communaute.listnoeud[j] + 1
        nomArête←communaute.listnoeud[j] + 1
         la matrice de noeud, aprés le nomage des arêtes, sera parcourir
         jusqu'à trouver une valeur égale au nomArête
         objet.source←colonne corespond au nomArête
         objet.target←nbrColonne + ligne correspond au nomArête
         les communautés de groupe de communauté chevauchante, seront
              parcourir jusqu'à trouver celle qui correspond a ce noeud
           le noeud sera ajouté à la liste de cette communauté
         si le noeud ne trouve pas la communauté qui lui corespond
           une nouvelle communauté sera crée
            idcommunaute de la nouvelle communauté←communaute.idcommunaute
           le noeud sera inséré dans la liste de cette nouvelle communauté
        j + +
      fin
    i + +
   fin
fin
```

3.12. Evaluation de qualité de communautés chevauchantes

Pour évaluer la qualité du partitionnement en communautés chevauchantes, nous utilisons la métrique MQBover présenté dans le chapitre précédant. A cet effet, nous avons programmé la procédure dont : « Algorithme13 ».

Nous donnons d'abord des explications sur les variables utilisées et nous enchainons sur la description de l'algorithme.

taille: la taille du groupe de communautés chevauchantes

nbrLienInterne : est le nombre de liens interne pour une communauté quelconque

nbrLienExterne : est le nombre de liens entre deux communautés, c.à.d. que chaque lien doit avoir obligatoirement une extrémité dans la première communauté et l'autre extrémité dans la deuxième communauté.

nbrRouge : indique le nombre de nœuds de la première famille

nbrBleu : indique le nombre de nœuds de la deuxième famille

Algorithme13

Début

Initialiser MQBover, MQBplus et MQBmoins à zéro
Pour(i allons de 0 jusqu'à taille − 1)faire
Début
Initialiser nbrBleu et nbrRouge à zéro
Initialiser nbrLienInterne à zéro
Pour chaque communauté du groupeCommunauteChevauchante faire
Pour chaque nœud de cette communauté faire
début
Si le nœud appartient aux nœuds bleus alors
nbrBleu + +
sinon
nbrRouge + +
pour chaque voisin du nœud faire

si le voisin appartient à la même communauté que nœud alors

nbrLienInterne + + ;

fin

nbrLienInterne ← nbrLienInterne/2

MQBplus ← MQBplus + (nbrLienInterne/nbrBleu ∗ nbrRouge)

Fin

MQBplus ← MQBplus/nbr de communauté

Pour chaque communauté1 de groupeCommunauteChevauchante faire

Pour chaque communauté2 de groupeCommunauteChevauchante faire

Si(communaute1.IDcomm ! = communaute2.IDcomm)alors

Début

Initialiser nbrBleu1, nbrRouge1, nbrBleu2 et nbrRouge2 à zéro

Initialiser nbrLienExterne à zéro

Éliminer de la communauté2 tous les nœuds en commun entre communauté1 et communauté2 et insérer ces nœuds dans listCommun

Pour chaque nœud1 de communaute1 faire

Pour chaque nœud2 de communaute2 faire

Si noeud1 est un voisin de noeud2 alors

nbrLienExterne + +

calculer le nombre de nœud bleu de la communauté1 : nbrNoeudBleu1

calculer le nombre de nœud rouge de la communauté1 : nbrNoeudRouge1

calculer le nombre de nœud bleu de la communauté2 : nbrNoeudBleu2

calculer le nombre de nœud bleu de la communauté2 : nbrNoeudRouge2

MQBmoins ← MQBmoins + (nbrArcExterne/(nbrRouge1 ∗ nbrBleu2 + nbrRouge2 ∗ nbrBleu1))

Réinsérer dans communauté2 tous les nœuds ce trouvant dans listCommun

vider listCommun

fin

MQBmoins ← MQBmoins /(taille) (taille − 1)

MQBover ← MQBplus − MQBmoins

fin

3.13. Affichage des résultats

A la fin du cycle génétique, et après le partitionnement du graphe en communautés il ne reste qu'à afficher les résultats de l'algorithme.

Construction du groupe de communautés pour le meilleur chromosome

Affichage : *la fitness*, *nombre de communautés de ce chromosome*

Affichage du nombre total de générations et le temps d'exécution

de l'algorithme génétique, *sans prendre en considération le temps*

de chargement des données du graphe de puis le fichier.

Affichage des communautés résultantes.

Les mêmes étapes sont appliquées pour la détection de communautés chevauchantes à la différence que dans ce cas, il faut d'abord déduire les groupes de communautés chevauchantes, pour les afficher par la suite, et d'afficher MQBover qui indique la qualité du partitionnement du graphe biparti en communautés chevauchantes.

3.14. Génération du fichier.gexf

En plus de l'affichage des résultats du partitionnement, il est plus pratique de visualiser graphiquement les communautés résultantes, à cet effet, nous avons adopté un logiciel de visualisation qui reçoit en entré un fichier.gexf au forma xml pour représenter le graphe. Ce fichier se compose de deux parties : l'entête et le corps.

L'entête représente la partie commune de tous les documents GEXF, dans lequel est définit la version du XML et le type de codage employé, ainsi que l'élément racine du document gexf ; qui appartient à l'espace de nom : *http://www.gexf.net/1.1draft.*

Le corps est une partie obligatoire, qui consiste à définir la topologie du graphe, on déclare donc dans cette partie les balises suivantes : graphe, node, edge;

Ce type de fichier possède plusieurs avantages: en plus de sa simplicité, il permet de définir le type de graphe employé: orientés, non-orientés, statique ou dynamique ;

d'attribuer nom, couleur, taille, position à chacun des nœuds et d'attribuer nom, couleur, poids, pour chacune des arêtes.

La manière de générer ce fichier n'est pas identique pour les deux sortes de communautés (disjointes et chevauchantes), pour cela nous avons employé 3 procédures de génération de fichier (.gexf), la première sera générée pour un graphe quelconque avant de le partitionner en communautés, la deuxième pour les communautés disjointes et la dernière pour les communautés chevauchantes.

- **Procédure1 : Visualisation avant le partitionnement**

Dans ce cas, aucune information spéciale n'est attribuée aux nœuds, ni aux arêtes, les seules données à décrire sont leurs noms.

Algorithme14

Début

Ouvrire en ecriture le fichier("graphe.gexf")
ecrir dans le fichier(" <? xml version =\"1.0\" encoding =\"UTF − 8\" ? > ")
ecrir dans le fichier(" < gexf xmlns
=\"http://www.gexf.net/1.2draft\" xmlns: viz
=\"http://www.gexf.net/1.2draft/viz\" > ")
ecrir dans le fichier("\t < graph defaultedgetype = \"undirected\" > ")
ecrir dans le fichier("\t\t < nodes > ")
pour(i allons de 0 à *nbrnoeud − 1)faire*
ecrir dans le fichier("\t\t\t < node id =\"" + i + "\"label =\"" + i + "\"/
> ")
ecrir dans le fichier("\t\t </nodes > ")
ecrir dans le fichier("\t\t < edges > ")
idedge ←0
pour(i allons de 0 à *nbrnoeud − 1)faire*
pour(j allons de 0 à *nbrnoeud − 1)faire*
si il y a un lien entre i et j alors
début

("\t\t\t < edge id =\"" + idedge + "\" source
=\"" + i + "\" target =\"" + j + "\"/> ")
idedge + +
fin
ecrir dans le fichier("\t\t </edges > ")
ecrir dans le fichier("\t </graph > ")
ecrir dans le fichier(" </gexf > ")
fermer le fichier
fin

- **Procédure2 : Visualisation après partitionnement pour les communautés disjointes**

Avec cet algorithme, nous attribuons une même taille et une même couleur à tous les nœuds qui appartiennent à une communauté donné afin de distinguer les différentes communautés et les nœuds qui les composent

Algorithme15

Début
normalisation(individu)

ouvrir en écriture(fichier, "graphComDisj. gexf"))
ecrir dans le fichier(" <? xml version =\"1.0\" encoding =\"UTF − 8\" ? > ")
ecrir dans le fichier (" < gexf xmlns
=\"http://www. gexf. net/1.2draft\" xmlns: viz
=\"http://www. gexf. net/1.2draft/viz\" > ")
ecrir dans le fichier ("\t < graph defaultedgetype = \"undirected\" > ")
ecrir dans le fichier("\t\t < nodes > ")
pour(i allons de 0 à taille. groupcommunaute − 1)faire
début
attribuer la même couleur et la même taille pour tous
les noeud de la même communauté

ecrir dans le fichier ("\t\t\t < *node id* =\"" + *noeud* + "\" *label*
=\"" + *noeud* + "\" > ")
ecrir dans le fichier (" < *viz*: *color b* =\"" + *couleur*1 + "\" *g*
=\"" + *couleur*2 + "\" *r* =\"" + *couleur*3 + "\"/>
< *viz*: *size value* =\"" + *taille* + "\"/></*node* > ")
fin
fin
ecrir dans le fichier("\t\t </*nodes* > ")
ecrir dans le fichier("\t\t < *edges* > ")
int idedge ←0
associer pour chaque arête du graphe ses deux noeuds d'extrimités
ecrir dans le fichier ("\t\t\t < *edge id* =\"" + *idedge* + "\" *source* =
\"" + *noeudSource* + "\" *target* =\"" + *noeudDistination* + "\"/> ")
ecrir dans le fichier ("\t\t </*edges* > ")
ecrir dans le fichier ("\t </*graph* > ")
ecrir dans le fichier (" </*gexf* > ")
fermer le fichier
Fin

- **Procédure3 : Visualisation après partitionnement pour les communautés chevauchantes**

Dans le cas des communautés chevauchantes, c'est les arêtes de même communauté qui ont la même couleur, par contre les nœuds sont coloriés par rapport à leurs familles excepté les nœuds qui se chevauchent (auront une couleur différente) et nous avons attribué une même taille pour tous les nœuds d'une même communauté.

Algorithme16

Début

normalisation(*individu*)

ouvrir en écriture(*fichier*, "*graphComDisj*. *gexf*"))

ecrir dans le fichier(" <? *xml version* =\"1.0\" *encoding* =\"*UTF* − 8\" ? > ")

ecrir dans le fichier (" < *gexf xmlns*

=\"*http*://*www*. *gexf*. *net*/1.2*draft*\" *xmlns*: *viz*

=\"*http*://*www*. *gexf*. *net*/1.2*draft*/*viz*\" > ")

ecrir dans le fichier ("\t < *graph defaultedgetype* = \"*undirected*\" > ")

ecrir dans le fichier("\t\t < *nodes* > ")

objet communaute = *new objet*()

pour(*i allons de* 0 à *taille*. *groupCommunauteChevauchante* − 1)*faire*

début

affecter la même couleur et la même taille pour tous

les noeuds de la même communauté

ecrir dans le fichier ("\t\t\t < *node id* =\"" + *noeud* + "\" *label*

=\"" + *noeud* + "\" > ")

ecrir dans le fichier (" < *viz*: *color b* =\"" + *couleur*1 + "\" *g*

=\"" + *couleur*2 + "\" *r* =\"" + *couleur*3 + "\"/>

< *viz*: *size value* =\"" + *taille* + "\"/></*node* > ")

les noeuds, *qui se chevauchent*, *auront une couleur différente*

fin

fin

ecrir dans le fichier("\t\t </*nodes* > ")

ecrir dans le fichier("\t\t < *edges* > ")

int idedge ←0

associer pour chaque arête du graphe ses deux noeuds d'extrimités

ecrir dans le fichier ("\t\t\t < *edge id* =\"" + *idedge* + "\" *source* =

\"" + *noeudSource* + "\" *target* =\"" + *noeudDistination* + "\"/> ")

ecrir dans le fichier ("\t\t </*edges* > ")

ecrir dans le fichier ("\t </*graph* > ")

ecrir dans le fichier (" </*gexf* > ")

fermer le fichier

Fin

4. Conclusion

Après avoir présenté les différentes structures de données employées et les paramètres d'entrée de l'algorithme, nous avons schématisé l'architecture générale avec un organigramme, puis nous avons détaillé les différents algorithmes employés dans le but de réaliser notre logiciel.

Nous présenterons dans le chapitre suivant en plus, de l'environnement de développement, les résultats des tests effectués sur l'algorithme génétique afin d'évaluer ses performances.

Chapitre5

Mise en œuvre et expérimentation

1. Introduction

Apres avoir défini le problème de détection de communautés dans les graphes complexes ainsi que les deux méthodes de résolution que nous avons implémenté. Nous présentons dans ce chapitre la mise en œuvre de ces méthodes. La première section décrit la plate forme de développement utilisée. Les sections suivantes portent sur l'architecture générale de notre application ainsi que les résultats expérimentaux que nous avons effectués.

2. Environnement de développement

Pour l'implémentation des méthodes vues dans le chapitre précédant nous avons opté pour le langage java comme outil de développement dans un environnement Windows, et les résultats du partitionnement du graphe obtenus par notre application sont visualisés à l'aide du logiciel Gephi (Gephi 0.8 alpha).

2.1. environnement java

Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C.

Il possède un certain nombre de caractéristiques qui ont largement contribué à son énorme succès :

❖ **Java est interprété :** La source est compilé en pseudo code ou byte code puis exécuté par un interpréteur Java : une machine virtuelle JAVA (JVM). Ce concept est à la base du slogan de Sun pour Java : WORA (Write Once, Run Anywhere : écrire une fois, exécuter partout.

❖ **Java est portable :** Il est indépendant de toute plate-forme. Il n'y a pas de compilation spécifique pour chaque plate forme. Le code reste indépendant de la machine sur laquelle il s'exécute. Il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une machine virtuelle JAVA. Cette indépendance est assurée au niveau du code source grâce à Unicode et au niveau du byte code.

❖ **Java est orienté objet :** comme la plupart des langages récents, Java est orienté objet. Chaque fichier source contient la définition d'une ou plusieurs classes qui sont utilisées les unes avec les autres pour former une application. Java n'est pas complètement objet car il définit des types primitifs (entier, caractère, flottant, booléen,...).

❖ **Java est simple :** le choix de ses auteurs a été d'abandonner des éléments mal compris ou mal exploités des autres langages tels que la notion de pointeurs (pour éviter les incidents en manipulant directement la mémoire), l'héritage multiple et la surcharge des opérateurs, ...

❖ **Java est fortement typé :** toutes les variables sont typées et il n'existe pas de conversion automatique qui risquerait une perte de données. Si une telle conversion doit être réalisée, le développeur doit obligatoirement utiliser un cast ou une méthode statique fournie en standard pour la réaliser.

❖ **Java assure la gestion de la mémoire :** l'allocation de la mémoire pour un objet est automatique à sa création et Java récupère automatiquement la mémoire inutilisée grâce au garbage collector qui restitue les zones de mémoire laissées libres suite à la destruction des objets.

❖ **Gestion et récupération des erreurs d'exécution (exceptions) :** Comme la plupart des langages modernes, Java dispose de mécanismes permettant de récupérer les erreurs d'exécution et de reprendre la main. Ce qui permet d'éviter les Bugs des logiciels.

❖ **Gratuit** : Sun fournit gratuitement la machine virtuelle Java avec les outils de développement de base et certains outils plus évolués. D'autres outils professionnels payants qui bénéficient de support technique plus développés sont mis à la disposition des développeurs.

❖ **Exécution sécurisée** : Les programmes s'exécutent sur une machine virtuelle permettant de protéger le système d'exploitation et les autres programmes utilisateurs des codes malveillants.

❖ **Bibliothèque riche**: Des composants fournis en standard avec Java couvrent de nombreux domaines; interface utilisateur graphique, accès aux bases de données, accès aux fichiers, accès au réseau, …etc.

2.2. Environnement gephi

Gephi est l'un des logiciels les plus employés à l'heure actuelle pour visualiser, analyser et explorer en temps réel des graphes de tout type. Il est écrit en Java et basé sur la plateforme Netbeans. Il se distingue par sa facilité d'utilisation. Il propose différents algorithmes de spatialisation paramétrables et permet également de travailler l'aspect du graphe (couleur et taille des nœuds et des arêtes), manuellement ou en fonction de données numériques.

Quelque une des fonctionnalités de gephi:

- Visualisation
- Algorithmes de placement (**layout**) efficaces
- Métriques de réseaux : centralités, densité, diamètre, détection de communautés...
- Plugins communautaires permettant d'étendre facilement les fonctionnalités
- Tableur adapté aux données de réseaux pour combiner visualisation et analyse statistique
- Editeur de cartographie vectorielle
- Filtrage
- Support des réseaux dynamiques et des graphes hiérarchiques
- Toolkit dont les APIs permettent de construire d'autres applications

Les deux premières fonctionnalités correspondent à nos besoins :

– **Visualisation :** nous a permis de mieux manipuler les nœuds de différentes communautés avec ces divers éléments :

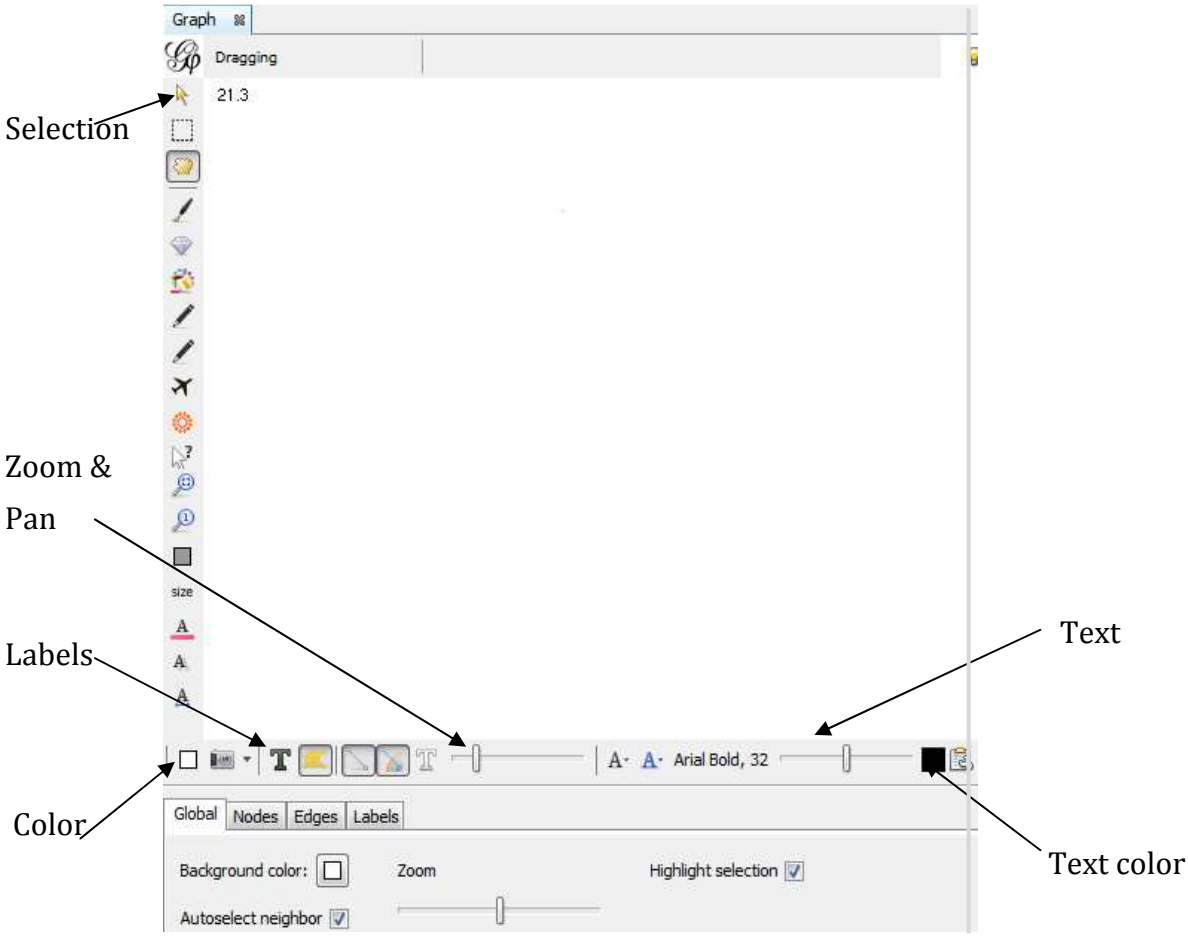


Figure5. 1. Vue générale de l'interface de visualisation de Gephi

–**Layout :** nous avons choisi d'appliquer l'algorithme Force Atlas car il correspond à nos besoins. Il permet de rapprocher les nœuds de même taille (lors de la programmation, nous avons affecté la même taille aux nœuds de la même communauté au moment de la génération du fichier gexf).

La procédure est comme suite :

1-Cocher la case « Atraction Distrib ».

2-cocher la case « Ajust by Sizes ».

3-cliquer sur « Run ».

4-lorsque le graph s'est stabilisé, cliquer sur « Stop».

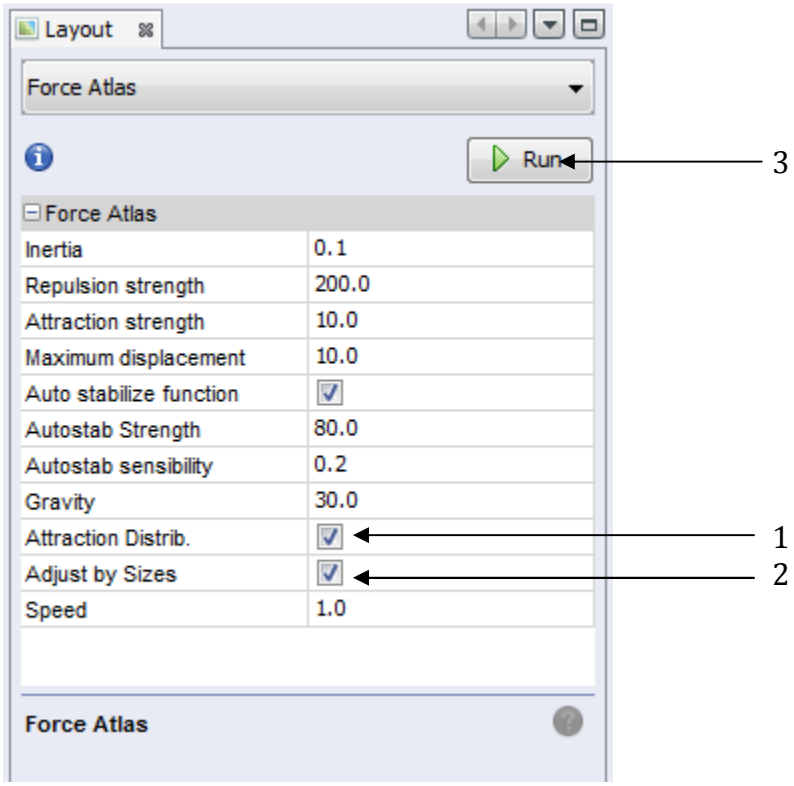


Figure5. 2. Vue générale de l'interface de Layout

➢ **Interopérabilité**

・Fichiers d'entré des données : CSV, GDF, GEXF, GML, GraphML, Graphviz DOT, Pajek NET, Tulip TLP, Ucinet DL, XGMML.

・fichiers de sortie des données : CSV, GDF, GEXF, GraphML

・Fichiers de sorties graphiques : PDF, SVG, PNG

3. Description des interfaces de notre logiciel

3.1. La fenêtre principale

Cette fenêtre est constituée de 2 parties :

➢ **1ère partie:** Celle qui permet à l'utilisateur d'appliquer l'algorithme génétique pour détécter des communautés disjointes sur un graphe monoparti.

➢ **2ème partie:** Celle qui permet à l'utilisateur d'appliquer l'algorithme génétique pour détécter des communautés chevaucantes sur un graphe biparti.

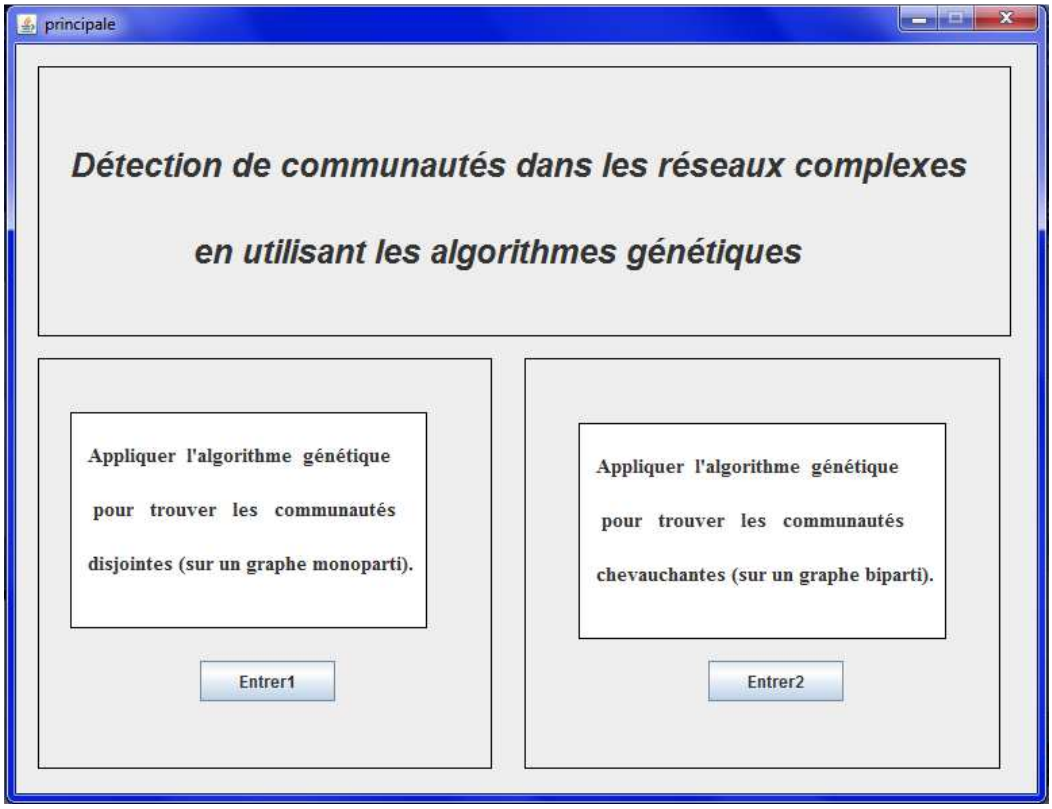


Figure5. 3. La fenêtre principale

➢ **Entré1 :** ce bouton renvoie à la 2ème fenêtre qui permet la détection de communautés disjointes dans un graph monoparti.

➢ **Entré2 :** ce bouton renvoie à la 3éme fenêtre qui permet la détection de communautés chevauchantes sur un graphe biparti.

3.2. La fenêtre de détection de communautés disjointes

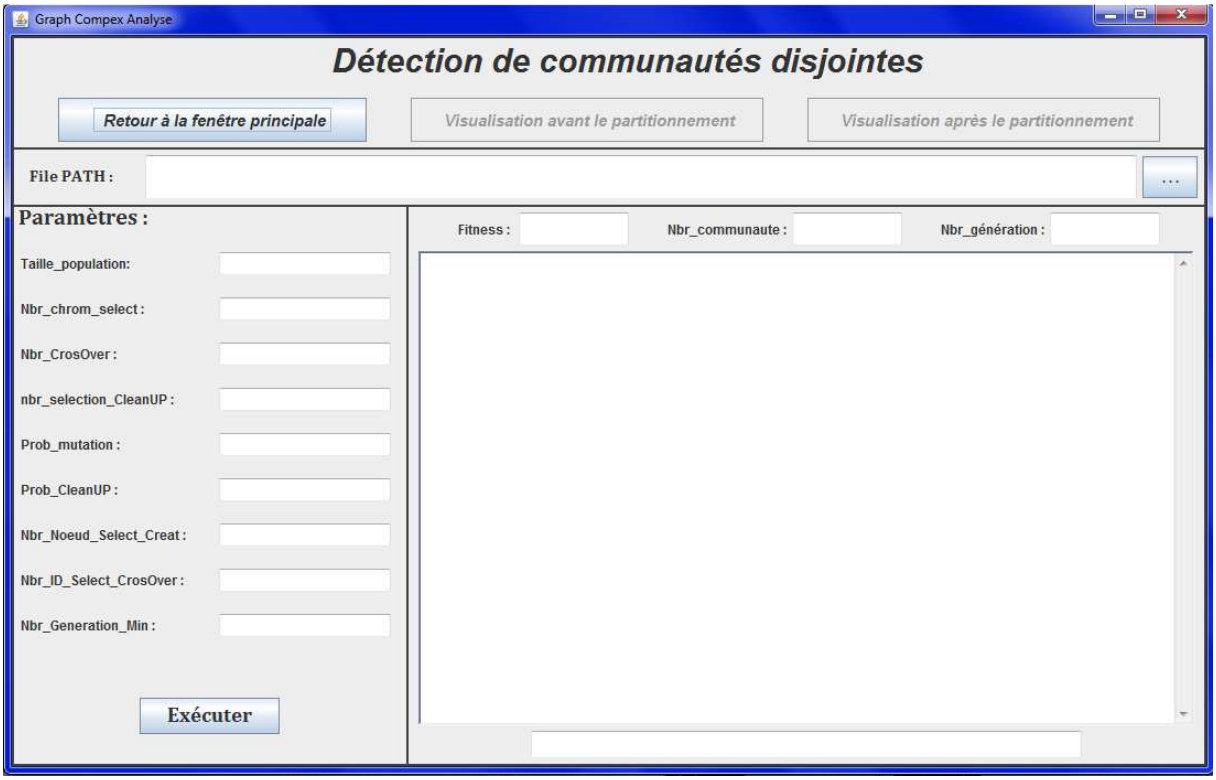


Figure5. 4. Fenêtre de détection de communautés disjointes

L'utilisation de cette fenêtre est comme suite :

1- Cliquer sur le bouton « … » afin de charger le fichier (qui contient la description du graphe) sur lequel l'algorithme génétique sera appliqué et son emplacement sera affiché dans la zone « File PATH ».

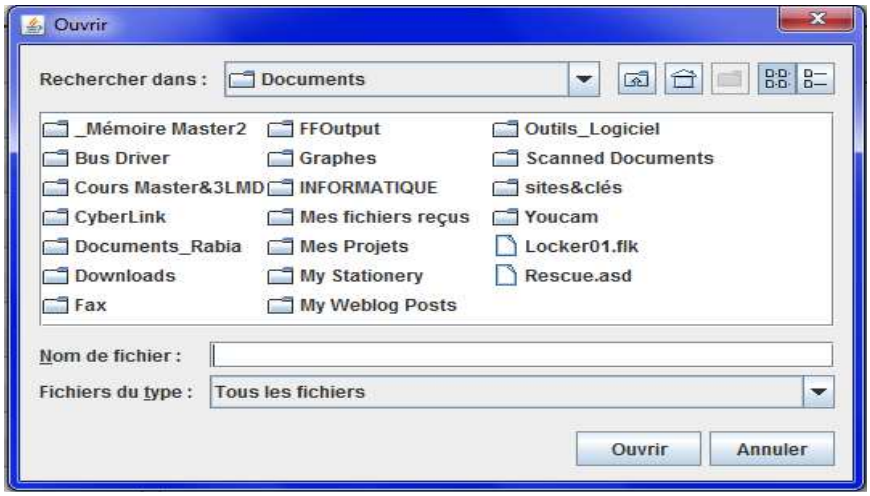


Figure5. 4. Fenêtre de chargement du PATH

2- Saisir les paramètres d'entrée de l'algorithme :

-**Taille_population** : taille de la population

- **Nbr_Chrom_Select:** nombre de chromosomes sélectionnés pour le croisement

-Nbr_croisement: nombre de fois que le croisement sera appliqué

-**Nbr_selection-CleanUP** : nombre de nœuds sélectionnés pour l'opérateur cleanUP

- **Prob_Mutation**: probabilité de mutation est entre 0 et 1

- **Prob_CleanUP**: probabilité de clean up entre 0 et 1

- **Nbr_Noeud_Select_Creat:** nombre de nœuds sélectionnés pour l'initialisation du chromosome dans la population initial

- **Nbr_ID_Select_Crosover**: nombre d'identifiants qui doivent être sélectionnés pour le croisement

-**Nbr_Generation_Min:** nombre de génération minimum

3-Cliquer sur le bouton « Exécuter », ce qui va lancer l'exécution de l'algorithme génétique.

Les résultats retournés (pour le meilleur chromosome obtenu) sont :

-La valeur de fitness

-Nombre de communautés

-Nombre de générations

-Le temps d'exécution de l'algorithme

-Affichage de toutes les communautés avec les nœuds qui les composent

Le bouton « **Visualisation après le partitionnement** » permet de visualiser le résultat à l'aide du logiciel gephi.

3.3. La fenêtre de détection de communautés chevauchantes

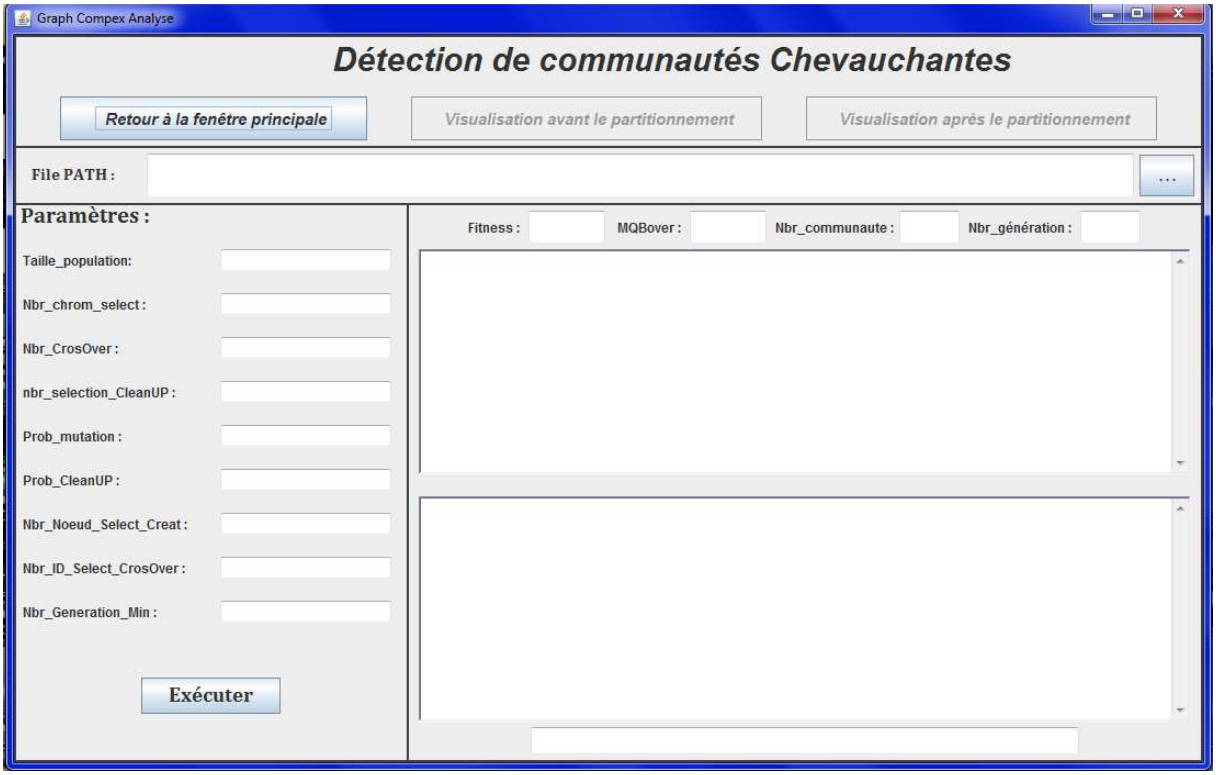


Figure5. 5. Fenêtre de détection de communautés chevauchantes

L'utilisation de cette fenêtre est identique à l'utilisation de la 2ème fenêtre à la différence, qu'ici nous avons 2 affichages en plus :

– la mesure de qualité pour le partitionnement en communautés chevauchantes

– Des communautés disjointes avec les nœuds qui les composent (pour le graphe de lien).

Remarque : les paramètres d'entrée saisis par l'utilisateur peuvent être incohérents et provoquer des erreurs d'exécution. Pour cela, des fenêtres d'erreur sont générées.

Exemple :

- si les paramètres d'entrée ne sont pas tous introduits
- si le nombre de chromosomes sélectionnés est supérieur à la taille de la population
-si la probabilité introduite n'est pas dans l'intervalle [0,1]
- Etc…

4. Résultats expérimentaux

Nous avons réglé les paramètres de l'algorithme génétique pour chacun des problèmes traité et ça en un nombre considérable d'exécutions et n'avons pas changé ces valeurs après ce réglage.

Nous avons évalué l'exactitude de l'algorithme sur trois jeux de données. Les deux premiers sont bien connus, à savoir le « **Zachary Karate Club »** et les ensembles de données du « **College Football Network»**, le troisième est un graphe généré a partir d'un benchmark (il génère un graphe avec une meilleur partition qu'on utilise comme point de référence), afin de détecté des communautés disjointes. Le dernier jeu de donné est appliqué sur graphe biparti afin de détecté des communautés chevauchantes.

4.1. Graphes monopartis

- **Zachary Karate Club**

Les données de **Zachary Karate Club** contiennent la structure communautaire dans un club de karaté.

Le reseau est constitué de 34 nœuds et 78 arêtes. Nous exécutons notre algorithme sur un ensemble de données plusieurs fois. Dans nos expériences nous avons utilisé une valeur entre 200 et 500 comme le nombre de génération et une valeur entre 100 et 250 comme la taille de population.

Les résultats de la meilleure solution qu'on a obtenue sont :

- Fitness : 0.406
- Nombre de communautés : 4
- Temps d'exécution : ne dépasse pas 15 seconds
- Nombre de nœuds mal placés : 2

L'algorithme approche la solution idéale de 94%.

L'algorithme constate que 97 %-100 % corrigent la structure communautaire. Il place (situe) parfois le nœud 10 dans la fausse communauté.

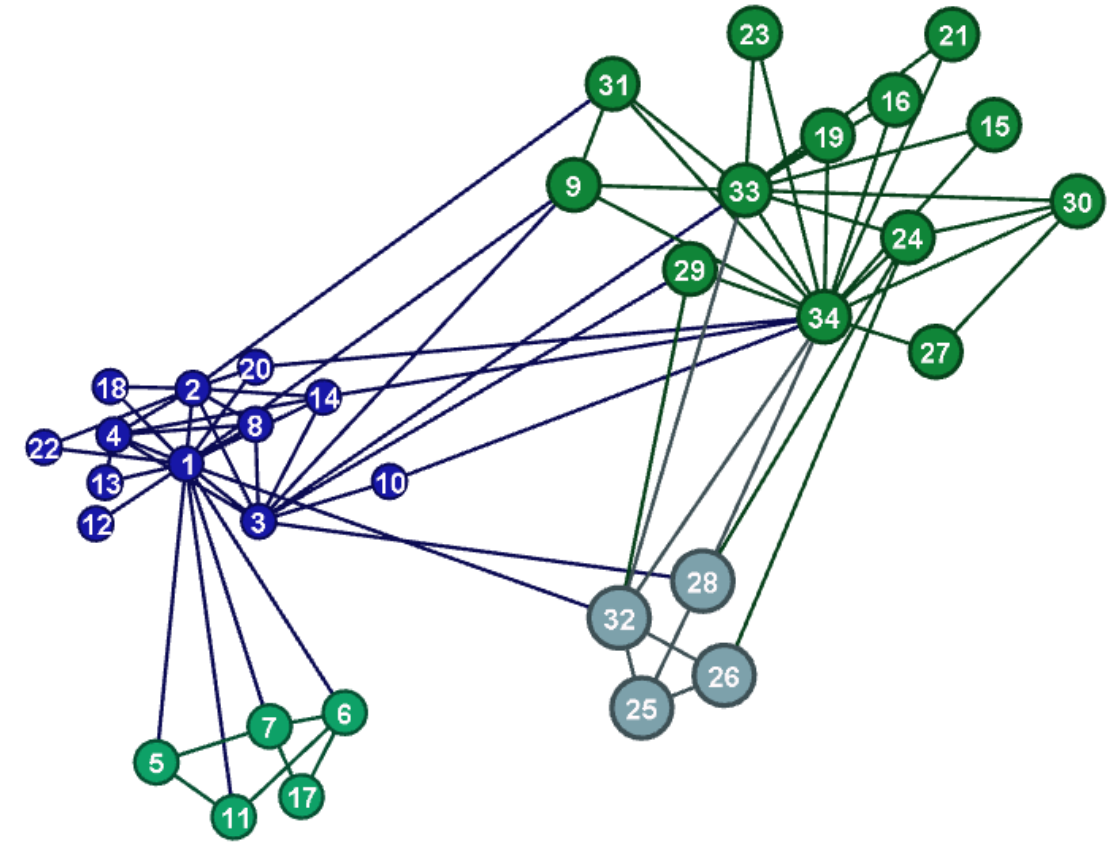


Figure5.6. Structure en communautés de Zachary Karate Club

- **College Football Network**

Le réseau **College Football Network** est composé des matchs de football dans une université aux USA, pour la première division en 2000. Les nœuds dans le réseau représentent les équipes de football et deux équipes qui ont joué un match sont reliées par une arête.

Le réseau est constitué de 115 nœuds et 613 arêtes. Nous exécutons notre algorithme sur un ensemble de données plusieurs fois. Dans nos expériences nous avons utilisé une valeur entre 300 et 500 comme le nombre de génération et une valeur entre 500 et 800 comme la taille de population.

Les résultats de la meilleure solution qu'on a obtenue sont :

- Fitness : 0.530
- Nombre de communautés : 12
- Temps d'exécution : entre 3 et 5 minutes
- Nombre de nœuds mal placés : 8

L'algorithme approche la solution idéale de 93%.

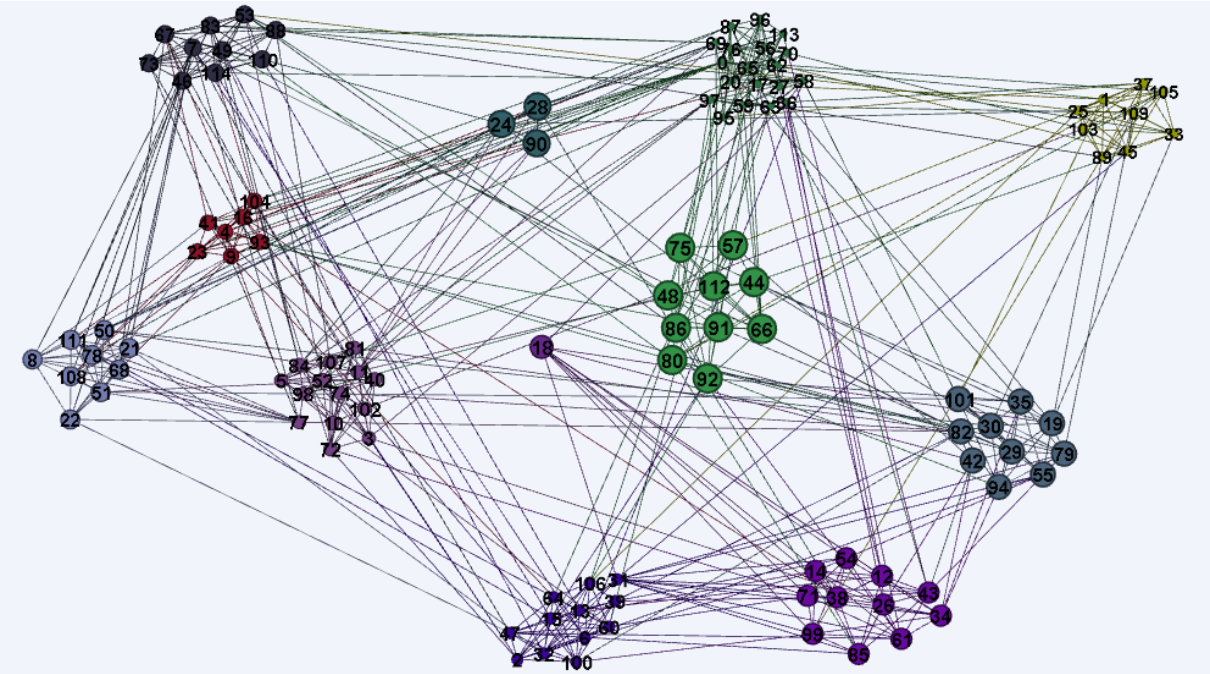


Figure5.7. Structure en communautés de College Football Network

➢ Graphe synthétique

Pour générer des graphes synthétiques, nous avons utilisé un logiciel qui produit des Benchmarks. Ce logiciel a été mis en ligne par Fortunato, il est téléchargeable sur la page http://sites.google.com/site/santofortunato/inthepress2. Le graphe synthétique généré contient 128 nœuds répartit en 4 communautés de 32 nœuds chacune, de 1024 arêtes et d'une densité de 0.2 entre les communautés.

Dans nos expériences nous avons utilisé une valeur entre 200 et 500 comme le nombre de génération et une valeur entre 500 et 800 comme la taille de population.

Les résultats de la meilleure solution qu'on a obtenue sont :

- Fitness : 0.485
- Nombre de communautés : 4
- Temps d'exécution : entre 5 et 15 minutes
- Nombre de nœuds mal placés : 6

L'algorithme approche la solution idéale de 95%.

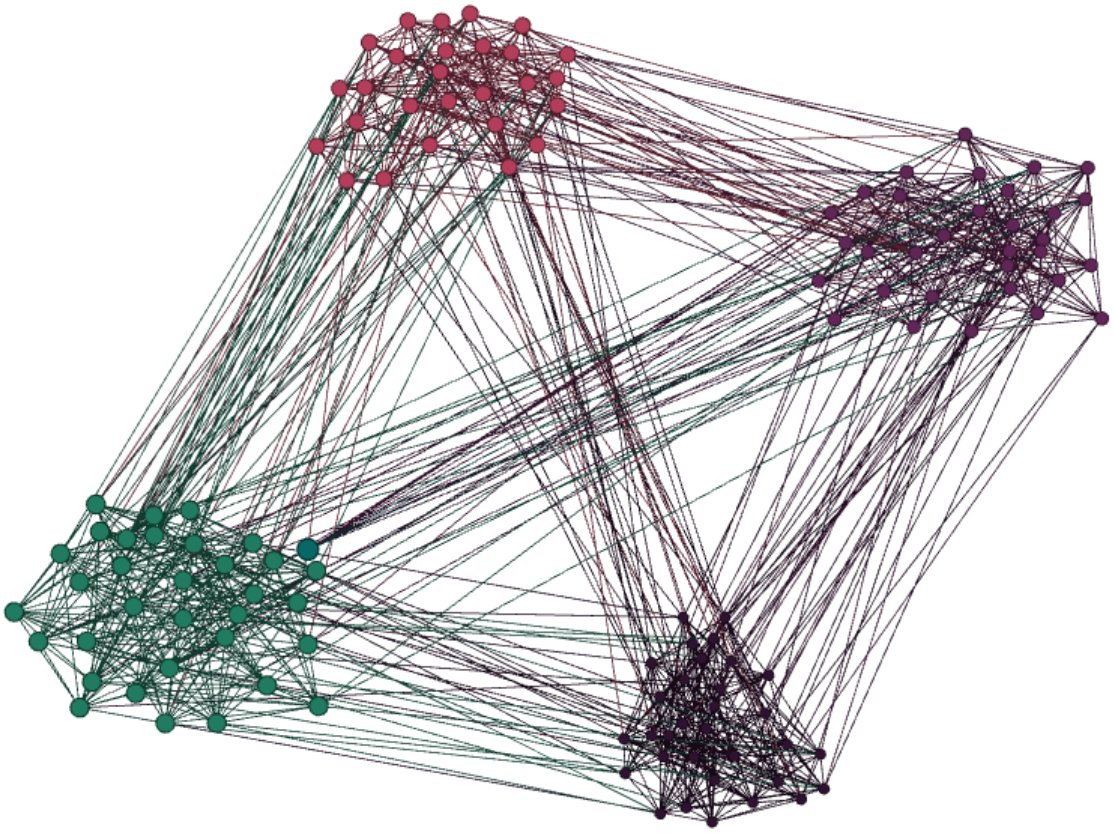


Figure5.8. Structure en communautés du graphe synthétique

4.2. Graphe biparti

La détection de communautés chevauchantes a été testée avec un graphe bien structuré qui est constitué de 26 nœuds et de 35 arêtes.

Dans nos expériences nous avons utilisé une valeur entre 20 et 50 comme le nombre de génération et une valeur entre 100 et 200 comme la taille de population.

Les résultats de la meilleure solution qu'on a obtenue sont :

- Fitness : 0.610
- Mesure de qualité bipartie : 0.600
- Nombre de communautés : 4
- Temps d'exécution : inférieur à 3 seconds

Dans le cas de ce graphe, l'algorithme effectue un bon partitionnement en communautés chevauchantes.

Le numéro de chaque nœud (Figure5.9) indique l'identifiant de communauté

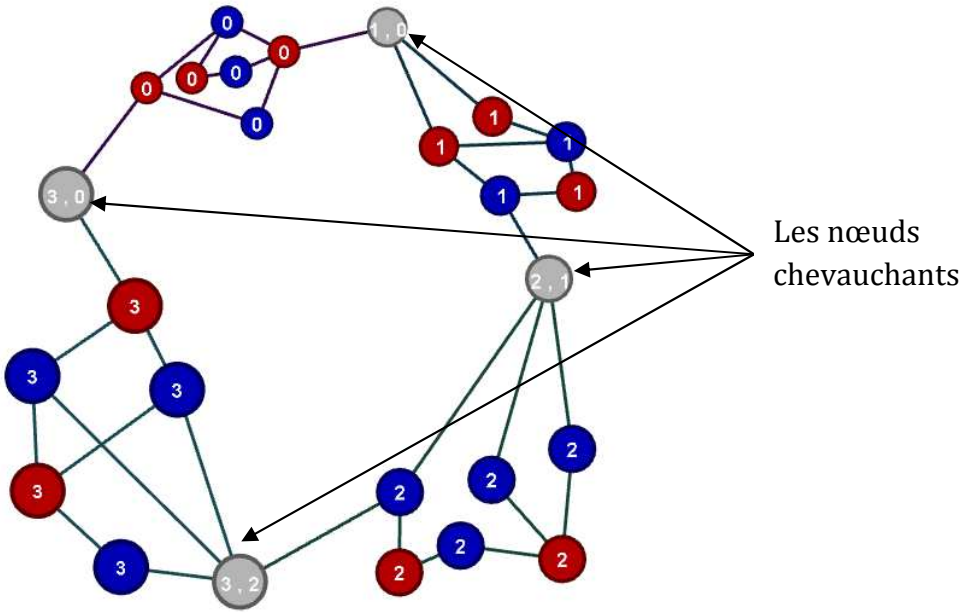


Figure5.9. Structure en communautés du graphe biparti

5. Conclusion

Nous avons constaté que l'ajustement des paramètres d'un algorithme génétique est toujours une opération délicate. Il doit être optimisé pour chaque problème traité, toute la connaissance du problème peut être traduite et exploitée afin d'améliorer la recherche de solutions optimales. Ainsi, l'utilisateur doit être judicieux avec chacun des paramètres de l'algorithme telles que, la taille de la population, nombre de chromosomes sélectionnés pour le croisement, nombre de croisements effectués pour chaque génération, nombre de nœuds sélectionnés pour l'operateur cleanUP, probabilité de mutation, probabilité cleanUP, nombre de nœuds sélectionnés afin d'initialiser chaque chromosome de la population initiale, nombre d'identifiants de communautés sélectionnés pour l'operateur de croisement et enfin le nombre de génération à dépasser.

Conclusion générale

Dans ce rapport nous avons implémenté un nouvel algorithme de détection de communautés, qui essaye d'optimiser la modularité du réseau en utilisant des méthodes des algorithmes génétiques. Cet algorithme est destiné pour la détection de communautés disjointes dans les graphes monopartis. Nous avons aussi utilisé cet algorithme dans une approche qui détecte des communautés chevauchantes sur des graphes bipartis. Dans cette approche la détection de communautés chevauchantes se fait par le partitionnement de l'ensemble d'arêtes.

L'algorithme est rapide et évolutif, appliqué sur de très grands réseaux en raison de sa complexité temporelle O (e) où e est le nombre d'arêtes dans le réseau. À la différence de la plupart des algorithmes précédents, le nouvel algorithme n'a pas besoin de connaissance à priori de la structure des communautés, comme le nombre de communautés dans le réseau ou la valeur d'un seuil.

Un nouveau mécanisme appelé cleanUP a été ajouté pour réduire le nombre de nœuds mal placé et d'améliorer la convergence de l'algorithme.

La détection de communautés est un domaine de recherche extrêmement actif et fécond, la course à des algorithmes toujours meilleurs continue. Certaines problématiques n'en sont encore qu'à leurs balbutiements : on ne comprend encore pas bien les communautés qui se chevauchent, les communautés dans des graphes orientés, ou l'évolution des communautés dans un graphe dynamique.

Bibliographie

[1] R. ALBERT and A.-L. BARABASI. Statistical mechanics of complex networks. Reviews of Modern Physics, 74(1) :47, 2002.

[2] ULRIK BRANDES and THOMAS ERLEBACH, editors. Network Analysis : Methodological Foundations, volume 3418 of Lecture Notes in Computer Science. Springer, 2005.

[3] GAREY M., JOHNSON S., Computers and Intractability: A Guide to the Theory

of NP-Completeness, Freeman, San Francisco, 1979.

[4] PAPADIMITRIOU C. H., STEIGLITZ K., Combinatorial Optimization Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[5] GAREY M., JOHNSON D., STOCKMEYER L., "Sorne Simplified NP-Complete

Graph Problems", Theorical Computer Science, vol. 1(3), pp. 237-267, 1976.

[6] BUI T. N., MOON B. R., "Genetic Algorithm and Graph Partitioning", IEEE

Transactions on Computers, vol. 45, nO 7, pp. 841-855, 1996.

[7] NICOLAS KRÜMMENACKER, Heuristiques de conception de topologies réseaux: application aux réseaux locaux industriels, Thèse de Doctorat, 2002.

[8] CHARLES-EDMOND BICHOT, La méthode de fusion-fission. Application au découpage de l'espace aérien, Thèse de Doctorat, 2007.

[9] PASCAL PONS, Détection de communautés dans les grands graphes de terrain, Thèse de Doctorat, 2007.

[10] GEORGE KARYPIS et VIPIN KUMAR**.** A fast and high quality multilevel schemefor partitioning irregular graphs. s.l. : SIAM journal of scientific computing, 1998.

[11] W.E. DONATH et A.J. HOFFMAN.ALGORITHMS for partitioning graphs and computer logic based on eigenvectors of connection matrices. s.l. : IBM Technical Disclosure Belletin, 1972.

[12] KERNIGHAN B. W., LIN S., "An Efficient Heuristic Procedure for Partitioning Graphs", The Bell System Technical Journal, vol. 49(2), pp. 291-307, 1970.

[13] STEPHEN T. BARNARD et HORST D. SIMON. A fast multilevel implementation of recursive spectral bisection for patitioning unstructured problems. s.l.: Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing, 1993.

[14] T.BUI et C.JONES. A heuristic for reducing fill in sparse matrix factorization. s.l. : Proceedings of 6th SIAM Conference on Parallel Processing for Scientific Computing, 1993.

[15] S.KIRKPATRICK, C.D. GELATT ET M.P. VECCHI**.** Optimization by Simulated Annealing. s.l. : Science, 1983.

[16] GLOVER F., "Tabu Search - Part 1", ORSA Journal on Computing, vol. l, pp. 190-206, 1989.

[17] GLOVER F., "Tabu Search - Part II'', ORSA Journal on Computing, vol. 2, pp. 4-32, 1990.

[18] ROLLAND E., PIRKUL H., GLOVER F., "A Tabu Search for Graph Partitioning", Annals of Operations Research, Metaheuristics in Combinatorial Optimization, vol. 63 pp. 209-232, 1996.

[19] A.COLORNI, M.DORIGO et V.MANIEZZO.Distributed Optimization by Ant Colonies. s.l. : Proceedings of the first European Conference on Artificial Life, 1992.

[20] GOLDBERG D. E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison- Wesley, 1989.

[21]FATIMA-ZOHRA KETTAF « des algorithmes evolutionnaires pour la classification automatique » : article.

[22]CHARLES-EDMOND BICHOT, élaboration d'une nouvelle métaheuristique pour le partitionnement de graphe : la méthode de fusion-fission. Thèse de Doctorat, novembre 2007

[23]MATTHEW WALKER, « Introduction to Genetic Programming», October 7, 2001

[24]MARKUS BRAMEIER, « On Linear Genetic Programming», Dissertation zur Erlangung des Grades eines Doktors der Naturwissenschaften der Universit¨at Dortmund am Fachbereich Informatik von, Feb. 2004

[25]SILJA MEYER-NIEBERG and HANS-GEORG BEYERWHY NOISE MAY BE GOOD: Additive Noise on the Sharp Ridge

[26]MITCHELL MELANIE « An Introduction to Genetic Algorithms», A Bradford Book The MIT Press Cambridge, Massachusetts • London, England Fifth printing, 1999,First MIT Press paperback edition, 1998

[27] SARRA BOUALLAGUI, Techniques d'optimisation déterministe et stochastique pour la résolution de problèmes difficiles en cryptologie ; Thèse de Doctorat, 2010.

[28]A.E. EIBEN, M. SCHOENAUER, « Evolutionary computing» Faculty of Sciences, Free University Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, Netherlands INRIA Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, France

[29] TASGIN, M., HERDAGDELEN, A., BINGOL, A.: Communities detection in complex networks using genetic algorithms. oai:arXiv.org:0711.0491v1 [physics.soc-ph] (2007)

[30]C. PIZZUTI, GA-NET: A Genetic Algorithm for Community Detection in Social Networks, in: Lecture Notes in Computer Sciences, LNCS 5189, Springer Verlag, Berlin 2008, p. 1081.

[31]TASGIN, M., BINGOL, A.: Communities detection in complex networks using genetic algorithms. In: Proc. of the European Conference on Complex Systems (ECSS 2006) (2006)

[32] NEWMAN, M.E.J. “Fast algorithm for detecting community structure in networks”, Physical Review E, 69:066133, 2004.