Bat-Cluster : Une Approche De Clustering Automatique Des Graphes Basée Sur L'algorithme Des Chauves-Souris

Zakaria Boulouard (*), **Anass El Haddadi** (**), **Bernard Dousset** (***)
zboulouard@gmail.com , anass.elhaddadi@gmail.com , dousset@irit.fr

(*) Faculté des Sciences Agadir, BP 8106 Cité Dakhla, Agadir (Maroc),
(**) Ecole Nationale des Sciences Appliquées Al Hoceima, BP 03 Ajdir, Al Hoceima (Maroc),
(***) Institut de Recherche en Informatique de Toulouse, 118 Route de Narbonne, Toulouse (France).

Mots clefs :
Clustering Automatique, Bat Algorithm, Bat-Cluster, Larges Graphes, Intelligence par Essaims
Keywords:
Automated Clustering, Bat Algorithm, Bat-Cluster, Large Graphs, Swarm Intelligence
Palabras clave :
Agrupación Automatizada, Bat Algorithm, Bat-Cluster, Gráficos Grandes, Inteligencia de Enjambre

Résumé

Définir le nombre correct de clusters est l'une des tâches les plus fondamentales en clustering des graphes. Lorsqu'il s'agit de larges graphes, la tâche devient encore plus difficile à cause d'un manque d'informations préalables. Ce chapitre présente une approche pour résoudre ce problème en se basant sur l'algorithme des chauves-souris (Bat Algorithm), qui est l'un des algorithmes les plus prometteurs dans le domaine de l'intelligence par essaim. Cette approche que nous avons choisi d'appeler ``Bat-Cluster, ou (BC)", permet une automatisation du clustering des graphes basée sur un équilibre entre deux processus, la recherche globale et la recherche locale. Une simulation de quatre graphes de référence de différentes tailles, démontre l'efficacité de l'algorithme que nous proposons, ainsi que sa capacité à fournir une meilleure précision et d'outrepasser les clusterings proposés par des solutions plus connues au niveau de la littérature.

1 Introduction

Les graphes permettent aux analystes de visualiser les données interconnectées et de s'appuyer sur leurs capacités visuelles afin de déchiffrer des connaissances d'une grande importance et qui pourraient s'avérer très utiles pour améliorer le processus de prise de décision au sein d'un organisme. La visualisation de ces larges graphes issus de ce gros tas de données disponibles qui ne cesse d'augmenter de manière continue, est devenu une tâche de plus en plus complexe et qui a sûrement dépassé l'habilité humaine de visualiser, d'analyser, ou même de comprendre ces graphes. Pour cela, on aura besoin d'un processus capable de réduire ces larges graphes en des graphes beaucoup plus petits et beaucoup plus représentatifs. Pour répondre à ce besoin, le clustering des graphes s'est dernièrement imposé comme un domaine de recherche très prometteur.

Etant une branche du clustering de données standard, l'objectif principal du clustering des graphes est de partitionner un large graphe en un ensemble de regroupements de nœuds similaires. Chacun de ces regroupements, appelé ``Cluster", contient des nœuds qui, en se basant sur une certaine mesure de similarité, sont semblables les uns aux autres et différents des nœuds appartenant aux autres clusters. Le clustering des graphes est une pratique très connue ayant des applications dans différents domaines tels que la segmentation des images [1], le World Wide Web [2], ou encore, les réseaux de neurones biochimiques [3] entre autres...

L'un des plus grands problèmes liés au clustering des graphes est l'estimation du nombre correct de clusters, à titre d'exemple, le bon vieux K-Means par [4] qui nécessite une connaissance du nombre exact de clusters avant de s'exécuter. Le problème persiste toujours avec les plus récentes mises-à-jour de cet algorithme, comme celles proposées par [5] ou par [6] par exemple.

Toutefois, dans une bonne partie de cas d'utilisation réels, il est impossible de déterminer le nombre exact de clusters à priori, et c'est pour cette raison que la littérature a rapporté plusieurs tentatives d'algorithmes capables de déterminer le nombre corrects de clusters avant de passer au processus de clustering. Ce domaine de recherche, communément appelé ``Clustering Automatique" a connu ses débuts dans la fin des années 1990 mais il n'a pu se développer que vers la fin des années 2000 et le début des années 2010 avec l'introduction des concepts liés à l'intelligence artificielle tels que les algorithmes inspirés par la nature [7], [8].

L'intelligence par essaim, ou ``Swarm Intelligence", étant l'une des tendances les plus récentes dans le domaine des algorithmes inspirés par la nature, a aussi était appliquée au clustering automatique. Elle a pu montrer sa performance et sa capacité de définir un clustering très correct à travers plusieurs travaux issus de la littérature la plus récente [9]–[11].

L'intelligence par essaim a été définie par [11] comme étant une classe de métaheuristiques inspirée par l'intelligence collective d'une population d'agents qui interagissent entre eux ainsi qu'avec leur environnement. Il est possible de retrouver ce genre d'intelligence chez les animaux et les organismes biologiques qui ont tendance à vivre de manière communautaire tels que les colonies de fourmis, les nuées d'oiseaux, ou encore, les bancs de poissons [12].

Parmi les algorithmes de cette classe, le plus connu est l'Optimisation par Essaim de Particules, communément connue sous son nom anglais que nous allons utiliser par la suite dans ce document, ``Particle Swarm Optimization" ou PSO. Cet algorithme développé par [13] est utilisé dans le but de définir la solution optimale d'une fonction objectif en se basant sur le comportement d'une population de particules. Chacune de ces particules explore l'espace de recherche en ajustant sa trajectoire d'une manière itérative en se basant sur sa propre expérience ainsi que celle collectée depuis les autres particules de l'essaim. Nous parlerons d'une manière plus détaillée du PSO, ainsi que d'autres algorithmes que nous trouvons intéressants, dans la Section 3 où nous allons aussi évoquer leur utilité quant à la résolution de la problématique du clustering des graphes.

En suivant la même logique, et après avoir défini, en Section 4, la fonction objectif à optimiser, nous allons présenter, en Section 5, l'algorithme ``Bat-Cluster''. C'est une adaptation de l'algorithme des chauves-souris développé par [14]. Cet algorithme, plus connu sous le nom de ``Bat Algorithm'' (ou BA), a gagné une bonne réputation dans la littérature comme étant l'un des algorithmes d'essaim les plus prometteurs. Il s'agit d'une amélioration du PSO basée sur le comportement des chauves-souris.

BA a déjà été proposé comme une solution à la problématique du clustering automatique par [15]. Notre solution, ``Bat-Cluster'' différera du KMBA proposé par [15] en trois points essentiels :

- Les sources de données utilisées dans les tests du KMBA sont standard, définies et de petite taille. Ceci a permis aux développeurs de cet algorithme de comparer leurs résultats avec des vrais positifs connus d'avances. Dans mon cas, les sources de données sont fondamentalement des larges graphes dont nous n'avons aucune connaissance préalable des supposés vrais positifs.
- Au lieu d'utiliser la distance Euclidienne séparant les données des centres des clusters comme mesure de qualité du clustering, nous avons opté pour l'une des métriques de clustering les plus connues de la littérature, l'indice de Davies-Bouldin, ou ``DBIndex'' [16]. Cet indice permet de mesurer la qualité d'un clustering en se basant sur un ratio entre, d'un côté, la distance entre les nœuds d'un même cluster, et d'un autre côté, de la distance entre les centres des clusters. Plus ce ratio est petit, mieux est la qualité du clustering.
- Au lieu de traiter des données standards comme en KMBA, ``Bat-Cluster'' va traiter des données de graphes, qui ont leurs propres structures ainsi que leurs propres contraintes. Des contraintes qu'on saura éventuellement contourner, comme ce sera détaillé dans la Section 4, en transformant les données de graphe en des données standards grâce à l'algorithme ``FFDP'' que nous avons développé [17].

Les résultats des tests effectués sur le clustering fourni par ``Bat-Cluster'' comparé à celui fourni par d'autres algorithmes très prisés dans la littérature tels que le PSO, le recuit simulé ou ``Simulated Annealing (SA)'', l'optimisation par les colonies de fourmis ou ``Ant Colony Optimization (ACO)'', ainsi que l'évolution différentielle ou ``Differencial Evolution (DE)'', seront discutés dans la Section 6. La Section 7 conclura ce papier.

2 Métaheuristiques et Optimisation Complexe

Chaque jour, les décideurs font face à des problématiques de complexité croissante dans leurs domaines. A titre d'exemple, la recherche opérationnelle, le traitement des images, ou encore, dans le cas qui nous intéresse le plus, le clustering des larges graphes.

Ce genre de problématiques à résoudre peut être modélisé sous la forme d'un problème d'optimisation, dans lequel, on définit une fonction objectif (ou plusieurs dans certains cas), qu'on cherche à maximiser ou à minimiser en tenant compte de tous les paramètres qui peuvent entrer en jeu. Il est assez fréquent d'accompagner ces fonctions objectifs par des contraintes représentant les limitations liées aux paramètres. Dans cette section, nous allons introduire brièvement le concept des métaheuristiques et les avantages qu'elles présentent quant à l'optimisation complexe.

2.1 Optimisation Complexe

On peut distinguer deux types de problèmes d'optimisation : les problèmes discrets et les problèmes continus. L'exemple le plus connus des problèmes discrets est le problème du voyageur de commerce, où l'objectif est de minimiser le trajet parcouru par un agent commercial qui devrait visiter plusieurs villes avant de revenir vers sa ville de départ. Un exemple traditionnel de problèmes continus est celui de la méthode des moindres carrés pour la régression qui tente de minimiser l'erreur

quadratique. Dans certains cas pratiques, il est possible de faire face à des problèmes dits ``mixtes'', où il est possible d'avoir des paramètres discrets et d'autres continus.

Cette différenciation entre les types de problèmes d'optimisation est nécessaire afin de déterminer si le problème à résoudre peut être considéré comme ``complexe''. En effet, nous trouvons, dans la littérature, deux types de problèmes considérés comme complexes :

- Des problèmes d'optimisation discrète n'ayant pas d'algorithme capable de les résoudre en un temps polynomial (algorithmes ayant un temps de calcul proportionnel à N^n, où N est le nombre des paramètres du problème tandis que n est un entier constant). Ce genre de problèmes est appelé ``NP-Complexe''.
- Des problèmes d'optimisation continue pour lesquels il n'existe pas d'algorithme connu, capable de déterminer l'optimum global (la meilleure solution possible), d'une manière définitive, et dans un nombre fini de calculs.

La littérature regorge de tentatives de résolution de ces deux types de problèmes, chaque type de son côté.

Dans le domaine de l'optimisation continue, il est possible de trouver un nombre très important de méthodes traditionnelles d'optimisation globale [18], mais ces méthodes sont, souvent, inefficaces si la fonction objectif ne possède pas de propriétés particulières telles que la convexité. En ce qui concerne l'optimisation discrète, il est possible de trouver un grand nombre d'heuristiques capable de trouver des solutions proches de l'optimum, mais la majorité de ces algorithmes ont été conçus pour résoudre des problèmes bien spécifiques.

L'arrivée des métaheuristiques constitue une sorte de réconciliation entre les deux domaines de recherche. En effet, elles peuvent être appliquées à n'importe quel type de problèmes discret, comme elles peuvent résoudre des problèmes d'ordre continu.

Les métaheuristiques ont en communs plusieurs caractéristiques, dont essentiellement :

- Le fait qu'elles aient une nature stochastique, à un certain degré : Ceci les rend capables de faire face aux éventuelles explosions combinatoires des possibilités.
- Leur aspect direct.
- Elles sont souvent inspirées par des analogies : A titre d'exemple, de la physique (recuit simulé, système à réaction-diffusion, etc...), ou de la nature (algorithmes évolutionnaires, intelligence collective des essaims, etc...).

Parmi les inconvénients les plus connus des métaheuristiques, les plus connus sont :

- La difficulté d'ajuster les paramètres de l'algorithme pour garantir un meilleur résultat. Il faut, souvent, avoir recours à tests empiriques préalables pour pouvoir juger si une configuration des paramètres est correcte ou pas.
- La longue durée que les calculs peuvent prendre.

Le plus intéressant concernant ces méthodes, c'est qu'elles ne sont pas mutuellement exclusives. En effet, il est très courant de rencontrer, dans les nouvelles tendances de la recherche, la possibilité d'améliorer l'efficacité d'une certaine métaheuristique, ou de contourner ses faiblesses, en tirant profit des avantages que peut présenter une autre métaheuristique. C'est ce qu'on appelle, les méthodes ``hybrides''.

Les métaheuristiques peuvent être étendues pour répondre à des problèmes d'optimisation plus complexes tels que :

- Les problèmes multiobjectifs : Pour lesquels il faut optimiser, simultanément, plusieurs fonctions objectif contradictoires.
- Les problèmes multimodaux : Où l'objectif est de trouver un ensemble d'optima locaux ou globaux.
- Les problèmes dynamiques : Où la fonction objectif peut être sujette à des variations temporelles.
- Le parallélisme.

2.1.1 Avantages des Métaheuristiques

Supposons qu'on veut monter un circuit électronique. Les composantes de ce circuit devraient être placées selon une disposition c^* qui permettrait d'avoir un nombre minimum de connexions entre elles, et donc, d'assurer une meilleure performance. La Figure 1 présente l'évolution de la fonction objectif de ce problème standard d'optimisation.

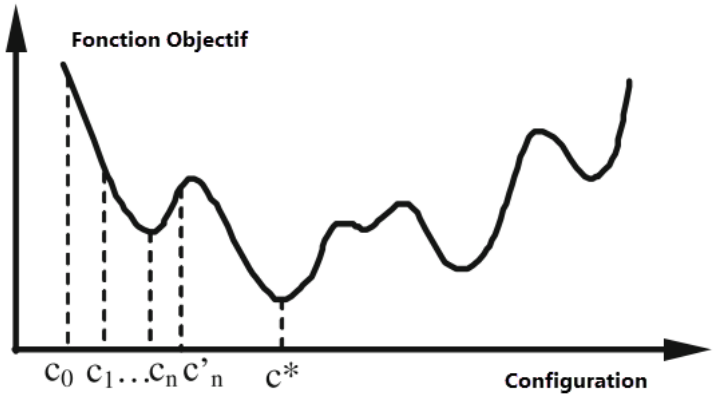

Figure 1 : Evolution de la fonction objectif du problème du circuit électronique

Dans ce paragraphe, nous allons présenter les limites que les méthodes classiques peuvent rencontrer dans leur quête du minimum global de ce problème. Puis nous allons expliquer l'avantage que les métaheuristiques peuvent présenter dans ce sens.

2.1.1.1 Les méthodes classiques et les minima locaux

Pour résoudre ce problème, ou tout autre problème d'optimisation, la littérature présente un grand nombre de méthodes itératives, dites ``classiques''. Le principe de ces méthodes est le suivant : Elles commencent par proposer une configuration initiale des positions des composantes électroniques c_0, qui peut soit être choisie au hasard, soit déterminée préalablement par le décideur.

Une modification élémentaire de cette configuration, souvent appelée ``Mouvement'', est testée. A titre d'exemple, deux composantes choisies au hasard peuvent être échangées. Les valeurs de la fonction objectif, avant et après le mouvement, vont être comparées.

Si ce mouvement a permis de constater une réduction de la valeur de la fonction objectif, il est accepté, et la configuration c_1 est adoptée et choisie comme point de départ pour un nouveau mouvement. Dans le cas contraire, on revient vers la configuration précédente et on lance un nouveau mouvement.

La procédure continue de manière itérative jusqu'à ce que n'importe quelle modification proposée ne donne que des mauvais résultats.

La Figure 1 montre clairement que la méthode classique, aussi appelée ``de descente", ne va pas permettre d'arriver à la disposition optimale c^*, mais plutôt à une configuration offrant un minimum local c_n.

2.1.2 La capacité d'extraction des minima locaux

L'idée qui constitue la base des métaheuristiques, et qui leur permet d'atteindre les optima globaux, c'est leur capacité à tolérer des dégradations temporaires des résultats de la fonction objectif.

Dans le cas de l'exemple du circuit intégré, cette dégradation est présentée par le passage de la solution c_n vers la solution $c_n^{'}$. Un mécanisme de contrôle des dégradations, spécifique à chaque métaheuristique, permet d'éviter toute divergence. De cette manière, il est possible d'extraire le processus de calcul des optima locaux et de lui permettre d'explorer de meilleures possibilités.

3 Définition de la Problématique

3.1 Description

Le clustering des graphes peut être défini comme étant un problème de collection des nœuds similaires dans les mêmes groupes appelés ``clusters". Les nœuds faisant partie d'un même cluster sont plus semblables entre eux qu'avec des nœuds d'autres clusters. En général, le nombre de clusters est connu d'avance et est fourni à l'algorithme de clustering en tant qu'entrée.

Cependant, avec le Big Data et les graphes qui deviennent de plus en plus larges, il est devenu très difficile d'avoir une idée préalable sur le nombre éventuel de clusters. C'est pour cette raison qu'il était devenu impératif de proposer des solutions où c'est à l'algorithme de clustering des graphes de déterminer le nombre éventuel de clusters avant de lancer le processus de clustering. Ce domaine de recherche est plus connu sous le nom de ``Clustering Automatique des Graphes".

Pour procéder à un clustering automatique des graphes, il faut définir une ``mesure de similarité", puis de disposer les clusters selon cette mesure.

Trouver la meilleure disposition des clusters selon une certaine mesure de similarité peut être exprimé sous forme d'un problème d'optimisation.

Plusieurs travaux ont tendance à proposer une formulation discrète de cette mesure de similarité, et devaient donc proposer des adaptations des algorithmes inspirés par la nature, qui sont à la base, continus.

Dans le présent travail, nous avons procédé autrement, en présentant une adaptation du problème de clustering des graphes de telle sorte à ce qu'il ait une formulation continue.

Cette adaptation sera grâce à l'algorithme ``Bat-Cluster'', qui présente une combinaison entre le FFDP développé au sein de notre équipe de recherche [17], et le Bat Algorithm tel que décrit par [14]. FFDP va définir une position d'équilibre stable du large graphe, puis va fournir des positions finales des nœuds sous la forme d'un vecteur de coordonnées au Bat Algorithm. Ce dernier va prendre le vecteur de coordonnées en considération et va trouver la meilleure configuration de clustering possible.

Le prochain paragraphe va décrire la mesure de similarité que nous avons utilisée comme fonction objectif à optimiser par l'algorithme ``Bat Cluster (BC)''.

3.2 Fonction Objectif

La fonction objectif que nous avons utilisée pour mon algorithme est la mesure de qualité qui va juger quelle configuration de clustering est la meilleure. [11] ont proposé une liste des métriques de qualité les plus prisées en clustering.

Nous avons besoin d'une métrique capable d'assurer que les nœuds similaires sont proches les uns des autres et les nœuds non-similaires sont éloignés entre eux.

L'une des métriques les plus populaires dans la littérature est le ``DBIndex''. Cet indice, proposé par [16], représente un ratio entre la distance intra-cluster (la distance entre les nœuds du cluster et son centre) et la distance inter-cluster (la distance entre les centres des différents clusters).

DBIndex est défini tel que :

$$DB = \frac{1}{n_c} \sum_{i=1}^{n_c} R_i \qquad (1)$$

Avec:

$R_i = \max_{j=1...n_c, i \neq j} (R_{ij}),\ i = 1...n_c$

$R_{ij} = \frac{s_i + s_j}{d_{ij}}$

$d_{ij} = d(g_i, g_j),\ s_i = \frac{1}{\|c_i\|} \sum_{x \in c_i} d(x, g_i)$

$d(x, y)$ est la distance Euclidienne entre x et y.

c_i est le cluster i.

g_i est le centre du cluster i.

$\|c_i\|$ est la norme de c_i.

Selon [16], un bon clustering devrait minimiser le DBIndex tel que décrit dans l'équation (1). Ceci-dit, la fonction objectif pour notre algorithme devrait être la suivante :

$$\min DB = \frac{1}{n_c} \sum_{i=1}^{n_c} R_i \qquad (2)$$

4 Travaux Connexes

Dans cette section, nous allons explorer quelques-uns des plus importantes solutions inspirées par la nature et qui ont été capables de tacler la problématique du clustering automatique des graphes. Ces solutions seront, par la suite, testées et comparées, en Section 5, à notre solution proposée, ``Bat-Cluster''.

4.1 Optimisation par Essaim de Particules (PSO)

L'optimisation par Essaim de Particules, plus connue sous le nom de ``Particle Swarm Optimization (PSO)'', est un processus de recherche basé sur une population de particules. Cet algorithme, introduit par [13], a été utilisé pour résoudre plusieurs problématiques liées à l'Apprentissage Automatique (Machine Learning), telles que le clustering [9], [19], ou la classification [20].

L'idée derrière le PSO est d'optimiser une fonction objectif, ou de fitness, en découvrant un ensemble de positions de particules représentant des solutions potentielles à cette fonction objectif. Ces solutions seront améliorées de manière itérative à travers l'évolution de l'algorithme, et s'approcher graduellement de l'optimum de la fonction objectif spécifiée. Durant chaque itération du PSO, les particules se déplacent à travers l'espace de recherche en se basant sur leur intelligence de groupe et selon certaines règles qu'elles suivent en se rapprochant graduellement vers la position optimale.

Au début de l'algorithme, une population de particule est définie. Chaque particule de cette population aura une position et une vélocité initialisées, généralement, de manière aléatoire.

Un algorithme de PSO contient quatre étapes qui seront conduites de manière itérative jusqu'au moment où il atteindra un résultat optimal :

1. Chaque particule se déplacera et mettra sa position à jour en lui ajoutant la valeur de la vélocité.
2. Les particules seront évaluées selon la valeur de la fonction objectif, ou de fitness, calculée à leur niveau.
3. Chaque particule mettra à jour sa propre meilleure valeur et la communiquera aux autres particules afin de calculer la meilleure valeur globale.
4. Chaque particule mettra à jour sa vélocité.

La littérature présente plusieurs approches ayant adopté le PSO en clustering des graphes, souvent appelé ``Détection de Communautés''.

Ces approches sont, souvent, basées sur une modification du PSO (conçu pour résoudre des problèmes d'optimisation continue), de telle façon à ce qu'il soit capable de résoudre des problèmes d'ordre discret.

[21], [22] présentent une modification de la définition de la position et de la vélocité de telle sorte que le vecteur de position représente une partition d'un graphe signé, tandis que la vélocité représente une éventuelle permutation de cette partition.

[23] ont, quant à eux, appliqué le PSO dans son aspect continu, mais ils ont suggéré d'utiliser une optimisation à multiples essaims au lieu du PSO standard basé sur un seul essaim.

[24] ont utilisé le PSO pour empêcher le processus du clustering des graphes de laisser des nœuds résiduels (des nœuds n'appartenant à aucun cluster) après son exécution. Leur idée a été appliquée pour optimiser la consommation en énergie, le débit, le ratio de livraison des paquets, ainsi que la durée de vie des réseaux de capteurs sans fil.

4.2 Optimisation par Colonies de Fourmis (ACO)

L'optimisation par les colonies de fourmis ou ``Ant Colony Optimization (ACO)'', est un algorithme qui essaie d'imiter la manière avec laquelle les fourmis interagissent et s'entraident afin de trouver et d'exploiter une source potentielle de nourriture.

La première simulation algorithmique du comportement des fourmis a été introduite par [25]. Un peu plus tard, [26] ont proposé cette simulation comme une solution pour des problèmes d'optimisation. Depuis, elle a été utilisée pour résoudre plus problèmes couvrant différents domaines tels que la reconnaissance automatique des espèces végétales [27], ou encore la détection des congestions au niveau des larges MIMO[1] [28].

L'idée derrière l'ACO est de bénéficier de la communication indirecte entre les différents agents de la colonie, à savoir, les ``fourmis artificielles''. Cette communication se fait à travers les traînées de phéromone que les fourmis laissent derrière elles après avoir trouvé une source potentielle de nourriture, c'est-à-dire, une solution optimale potentielle.

L'algorithme est initialisé par la définition des paramètres essentiels liés à l'algorithme et aux traînées de phéromone. Puis, la boucle principale exécute ces trois instructions à chaque itération :

1. Les fourmis construisent des solutions à base de l'information tirée de la phéromone.
2. Une fois les fourmis trouvent leurs solutions, une recherche locale optionnelle peut être adoptée pour améliorer la solution potentielle.
3. Pour terminer, les valeurs de la traînée de phéromone sont mises à jour afin de communiquer les résultats trouvés par les fourmis.

[29] ont proposé une technique de clustering des graphes basée sur l'ACO et ils l'ont appliquée dans le domaine du e-marketing afin de détecter des communautés de clients.

[30] ont suggéré une solution pour la détection des communautés dans les larges graphes en se basant sur une stratégie de diffusion et de mise à jour des phéromones des fourmis afin de trouver une partition optimale des graphes.

[31] ont suivi un processus similaire tout en prenant en considération la problématique du chevauchement des larges communautés.

[32] ont utilisé l'ACO ainsi que la technique de ``Feature Selection'' afin de définir des clusters d'attributs.

[1] Multiple-Input Multiple-Output, une technique de multiplexage utilisée dans les réseaux sans fil et les réseaux mobiles permettant des transferts de données à plus longue portée et avec un débit plus élevé

[33] ont proposé une combinaison entre l'ACO et le K-Means comme solution à la problématique d'emplacement dynamique d'installations. Le K-Means est utilisé pour détecter l'emplacement des dépôts (les centres des clusters), tandis que l'ACO s'occupera du problème du VRP[2] dynamique.

Pour une meilleure comparaison des performances de ``Bat-Cluster", nous avons introduit deux autres algorithmes issus d'autres catégories d'algorithmes inspirés par la nature : le Recuit Simulé et l'Evolution Différentielle.

4.3 Recuit Simulé

Le Recuit Simulé, ou ``Simulated Annealing (SA)", a été introduit par [34], puis par [35] dans un autre travail complétement indépendant.

Comme son nom le suggère, cet algorithme imite le comportement d'un phénomène physique appelé ``recuit". Le recuit d'un métal revient à l'exposer à une très haute température, puis à réduire très lentement cette dernière selon différents niveaux de température. L'idée ici est faire en sorte que le métal puisse atteindre sa position la plus stable ayant l'état d'énergie le plus faible.

D'un autre côté, on a ce que l'on appelle, ``la trempe". Elle consiste à réduire la température du métal d'une manière rapide et brusque, causant ainsi une position métastable du métal correspondant à un minimum local d'énergie.

La transposition du recuit physique au recuit ``simulé" utilisé pour résoudre des problèmes d'optimisation est basée sur les analogies suivantes :

- La fonction objectif représente l'énergie.
- Les paramètres du problème représentent les coordonnées des particules du métal.
- Trouver des solutions bonnes ou optimales du problème revient à trouver des états d'énergie minimums du métal.
- La trempe représente un minimum local.

Le SA commence par la définition de la température initiale et d'une solution aléatoire avec son coût. Puis, la boucle principale itère en exécutant les instructions suivantes :

1. Génération d'une solution voisine aléatoire.
2. Evaluation de la variation au niveau de l'énergie de la solution ΔE.
3. Si elle est négative, la solution est acceptée et la température est mise à jour.
4. Sinon, la solution pourrait être acceptée avec une probabilité de $e^{-\frac{\Delta E}{T}}$ et la température est mise à jour.
5. Si elle est négative, la solution est acceptée et la température est mise à jour.

[36] ont utilisé SA pour résoudre le problème de partitionnement des circuits intégrés. Le circuit est partitionnée de telle sorte à minimiser le nombre d'interconnections.

[2] Vehicle Routing Problem, ou Problème de Tournée des Véhicules, est une classe de problèmes de recherche opérationnelle et d'optimisation combinatoire. Il s'agit de déterminer les tournées d'une flotte de véhicules afin de livrer une liste de clients, ou de réaliser des tournées d'interventions (maintenance, réparation, contrôles) ou de visites (visites médicales, commerciales, etc.)

[37] ont répondu à la même problématique mais à une plus grande envergure puisqu'ils ont travaillé sur le réseau de haute tension de Floride. Leur solution est une version Monte Carlo du SA avec comme objectif, maximiser la connectivité interne des clusters.

[38] ont proposé une version déterministe du SA applicable au clustering des petits graphes. Plusieurs propositions de combinaison du recuit simulé avec d'autres algorithmes sont disponibles dans la littérature.

A titre d'exemple, [39] ont suggéré un algorithme mémétique combinant le SA avec le Tightness Greedy Algorithm (TGO) de telle sorte à utiliser cette combinaison comme une stratégie de recherche locale pour résoudre le problème de détection de communautés.

4.4 Evolution Différentielle (DE)

L'Evolution Différentielle, ou ``Differential Evolution (DE)", développée par [40], est considérée comme l'un des algorithmes évolutionnaires les plus performants.

Tout comme l'algorithme génétique (GA), DE applique des concepts tirés de la théorie de l'évolution, tels que la mutation et le croisement. Cependant, la plus importante différence entre la DE et le GA réside au niveau de la mutation. En effet, en GA (ou tout autre algorithme évolutionnaire traditionnel), la mutation est appliquée après le croisement, et dans certains cas, selon une certaine probabilité. Dans le cas de la DE, la mutation s'applique avant le croisement et d'une manière régulière pour chaque progéniture.

La littérature liée à la DE propose plusieurs stratégies de mutation et de croisement. Afin de différencier ces stratégies, une convention de nomenclature a été mise en place de cette manière ``$DE/x/y/z$". Où ``x" représente le vecteur à muter, ``$y$" représente le nombre de vecteurs de perturbation utilisés, tandis que ``z" représente le type de croisement.

L'algorithme d'évolution différentielle commence par la définition d'une population de vecteurs réels. Puis, la boucle principale se lance pour exécuter les instructions suivantes :

1. Calculer un vecteur mutant v_i.
2. Créer un nouvel individu u_i en utilisant un croisement entre le vecteur mutant et un autre individu de la population x_i.
3. Evaluer la valeur de la fonction objectif pour u_i et x_i.

[41] ont proposé une application directe de la DE pour résoudre le problème de partitionnement des graphes et une étude comparative avec l'algorithme génétique (GA), et qui a montré que DE était plus efficace.

[42] ont proposé une adaptation de la DE en s'inspirant du phénomène d'apprentissage social au sein des communautés animales. Leur amélioration consiste à introduire la stratégie de sélection ASL qui permettra à la DE de conduire la sélection des parents éligibles au croisement à base des relations de voisinage des individus de la population.

Plusieurs tentatives d'hybridation de la DE avec d'autres algorithmes sont disponibles dans la littérature la plus récente. A titre d'exemple, [43] ont proposé une combinaison entre la DE et l'algorithme la colonie d'abeilles artificielles (ABC). Cette combinaison a été appliquée pour résoudre la problématique de sélection d'attributs.

5 Bat-Cluster

5.1 Bat Algorithm

``Bat Algorithm (BA)" est un algorithme développé en 2010 par [14]. Il imite le comportement des ``microbats", des chauves-souris de petite taille et qui se basent essentiellement sur l'écholocalisation dans leur mouvement.

Le BA se concentre, dans son imitation du comportement des chauves-souris, sur le rôle que joue l'écholocalisation afin de permettre à la chauve-souris de trouver sa proie. L'écholocalisation a été définie comme étant le processus de détection de la position à travers les impulsions.

La chauve-souris émet des ultrasons vers son environnement et écoute ses échos afin d'éviter d'éventuels obstacles et de localiser d'éventuelles proies. Ces ultrasons varient leur fréquence et intensité en se basant sur la distance entre la chauve-souris et sa proie. Plus la chauve-souris s'approche de sa proie, plus l'intensité des ultrasons diminue et plus leurs fréquence augmente.

Le processus d'écholocalisation est décrit dans la Figure 2.

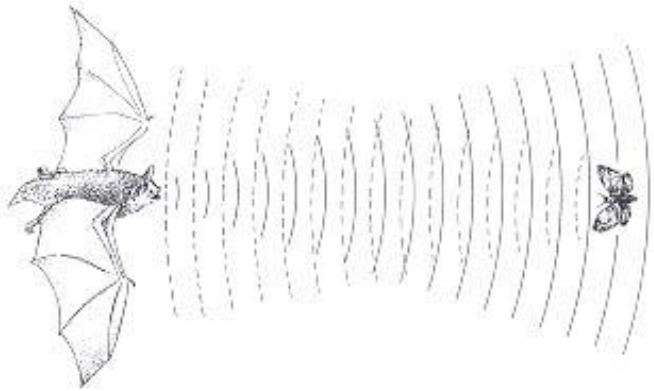

Figure 2 : Le processus d'écholocation des chauves-souris

Pour des raisons d'optimisation, [14] a idéalisé le comportement des chauves-souris selon les règles suivantes :

1. Le mouvement de la chauve-souris dépend uniquement de l'écholocalisation.
2. Les chauves-souris chassent en groupe la position de chaque chauve-souris est déterminée par x_i.
3. Chaque chauve-souris se déplace avec une vélocité v_i.
4. Chaque chauve-souris ajuste sa fréquence f_i, son pouls (pulse rate) r_i, et son intensité d'écholocalisation A_i.

La position et la vélocité de la chauve-souris i à l'itération t doivent être mises à jour au fur et à mesure de l'évolution des itérations.
Les mises à jour de la position et de la vélocité à l'itération $t+1$ sont décrites par les équations 3, 4 et 5.

Dans l'équation 5, l'expression du déplacement de la chauve-souris devrait, normalement, être comme suit : $x_i^{t+1} = x_i^t + \Delta t * v_i^{t+1}$.

Ceci-dit, vu que le temps s'incrémente d'une manière discrète entre t et $t+1$, nous avons donc $\Delta t = 1$.
Donc le déplacement de la chauve-souris est exprimé par l'expression simplifiée décrite par l'équation 5.

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta \quad (4)$$

$$v_i^{t+1} = v_i^t + (x_i^t + x^*) f_i \quad (5)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (6)$$

f_i est la fréquence de l'écholocalisation émise par la chauve-souris i. La valeur de cette fréquence est générée de manière uniforme et reste comprise dans un itervalle de $[f_{\min}, f_{\max}]$ préalablement défini.

β est un nombre aléatoire de l'intervalle $[0,1]$.

x^* est la meilleure position (ou solution) globale résultant de la comparaison de toutes solutions trouvées par les chauves-souris. Une fois une meilleure solution globale est trouvée, un processus de recherche locale est lancé en générant une nouvelle solution pour chaque chauve-souris à travers une marche aléatoire (Random Walk). Cette recherche locale est exprimée par l'équation 7.

$$x_{new} = x_{old} + \varepsilon A^t \quad (7)$$

Où x_{old} est la meilleure solution globale actuelle et x_{new} est la meilleure solution globale fraîchement découverte.

ε est une valeur aléatoire de l'intervalle $[-1,1]$ et A^t est l'intensité moyenne des chauves-souris à l'itération t telle que : $A^t = \langle A_i^t \rangle$.

Les valeurs du pouls et de l'intensité sont mises à jour à chaque itération en se basant sur si la chauve-souris i a trouvé une proie ou pas. Si c'est le cas, l'intensité diminue tandis que le pouls augmente.

Les valeurs maximum et minimum de l'intensité peuvent être choisies à convenance. Pour faire simple, nous avons choisi la valeur maximum $A_0 = 1$ et la valeur minimum $A_{\min} = 0$ en supposant que la chauve-souris arrête temporairement d'émettre des sons lorsqu'elle trouve une proie.

Le rôle du pouls est de choisir la bonne décision entre exploiter la meilleure solution globale actuelle et mettre à jour le pouls et l'intensité, ou permettre plus de marches aléatoires autour de la meilleure solution trouvée jusqu'à présent. On peut assumer que cette prise de décision basée sur le pouls est similaire à celle basée sur la température au niveau du recuit simulé, où la température contrôle le choix entre exploiter la meilleure solution globale actuelle, ou bien, explorer un peu plus l'espace de recherche.

Les valeurs de l'intensité et du pouls sont mises à jour au fur et à mesure de l'évolution de l'algorithme en se basant sur les équations 8 et 9.

$$A_i^{t+1} = \alpha A_i^t \quad (8)$$

$$r_i^{t+1} = r_i^0 \left[1 - e^{(-\gamma t)}\right] \quad (9)$$

Où $\alpha \in]0,1[$ et $\gamma \succ 0$.

Pour des raisons de simplification, nous avons choisi $\alpha = \gamma$. Donc au fur et à mesure de l'évolution des itérations, nous allons éventuellement avoir $A_i^t \to 0$ et $r_i^t \to r_i^0$ lorsque $t \to +\infty$.

L'intensité et le pouls initiaux sont choisis tels que $A_i^0 \in [1, 2]$ et $r_i^0 \in [0, 1]$. Leurs valeurs ne seront mises à jour que lorsque les chauves-souris trouvent de meilleures solutions.

Le pseudo-code du Bat Algorithm peut être décrit comme suit :

Algorithm 2 BatAlgorithm(M , N , A^0 , r^0)
input: Maximum iterations number M , Bats total population N , initial loudness A^0 and initial pulse rate r^0
Initialize the bats positions x_i , velocities v_i and frequencies f_i
While ($t \prec M$) {
    // Generate a new solution
    Adjust the frequencies of the bats according to the equation 4
    Update the velocities and the positions of the bats according to the equations 5 and 6
    If ($rand \succ r_i$) {
        Generate a new solution around the best solution according to the equation 7
    }
    If ($rand \prec A_i$ *and* $f(x_i) \prec f(x^*)$) {
        Accept the new solutions
        Update A_i and r_i according to equations 8 and 9
    }
    Select the current best solution $f(x^*)$
    Increment t
}

5.2 Bat-Cluster

``Bat-Cluster (BC)'' est une combinaison des deux algorithmes décrits précédemment, FFDP et Bat Algorithm. FFDP va s'exécuter en premier et définir une position d'équilibre pour les nœuds du graphe. Les coordonnées finales de ces nœuds seront fournies comme entrées pour le Bat Algorithm.

BA commencera par générer une population de chauves-souris dont chacune aura son propre pouls, sa propre intensité, position et vélocité. Les positions initiales des chauves-souris représentent les centres initiaux des clusters.

Quand l'algorithme se lance, chaque chauve-souris se voit assignée à un centre de cluster. Pour chaque centre de cluster, l'algorithme va calculer la valeur moyenne des positions des nœuds qui lui sont les plus proches. La position du centre de cluster est ainsi mise à jour, puis la fonction objectif est calculée selon l'équation 2. Si la valeur de la fonction objectif converge, on retourne les positions des centres des clusters. Sinon, on réaffecte chaque chauve-souris au centre de cluster correspondant.

Si la valeur aléatoire *rand* est plus grande que les pouls des chauves-souris, l'algorithme sélectionne une solution parmi les meilleures solutions trouvées puis génère une solution autour de la meilleure solution sélectionnée selon l'équation 6. Si la valeur aléatoire *rand* est plus petite que l'intensité A_i, et la valeur de la fonction objectif pour la position actuelle de la chauve-souris est meilleure (dans notre cas plus petite) que la valeur de la meilleure solution globale trouvée jusqu'à présent $f(x_i) < f(x^*)$, la nouvelle solution est acceptée, le pouls est augmenté, et l'intensité est réduite. Les solutions trouvées sont triées et la meilleure solution globale actuelle est sauvegardée.

L'algorithme continue à fonctionner jusqu'à atteindre son critère d'arrêt. Dans notre cas, l'algorithme devrait s'arrêter si le nombre d'itérations t atteint sa valeur maximale M prédéfinie.

Le pseudo-code de l'algorithme Bat-Cluster est défini comme suit :

Algorithm 3 BatCluster(G, X, tol, K, M, N, A^0, r^0)
input: Graph G, Nodes initial positions X, tolerance tol, nominal edge length K, Maximum iterations number M, Bats total population N, initial loudness A^0 and initial pulse rate r^0
// Set the graph's positioning
$coords = FFDP(G, X, tol, K)$
//coords is 2-D or 3-D vector
/*Initialize the bats positions x_i, velocities v_i and frequencies f_i */
For all bats i **do** {
$A_i = A_0$
$r_i = r_0$
$f_i = 0$
$v_i = 0$
$x_i = rand$
/* Calculate the solution of the objective function for the bat i according to the equation (2) */

$DB(x_i, coords)$

}

Select the best solution

While ($t \prec M$) {

/* Generate a new solution by adjusting the frequencies of the bats according to the equation (4), updating the velocities and the positions of the bats according to the equations (5) and (6) */

For all bats i **do** {

$f_i = f_{\min} + (f_{\max} - f_{\min})\beta$

$v_i^{t+1} = v_i^t + (x_i^t + x^*) f_i$

$x_i^{t+1} = x_i^t + v_i^{t+1}$

If ($rand \succ r_i$) {

/* Generate a new solution around the best solution according to the equation (7) */

$x_{new} = x_{old} + \varepsilon A^t$

}

}

If ($rand \prec A_i$ *and* $DB(x_i, coords) \prec DB(x^*, coords)$) {

//Accept the new solutions

$x^* = x_i$

/* Update A_i and r_i according to equations (8) and (9) */

$A_i^{t+1} = \alpha A_i^t$

$r_i^{t+1} = r_i^0 \left[1 - e^{(-\gamma t)}\right]$

}

}

Select the current best solution

$t = t + 1$

}

6 Tests et Résultats

6.1 Environnement de Test

Pour des raisons de simulations, nous avons lancé l'algorithme Bat-Cluster sur un ordinateur ayant les caractéristiques techniques suivantes :

- Environnement de développement : Java 8 + Matlab R2017a.
- Système d'Exploitation : Windows 7.
- CPU: Intel i5 2450M 2.5Ghz.
- RAM: 4Go.

Pour comparer les performances de BC, je l'ai comparé avec les quatre algorithmes suivants :

- Optimisation par Essaim de Particules (PSO).
- Optimisation par les colonies de fourmis (ACO).
- Recuit Simulé (SA).
- Evolution Différentielle (DE).

Nous allons utiliser ces algorithmes dans leur aspect continu, et la fonction à optimiser sera le DBIndex tel que décrit dans l'équation 2. Cette approche me permettra de comparer les performances de ces algorithmes sur le même pied d'égalité.

6.2 Graphes de Benchmark

Les graphes que nous avons utilisés pour les tests sont quatre graphes de benchmark de différentes tailles et issus de différents domaines. Ces graphes[3] sont les suivants :

Figure 3 : Graphe facebook_ego_686

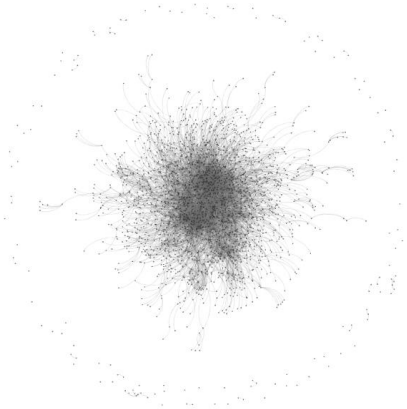

Figure 4 : Graphe yeast

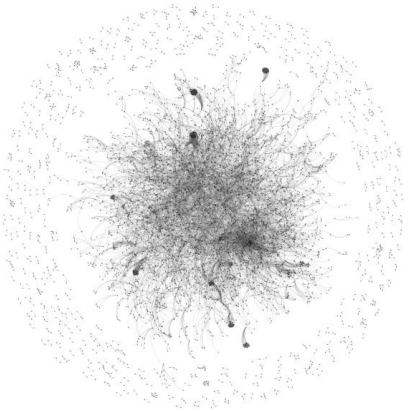

Figure 5 : Graphe arxiv_general_relativity

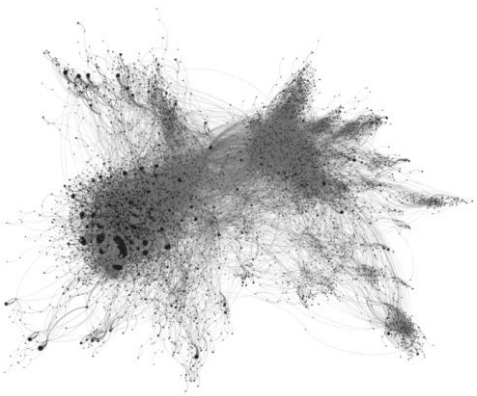

Figure 6 : Graphe oregon2_010331

[3] Ces graphes sont disponibles sur la base de données de graphes standards de Gephi, accessible depuis ce lien : https://goo.gl/7tzNEK (dernière visite le 25/12/2017).

6.3 Paramétrage

La définition des bons paramètres pour un algorithme inspiré de la nature est, en général, une étape qui nécessite un testing préalable rigoureux. Il en est de même pour BC et les algorithmes contre lesquels il va être testé.

Après plusieurs expérimentations, les paramètres que nous avons jugés capables de répondre correctement à nos besoins sont les suivants :

Bat-Cluster	**PSO**	**ACO**	**SA**	**DE**
Nombre d'Itérations Maximum $M = 200$ Taille de la Population des Chauves-souris $N = 50$ Intensité Initiale $A_0 = 1.1$ Pouls Initial $r_0 = 0.5$	Nombre d'Itérations Maximum $M = 200$ Taille de la Population des Particules $N = 50$ Poids d'Inertie $w = 1$ Rapport d'Amortissement du Poids d'Inertie $w_{dump} = 0.99$ Coefficient d'Apprentissage Personnel $c_1 = 1.5$ Coefficient d'Apprentissage Global $c_2 = 2$	Nombre d'Itérations Maximum $M = 200$ Taille de la Population des Fourmis $N = 10$ Taille de l'échantillon $nSample = 40$ Facteur d'Intensification $q = 0.5$ Ratio de Distance d'Ecart $\zeta = 1$	Nombre d'Itérations Maximum $M = 200$ Température Initiale $T_0 = 0.1$ Taux de Réduction de la Température $\alpha = 0.99$	Nombre d'Itérations Maximum $M = 200$ Taille de la Population des Individus $N = 50$ Probabilité $pCR = 0.2$

6.4 Résultats Expérimentaux

Les tableaux de 1 jusqu'à 4 montrent les performances de Bat-Cluster comparé aux autres algorithmes précités appliqués sur les graphes de benchmark.

Tableau 1 - facebook_ego_686

Algorithme	Nombre de Clusters	Valeurs du DBIndex
BC	4	0,72657
PSO	3	0,72731
ACO	3	0,80209
SA	3	0,76827
DE	3	0,73884

Au niveau du graphe ``facebook_ego_686", le Bat-Cluster a fourni la meilleure valeur optimale du DBIndex, suivi de très près par PSO. Cependant, BC était le seul algorithme capable de fournir quatre clusters, tandis que les autres algorithmes n'ont pu fournir que 3 clusters.

Les figures de 7 à 11 présentent l'évolution de la valeur du DBIndex pour chacun des algorithmes testés sur le graphe ``facebook_ego_686".

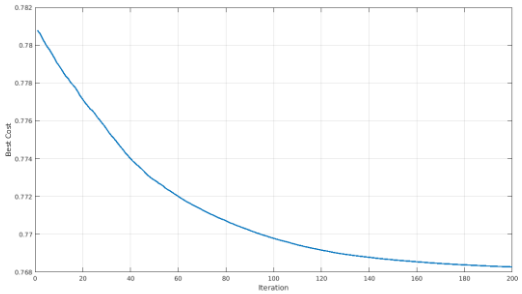

Figure 7 : L'évolution des valeurs optimales du DBIndex fournies par SA pour le graphe facebook_ego_686

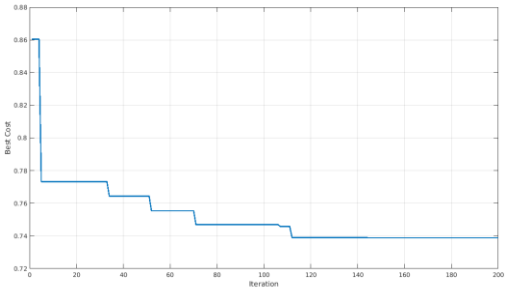

Figure 8 : L'évolution des valeurs optimales du DBIndex fournies par DE pour le graphe facebook_ego_686

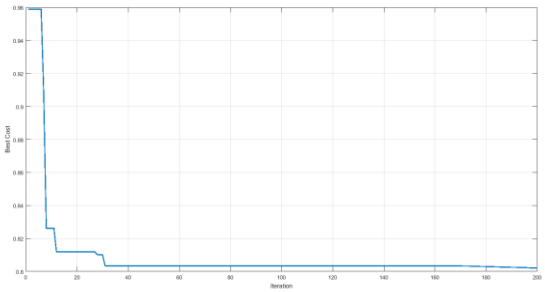

Figure 9 : L'évolution des valeurs optimales du DBIndex fournies par ACO pour le graphe facebook_ego_686

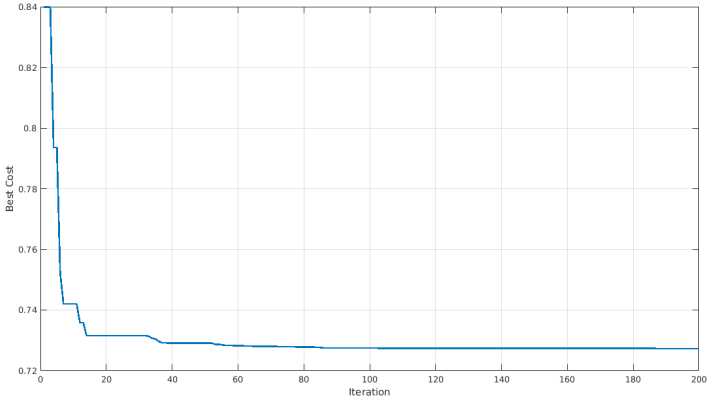

Figure 10 : L'évolution des valeurs optimales du DBIndex fournies par PSO pour le graphe facebook_ego_686

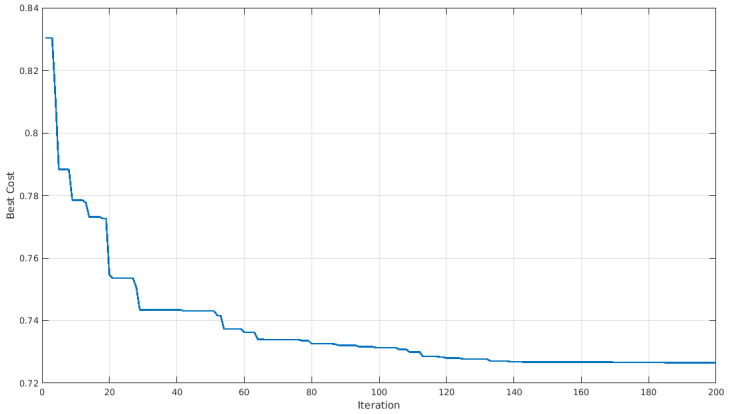

Figure 11 : L'évolution des valeurs optimales du DBIndex fournies par BC pour le graphe facebook_ego_686

La figure 12 présente le clustering du graphe facebook_ego_686 fourni par BC.

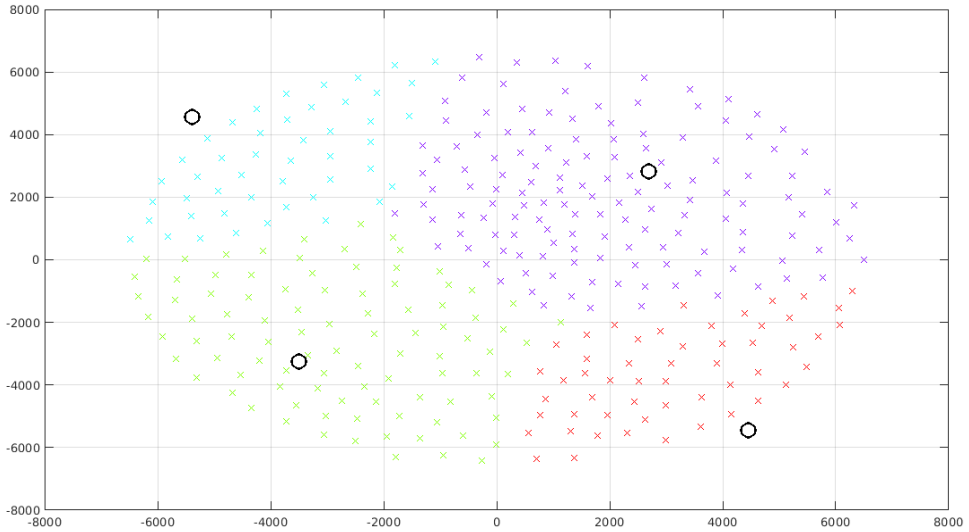

Figure 12 : Clustering du graphe facebook_ego_686 fourni par BC

Tableau 2 - yeast

Algorithme	Nombre de Clusters	Valeurs du DBIndex
BC	4	0,69332
PSO	5	0,69423
ACO	3	0,81152
SA	2	0,73692
DE	3	0,71968

Au niveau du graphe ``yeast", a première vue, on a cette envie de dire que PSO a trouvé le plus grand nombre de clusters, donc c'est lui qui a donné la meilleure performance.

Cependant, si compare les valeurs du DBIndex, on trouve que BC a donné une meilleure valeur pour cette métrique de qualité. Surtout quand on voit, dans la Figure \ref{5-8}, que la valeur du DBIndex retournée par PSO n'a plus évolué depuis l'itération 100.

Donc, on peut conclure qu'avoir 5 clusters n'aurait peut-être pas pu être le meilleur scénario de clustering possible.

Les figures de \ref{5-8} vers \ref{5-204} présentent l'évolution de la valeur du DBIndex pour chacun des algorithmes testés sur le graphe ``yeast".

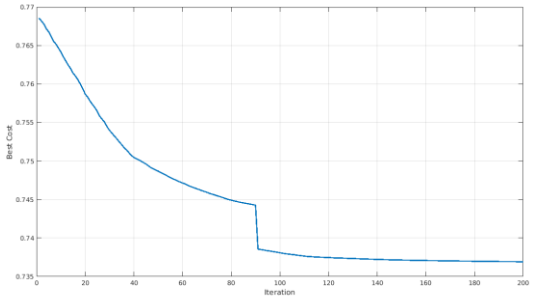

Figure 13 : L'évolution des valeurs optimales du DBIndex fournies par SA pour le graphe yeast

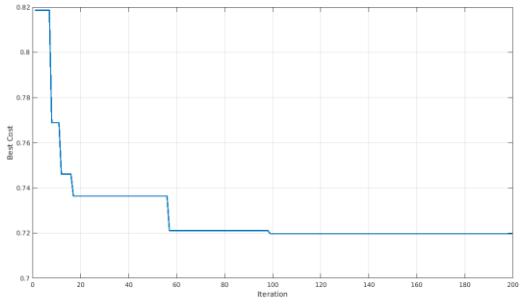

Figure 14 : L'évolution des valeurs optimales du DBIndex fournies par DE pour le graphe yeast

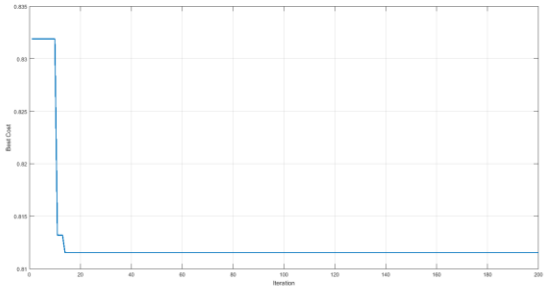

Figure 15 : L'évolution des valeurs optimales du DBIndex fournies par ACO pour le graphe yeast

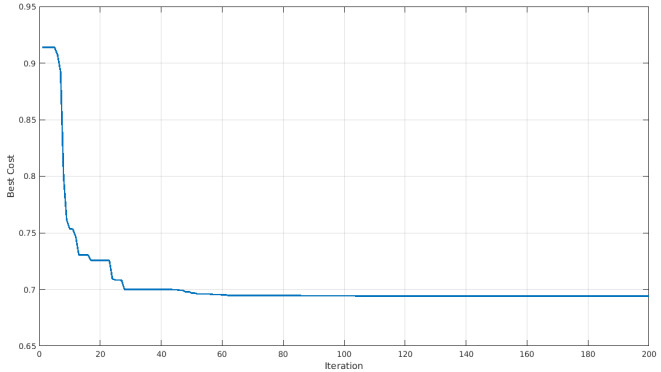

Figure 16 : L'évolution des valeurs optimales du DBIndex fournies par PSO pour le graphe yeast

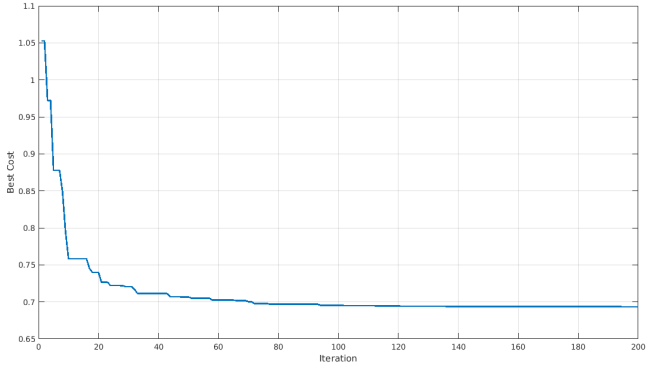

Figure 17 : L'évolution des valeurs optimales du DBIndex fournies par BC pour le graphe yeast

La figure 18 présente le clustering du graphe yeast fourni par BC.

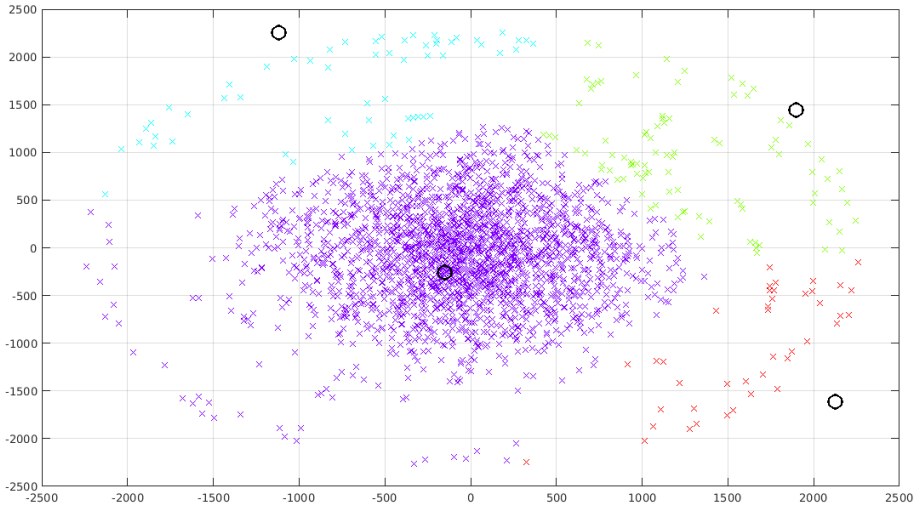

Figure 18 : Clustering du graphe yeast fourni par BC

Tableau 3 - arxiv_general_relativity

Algorithme	Nombre de Clusters	Valeurs du DBIndex
BC	6	0,74797
PSO	4	0,7558
ACO	3	0,81471
SA	3	0,79505
DE	3	0,76049

Au niveau du graphe ``arxiv_general_relativity", BC a fourni la plus petite valeur optimale du DBIndex et beaucoup plus de clusters (6 contre 4 fournis par le second PSO).

Les figures de 19 à 21 présentent l'évolution de la valeur du DBIndex pour chacun des algorithmes testés sur le graphe ``arxiv_general_relativity".

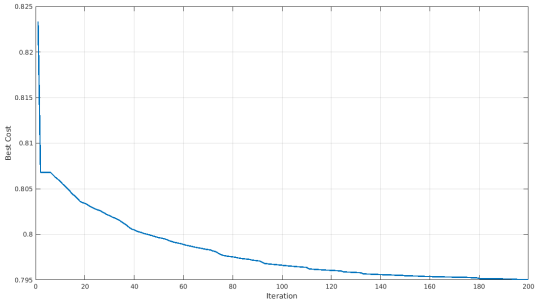

Figure 19 : L'évolution des valeurs optimales du DBIndex fournies par SA pour le graphe arxiv_general_relativity

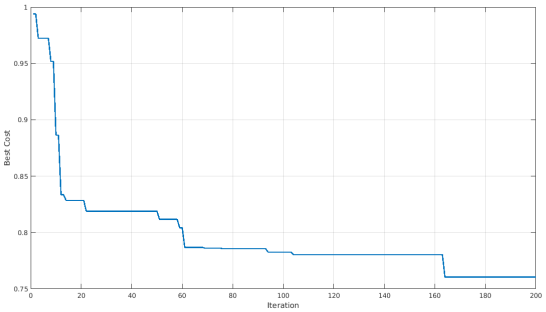

Figure 20 : L'évolution des valeurs optimales du DBIndex fournies par DE pour le graphe arxiv_general_relativity

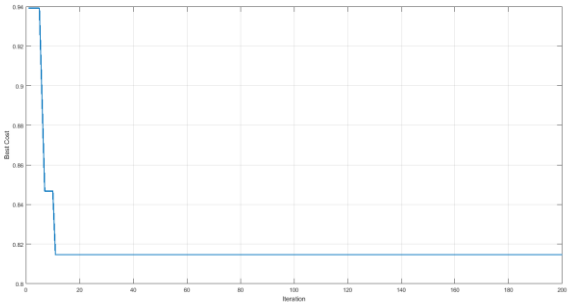

Figure 21 : L'évolution des valeurs optimales du DBIndex fournies par ACO pour le graphe arxiv_general_relativity

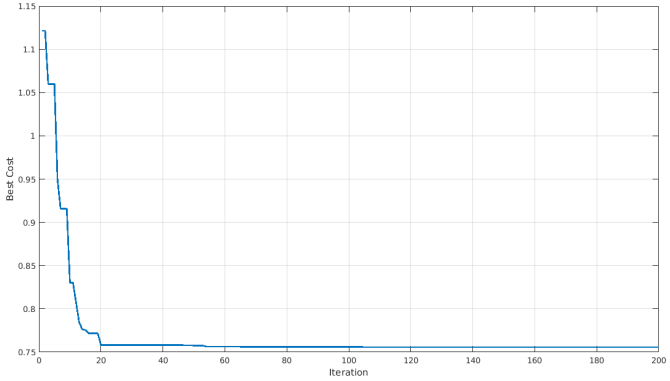

Figure 22 : L'évolution des valeurs optimales du DBIndex fournies par PSO pour le graphe arxiv_general_relativity

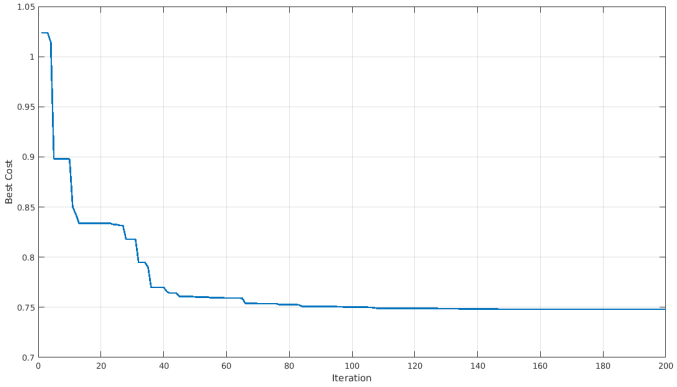

Figure 23 : L'évolution des valeurs optimales du DBIndex fournies par BC pour le graphe arxiv_general_relativity

La Figure 24 affiche le clustering de BC pour ce graphe.

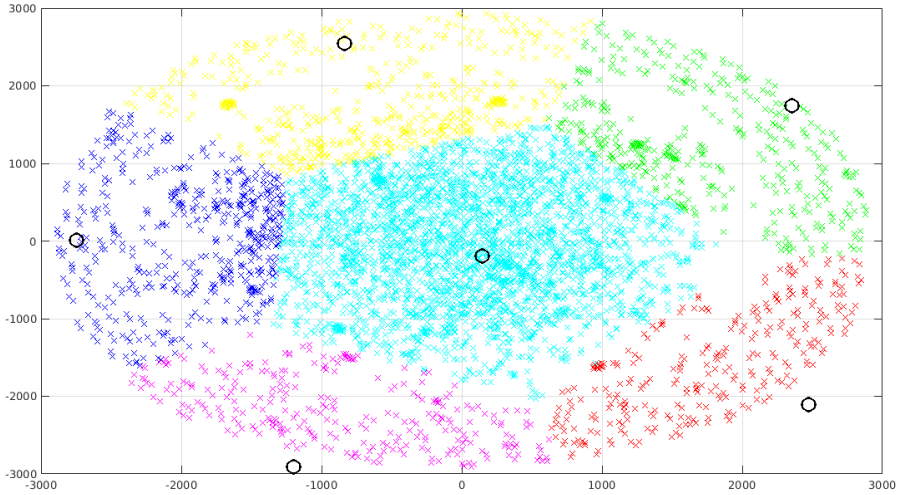

Figure 24 : Clustering du graphe arxiv_general_relativity fourni par BC

Tableau 4 - oregon2_010331

Algorithme	Nombre de Clusters	Valeurs du DBIndex
BC	3	0,58093
PSO	3	0,58101
ACO	2	0,71037
SA	2	0,69676
DE	2	0,62902

Au niveau du graphe ``oregon2_010331", BC et PSO ont été les seuls algorithmes capables de fournir trois clusters tandis que les autres n'ont pu fournir que deux. Les valeurs optimales du DBIndex fournies par BC et PSO étaient très proches avec un léger avantage pour BC.

Les figures de 25 à 29 présentent l'évolution de la valeur du DBIndex pour chacun des algorithmes testés sur le graphe ``oregon2_010331".

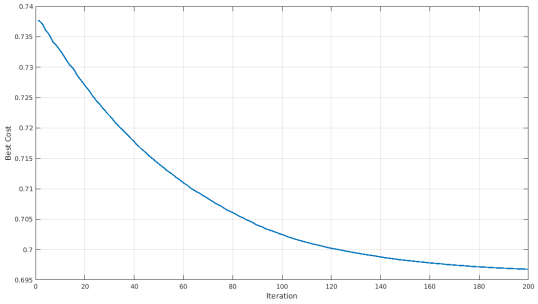

Figure 25 : L'évolution des valeurs optimales du DBIndex fournies par SA pour le graphe oregon2_010331

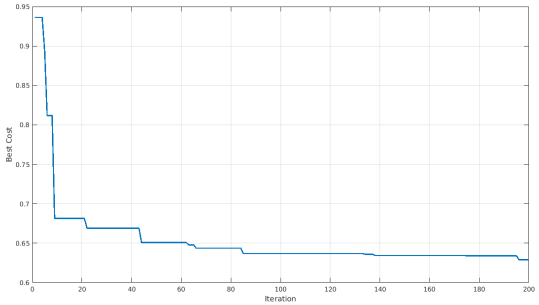

Figure 26 : L'évolution des valeurs optimales du DBIndex fournies par DE pour le graphe oregon2_010331

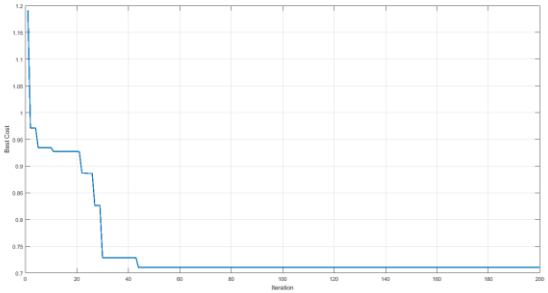

Figure 27 : L'évolution des valeurs optimales du DBIndex fournies par ACO pour le graphe oregon2_010331

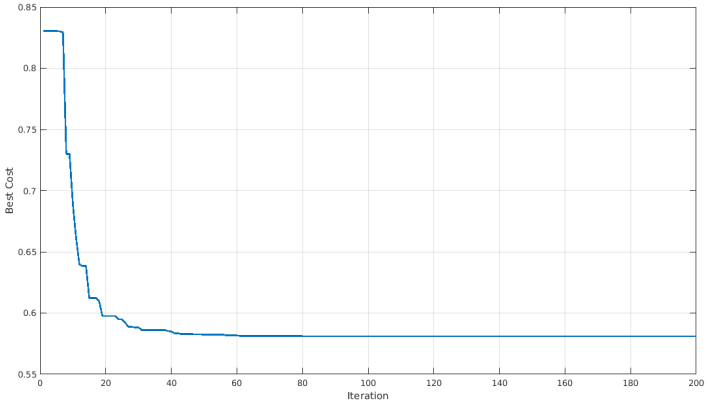

Figure 28 : L'évolution des valeurs optimales du DBIndex fournies par PSO pour le graphe oregon2_010331

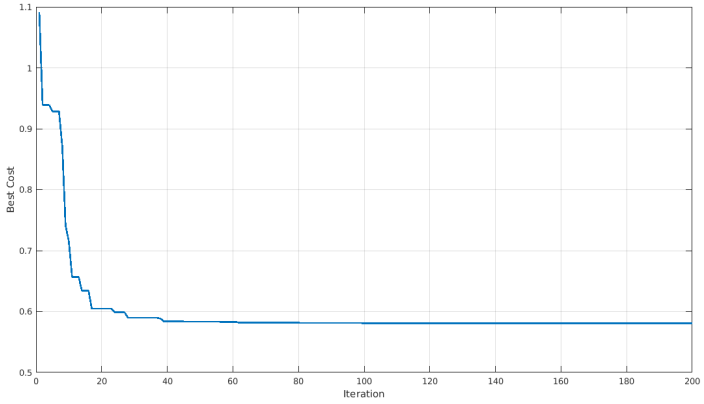

Figure 29 : L'évolution des valeurs optimales du DBIndex fournies par BC pour le graphe oregon2_010331

La Figure 30 affiche le clustering de ce graphe tel que fourni par BC.

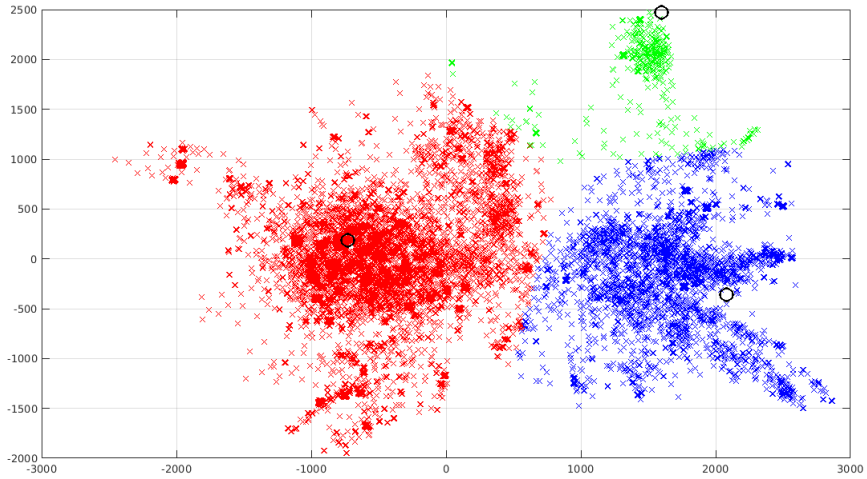

Figure 30 : Clustering du graphe oregon2_010331 fourni par BC

De manière générale, Bat-Cluster était capable de fournir les meilleures valeurs optimales du DBIndex au niveau de tous les graphes de benchmark testés.

Le fait qu'il soit suivi de très près par le PSO démontre la capacité des algorithmes d'optimisation par essaims à résoudre ce genre de problème.

Toutefois, le PSO était capable de trouver une solution optimale un peu plus rapidement que le BC (une moyenne entre 100 et 120 itérations pour le PSO contre une moyenne entre 120 et 140 pour BC).

7 Conclusion

Ce papier a présenté ``Bat-Cluster'', un algorithme capable de donner une solution à la problématique du clustering automatique des larges graphes.

BC est une combinaison entre l'algorithme FFDP que nous avons développé [17] et le Bat Algorithm développé par [14].

Contrairement à la majorité des algorithmes de clustering disponibles dans la littérature qui ont tendance à donner à ce problème une représentation discrète, BC était capable de translater le problème de clustering des larges graphes en un problème continu.

L'idée était d'exécuter un algorithme de visualisation des larges graphes, le FFDP, et faire en sorte qu'il fournisse les coordonnées des positions d'équilibre des nœuds du graphe. Le fait d'avoir ces coordonnées m'a permis de représenter le graphe sous forme d'un vecteur réel facile à résoudre par la version continue du Bat Algorithm.

La métrique que nous avons utilisée pour mesurer la qualité du clustering est l'indice DBIndex de [16].

Le Bat-Cluster a été testé sur quatre graphes de benchmark de différentes tailles et issus de différents domaines.

BC a prouvé qu'il était une bonne alternative pour résoudre la problématique du clustering automatique des larges graphes lorsque je l'ai comparé avec des algorithmes considérés parmi les meilleurs de la littérature.

L'algorithme Bat-Cluster va, par la suite, être intégré dans ``XEWGraph" [44], le service de visualisation des larges graphes offert par le système d'intelligence économique ``Xplor EveryWhere" [45]. Ceci donnera à l'utilisateur de ce service la possibilité d'avoir des graphes clustérisés et étendus à la demande pour les deux interfaces web et mobile de ``XEWGraph".

8 Bibliographie

[1] S. Jianbo and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[2] G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee, “Self-organization and identification of Web communities,” *Computer*, vol. 35, no. 3, pp. 66–70, Mar. 2002.

[3] H. Zhou and R. Lipowsky, “Dynamic pattern evolution on scale-free networks.,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 29, pp. 10052–7, Jul. 2005.

[4] J. a. Hartigan and M. a. Wong, “Algorithm AS 136: A K-Means Clustering Algorithm,” *Journal of the Royal Statistical Society C*, vol. 28, no. 1, pp. 100–108, 1979.

[5] A. K. Jain, “Data clustering: 50 years beyond K-means,” *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[6] A. Fahad *et al.*, “A survey of clustering algorithms for big data: Taxonomy and empirical analysis,” *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 3, pp. 267–279, 2014.

[7] D. Camacho, “Bio-inspired Clustering: basic features and future trends in the era of Big Data,” in *Cybernetics IEEE Conf on*, 2015.

[8] E. R. Hruschka, R. J. G. B. Campello, A. A. Freitas, and A. C. P. L. F. de Carvalho, “A survey of evolutionary algorithms for clustering,” *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 39, no. 2, pp. 133–155, 2009.

[9] H.-L. Ling, J.-S. Wu, Y. Zhou, and W.-S. Zheng, “How many clusters? A robust PSO-based local density model,” *Neurocomputing*, 2016.

[10] H. D. Menendez, D. F. Barrero, D. Camacho, D. M. E. N. Endez, U. Aut, and D. F. Barrero, “A genetic graph-based approach for partitional clustering,” *International journal of neural systems*, vol. 24, no. 03, p. 1430008, 2014.

[11] S. J. Nanda and G. Panda, “A survey on nature inspired metaheuristic algorithms for partitional clustering,” *Swarm and Evolutionary Computation*, vol. 16, pp. 1–18, 2014.

[12] A. K. Kar, “Bio inspired computing - A review of algorithms and scope of applications,” *Expert Systems with Applications*, vol. 59, pp. 20–32, Oct. 2016.

[13] J. Kennedy and R. C. Eberhart, “Swarm intelligence,” *Scholarpedia*, vol. 2, no. 9, p. 541, 2001.

[14] X.-S. Yang, “A New Metaheuristic Bat-Inspired Algorithm,” in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, 2010th ed., vol. 284, J. R. Gonzalez, D. A. Pelta, C. Cruz, T. German, and N. Krasnogor, Eds. Granada: Springer Springer, 2010, pp. 65–74.

[15] G. Komarasamy and A. Wahi, “An Optimized K-Means Clustering Technique Using Bat Algorithm,” *European Journal of Scientific Research*, vol. 84, no. 2, pp. 63–81, 2012.

[16] D. L. Davies and D. W. Bouldin, “DBIndex : A Cluster Separation Measure,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, Apr. 1979.

[17] Z. Boulouard, L. Koutti, A. El Haddadi, and B. Dousset, “‘Forced’ Force Directed Placement: a New Algorithm for Large Graph Visualization,” *International Review on Computers and Software (IRECOS)*, vol. 12, no. 2, pp. 75–83, Mar. 2017.

[18] G. Berthiau and P. Siarry, “État de l’art des méthodes ‘d’optimisation globale,’” *RAIRO - Operations Research*, vol. 35, no. 3, pp. 329–365, Jul. 2001.

[19] S. Alam, G. Dobbie, Y. S. Koh, P. Riddle, and S. Ur Rehman, “Research on particle swarm optimization based clustering: A systematic review of literature and techniques,” *Swarm and Evolutionary Computation*, vol. 17, pp. 1–13, Aug. 2014.

[20] R. Liu, Y. Chen, L. Jiao, and Y. Li, “A particle swarm optimization based simultaneous learning framework for clustering and classification,” *Pattern Recognition*, vol. 47, no. 6, pp. 2143–2152, Jun. 2014.

[21] Q. Cai, M. Gong, B. Shen, L. Ma, and L. Jiao, “Discrete particle swarm optimization for identifying community structures in signed social networks,” *Neural Networks*, vol. 58, pp. 4–13, Oct. 2014.

[22] Q. Cai, M. Gong, L. Ma, S. Ruan, F. Yuan, and L. Jiao, “Greedy discrete particle swarm optimization for large-scale social network clustering,” *Information Sciences*, vol. 316, pp. 503–516, Sep. 2015.

[23] S. Suganthi and S. P. Rajagopalan, “Multi-Swarm Particle Swarm Optimization for Energy-Effective Clustering in Wireless Sensor Networks,” *Wireless Personal Communications*, vol. 94, no. 4, pp. 2487–2497, Jun. 2017.

[24] J. Rejina Parvin and C. Vasanthanayaki, “Particle Swarm Optimization-Based Clustering by Preventing Residual Nodes in Wireless Sensor Networks,” *IEEE Sensors Journal*, vol. 15, no. 8, pp. 4264–4274, Aug. 2015.

[25] S. Goss, D. Fresneau, J. L. Deneubourg, J.-P. Lachaud, and J. Valenzuela-Gonzalez, “Individual foraging in the antPachycondyla apicalis,” *Oecologia*, vol. 80, no. 1, pp. 65–69, Mar. 1989.

[26] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: optimization by a colony of cooperating agents,” *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[27] M. Ali Jan Ghasab, S. Khamis, F. Mohammad, and H. Jahani Fariman, “Feature decision-making ant colony optimization system for an automated recognition of plant species,” *Expert Systems with Applications*, vol. 42, no. 5, pp. 2361–2370, Apr. 2015.

[28] M. Mandloi and V. Bhatia, “Congestion control based ant colony optimization algorithm for large MIMO detection,” *Expert Systems with Applications*, vol. 42, no. 7, pp. 3662–3669, May 2015.

[29] S. R. Mandala, S. R. T. Kumara, C. R. Rao, and R. Albert, “Clustering social networks using ant colony optimization,” *Operational Research*, vol. 13, no. 1, pp. 47–65, Apr. 2013.

[30] J. Ji, X. Song, C. Liu, and X. Zhang, “Ant colony clustering with fitness perception and pheromone diffusion for community detection in complex networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 15, pp. 3260–3272, Aug. 2013.

[31] X. Zhou, Y. Liu, J. Zhang, T. Liu, and D. Zhang, “An ant colony based algorithm for overlapping community detection in complex networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 427, pp. 289–301, Jun. 2015.

[32] P. Moradi and M. Rostami, “Integration of graph clustering with ant colony optimization for feature selection,” *Knowledge-Based Systems*, vol. 84, pp. 144–161, 2015.

[33] S. Gao, Y. Wang, J. Cheng, Y. Inazumi, and Z. Tang, “Ant colony optimization with clustering for solving the dynamic location routing problem,” *Applied Mathematics and Computation*, vol. 285, pp. 149–173, Jul. 2016.

[34] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[35] V. Černý, “Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm,” *Journal of Optimization Theory and Applications*, vol. 45, no. 1, pp. 41–51, Jan. 1985.

[36] S. G. Sandeep, C. Rajeevan, and K. C. Ashwani, “Efficient clustering and simulated annealing approach for circuit partitioning,” *Journal of Shanghai Jiaotong University (Science)*, vol. 16, no. 6, pp. 708–712, Dec. 2011.

[37] I. A. Hamad, P. A. Rikvold, and S. V. Poroseva, “Floridian high-voltage power-grid network partitioning and cluster optimization using simulated annealing,” *Physics Procedia*, vol. 15, pp. 2–6, 2011.

[38] F. Rossi and N. Villa-Vialaneix, “Optimizing an organized modularity measure for topographic graph clustering: A deterministic annealing approach,” *Neurocomputing*, vol. 73, no. 7–9, pp. 1142–1163, Mar. 2010.

[39] C.-H. Mu, J. Xie, Y. Liu, F. Chen, Y. Liu, and L.-C. Jiao, “Memetic algorithm with simulated annealing strategy and tightness greedy optimization for community detection in networks,” *Applied Soft Computing*, vol. 34, pp. 485–501, Sep. 2015.

[40] R. Storn and K. Price, “Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces,” *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[41] S. Paterlini and T. Krink, “Differential evolution and particle swarm optimisation in partitional clustering,” *Computational Statistics & Data Analysis*, vol. 50, no. 5, pp. 1220–1247, Mar. 2006.

[42] Y. Cai, J. Liao, T. Wang, Y. Chen, and H. Tian, “Social learning differential evolution,” *Information Sciences*, Oct. 2016.

[43] E. Zorarpaci and S. A. Özel, “A hybrid approach of differential evolution and artificial bee colony for feature selection,” *Expert Systems with Applications*, vol. 62, pp. 91–103, Nov. 2016.

[44] Z. Boulouard, A. El Haddadi, A. El Haddadi, L. Koutti, and A. Fennan, “XEWGraph : Outil de Visualisation et Analyse des Hypergraphes pour un Système d’Intelligence Economique,” *Revue des Nouvelles Technologies de l’Information*, pp. 479–480, 2015.

[45] A. El Haddadi, “Fouille multidimensionnelle sur les données textuelles visant à extraire les réseaux sociaux et sémantiques pour leur exploitation via la téléphonie mobile,” Université de Toulouse III, Paul Sabatier, 2011.