A Polymorphic Ant-Based Algorithm for Graph Clustering

by

Ying Ying Liu

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
March 2016

Thesis advisor

Parimala Thulasiraman

Author

Ying Ying Liu

A Polymorphic Ant-Based Algorithm for Graph Clustering

Abstract

In this thesis, I introduce two new algorithms: *Ant Brood Clustering-Intelligent Ants (ABC-INTE)* and *Ant Brood Clustering-Polymorphic Ants (ABC-POLY)* for the graph clustering problem. ABC-INTE uses techniques such as hopping ants, relaxed drop function, ants with memories, stagnation control, and addition of k-means cluster retrieval process, as an improvement of the basic ABC-KLS algorithm. ABC-POLY uses two types of ants, inspired by the division of labour between the major and minor ants in *Pheidole* genus, as an improvement of ABC-INTE. For comparison purpose, I also implement MMAS, an ACO clustering algorithm. When tested on the benchmark networks, ABC-POLY outperforms or achieves the same *modularity* values as MMAS and ABC-INTE on 7 out of 10 networks and is robust against different graphs. In practice, the speed of ABC-POLY is at least 10 times faster than MMAS, making it a scalable algorithm compared to MMAS. ABC-POLY also outputs a direct visual representation of the natural clusters on the graph that is appealing to human observation. This thesis opens an interesting research topic to apply polymorphic ants for graph clustering in the ABC-POLY algorithm. The distributive and self-organization nature of ABC-POLY makes it a candidate for analyzing clusters in more complex and dynamic graphs.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to begin by thanking my advisor, Dr. Parimala Thulasiraman, a phenomenal mentor who saw the potential in me and showed me the blueprint of being a computer scientist. I would not be here without her tremendous encouragement and trust.

I am fortunate to have learned from many outstanding professors in the department of Computer Science. I have a lot of respect for their knowledge and integrity.

I am grateful to Natural Sciences and Engineering Research Council of Canada for their generous support through the Canada Graduate Scholarship-Masters (CGS M) program.

Lastly, to my wonderful family: my parents, my husband, and my beautiful children Viyanna and baby Yona, who lightened my life.

For the people I love.

Chapter 1

Introduction

Over the past years we have moved towards the big data era. Real systems such as biological, social and information networks produce tremendous amounts of data that require a systematic approach to capture the relevant information in the data. One approach is to model these systems as graphs. The objects in the graph can be represented as vertices while the relationship between these objects can be represented as edges. These graphs can be studied and analyzed to capture important features of the system. The graphs representing real systems are called *real networks*.

One of the important problems in many real networks is the detection of communities. The communities are organized such that some groups of vertices are clustered together (many edges connected to vertices). This problem is called the *graph clustering* problem. There is no universally accepted definition of graph clustering (Schaeffer [2007]). Traditionally, the problem of graph clustering is closely related to graph partitioning problem, which refers to the task of dividing the vertex set V of a given graph G into k non-empty groups such that number of edges (or edge weights for

weighted graphs) between vertices in different groups is minimized (Newman [2004]). For an arbitrary k, the graph clustering problem is NP-complete (Garey et al. [1976]). The partition heuristics commonly use similarity measurements based on vertex connectivity to guide the algorithms (Fortunato [2010]).

Alternatively, Newman and Girvan (Newman [2004]) argue that in the case of graph clustering, it is not necessary to strictly minimize the number of inter-cluster edge. This is because, they mention, that graph clustering has no restriction on cluster sizes and large clusters tend to have more inter-cluster edges contrary to small clusters . They propose the concept of *modularity* to measure the difference between intra-cluster connectivity and the expected random connectivity given the same vertex degrees. If a clustering C is of good quality, its intra-cluster vertices should be more strongly connected than it would occur by random chance. The problem of graph clustering is then an optimization problem, which is maximizing the value of *modularity*.

There are numerous approximate methods to solve the graph clustering problem. Well-known traditional heuristics such as graph partition algorithms, hierarchical clustering algorithms, random walks (Fisher [1984]), greedy modularity optimization (Newman [2004]), and simulated annealing (Kirkpatrick et al. [1983]) are described in chapter 2 of this thesis. (Fortunato [2010]) and (Schaeffer [2007]) provide a comprehensive survey of graph clustering algorithms.

Although the above mentioned existing techniques in the literature work well for static networks, dynamic networks such as online social networks with millions of nodes, and with edge insertions and deletions that happen almost instantly and dy-

namically, require new techniques. More specifically, the challenges of modern graph clustering can be summarized as three-fold: *scalability*, *optimality*, and *adaptability*. The massive graph size creates challenges in producing efficient graph clustering algorithms. On the other hand, scalable algorithms such as the greedy algorithm do not have good optimality (Newman [2004]). Furthermore, scalable clustering algorithm development that adapts to the dynamic change of graph structure is still in its early stage (Fortunato [2010]).

Swarm intelligence refers to models and systems that are inspired by the collective behavior of social insects and animals to solve complex problems (Bonabeau et al. [1999]). For example, with limited individual capabilities, the *Atta* (leafcutter ants) find leaves hundreds of meters away from the nest ([Bonabeau et al., 1999, [figure 1.1]), and the *Parachartergus* (tropical wasp) build nests with parallel horizontal combs ([Bonabeau et al., 1999, figure 1.6(a)]), in the complex and unpredictable environment. The research of several nature-inspired algorithms, such as Ant Colony Optimization (Dorigo et al. [1991]), Particle Swarm Optimization (Kennedy [2011]), and Artificial Bee Colony (Karaboga and Basturk [2007]), has shown that swarm intelligence is not just a proof-of-concept, but a valuable alternative to traditional approaches.

In this thesis, I study two ant based algorithms: Ant Colony Optimization (ACO) and Ant Brood Clustering (ABC). Both inspired by the behaviors of ant species in nature, these two algorithms are applied to the graph clustering problem in very different approaches. ACO imitates the capability of ants to find the shortest path from their nest to the food source. Using the metaphors from ant systems, ACO

has a general-purpose framework that can be tailored to solve different optimization problems. In the case of graph clustering, the solution of ACO is a locus-based adjacency representation (Park and Song [1998]) of the connected components in a subgraph G' that reveals the community structure of the original graph G (Chang et al. [2013]). The objective of ACO is to optimize *modularity* value. While ACO is less related to the real ants, ABC is a direct simulation of ants moving and segregating brood items according to their characteristics (Deneubourg et al. [1991]; Lumer and Faieta [1994]; Kuntz et al. [1997].) The input to this algorithm is a random projection of the graph vertices on a geometric space, and its output is a direct visual display of *natural* clusters in the graph such that vertices with more connections are placed close to each other, and those with less connections are placed far from each other. The extraction of explicit membership information is done based on the spatial (Euclidean) distance among the vertices. Compared to ACO, the study of ABC is still in it's infancy. Questions raised by previous research on this algorithms include the quality measurement and efficiency of this algorithm (Bonabeau et al. [1999]; Handl et al. [2006]).

In this thesis, I conduct an extensive study of this algorithm, and show that with modifications, ABC can perform well in terms of both solution quality and speed. Given its *self-organization* and *distributed* nature, ABC shows the potential of becoming a promising method to find clusters in complex, large and dynamic graphs.

My contribution in the thesis is as follows: I first study the ACO Max-Min Ant System (MMAS) algorithm proposed by (Chang et al. [2013]) to solve graph clustering

problem. Then, I take a different approach by looking at the more natural way of clustering offered by ABC. I start by studying the ABC-KLS algorithm proposed by (Kuntz et al. [1997]). Then, I introduce *ABC-INTE* (Chan et al. [2016]) that employs more intelligent ants than the KLS algorithm. Next, I propose *ABC-POLY* that contains heterogeneous worker types within the ants, inspired by *division of labor* in polymorphic species of ants. To my knowledge, the design of ABC-POLY has not been studied in the literature.

The rest of the thesis is as follows. Chapter 2 states the problem of graph clustering and discusses literature review for this problem. Chapter 3 explains the ACO-MMAS algorithm. Chapter 4 explains ABC-KLS, ABC-INTE and ABC-POLY algorithms. Chapter 5 evaluates the performance of ACO-MMAS, ABC-INTE and ABC-POLY on 10 real-life networks, and Chapter 6 concludes this thesis.

Chapter 2

Literature Review

Depending on the graph instance, there might be different quantitative definitions of a community for the graph clustering problem (Fortunato [2010]). Intuitively, it is expected that edges within the same communities (intra-cluster edges) are denser than edges between communities (inter-cluster edges).

The local definition of graph clustering deals with a single community. A *strong* community is a subgraph such that each vertex within it has more intra-cluster edges than inter-cluster edges, and a *weak* community is a subgraph such that the sum of its intra-cluster edges exceeds the sum of its inter-cluster edges. A strong community is also a weak community, but a weak community is not necessarily a strong community (Radicchi et al. [2004]).

In the global definition of graph clustering, the graph is considered as a whole. If a clustering C is of good quality, its intra-cluster vertices should be more strongly connected than it would occur by random chance (Newman [2004]). This difference in connectivity is measured by *modularity*.

Another definition of graph clustering is based on vertex *similarity* (Fortunato [2010]). It is assumed that vertices within a community are more similar to each other than to those in other communities. If there are no attributes associated with the graph nodes, the measurement of similarity can be based on adjacency relationships of nodes. For example, two nodes are more similar if they share more common neighbours.

The ant based algorithms I study in this thesis use mainly vertex *similarity* measurements to guide the ants' behavior, but also use *modularity* to influence the ants and measure the quality of the clustering. Nonetheless, I define the problem as follows to reflect the major role of similarity.

2.1 Problem Statement

Given an undirected graph $G = (V, E)$, where V denotes the set of vertices, E the set of edges and $|V|$ the number of vertices, the graph clustering problem can be defined as dividing the vertices into k disjoint sets, V_1, V_2, $\ldots$, V_k, such that the following criteria are met for all $1 \leq i \leq k$:

- $\bigcup_{i=1}^{k} V_i = V$
- $\bigcap_{i=1}^{k} V_i = \emptyset$
- $\forall u_i \in V_i, u_j \in V_i, u_l \notin V_i$, $Similarity(u_i, u_j) \geq Similarity(u_i, u_l)$

In this thesis, I consider graphs with no edge weights. Clustering of weighted graphs is a topic for the future work.

2.2 Heuristics For Graph Clustering Problem

2.2.1 Traditional Methods

The concept of graph clustering is closely related to graph partitioning. Therefore, graph partitioning methods can be used to retrieve the clusters in a graph. Well-known partitioning algorithms include Kernighan-Lin algorithm (Kernighan and Lin [1970]) that make local interchanges to an initial partition, spectral bisection method (Pothen et al. [1990]) that computes vertex separator based on eigenvalues of the Laplacian matrix associated with a graph, and k-means algorithm (MacQueen et al. [1967]) that assigns all vertices to k centroids in each iteration. Graph partitioning algorithms require the number of clusters, k, to be known in advance. It is possible to try different k values and choose the k that results in the best clustering result. However, the process is at least tedious for small graphs, and unfeasible for large graphs with unknown structures.

Another important class of algorithms are the hierarchical clustering algorithms. These algorithms are common to analyze social and biological network graphs that exhibit hierarchical structure, i.e. a node may belong to a small cluster, which in turn belongs to a big cluster, and even a bigger cluster, and so forth. The clusters in such graphs can be best illustrated as trees, or dendrograms (Newman [2004]), in which the leave nodes represent all the vertices of the graph, and the root node represents the entire graph as a single cluster. In general, hierarchical clustering algorithms can be grouped into two categories: agglomerative algorithms and divisive algorithms (Newman [2004]). Agglomerative algorithms work in a bottom-up fashion. They

start by making each node a unique cluster, and iteratively merge clusters with high similarities. The dendrogram is built from the leave nodes towards the root node. Divisive algorithms work in the opposite direction. They take the entire graph as input and iteratively removes edges with high edge betweenness.The algorithm of Girvan and Newman (GN) (Newman [2004]) is an example of a divisive algorithm.

Random walk (Zhou [2003]) is an approach to detect community in dynamic graphs. It is assumed that a random walker spends more time travelling inside a community than outside the community. The distance between two vertices is calculated as the average number of edges that the random walker encounters along the way. The shorter the distance, the more likely that the two vertices belong to the same community. The divisive iteration of the algorithm starts from the entire graph, and ends when the dissimilarity within the community is less than a threshold. The complexity of random walk is $O(n^3)$, making it a slow algorithm.

2.2.2 Modularity Based Methods

The concept of *modularity* was first proposed in GN as a stopping criteria of the algorithm. It is defined as the difference of the fraction of edges within the communities and would occur in random connection with the same degrees. Consider an undirected graph $G = (V, E)$. Let m be the number of edges of G. Let A be the adjacency matrix to represent G. $A_{ij} = 1$ if there is an edge between vertex i and j and 0, otherwise. The degree of vertex i is then defined as $a_i = \sum_j A_{ij}$. The expected probability of vertex i to form an edge is then $a_i/2m$, where $2m$ is all the *stubs*, or half-edges belonging to each vertices. The expected number of edges between i and

j is then $2m * (a_i/2m) * (a_j/2m) = a_i a_j/2m$. The modularity, Q, is then defined as

$$Q = \frac{1}{2m} \sum_{i,j \in V} [A_{ij} - \frac{a_i a_j}{2m}] \delta(C_i, C_j) \tag{2.1}$$

where C_i and C_j refer to the cluster IDs of vertices i and j, and $\delta(C_i, C_j)$ refers to Kronecker delta function that yields 1 if $C_i = C_j$ and 0, otherwise. Instead of measuring the quality of each individual cluster, *modularity* measures the overall clustering quality of the entire clustering and is considered a quality function of *global* clustering. Despite some known problems such as fluctuations in random graphs (Guimera et al. [2004b]) and resolution limit (Lancichinetti and Fortunato [2011]), *modularity* has become the most popular choice of quality measurements of graph clustering (Fortunato [2010]) and the basis of a large group of clustering algorithms.

The value of *modularity* lies in $[-\frac{1}{2}, 1]$ (Brandes et al. [2008]). Modularity tends to be 0 for a completely random graph, and $1 - \frac{1}{|C|}$ for a perfectly modular graph with $|C|$ equally sized clusters (Ziv et al. [2005]). The value of modularity reaches 1 when $|C|$ goes to infinity. It is shown that this maximum value can only be obtained from a specific graph with no edges. The minimum value $-\frac{1}{2}$ is obtained from a bipartite graph with the canonic clustering where all the edges are inter-cluster edges (Brandes et al. [2008]). Since a high value of *modularity* seems to indicate good clustering, it is natural to transform the objective of graph clustering to the maximization of modularity. Unfortunately,*modularity* optimization is again an NP-complete problem (Brandes et al. [2008]). In the following, a few heuristics for optimizing *modularity* are discussed.

The greedy algorithm by (Clauset et al. [2004]) amalgamates communities in pairs

as long as the union increases (or does not decrease) the modularity value. A matrix of ΔQ_{ij} is maintained and updated to avoid repetitive computation. This algorithm has a complexity of $O(mdlogn)$ for a graph with n vertices, m edges and d, the depth of the dendrogram. It can be applied to graphs with up to 1 million vertices (Fortunato [2010]). Its disadvantage is that the accuracy is not as good as other techniques (Newman [2004]).

Simulated Annealing (SA) (Kirkpatrick et al. [1983]) is a probabilistic metaheuristic for global optimization. The motivation of SA comes from heating and slow cooling of a material to find the empirical ground state of the matter the optimal configuration of which is statistically unlikely. Given a function F, the task of the SA algorithm is to find the global optimum of F, in this case, the maximum number of *modularity*. In each state of the algorithm, an atom is given a small random displacement, and if F increases, then the transition occurs with probability 1, otherwise, the probability is inversely related to the decrease of F. A smaller decrease in F results in a bigger probability of the change. SA provides good solution quality, but like many other metaheuristics, it is slow and not feasible for large graphs (Guimera et al. [2004a]).

2.2.3 Ant Colony Optimization Clustering

For the graph clustering problem, recent literature shows solution optimality when using ACO to find clusters on small graphs. (Chang et al. [2013]) propose a tailored ACO Max-Min Ant System (MMAS) (Stützle and Hoos [2000]) algorithm to solve the community detection problem. The authors implement their algorithm on one synthetic network and four small real networks and compare the results with other

approximation algorithms in terms of *modularity* and normalized mutual information (NMI). The authors show that MMAS is as good as other algorithms for the synthetic network, and achieved better *modularity* than other algorithms did for real-life networks. Similar comments on the optimality of ACO were made by (Jin et al. [2011]) who developed a hybrid ACO-random walk algorithm (ACOMRW) and (Ji et al. [2013]) who took the framework of ACO and replaced the solution construction with a fitness function in their Ant Colony Clustering algorithm based on Fitness perception and Pheromone diffusion (ACC-FP) algorithm.

2.2.4 Ant Brood Clustering

Ant brood clustering (ABC), also called Ant-based Clustering in the literature, is based on how ants cluster their brood or corpse into different shapes and sizes. (Handl et al. [2006]) compare an improved version of the basic algorithm for data clustering, called ATTA, with k-means, average-link agglomerative hierarchical clustering and one-dimensional self-organizing maps (1D-SOM) and conclude that the strengths of the algorithm include scalability, capability to work with any kind of data that can be described in terms of symmetric dissimilarities and automatically determine the number or clusters k, with no assumption on the shape of the clusters.

(Kuntz et al. [1999]) propose to apply ant brood clustering (the ABC-KLS algorithm) to solve graph partitioning problem. Instead of directly working with graph, the authors propose a bijective mapping between the graph vertices and points on a 2-D geometric space. The graph partitioning problem is then transformed to finding the "natural" clusters on the plane by moving similar points close to each other. The

KLS algorithm is one of the few attempts to apply ABC to graph problems. Although the algorithm was able to derive clearly separated clusters of points from an initial random distribution on the plane, it was only tested on small pseudo-random graphs with a few hundred vertices and small number of clusters. There was also no qualitative comparison with existing algorithms in graph partitioning/clustering.

Chapter 3

Ant Colony Optimization

Ant Colony Optimization (ACO) (Dorigo et al. [1991]) is inspired by the foraging behaviour of ants searching for food. In nature, ants leave pheromone trails for the other members of the colony to follow their routes from nest to food source. The pheromones evaporate over time. As more ants travel a given path, the intensity of the pheromone increases. This leads to a better route and a shorter route.

The ACO algorithm was first applied to the traveling salesman problem (TSP). The algorithm works in an iterative fashion. Initially, all the routes between two cities are initialized with the same amount of artificial pheromone. In each iteration, two steps are performed: *solution construction* and *pheromone update*. In the phase of solution construction, m virtual ants are scattered on n cities. Each ant follows a stochastic rule to choose the next city to visit, using a probability function that favors shorter tour and strong pheromone level. The phase ends when each ant completes a closed tour containing all n cities. In the phase of pheromone update, pheromone level on all the routes evaporates (decreases) followed by an increment on the shortest

tour in the iteration. The pheromone serves as a global memory, or an *exploitation*, to reinforce the local optimal. The stochastic rule allows different ants to find different solutions in different iterations and gives the algorithm enough *exploration* to find possibly better solutions.

The framework of ACO is, indeed, a higher level abstraction that can be tailored to solve different optimization problems. There is a large amount of literature in the application of ACO. For example, it has been applied to job-shop scheduling (Colorni et al. [1994]), network routing (Wang et al. [2009]; Rana et al. [2013]; Di and Dorigo [2011]), option pricing (Kumar et al. [2009]), and graph clustering (Chang et al. [2013]; Jin et al. [2011]; Ji et al. [2013]), to name a few.

3.1 Max-Min Ant System (MMAS) Clustering

In my thesis, I implement the MMAS (Chang et al. [2013]) algorithm for experimentation and comparison.

3.1.1 Locus-based Adjacency Representation

Given a graph G, a locus-based adjacency representation (Park and Song [1998]) of the clustering solution is an array of n, the number of nodes. Each index-element pair represents an edge in the subgraph G', the connected components of which represent communities in the original graph G. To explain this representation, let's consider a toy graph in figure 3.1 with three clusters, and each cluster includes a k-5 subgraph. The locus-based adjacency representation of a clustering solution, S_{locus}, and its corresponding G' are represented in figure 3.2. A breath-first search (BFS) is

then applied to S_{locus} to find all the connected components, which are given unique membership ID's, as shown in figure 3.3.

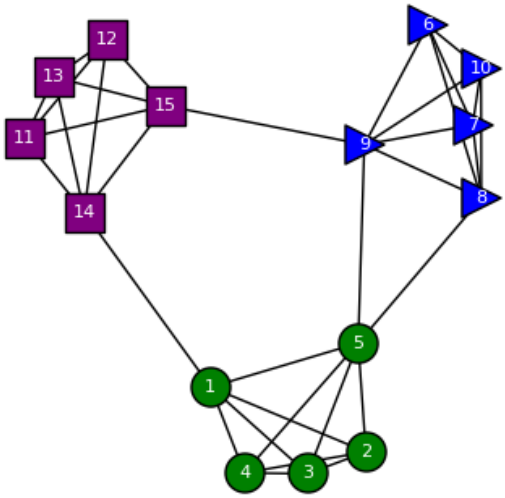


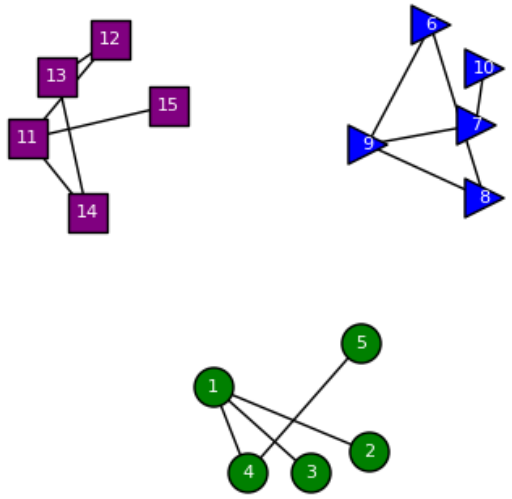
Figure 3.1: A Toy Graph with 3 Clusters

3.1.2 Solution Construction

The algorithm is divided into three phases: pheromone initialization, iterative solution construction and pheromone update.

The value of initial pheromone can be arbitrary, as long as all the edges are deposited with the same amount of pheromone level. In this implementation, the value is set to 1 for simplicity.

$$\tau_{ij} = 1, \quad \forall e_{ij} \in E \tag{3.1}$$



Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_{locus}	4	1	1	5	4	9	9	6	8	7	12	13	14	11	11

Figure 3.2: One Locus-based Adjacency Representation of Figure 3.1

Vertex	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Membership	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3

Figure 3.3: Clustering Solution of Figure 3.1

At the beginning of each iteration, each ant is assigned a different vertex to construct S_{locus} from an empty array. Let v_i denote the current vertex that the ant is at. Let v_j be one of the vertex in v_i's neighbourhood N_i. The probability for an ant to travel to v_j is evaluated with equation 3.2, where τ_{ij} is pheromone level on edge e_{ij}, η_{ij} is heuristic information on e_{ij}, and α and β are ACO parameters that determine the relative importance of pheromone and heuristic information.

$$p(i,j) = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{l \in N_i}(\tau_{il})^{\alpha}(\eta_{il})^{\beta}} \tag{3.2}$$

The heuristic information η_{ij} incorporates information in the graph structure that forms the basis of the algorithm. Pearson correlation is widely used in statistics to determine linear dependence between two items. It has also been used in some graph clustering algorithms as a measurement of vertex similarity (Fortunato [2010]). In equation 3.3, C_{ij} denotes the Pearson correlation between vertex i and j, A_{il} denotes one entry in row i and column l of adjacency matrix A, μ_i is the average degree of vertex i, or average of row i, calculated by $\sum A_{il}/n$. σ_i is standard deviation of row i, calculated by $\sqrt{\sum (A_{il} - \mu_i)^2/n}$. The value of C_{ij} is in $[-1, 1]$, with -1 for total negative correlation, 1 for total positive correlation, and 0 for no correlation. Since C_{ij} can have negative values, it can not be used directly as η_{ij}. Therefore, we normalize C_{ij} in equation 3.4. The more similarity between vertex v_i and v_j, the closer to 1 is C_{ij}, and the larger is η_{ij}. As will be described shortly, a higher pheromone level indicates that this particular edge was in the solution vector in at least one of the previous iterations. Therefore, the higher the pheromone level, and the more similar the vertices, the more likely that the edge is included in the solution vector.

$$C_{ij} = \frac{\sum_{l \in V}(A_{il} - \mu_i)(A_{jl} - \mu_j)}{n\sigma_i\sigma_j} \tag{3.3}$$

$$\eta_{ij} = \frac{1}{1 + e^{-C_{ij}}} \tag{3.4}$$

3.1.3 Pheromone Update

At the end of each iteration, all the solutions found by ants are evaluated with *modularity* value. The highest *modularity* in the iteration is called Q_{ib}, and its corresponding solution is called S_{ib}. Now, pheromone level on the edges are updated in two opposite directions. First, pheromone level on all the edges only pertains a fraction $\rho \in (0, 1)$ of the original trail, imitating the pheromone evaporation in nature and acting as negative feedback to suppress non-optimal solutions from the search space in later iterations. Second, edges in S_{ib} receives extra pheromone deposition. This pheromone increment acts as a positive feedback to "remember" good solutions and give them higher probabilities to be chosen in later iterations.

In order to avoid stagnation, a maximum, η_{max}, and minimum, η_{min}, are applied to limit the range of pheromone level. These two values are calculated in equation 3.5:

$$\begin{aligned} \eta_{max} &= Q_{gb}/(1-\rho) \\ \eta_{min} &= \epsilon \eta_{max} \end{aligned} \tag{3.5}$$

where Q_{gb} is global best modularity so far, ρ is the pheromone persistence parameter, and $\epsilon \in (0, 1)$ is a rate coefficient parameter.

The complete formula to update pheromone is shown in equation 3.6. The pseudo code of the MMAS algorithm given in Algorithm 1.

An obvious advantage of the MMAS algorithm is that it automatically determines the number of clusters, k.

$$\eta_{ij} = \begin{cases} \rho\eta_{ij} + Q_{ib} & \text{if } e_{ij} \in S_{ib} \\ \rho\eta_{ij} & \text{if } e_{ij} \notin S_{ib} \\ \eta_{max} & \text{if } \eta_{ij} > \eta_{max} \\ \eta_{min} & \text{if } \eta_{ij} < \eta_{min} \end{cases} \tag{3.6}$$

Algorithm 1: Max-Min Ant System Clustering

Input: G, N_{ants}, N_{iter}, α, β, ρ, ϵ

Output: C after N_{iter} iterations

`// Initialization`

Initialize pheromone (equation 3.1);

Calculate heuristic information (equation 3.4);

for $iter \leftarrow 1$ **to** N_{iter} **do**

`// Solution Construction`

  foreach *ant,* a_i **do**

    $S_i \leftarrow \emptyset$;

    while $|S_i| < n$ **do**

      construct solution (equation 3.2);

    end

    BFS solution vector S_i to construct clustering vector C_i;

    calculate Q_i (equation 2.1);

  end

`// Pheromone Update`

  calculate Q_{ib} and Q_{gb};

  calculate η_{max} and η_{min} (equation 3.5) ;

  apply pheromone evaporation on all edges (equation 3.6);

end

Chapter 4

Ant Brood Clustering

4.1 Ant Brood Clustering

Ant Brood algorithm was proposed by (Deneubourg et al. [1991]) as a distributed sorting algorithm, inspired by how ant colonies sort their brood. It was initially used by robot teams. In the algorithm, the ALRs or RLAs (ant-like robots or robot-like ants) move randomly and decide whether to pick up or drop off an object based on the fraction of nearby points occupied by objects of the same type in their limited memory. The less objects of the same sort there are in the immediate environment, the greater the probability that it will be picked up, as given by the function $p_{pickup} = ((k^+)/(k^+) + f)^2$, where f is an estimation of the fraction of nearby points occupied by objects of the same type, and k^+ is a constant that is usually between 0 and 1. The pickup probability decreases with f. p_{pickup} is 1 when $f = 0$, 1/4 when $f = k^+$, and $((k^+)/(k^+) + 1)^2$ as $f = 1$. The more objects of the same sort there are in the immediate environment, the greater the probability that an ant will drop

down an object. $p_{putdown} = (f/((k^-) + f))^2$, where f is as described above, and k^- is a constant between 0 and 1. The drop down probability increases with f, from 0 ($f = 0$), to 1/4 ($f = k^-$), and $(1/((k^-) + 1))^2$ ($f = 1$). This simple behaviour, along with the the indirect feedback from the environment that is modified through the iterations, allows the algorithm to cluster randomly distributed objects into piles just like the behaviour of real ants. Deneubourg *et.al.* point out that although the algorithm is less efficient, it is autonomous and provides simplicity and flexibility to the programming of a robot team.

Deneubourg *et.al.*'s model is extended by Lumer and Faieta to work with objects which are comparable according to a measure of dissimilarity (Lumer and Faieta [1994]). The simulation evolves in discrete time steps. At each step, an ant is selected at random and can either pick or drop an item at its current location. The probability of picking the element increases with low density and decreases with high density of the similar elements in a small surrounding area of $d \times d$ nodes.

Handl and Dorigo (Handl et al. [2006]) propose an improved version based on the model of Lumer and Faieta, called Adaptive Time-dependent Transporter Ants (ATTA), for cluster analysis and topographic mapping in data mining. ATTA employs techniques including modified neighborhood function, short-term memory with lookahead, increasing radius of perception, time-dependent modulation of the neighborhood function, modified threshold functions, and activity-based α adaptation. ATTA is experimented on synthetic and real data sets for machine learning. The improvements lead to gain in efficiency, robustness and solution quality.

4.2 ABC-KLS Algorithm for Graph Clustering

The dissimilarity measure is extended from Euclidean distance to the graph dissimilarity measure by Kuntz, Layzell, and Snyers (KLS) (Kuntz et al. [1997]) in solving graph partitioning problem. The dissimilarity between two objects, now represented as vertices, depends only on their respective neighborhood relationships.

I describe the ABC-KLS algorithm in details below.

Let N_i be the set of vertices adjacent to v_i, including v_i. Then the dissimilarity matrix can be expressed as

$$d(i,j) = \frac{|N_i \triangle N_j|}{|N_i| + |N_j|} \tag{4.1}$$

where the $\triangle$ operator is the symmetric difference (union minus intersection), while the $|\cdot|$ is the set cardinality operator. For example, if v_i and v_j are not adjacent and share no common edges, then $d(i,j) = 1$, indicating the most dissimilar vertex pair. Obviously, $d(i,i) = 0$. Most of the d values lie between 0 and 1. Larger d indicates more dissimilarity.

The probability of picking an element i is

$$P_{pick}(i) = \left(\frac{k_p}{k_p + f(i)}\right)^2 \tag{4.2}$$

$$P_{drop}(i) = \left(\frac{f(i)}{k_d + f(i)}\right)^2 \tag{4.3}$$

where k_p and k_d are set to educated guesses that allow the customization of the probability of pick up, P_{pick}, and drop, P_{drop}. $f(i)$ is the local density estimator of the object i and the ant's current neighborhood. The local density estimator is defined

by

$$f(i) = \begin{cases} \frac{1}{l} \sum_j \left(1 - \frac{d(i,j))}{\alpha}\right) & \text{if } f > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

The dissimilarity, $d(i, j)$, between the object i and current neighbor object j is scaled by a constant $\alpha \in (0, 1]$. Then, it is normalized by l representing the neighborhood size. l is calculated by the ants' radius of reception, r, usually a small integer number. For example, if $r = 1$, then $l = 8$ for ants in the middle area of the grid, $l = 5$ for ants at the boundaries, and $l = 3$ for ants at the corners.

The ABC-KLS algorithm is described in Algorithm 2. In this model, both vertices (objects) and ants are randomly scattered on a 2D grid with dimension *dim*. The ants start their random walk with up to r grid cells at a time in four possible directions: up, down, left, and right. If the next walk falls out of boundary, the ant simply bounces back in the opposite direction. If one unloaded ant comes to a site (grid cell) occupied by an object, the ant measures the similarity density in this neighborhood and picks up the object with a probability that is inversely related to the similarity density. That is, the less similar objects in the neighbourhood, the more likely the object will be picked up. After the ant is loaded, it carries the object and continues to walk step-wise. Whenever the ant comes to an empty site, it evaluates the neighbourhood similarity density again and drops down the object with a probability that is proportional to the similarity density. To maintain the algorithmic simplicity, it is assumed that no two ants should be allowed to come to the same site, and each site can only accommodate one vertex.

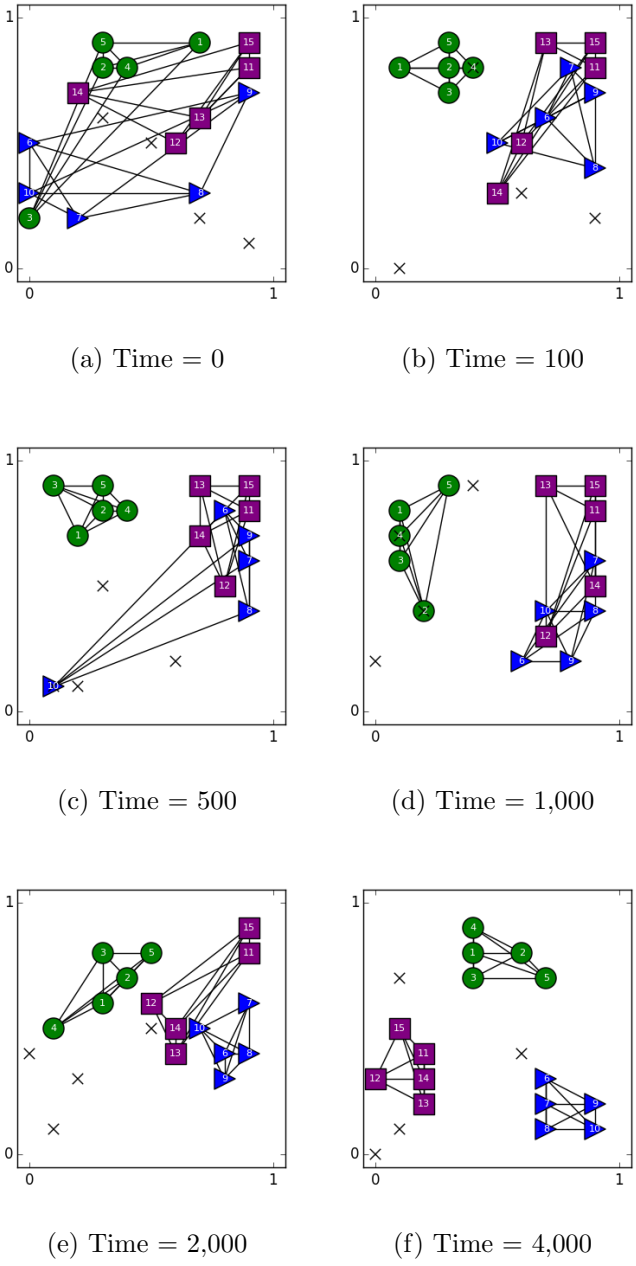
Algorithm 2: ABC-KLS Algorithm

Input: G, N_{ants}, N_{iter}, dim, r, k_p, k_d, α

Output: $v_{coordinates}$ after N_{iter} iterations

```
1  Assign random coordinates to all v ∈ V;
2  Assign random coordinates to N_ants ants;
3  for iter ← 1 to N_iter do
4      foreach ant, a_k do
5          if a_k is unloaded and the site is occupied by v_i then
6              Compute f(i) (equation 4.4) and P_pick(i) (equation 4.2);
7              Generate a random real number R ∈ [0, 1];
8              if R ≤ P_pick(i) then
9                  Pick up v_i;
10             end
11         end
12         else
13             if a_k is loaded with v_i and the site is empty then
14                 Compute f_i (equation 4.4) and P_drop(i) (equation 4.3);
15                 Generate a random real number R ∈ [0, 1];
16                 if R ≤ P_drop(i) then
17                     Drop down v_i;
18                 end
19             end
20         end
21         move to next position;
22     end
23 end
```

The ABC-KLS algorithm is loyal to the "natural" behaviour of ants. Although the ants are extremely simple agents with no memory and no capability to know the environment beyond its immediate neighbhourhood, the changing environment provides positive and negative feedback for the pick-up and drop-down actions of ants. Collectively, the ants are able to put similar nodes together and those that are dissimilar far away from each other. Figure 4.1 illustrates how KLS works on a toy graph composed of three disconnected k-5 subgraphs.

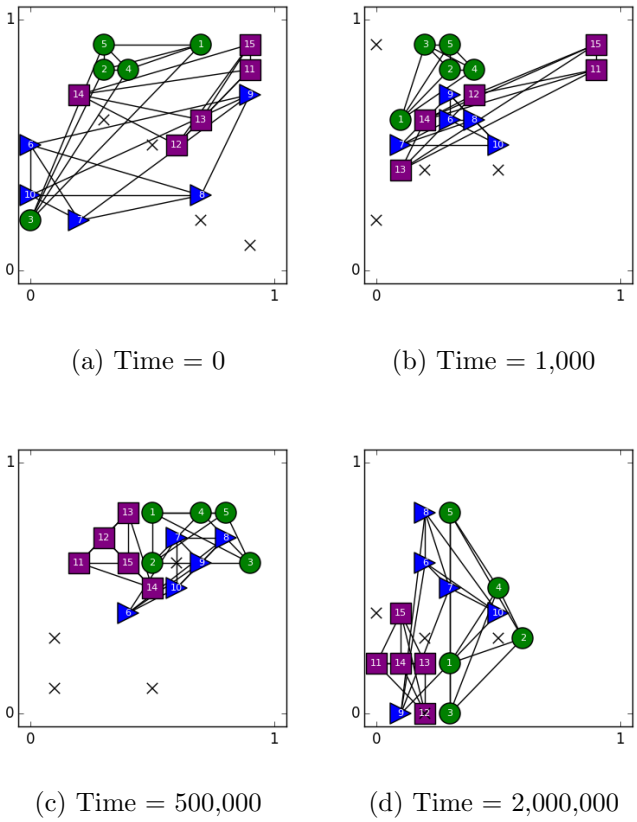


(a) Time = 0 (b) Time = 100

(c) Time = 500 (d) Time = 1,000

(e) Time = 2,000 (f) Time = 4,000

Figure 4.1: Different time steps of running ABC-KLS algorithm on a toy graph with three disconnected k-5 subgraphs. The expected clusters of the nodes are drawn with different colors and shapes. The ants are represented as X on the grid. All the coordinates are normalized to lie in [0, 1]. Parameters: $N_{ants} = 4$,$N_{iter} = 4,000$, $dim = 10$, $r = 2$, $k_p = 0.3$, $k_d = 0.05$, $\alpha = 0.9$



(a) Time = 0 (b) Time = 1,000

(c) Time = 500,000 (d) Time = 2,000,000

Figure 4.2: Different time steps of running ABC-KLS algorithm on a toy graph with three *connected* k-5 subgraphs. The expected clusters of the nodes are drawn with different colors and shapes. The ants are represented as X on the grid. All the coordinates are normalized to lie in [0, 1]. Parameters: $N_{ants} = 4$,$N_{iter} = 2,000,000$, $dim = 10$, $r = 2$, $k_p = 0.3$, $k_d = 0.05$, $\alpha = 0.9$

Although ABC-KLS algorithm seems to work well for the toy graph, it suffers from a few drawbacks.

- The algorithm is very inefficient in terms of speed. The ants tend to spend most of their time on random walks, instead of actually moving objects. As we

can observe in figure 4.1, none of the four ants were carrying objects at time 0, 2000, and 4000. At time 100 and 500, only one ant was "working". The busiest time was at time 1000, when two ants were associated with objects.

- The algorithm does not cope with more complicated graphs. In figure 4.2, a few inter-cluster edges are added to introduce some noise in the toy graph in figure 4.1. This is the same graph as figure 3.1. At time 1000, the ants were able to put similar objects together. However, once objects are aggregated in a close neighbourhood, the ants have a hard time to separate the clusters. The noise affects ants' judgment on the neighbourhood's dissimilarity density. Although the stochastic pick-up process allows ants to pick up some objects, these objects are soon dropped at an empty site close by the blob. Overtime, the one large cluster tends to drift on the grid but is not easily separated. This observation is obvious at time 500,000 and 2,000,000.

- One main disadvantage of most ant brood clustering algorithms, including ABC-KLS and other ABC algorithms, as pointed out by previous studies (Bonabeau et al. [1999]; Handl et al. [2006]), is a lack of quantitative measurement of the clustering quality. The interpretation of the clustering result, has, unfortunately, largely depended on human eyes. On one hand, the fact that ABC can produce an output that is appealing to human observation may be a valuable asset of the algorithm. On the other hand, relying solely on human vision to judge the goodness of an algorithm is, arguably, largely inaccurate, or easily fooled. The lack of quantitative measurement also prevents ABC to be taken seriously and comparably in solving real problems. Nonetheless, the task of

finding *explicit* clustering result based on *implicit* geometric embedding is not necessarily difficult. As the ideal output of ABC is an embedding of clearly separated clusters, an Euclidean-distance based cluster retrieval algorithm can be applied on the top of the embedding. (Kuntz et al. [1999]) use k-means for this task. (J. Handl et al. [2003]) use an agglomerative hierarchical clustering algorithm that automatically determines k.

4.3 ABC-INTE

I introduce an improved algorithm based on the ABC-KLS algorithm. Because the algorithm focuses on improving the capabilities of the ants, I call it *Ant Brood Clustering-Intelligent Ants*, or *ABC-INTE*.

4.3.1 Hopping Ants

As discussed previously, the ABC-KLS algorithm is inefficient because the ants seem to spend most of their time on random walk without carrying any objects. I use the concept of "hopping" ants, an idea that worked well in ant colony optimization for mobile ad hoc networks (Rana et al. [2013]).

The hopping ants work as follows. In order to reduce the unnecessary walks, I add a queue of free objects, Q_{obj}, to the algorithm. If an object is not carried by an ant, it is immediately pushed to Q_{obj}. Once an ant becomes unladen, it jumps to the first free object at the top of Q_{obj} directly. The hopping action of an ant only happens when it is unladen. If the ant is laden, it continues to walk stepwise on the grid. The stepwise walk is very important for the exploration of ants and the optimality of the

algorithm. So the behavior of an ant is a combination of hopping and walking. The hopping is for efficiency, and the walking is for exploration. The additional benefit of putting free objects in a First-In-First-Out (FIFO) queue is that it makes sure *all* the free objects are equally and frequently assessed for their eligibility of being moved. In a completely random walk mood, it is possible that some objects are missed out by the ants even though they should have been moved.

4.3.2 Enhanced Drop Behavior

The drop behavior is perhaps the most important portion of ABC. As we have seen in figure 4.2, the basic drop behavior of ants does not respond well to noisy environment. To improve the robustness of this behavior, I apply the following techniques.

Relaxed Drop Function

An issue I observe in the ABC-KLS algorithm is that an ant tends to carry an object for a lot of iterations before it can find a proper site to drop the object. While the original drop function aims to find a perfect empty site surrounded by similar objects, it may fail when all the similar objects are placed tightly and interleaved with dissimilar objects with no empty cell inside the blob. Another side-effect of this non-dropping behavior is that it minimizes the environment change over iterations, which is the main mechanism of *stigmergy* in ABC. In order to encourage ants to drop objects, I modify the drop function to equation 4.5 (Lumer and Faieta [1994]). The drop behavior is only stochastic if the dissimilarity density, $f(i)$, is less than a

threshold k_d. If the threshold is passed, the drop behavior is completely deterministic.

$$P_{drop}(i) = \begin{cases} 2f(i) & \text{if } f(i) < k_d \\ 1 & \text{otherwise} \end{cases} \tag{4.5}$$

Ants With Memories

Ants with memories (Lumer and Faieta [1994]) is a common technique used in the improvement of ant brood algorithm in data clustering. With this technique, each ant is assigned a small amount of memory to remember the past few positions it resided. In this thesis, I define the memory size to be 10 to stay faithful to the simple-agent design principle. In literature, when a laden ant comes to a site, it assesses the drop probabilities at the current site, *and* all the positions in its memory. The object is dropped stochastically at the position with highest probability. This computation, although relatively simple for a single scan of a small memory, can be expensive when the number of ants is large and/or the algorithm runs for a lot of iterations. In addition, this computation will be unnecessary if current site is already a good place for dropping the object with high probability. Therefore, I argue that the ant's memory should only be assessed when the ant is stuck, as will be described shortly in the next section.

Stagnation Control

If an ant is stuck, that is, it cannot find a site to drop the object, then the ant starts to "think": "Should I continue to run around, or should I make a decision now?" An intelligent ant, as in ABC-INTE, chooses to make a decision now. Therefore, the

drop action becomes deterministic. If the number of unsuccessful drops exceeds a threshold *dropLim*, the ant will access its memory and find a site with maximum similarity density to drop the object. If such a site does not exist in memory, the ant drops the object at current site, *even* if it's not favorable. The unfavorable drop position is permissible because it can be fixed in later iterations. The stagnation control is an essential improvement because it ensures a fast-changing environment with quality assurance from the choices in ants' memory.

4.3.3 Cluster Retrieval

As discussed earlier, a cluster retrieval process is needed to interpret the geometric embedding of ABC to explicit clustering of the graph in order to evaluate the quality of the algorithm. For simplicity, I use k-means (Hendrickson and Kolda [2000]) for this step. Note that cluster retrieval is not part of the ABC algorithm. The pseudo code of cluster retrieval is described in Algorithm 6.

4.3.4 Program flow

The program flow of ABC-INTE can be found in Algorithm 3. Note the difference between ABC-INTE and ABC-KLS algorithm. At the beginning of the iterations, the ants jump to free objects directly (line 3). The ant's trail is stored in its memory (line 8). To keep the ants simple, I limit the memory size to be 10 in ABC-INTE. If one ant picks up an object, a counter for its drop attempts becomes effective (line 12). The counter increments in the later iterations for every drop attempt this ant makes (line 18). A forced drop happens when drop counter exceeds a threshold (line 20), or the

algorithm reaches the end of the iterations (line 44). During the deterministic forced drop, the ant chooses a drop site among its current location and all the locations in its memory with the highest similarity density (line 21). By design, each grid cell can only accommodate one object. Since locations in memory may be occupied now, an additional check is required to make sure the drop does not overwrite existing object on the cell (line 22-27). If an object is dropped on the grid, it is immediately pushed back to the free object queue, waiting for a visit from another ant later (line 33). Finally, the ants move in different paces. If an ant is free, it jumps to the next object in the queue directly (line 38), otherwise, it continues to walk stepwise to explore a drop site (line 40).

Algorithm 3: ABC-INTE Algorithm - Part 1

Require: G, N_{ants}, N_{iter}, dim, r, k_p, k_d, α, $dropLim$

Ensure: $v_{coordinates}$ after N_{iter} iterations

1: Initialize $v_{coordinates}$ with random integer coordinates $\in [0, dim - 1]$ for all $v \in V$
2: Push all $v \in V$ to Q_{obj} in a random order
3: Initialize $ant_{coordinates}$ to the coordinates of first N_{ant} in Q_{obj}
4: Allocate drop counters a_{dcnt}
5: Allocate ants memory ant_{mem}
6: **for** $i \leftarrow 1, N_{iter}$ **do**
7: **for** $ant \leftarrow 1, N_{ants}$ **do**
8: Push $ant_{coordinates}[ant]$ to $ant_{mem}[ant]$
9: **if** ant is unladen AND site is occupied by object o **then**
10: pick up o (equation 4.2)
11: **if** pick up successful **then**
12: $a_{dcnt}[ant] \leftarrow 0$
13: **end if**
14: **else**

Algorithm 4: ABC-INTE Algorithm - Part 2

```
15:       if ant carries object o AND site is empty then
16:         drop down o (equation 4.5)
17:         if drop down not successful then
18:           a_dcnt[ant] ← a_dcnt[ant] + 1
19:         end if
20:         if a_dcnt[ant] > dropLim then
21:           calculate Max(f(i)) for current site and all positions in ant_mem[ant]
22:           if Max(f(i)).pos in memory then
23:             if Max(f(i)).pos is occupied then
24:               drop o at an empty site near Max(f(i)).pos
25:             else
26:               drop o at Max(f(i)).pos
27:           end if
28:         else
29:           drop o at current site
30:         end if
31:       end if
32:       if o is dropped then
33:         Q_obj.push(o)
34:       end if
35:     end if
```

Algorithm 5: ABC-INTE Algorithm - Part 3

```
36:       end if
37:       if ant is unladen then
38:          ant jumps to Q_obj.peek()
39:       else
40:          ant moves stepwise
41:       end if
42:    end for
43: end for
44: if some ants still carry objects then
45:    force drop the objects according to line 21 to 30
46: end if
```

Algorithm 6: k-means Cluster Retrieval

Require: $v_{coordinates}$,n, k, dim

Ensure: C

1: Initialize μ_j with random real coordinates $x, y \in [0, dim - 1]$ for all $j \in [1, k]$

2: **repeat**

3: **for** $i \leftarrow 0, n - 1$ **do**

4: $C_i \leftarrow \text{argmin}_j \left\| v_{coordinates}[i] - \mu_j \right\|^2$

5: **end for**

6: **for** $j \leftarrow 1, k$ **do**

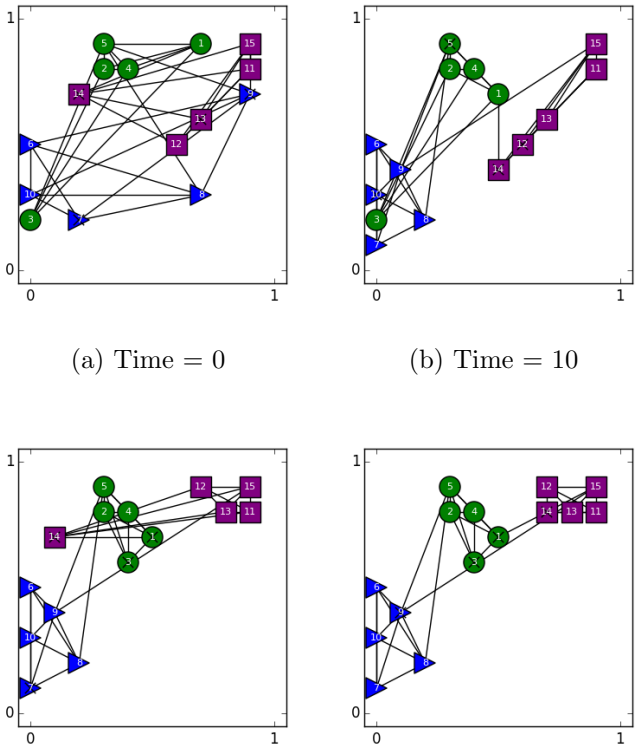
7: $\mu_j \leftarrow \text{mean } (v_{coordinates}[i]), \forall (C_i = j)$

8: **end for**

9: **until** Convergence

4.3.5 Algorithm Validation

To verify if the techniques used in ABC-INTE are effective, I run ABC-INTE on the same graph used by ABC-KLS algorithm in figure 4.2. The snapshots of different time stamps are shown in figure 4.3. Compared to ants in ABC-KLS, ants in ABC-INTE are always associated with objects, and they are able to reveal the clustering structure within 100 iterations whereas ants in ABC-KLS are not able to do so after 2 million iterations.

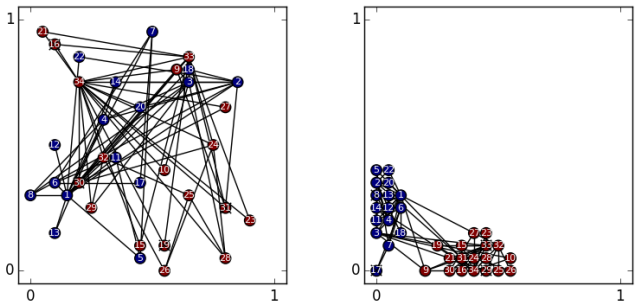


(a) Time = 0 (b) Time = 10

(c) Time = 50 (d) Time = 100

Figure 4.3: Different time steps of running ABC-INTE algorithm on a toy graph with three *connected* k-5 subgraphs. The expected clusters of the nodes are drawn with different colors and shapes. The ants are represented as X on the grid. All the coordinates are normalized to lie in [0, 1]. Parameters: $N_{ants} = 4$,$N_{iter} = 100$, $dim = 10$, $r = 2$, $k_p = 0.3$, $k_d = 0.05$, $\alpha = 0.9$, $dropLim = 100$

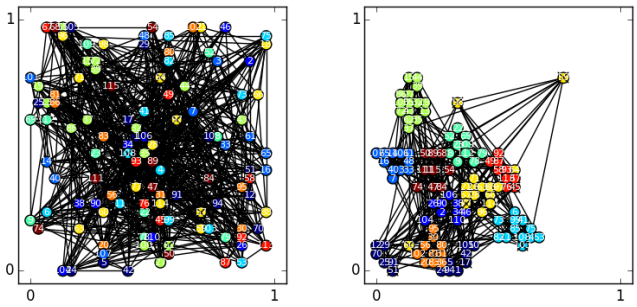
I now extend my experiment from toy graphs to real networks. I first consider a small social network called *karate* (Zachary [1977]). During a three-year period from 1970 to 1972, Wayne Zachary observed social interactions within a university karate club. Conflict arose between the club president and the instructor over the authority to raise lesson fees. Overtime the entire club became divided into two groups: supporters of the president and supporters of the instructor. This network has 34 nodes and 78 edges. Figure 4.4 shows the consensus network structure before the club is split into two clubs. The blue nodes represent supporters of the instructor and the red nodes refer to supporters of the president. At time 0, the nodes are mapped onto a grid with random positions. At time 1,000, ABC-INTE is able to reveal the network structure by separating the two clusters and placing nodes into the correct clusters.



(a) Time = 0 (b) Time = 1,000

Figure 4.4: Different time steps of running ABC-INTE algorithm on karate. The expected clusters of the nodes are drawn with different colors. The ants are represented as X on the grid. All the coordinates are normalized to lie in [0, 1]. Parameters: $N_{ants} = 4$,$N_{iter} = 1,000$, $dim = 20$, $r = 2$, $k_p = 0.49$, $k_d = 0.06$, $\alpha = 0.9$, $dropLim = 100$

I then look at another social network called *football* (Girvan and Newman [2002]). This network represents the schedule of Division I games for the 2000 season of United States college football. This network has 115 nodes representing teams and 613 edges representing regular-season games between two teams. The teams are divided into conferences. Games are more frequent between members of the same conference than those in different conferences. The reference clustering has the same *modularity* of 0.605 and 10 clusters as the result of the exact algorithm (Cafieri et al. [2011]). The membership is calculated by my implementation of MMAS (algorithm 1) since it is not given in literature. As shown in figure 4.5, ABC-INTE is able to cluster most of the nodes correctly after 30,000 iterations. Apply k-means cluster retrieval (algorithm 6) to the output at time 30,000 and I get a *modularity* of 0.504, which is better than 0.493 by divisive spectral heuristic and slightly worse than 0.538 by the divisive Kernighan-Lin based heuristic (Cafieri et al. [2011]).



(a) Time = 0 (b) Time = 30,000

Figure 4.5: Different time steps of running ABC-INTE algorithm on football. The expected clusters of the nodes are drawn with different colors. The ants are represented as X on the grid. All the coordinates are normalized to lie in [0, 1]. Parameters: $N_{ants} = 10$,$N_{iter} = 30,000$, $dim = 30$, $r = 2$, $k_p = 0.45$, $k_d = 0.02$, $\alpha = 0.9$, $dropLim = 100$

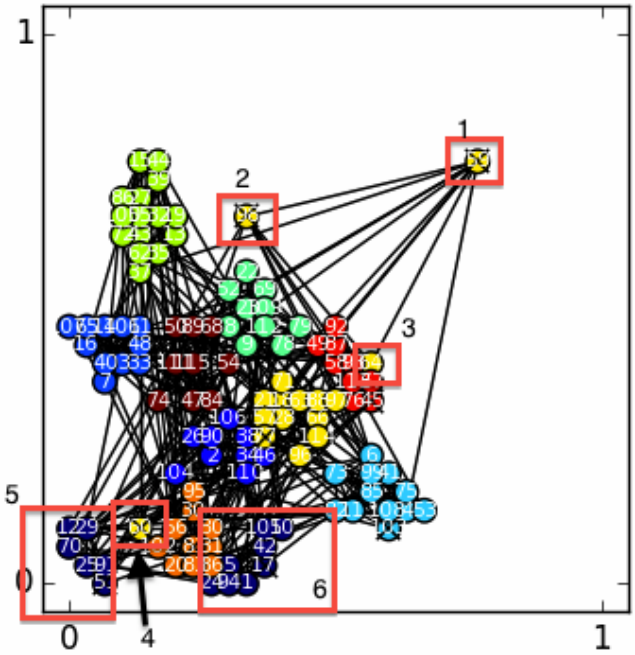


Figure 4.6: The discrepancy between the clustering of football network by ABC-INTE in figure 4.5 and the optimal solution. Differences are highlighted in numbered boxes.

4.3.6 The Drawbacks of ABC-INTE

The discrepancy between the clustering of football network by ABC-INTE and the optimal solution is caused by some outliners, as highlighted in numbered boxes in figure 4.6. The four nodes in boxes 1 to 4 belong to a cluster in the middle shaded with yellow color. The nodes in boxes 5 and 6 should belong to one single cluster instead of being separated. An analysis of these nodes reveals two drawbacks of ABC-INTE.

Separation of Clusters Needs to be Improved

I observe that although the ants are able to put most similar objects together, they are less capable of putting the clusters far away from each other. The vicinity of clusters has two effects on the algorithm. First, it affects the drop-down behavior of ants. For example, nodes in box 5 and 6 are separated because the space between them is occupied by another cluster. If the clusters are farther from each other, it is anticipated that the ants will be able to put box 5 and 6 together. The same is for the node in box 3, which is blocked by anther cluster away from its own cluster. Second, closely placed clusters also affects the accuracy of the cluster retrieval algorithm that is based on Euclidean distance.

Arguably, the small inter-cluster distance is the consequence of the small step size of the random walk. It is possible to increase the inter-cluster distance by increasing the radius of reception, r. However, the side effect of increasing r is that the intra-cluster distance will increase as well. Keeping inter-cluster distance large and intra-cluster distance small seems to be a contradictory task by merely changing r through parameter tuning. Increasing r also increases the neighbourhood size l and adds computation cost to the ants every time they evaluate the similarity density, $f(i)$, in the neighbourhood. Therefore, parameter tuning alone does not seem to solve this issue.

Lack of Global View

I notice that the nodes in boxes 1, 2, and 4 are associated with ants, meaning that they are dropped by these ants at the end of iterations. Although the ants can

exploit their memory for a favorable drop site, the fact that these three nodes are placed very far from their expected cluster indicates that there are no such positions in their limited memory that are close to the expected cluster. Until now, the ant brood clustering algorithm works entirely based on local similarity densities on the grid without a measurement of the global clustering quality to guide the ants' behavior. The locality contributes to the speed and novelty of the algorithm, but it also seems to be the bottleneck that prevents the ants towards optimality.

4.4 ABC-POLY

We have seen that ABC-INTE performs very well on finding the clusters in the 15-node toy graph (figure 4.3) and the karate network (figure 4.4), and reasonably well on the football network (figure 4.5). The analysis of ants' behavior on the football network reveals two drawbacks of ABC-INTE: small inter-cluster distance and lack of global view. In this section, I introduce a new algorithm, *Ant Brood Clustering-Polymorphic Ants (ABC-POLY)*, as an improvement of ABC-INTE.

4.4.1 Add Global View to the Ants

Lack of global view prevents ABC-INTE to achieve better performance. At some point of time, an ant may need to know the *best* position that *guarantees* the quality of the entire clustering. This "point of time" can be at the end of all iterations, as is the case for the three nodes in boxes 1, 2, and 4 in figure 4.6. If we let the three ants know the result of cluster retrieval, they might be able to find better positions to move the nodes to. Specifically, if we give the number of clusters, k, and the membership array to one such ant, the ant can calculate the *change* of modularity should the node it just dropped is placed in another cluster. Based on equation 2.1, the calculation of the partial *modularity* value of a single node, i, is shown below in equation 4.6:

$$\underset{i \in C}{Q}_i = \frac{1}{2m}\left[\left(A_{ii} - \frac{a_i a_i}{2m}\right) + \sum_{j \in C, j \neq i}\left(A_{ij} - \frac{a_i a_j}{2m}\right) * 2\right] \tag{4.6}$$

where C is the cluster that i belongs to, m is the number of edges in the graph, A is the adjacency matrix, and a_i is the degree of i. The first half of the equation

calculates the *modularity* of forming a self-edge. The second half of the equation calculates the sum of *modularity* from all intra-cluster edges that contain i on either end.

The change of *modularity*, ΔQ_i, of changing the membership of a single node i from C_1 to C_2, is calculated below.

$$\begin{aligned}
\Delta Q_i &= \underset{i \in C_1}{Q}{}_i - \underset{i \in C_2}{Q}{}_i \\
&= \frac{1}{2m}\left[\left(A_{ii} - \frac{a_i a_i}{2m}\right) + \sum_{j \in C_1, j \neq i}\left(A_{ij} - \frac{a_i a_j}{2m}\right) * 2\right] - \\
&\quad \frac{1}{2m}\left[\left(A_{ii} - \frac{a_i a_i}{2m}\right) + \sum_{j \in C_2, j \neq i}\left(A_{ij} - \frac{a_i a_j}{2m}\right) * 2\right] \\
&= \frac{1}{2m}\left[\sum_{j \in C_1, j \neq i}\left(A_{ij} - \frac{a_i a_j}{2m}\right) * 2 - \sum_{j \in C_2, j \neq i}\left(A_{ij} - \frac{a_i a_j}{2m}\right) * 2\right] \\
&= \frac{1}{m}\left[\sum_{j \in C_1, j \neq i}\left(A_{ij} - \frac{a_i a_j}{2m}\right) - \sum_{j \in C_2, j \neq i}\left(A_{ij} - \frac{a_i a_j}{2m}\right)\right] \qquad (4.7)
\end{aligned}$$

As shown in equation 4.7, the calculation of ΔQ_i is very easy and only requires information of the adjacent nodes to node i. If the memberships of all other nodes in the graph do not change, ΔQ_i is then the change to *global* modularity caused by membership change of i. Since k is now known by the ant that carries node i, it can calculate the optimal membership ID for i that causes the most positive ΔQ_i. If the optimal membership is different from the current membership of i, the ant simply moves i to an unoccupied cell at / near the centroid of the correct cluster.

By using the simple technique above, the ants carrying nodes in boxes 1, 2, and 4 in figure 4.6 can now move the nodes into their correct cluster. However, the node in box 3, which is force-dropped in an earlier iteration and is not associated with an ant, is not detected. Since the calculation of ΔQ_i is relatively simple, it may be beneficial

to use this technique during the iterations as well. In addition, if a few nodes are moved to cluster centroids during the iterations, they may become seeds that attract more nodes to be deposited nearby, and eventually lead to more separated clusters. If this is the case, then the nodes in boxes 5 and 6 may be placed together due to the separation.

4.4.2 The Major and Minor Ants

In polymorphic ant species, there are usually different types of workers specializing in different tasks. For example, in *Pheidole* genus (Bonabeau et al. [1999]), the major workers are larger and responsible for foraging for food or defending the nest, whereas the minors are smaller and perform some housekeeping tasks such as feeding the brood or cleaning the nest. Majors and minors work simultaneously to maintain a functioning colony. This *division of labour* is believed to be more efficient than using unspecialized individuals to perform different tasks sequentially.

Using this metaphor, I introduce the ABC-POLY algorithm. ABC-POLY involves two types of ants: the *Majors* and the *Minors*. The majors explore the grid and pick up / drop objects according to the similarity density in the neighbourhood. The minors exploit the solution optimality and clean up major's work by moving objects to better positions. To strengthen the exploitation, the minors exam *modularity* gain of membership change for every node on the grid and move the nodes if needed. This computation relies on the assumption that only one single node has a different membership when calculation of ΔQ_i based on equation 4.7. The calculation of ΔQ_i requires one row of the adjacent matrix, or $O(n)$ computation. The minor ants

exam all n node for all possible k clusters. The compuation cost of minor ants is then $O(kn^2)$, or $O(n^2)$ if k is a small constant. This cost does not include the cost of the k-means cluster retrieval algorithm, $O(nkt)$, where t is the number of iterations to reach convergence and is usually small thanks to the work of the major ants. Nonetheless, the computational cost of minor ants is expensive compared to major ants, which only spend constant time in checking similarity density in a small neighbourhood in each iteration. Therefore, the terms *major* and *minor* also infer the workload allocation. The majors are the main workers to form the clusters on grid, whereas the minors only intervene occasionally to maintain the efficiency of the entire algorithm. The infrequent intervention of minor ants is further justified by the claim that ABC-INTE can perform reasonably well, as shown in the previous section.

4.4.3 Program Flow

The program flow of ABC-POLY is formally described in algorithm 7. ABC-POLY requires three additional parameters than ABC-INTE: k, the expected number of clusters; N_{minor}, the number of minor ants; and $mPer$, the interval to activate the minors. The major ants perform ABC-INTE at each iteration (line 6). The minor ants remain dormant until the iteration reaches a checkpoint of $mPer$ (line 8). At such a checkpoint, ABC-POLY calls k-means to get the membership array, C, and the positions of the centroids of the k clusters, *kcentroids* (line 9). The workload of calculating ΔQ is distributed to the minor ants (line 10). This distribution may be unnecessary in a sequential implementation. But it is convenient in a parallel implementation since the calculation of ΔQ for each node is completely independent

and can be done in parallel. Next, the minor ants find the maximum ΔQ for each object (line 12 to 21) and move an object if needed (line 22 to 28).

Algorithm 7: ABC-POLY Algorithm - Part 1

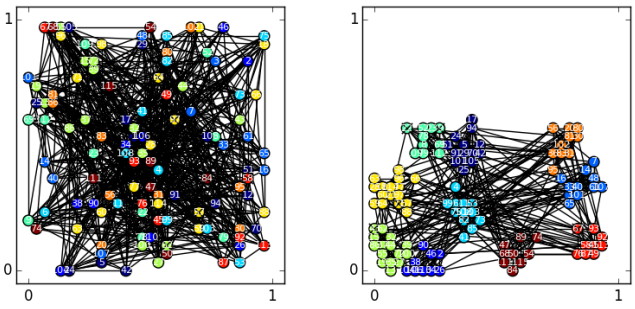
Require: G, N_{major}, N_{iter}, dim, r, k_p, k_d, α, $dropLim$, k, N_{minor}, $mPer$

Ensure: $v_{coordinates}$ after N_{iter} iterations

1: Initialization for ABC-INTE (algorithm 3)
2: Allocate membership array C
3: Allocate centroid array $kcentroids$
4: **for** $i \leftarrow 1, N_{iter}$ **do**
5: **for** $ant \leftarrow 1, N_{major}$ **do**
6: Perform ABC-INTE (algorithm 3)
7: **end for**
8: **if** $i \bmod mPer == 0$ **then**
9: Calculate C and $kcentroids$ with k-means cluster retrieval (algorithm 6)
10: Distribute C to N_{minor} ants
11: **for** $ant \leftarrow 1, N_{minor}$ **do**
12: **for** each $v \in C_{ant}$ **do**
13: $MaxDeltaQ.value = 0$
14: $MaxDeltaQ.id = C_{ant}[v]$
15: **for** $c \leftarrow 1, k$, $c \neq C_{ant}[v]$ **do**
16: Calculate $DeltaQ$ (equation 4.7)

Algorithm 8: ABC-POLY Algorithm - Part 2

```
17:             if DeltaQ > MaxDeltaQ.value then
18:                 MaxDeltaQ.value = DeltaQ
19:                 MaxDeltaQ.id = c
20:             end if
21:         end for
22:         if MaxDeltaQ.value > 0 then
23:             if kcentroids[MaxDeltaQ.id] is not occupied then
24:                 Move v to kcentroids[MaxDeltaQ.id]
25:             else
26:                 Move v to an unoccupied cell near kcentroids[MaxDeltaQ.id]
27:             end if
28:         end if
29:       end for
30:     end for
31:   end if
32: end for
```



(a) Time = 0 (b) Time = 9,000

Figure 4.7: Different time steps of running ABC-POLY algorithm on football. The expected clusters of the nodes are drawn with different colors. The ants are represented as X on the grid. All the coordinates are normalized to lie in [0, 1]. Parameters: $N_{major} = 10$,$N_{iter} = 9,000$, $dim = 30$, $r = 2$, $k_p = 0.3$, $k_d = 0.04$, $\alpha = 0.9$, $dropLim = 100$, $k = 10$, $N_{minor} = 1$, $mPer = 1,000$

4.4.4 Algorithm Validation

I now apply ABC-POLY to the *football* network and verify whether the work of the minor ants is efficient in terms of solution quality. Figure 4.7 shows the result of ABC-POLY after 9,000 iterations with 1 minor ant intervening every 1,000 iterations. It is obvious that the ants are now better at separating the clusters and the layout is very pleasing to human eyes. The final *modularity* of this layout is 0.605, the optimal value.

Chapter 5

Evaluations: Results and Discussion

5.1 Benchmark Graphs

I evaluate the clustering qualities of ACO-MMAS, ABC-INTE, and ABC-POLY on 10 benchmark real-life networks (Bader et al. [2014]) listed in table 5.1. Besides comparison among the three algorithms, I also compare the three algorithms with the popular Louvain Algorithm (Blondel et al. [2008]), a multi-level greedy modularity optimization algorithm for community detection on large networks. In addition, the *modularity* values of all the programs are compared to the highest values in literature.

Network	n	m	Description	Original Source
karate	34	78	friendships of members in a karate club	(Zachary [1977])
dolphins	62	159	associations of bottlenose dolphins	(Lusseau et al. [2003])
lesmis	77	254	character interactions in *Les Misérables*	(Knuth [1993])
polbooks	105	441	political books published in 2004 and sold on *Amazon.com*	(pol)
adjnoun	112	425	adjacencies of adjectives and nouns in *David Copperfield*	(Newman [2006])
football	115	613	conferences among college football teams	(Girvan and Newman [2002])
jazz	198	2742	jazz musicians network	(Gleiser and Danon [2003])
celegansneural	297	2148	neural network of C. Elegans	(Watts and Strogatz [1998])
celegans metabolic	453	2025	metabolic network of C. Elegans	(Jeong et al. [2000])
email	1133	5451	email interactions between members of the Univeristy Rovira i Virgili.	(Guimera et al. [2003])

Table 5.1: Benchmark Networks for Graph Clustering

5.2 Metrics

This thesis uses *modularity* value as the metric for clustering quality. Although *modularity* has its own limitations and high value of *modularity* is not necessarily the only way to determine a good clustering (Guimera et al. [2004b]; Lancichinetti and Fortunato [2011]), it is generally considered to be a good quality measurement that considers both minimizing inter-cluster edges and maximizing intra-cluster edges (Fortunato [2010]). Since modularity optimization is probably the most popular approach for graph clustering, there is a large number of literature that provide the base for a comparative study.

In addition to solution quality, I also measure *execution time* to understand the efficiency of the algorithms. The execution time is important for determining the *scalability* of applying an algorithm to larger networks.

5.3 Experiments

The programs for ACO-MMAS, ABC-INTE, and ABC-POLY are written in the C language and tested on a server with 4 CPUs (PD Core i5/3.3GHz), 16GB memory and Linux Scientific 7.x operating system. The Louvain algorithm is taken from the python module *community* (net), the core software of which is written in C++. The Louvain software is run on a MacBook with 2 CPUs (Intel Core i5/2.3GHz), 4GB memory and Mac OS X Version 10.7.5.

The experiments are set up in following sequence. First, I determine the parameters of ACO-MMAS, ABC-INTE, and ABC-POLY for each benchmark graph through

a parameter tuning process. Then, I determine the numbers of iterations that are required to get the highest *modularity* values by the algorithms. Long iterations are avoided due to speed and memory limitations. For MMAS, I set the typical value of N_{iter} to 100, and run longer iterations if the solution qualities are less optimal. For ABC-INTE and ABC-POLY, I limit the N_{iter} to be less than 50,000 for most of the instances. Even if a longer iteration is deemed necessary, it is still less than 100,000. Finally, once the parameters and N_{iter} for a program are chosen, I run the program on the networks for ten times and take the average execution time for performance evaluation.

5.4 Parameter Tuning

The parameter values of the ant algorithms are largely based on the consensus ranges in literature.

For MMAS, I start with the setting suggested by (Chang et al. [2013]), that is, $\alpha = 1$, $\beta = 2$, $\rho = 0.8$, $\epsilon = 0.005$, and $N_{ants} = 30$. However, during the experiments, I find that different β (relative importance of heuristic information versus pheromone level) and N_{ants} may be required to achieve higher *modularity* for some graphs. Therefore, I test run parameter combinations with $\beta \in [1, 10]$ and different N_{ants} on each network for 100 iterations and choose the ones that result in the optimal *modularity*. Table 5.2 lists the parameters I determine for each graph.

For ABC-INTE and ABC-POLY, the base settings are: $N_{ants} = 0.1n$, $dim = \lceil\sqrt{10n}\rceil$ (Handl et al. [2003]), $k_p = 0.3$, $k_d = 0.1$, $\alpha = 1.0$ (Bonabeau et al. [1999]), and $r = 2$. For parameter tuning, I test run parameter combinations with $k_p \in [0.2, 0.5]$

Network	N_{ants}	α	β	ρ	ϵ
karate	30	1	2	0.8	0.005
dolphins	30	1	2	0.8	0.005
lesmis	100	1	8	0.8	0.005
polbooks	30	1	2	0.8	0.005
adjnoun	30	1	2	0.8	0.005
football	30	1	2	0.8	0.005
jazz	500	1	8	0.8	0.005
celegansneural	500	1	9	0.8	0.005
celegans metabolic	100	1	2	0.8	0.005
email	100	1	9	0.8	0.005

Table 5.2: MMAS Parameters

with an interval of 0.01, $k_d \in [0.01, 0.1]$ with an interval of 0.01, $\alpha \in [0.8, 1.0]$ with an interval of 0.1, and $r \in [2, 3]$ with an interval of 1 for 1,000 iterations. For the number of clusters, k, I consider several possible values found in literature, and choose the one that results in best *modularity* value. I test two k values on the karate network that has a hierarchical structure. $k = 2$ is the number of clusters observed in the original literature and $k = 4$ is the number of clusters that achieves the highest *modularity*. I determine the two additional parameters in ABC-POLY, N_{minor} and $mPer$, to be 1 and 1,000 respectively. As discussed in chapter 4, N_{minor} is a parameter designed for parallel implementation and is therefore set to 1 since the implementation in this thesis is sequential. $mPer$ is set to 1,000 because it allows a good balance of the

Network	N_{ants}	dim	k_p	k_d	α	r	$dropLim$	k
karate	4	20	0.49	0.06	0.9	2	100	2
karate	4	20	0.22	0.05	0.9	2	100	4
dolphins	6	20	0.48	0.02	0.9	2	100	5
lesmis	7	30	0.20	0.04	0.8	2	100	9
polbooks	10	30	0.29	0.03	0.9	2	100	7
adjnoun	10	30	0.20	0.02	1.0	2	100	5
football	10	30	0.45	0.02	0.9	2	100	10
jazz	20	50	0.22	0.03	0.8	2	100	4
celegansneural	30	60	0.20	0.04	0.9	2	100	6
celegans metabolic	50	80	0.21	0.02	0.9	3	100	6
email	110	110	0.21	0.01	1.0	2	100	10

Table 5.3: ABC-INTE Parameters

workload allocation between the major and minor ants.

The parameters determined by this process are listed in table 5.3 and table 5.4.

5.5 Results and Discussion

5.5.1 Modularity

I present the highest *modularity* values and their corresponding k's achieved by MMAS, ABC-INTE and ABC-POLY in table 5.5. For comparison, the best *modularity* values found in literature and the results by the Louvain software are also

Network	N_{major}	dim	k_p	k_d	α	r	$dropLim$	k	N_{minor}	$mPer$
karate	4	20	0.35	0.04	1.0	2	100	2	1	1000
karate	4	20	0.49	0.03	0.9	2	100	4	1	1000
dolphins	6	20	0.48	0.02	0.9	2	100	5	1	1000
lesmis	7	30	0.23	0.09	0.9	2	100	9	1	1000
polbooks	10	30	0.33	0.02	0.9	2	100	7	1	1000
adjnoun	10	30	0.41	0.04	1.0	2	100	5	1	1000
football	10	30	0.30	0.04	0.9	2	100	10	1	1000
jazz	20	50	0.38	0.03	1.0	2	100	4	1	1000
celegansneural	30	60	0.34	0.01	0.8	2	100	6	1	1000
celegans metabolic	50	80	0.34	0.08	1.0	3	100	6	1	1000
email	110	110	0.36	0.01	1.0	2	100	10	1	1000

Table 5.4: ABC-POLY Parameters

present in this table. Some entries in the column for k of the best known *modularity* (the third column) are filled with '-' because the values of k are not always given in literature. The first row also contains the sign '-' because finding 2 clusters on karate network is not possible for Louvain and MMAS. Both algorithms determine the value k automatically based on the highest *modularity*, which is achieved when $k = 4$. Except for the second column that contains all the highest *modularity* values, *modularity* values that are highest or closest to the highest are highlighted as **bold**.

Among the 10 benchmark networks, MMAS achieves the highest or closest to the highest *modularity* values for karate, adjnoun, and football. It is worthwhile to mention that the solution quality found by MMAS on adjnoun network is significantly

	Best Known			Louvain		MMAS		ABC-INTE		ABC-POLY	
Network	Q	k	Source	Q	k	Q	k	Q	k	Q	k
karate	0.371	2	(Zachary [1977])	-	-	-	-	**0.371**	2	**0.371**	2
karate	0.420	4	(Cafieri et al. [2011])	0.419	4	**0.420**	4	0.310	4	0.388	4
dolphins	0.529	5	(Cafieri et al. [2011])	**0.523**	5	0.520	7	0.487	5	0.501	5
lesmis	0.560	6	(Cafieri et al. [2011])	**0.336**	5	0.331	11	0.123	9	0.324	9
polbooks	0.527	5	(Cafieri et al. [2011])	0.520	4	0.523	7	0.214	7	**0.524**	7
adjnoun	0.308	-	(Li and Schuurmans [2011])	0.286	7	**0.307**	7	0.044	5	0.259	5
football	0.605	10	(Cafieri et al. [2011])	0.604	10	**0.605**	10	0.504	10	**0.605**	10
jazz	0.445	5	(Duch and Arenas [2005])	0.438	4	0.340	27	0.069	4	**0.445**	4
celegansneural	0.495	-	(Catalyürek et al. [2012])	**0.235**	6	0.178	50	0.011	6	0.227	6
celegans metabolic	0.453	-	(Ovelgönne and Geyer-Schulz [2012])	**0.435**	9	0.346	70	0.040	6	0.366	6
email	0.583	10	(Aloise et al. [2012])	**0.541**	11	0.339	162	0.043	10	0.450	10

Table 5.5: Modularity Results

better than other algorithms, with only 0.001 less than the best known value. In average, the solution quality of MMAS is very good for all the smaller networks up until football network. The quality of MMAS starts to degrade from jazz network and reaches the worst point at email network, with 0.202 difference from Louvain algorithm. This result, despite of parameter tuning and more iterations, seems to be caused by large values of k. For example, MMAS finds 162 clusters on email network whereas the best known method finds 10 clusters. MMAS determines k automatically by breath-first-search (BFS) the connected components in the locus-based solution vector, which is built based on pearson correlation, a similarity measurement. Therefore, a node that is not similar enough to other nodes may be placed as a single cluster in this approach. As a result, if a large number of such nodes exist, MMAS will put each of them into a separate cluster. The failure to find the periphery nodes is a limitation (Fortunato [2010]) of applying similarity based clustering method to many real-life networks that exhibit a power-law distribution (Newman [2003]), that is, only a small proportion of the nodes has high degrees, and the majority of the nodes have lower degrees. This limitation is perhaps further verified by the result of ABC-INTE, a purely similarity-based clustering method with no global optimization objective like MMAS has. The solutions found by ABC-INTE for jazz, celegansneural, celegans metabolic and email all have *modularity* values close to 0, meaning that the ants are not able to reveal the cluster structures at all.

ABC-POLY, on the other hand, has some very impressive results. ABC-POLY performs consistently well on different networks. It tops Louvain, MMAS and ABC-INTE on polbooks, football and jazz. For celegansneural, celegans metabolic and

email, although ABC-POLY is still not as good as Louvain, it outperforms MMAS and ABC-INTE significantly. Overall, ABC-POLY outperforms or achieves the same *modularity* values as MMAS and ABC-INTE on 7 out of the 10 networks. Considering that the only difference between ABC-POLY and ABC-INTE is the intervention of one minor ant every 1000 iterations, the drastic improvement of ABC-POLY over ABC-INTE on solution quality confirms the effectiveness of using heterogeneous types of ants for ant brood clustering algorithm.

5.5.2 Performance

Complexity Analysis

In general, MMAS requires less iterations than ABC-INTE and ABC-POLY, but runs for much longer time due to its intensive computation in each iteration. During the solution construction stage, an ant evaluates each node for the probability to include it in a solution. This step requires $O(m)$ steps if the graph is sparse and stored in an adjacency list, or $O(n^2)$ if the graph is stored in an adjacency matrix. For the membership extraction step, the BFS requires $O(n+m)$ for a sparse graph, or $O(n^2)$ steps for an adjacency matrix graph representation. The calculation of *modularity* requires $O(n^2)$ steps and the pheromone update requires $O(m)$, or $O(n^2)$ steps. The complexity of MMAS in one iteration is therefore $O(n^2)$, and the overall complexity of MMAS is $N_{ants}N_{iter}O(n^2)$.

For ABC-INTE, the iteration complexity is much less than that of MMAS. In each iteration, N_{ants} ants meet at most the N_{ants} objects and evaluate the similarity density in a small neighbourhood in constant time. Because $N_{ants} = 0.1n$,

the iteration complexity of ABC-INTE is $O(n)$, and the overall complexity of ABC-INTE is $N_{ants}N_{iter}O(n)$. As discussed in chapter 4, ABC-POLY requires additional cost of $O(kn^2)$ by minor ants and $O(nkt)$ by the k-means cluster retrieval algorithm, where t is the number of iterations to reach convergence and is usually small thanks to the work of the major ants. The overall complexity of ABC-POLY is then $N_{ants}N_{iter}O(n) + \frac{N_{iter}}{mPer}(O(kn^2) + O(nkt))$, or $N_{ants}N_{iter}O(n) + \frac{N_{iter}}{mPer}(O(n^2) + O(nt))$ if k is a small constant.

Execution Time

The time taken to get the *modularity* values in table 5.5 is shown in table 5.6. For each algorithm, I present both N_{iter} and execution time in seconds to gain a better understanding of the programs' performance. I observe that in practice, ABC-INTE and ABC-POLY are at least 10 times faster than MMAS to achieve their optimality. The execution time of MMAS gets longer for the last four networks because more ants are used to look for better solutions.

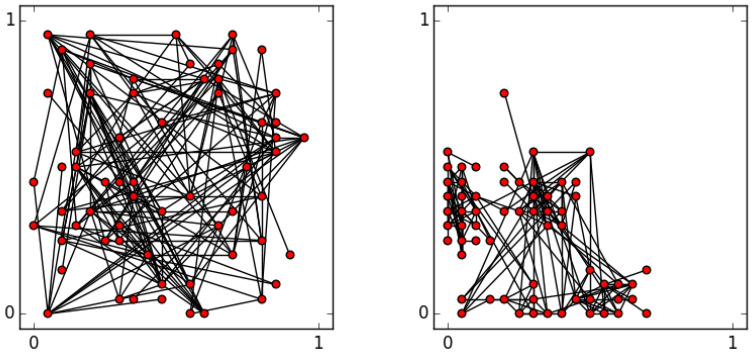
There is no significant difference in the seconds taken by ABC-INTE and ABC-POLY. Although ABC-INTE is faster for karate, lesmis, polbooks, adjnoun, jazz, and celegans metabolic, ABC-POLY is faster for the other networks because less iterations are used to achieve better result.

Network	MMAS		ABC-INTE		ABC-POLY	
	N_{iter}	Seconds	N_{iter}	Seconds	N_{iter}	Seconds
karate k=2	-	-	1000	0.00	1000	0.00
karate k=4	40	0.01	800	0.00	38000	0.17
dolphins	400	0.49	21000	0.13	12000	0.06
lesmis	1000	11.72	4200	0.03	21000	0.31
polbooks	5000	16.98	49000	1.06	45000	1.21
adjnoun	10000	36.61	10000	0.24	81000	1.86
football	3000	12.72	30000	0.72	9000	0.21
jazz	800	287.91	7000	0.18	12000	0.91
celegansneural	200	120.68	20000	5.17	22000	2.84
celegans metabolic	700	114.26	10000	2.88	19000	5.29
email	120	136.29	7000	7.47	6000	4.74

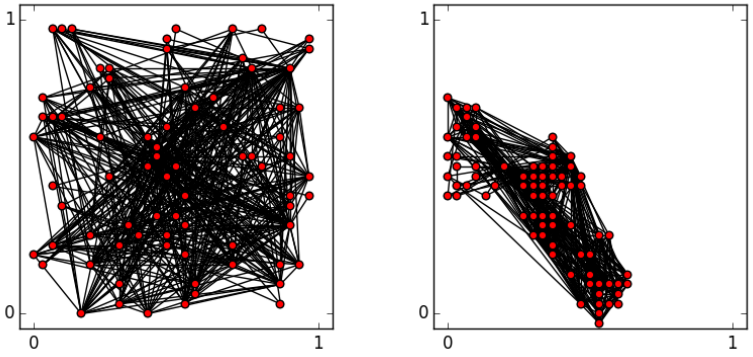
Table 5.6: Execution Time Results

5.5.3 Visual Display of ABC-POLY's Outputs

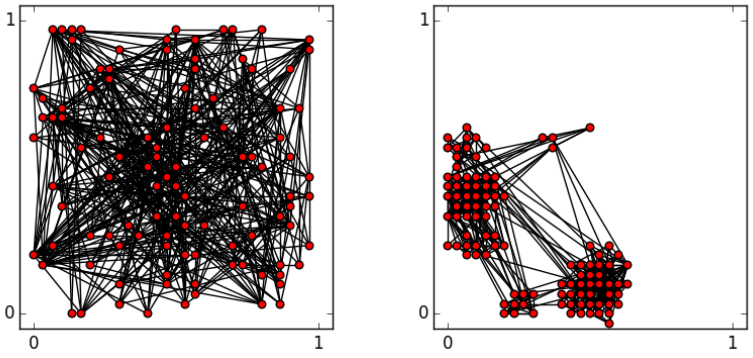
One unique advantage of the ant brood clustering, as we previously see in chapter 4, is the visual display of its output. I present the pleasant layouts of the benchmark graphs as a result of ABC-POLY in the figures on the next few pages. In all the cases, the ants turn a random projection of graph into clearly formed clusters. The reliability of such visual effect is guarded by the high *modularity* values achieved by ABC-POLY.



dolphins time 0 *dolphins* time 12,000

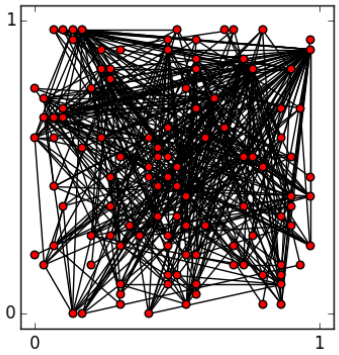


lesmis time 0 *lesmis* time 21,000

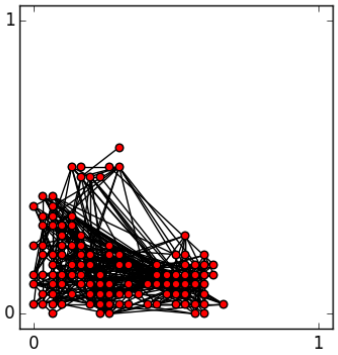


polbooks time 0 *polbooks* time 45,000

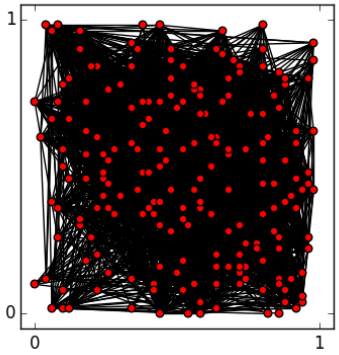
Figure 5.1: Graph Layout Before and After ABC-POLY (Part 1)



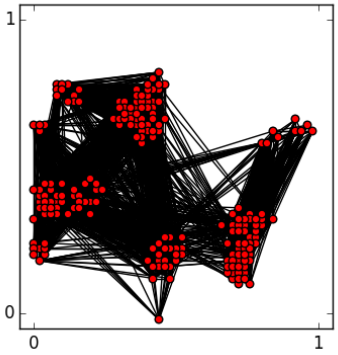
adjnoun time 0



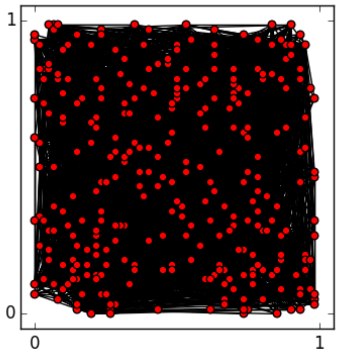
adjnoun time 81,000



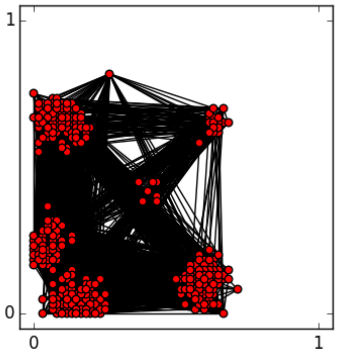
jazz time 0



jazz time 12,000

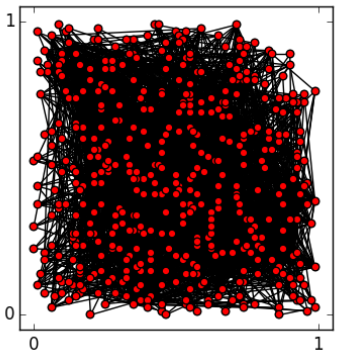


celegansneural time 0

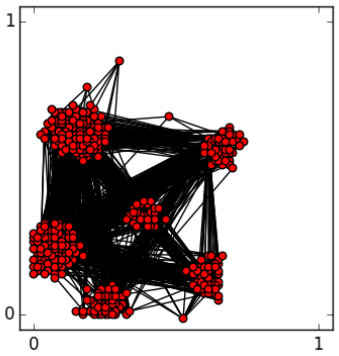


celegansneural time 22,000

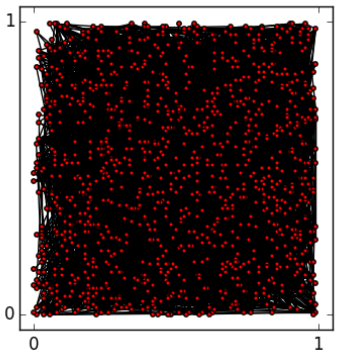
Figure 5.2: Graph Layout Before and After ABC-POLY (Part 2)



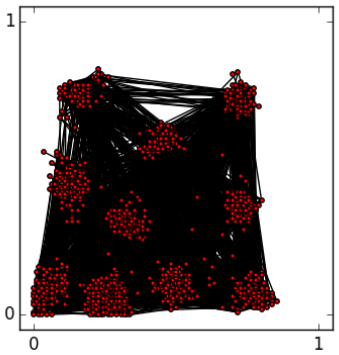
celegans metabolic time 0



celegans metabolic time 19,000



email time 0



email time 6,000

Figure 5.3: Graph Layout Before and After ABC-POLY (Part 3)

Chapter 6

Conclusion

6.1 Conclusion

In this thesis, I have studied a few ant based techniques for graph clustering.

The MMAS algorithm is proposed by (Chang et al. [2013]) and tailors the general framework of ACO to work with the graph clustering problem. I have implemented this algorithm and tested it on 10 real-life benchmark networks. Although MMAS achieves good solution quality for some of the smaller networks, it fails to cluster the periphery nodes on networks that exhibit a power-law distribution.

The focus of this thesis is on the ant brood clustering algorithm (ABC), a less well-known algorithm than ACO. The ABC algorithm directly simulates the brood clustering behavior of ants species by relocating graph vertices on a geometric 2D grid based on the similarity density in a small neighbourhood. This algorithm is very interesting because of its distributive and self-organizational nature. However, the basic ABC-KLS algorithm (Kuntz et al. [1997]) suffers from drawbacks such as

inefficiency, incapability to handle complicated graphs, and lack of quality control.

To improve ABC-KLS algorithm, I introduce *Ant Brood Clustering-Intelligent Ants*, or *ABC-INTE*. ABC-INTE uses techniques such as hopping ants, relaxed drop function, ants with memories, stagnation control, and addition of k-means cluster retrieval process. ABC-INTE is analyzed on a toy graph and the karate network, and shown to be effective. However, a further study of ABC-INTE on the football network exposes the disadvantages of ABC-INTE: small inter-cluster distance and lack of global view.

I further introduce a new algorithm, *Ant Brood Clustering-Polymorphic Ants*, or *ABC-POLY*, as an improvement of ABC-INTE. Inspired by the division of labour between the major and minor ants in *Pheidole* genus (Bonabeau et al. [1999]), I design two types of ants in ABC-POLY: the major and minor ants. Both types of ants move objects, but in different ways. The major ants follow the rules of ABC-INTE. The minor ants, although only intervening the majors' work infrequently, carries the important role of quality assurance. The minors scan each node for possible *modularity* gain if a node is moved to another cluster. After the minors decide which nodes should be moved to achieve a better *modularity* value, they move the objects to the corresponding clusters and these nodes serve as seeds to attract more nodes carried by the major ants in later iterations. This design enhances the *stigmergy* through the rapid and effective environment change.

When tested on the benchmark networks, ABC-POLY outperforms or achieves the same solution quality as both MMAS and ABC-INTE on 7 out of 10 networks and is robust against different graphs. In practice, the speed of ABC-POLY is at

least 10 times faster than ACO, making it a scalable algorithm compared to ACO.

Another unique advantage of ABC-POLY is a pleasant graph layout as an output of this algorithm. With the high quality collaboration of major and minor ants, ABC-POLY can serve as a reliable and intuitive source for direct observation of the community structures in a graph.

6.2 Future Work

This thesis opens an interesting research topic to apply polymorphic ants in the ant brood clustering algorithm for graph clustering. There are still a lot of improvements to be made to ABC-POLY, such as:

- Improve the cluster retrieval process. For example, the number of clusters, k, can be made non-parameterized using other geometric distance based cluster retrieval algorithms than k-means.

- Improve the initial mapping. There are better mapping techniques than random mapping. For example, if the initial mapping represents a solution from a fast greedy clustering algorithm, applying ABC-POLY on it means to find a better solution than the original one. This is essentially a hybrid of ABC-POLY and a common algorithm for graph clustering.

- Hybrid ABC-POLY and ACO. Specifically, it is interesting to study the effect of adding pheromone trails on the top of environmental change to strengthen the stigmergy.

- Parallel implementation of ABC-POLY. The simple agents and localized design of ABC-POLY make it ideal for parallel implementation, especially for parallel computing on GPUs that require simple threads and less communication.

- Experiment ABC-POLY on more graph instances, including large graphs, weighted graphs and dynamic graphs.

- Apply multi-level graph coarsening strategy (Korosec et al. [2003]; Oliker and Biswas [2000]) to improve the scalability of ABC-POLY.

- Experiment the alternative design of applying ABC-POLY on the graph directly instead of on a bijective map. The actions of pick-up and drop-down are then only conceptual. For example, they can be made equivalent to the actions of cluster labeling and unlabeling of the graph nodes.

Bibliography

implementation of Louvain algorithm for community detection, howpublished = `http://perso.crans.org/aynaud/communities/`, note = Accessed: 2016-02-22.

polbooks network by V. Krebs, author = Krebs howpublished = `http://www.orgnet.com/`, note = Accessed: 2016-02-22.

D. Aloise, G. Caporossi, P. Hansen, L. Liberti, S. Perron, and M. Ruiz. Modularity maximization in networks by variable neighborhood search. *Graph Partitioning and Graph Clustering*, 588:113, 2012.

D. A. Bader, H. Meyerhenke, P. Sanders, C. Schulz, A. Kappes, and D. Wagner. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*, pages 73–82. Springer, 2014.

V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 20(2):172–188, 2008.

S. Cafieri, P. Hansen, and L. Liberti. Locally optimal heuristic for modularity maximization of networks. *Physical Review E*, 83(5):056105, 2011.

Ü. V. Catalyürek, K. Kaya, J. Langguth, and B. Uçar. A divisive clustering technique for maximizing the modularity. In *The proceedings of*, volume 10, 2012.

J. Chan, Y. Mao, Y. Liu, P. Thulasiraman, and R. Thulasiram. Parallel ant brood graph partitioning in Julia. In *11th International Conference on Parallel Processing and Applied Mathematics 2015 Chapter 17*, volume 9574 of *LNCS*. Springer, 2016.

H. Chang, Z. Feng, and Z. Ren. Community detection using ant colony optimization. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 3072–3078. IEEE, 2013.

A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–53, 1994.

J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. The dynamics of collective sorting robot-like ants and ant-like robots.

In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 356–363, 1991.

G. C. Di and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *arXiv preprint arXiv:1105.5449*, 2011.

M. Dorigo, V. Maniezzo, A. Colorni, and V. Maniezzo. Positive feedback as a search strategy. Technical report, 1991.

J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.

D. S. Fisher. Random walks in random environments. *Physical Review A*, 30(2):960, 1984.

S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

P. M. Gleiser and L. Danon. Community structure in jazz. *Advances in complex systems*, 6(04):565–573, 2003.

R. Guimera, L. Danon, A. Diaz-Guilera, F. Giralt, and A. Arenas. Self-similar community structure in a network of human interactions. *Physical review E*, 68(6): 065103, 2003.

R. Guimera, M. Sales-Pardo, and L. A. N. Amaral. Modularity from fluctuations in random graphs and complex networks. *Physical Review E*, 70(2):025101, 2004a.

R. Guimera, M. Sales-Pardo, and L. A. N. Amaral. Modularity from fluctuations in random graphs and complex networks. *Physical Review E*, 70(2):025101, 2004b.

J. Handl, J. Knowles, and M. Dorigo. Ant-based clustering: a comparative study of its relative performance with respect to k-means, average link and id-som. In *Proceedings of the Third International Conference on Hybrid Intelligent Systems, IOS Press*, 2003.

J. Handl, J. Knowles, and M. Dorigo. Ant-based clustering and topographic mapping. *Artificial life*, 12(1):35–62, 2006.

B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26:1519–1534, 2000.

J. J. Handl, J. D. Knowles, and M. Dorigo. On the performance of ant-based clustering. In *HIS*, pages 204–213, 2003.

H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabási. The large-scale organization of metabolic networks. *Nature*, 407(6804):651–654, 2000.

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*, 392(15):3260–3272, 2013.

D. Jin, D. Liu, B. Yang, J. Liu, and D. He. Ant colony optimization with a new

random walk model for community detection in complex networks. *Advances in Complex Systems*, 14(05):795–815, 2011.

D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471, 2007.

J. Kennedy. Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer, 2011.

B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307, 1970.

S. Kirkpatrick, D. G. Jr., and M. P. Vecchi. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

D. E. Knuth. The stanford graphbase: a platform for combinatorial algorithms. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 41–43. Society for Industrial and Applied Mathematics, 1993.

P. Korosec, J. Silc, and B. Robic. Mesh partitioning: a multilevel ant-colony-optimization algorithm. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 8–pp. IEEE, 2003.

S. Kumar, G. Chadha, R. K. Thulasiram, and P. Thulasiraman. Ant colony optimization to price exotic options. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 2366–2373, Trondheim, Norway, May 2009.

P. Kuntz, P. Layzell, and D. Snyers. A colony of ant-like agents for partitioning in vlsi technology. In *Proceedings of the Fourth European Conference on Artificial Life*, pages 417–424. MIT Press, Cambridge, MA, 1997.

P. Kuntz, D. Snyers, and P. Layzell. A stochastic heuristic for visualising graph clusters in a bi-dimensional space prior to partitioning. *Journal of Heuristics*, 5(3): 327–351, 1999.

A. Lancichinetti and S. Fortunato. Limits of modularity maximization in community detection. *Physical review E*, 84(6):066122, 2011.

W. Li and D. Schuurmans. Modular community detection in networks. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1366, 2011.

E. D. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In *Proceedings of the third international conference on Simulation of adaptive behavior: from animals to animats 3*, pages 501–508. MIT Press, 1994.

D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

M. E. Newman. The structure and function of complex networks. *SIAM review*, 45 (2):167–256, 2003.

M. E. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

M. E. J. Newman. Detecting community structure in networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):321–330, 2004.

L. Oliker and R. Biswas. Parallelization of a dynamic unstructured algorithm using three leading programming paradigms. *Parallel and Distributed Systems, IEEE Transactions on*, 11(9):931–940, 2000.

M. Ovelgönne and A. Geyer-Schulz. An ensemble learning strategy for graph clustering. *Graph Partitioning and Graph Clustering*, 588:187, 2012.

Y. Park and M. Song. A genetic algorithm for clustering problems. In *Proceedings of the third annual conference on genetic programming*, pages 568–575, 1998.

A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11(3):430–452, 1990.

F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2658–2663, 2004.

H. Rana, P. Thulasiraman, and R. K. Thulasiram. MAZACORNET: Mobility aware

zone based ant colony optimization routing for VANET. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 2948–2955. IEEE, 2013.

S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

T. Stützle and H. H. Hoos. Max–min ant system. *Future generation computer systems*, 16(8):889–914, 2000.

J. P. Wang, E. Osagie, P. Thulasiraman, and P. K. Thulasiram. HOPNET: A hybrid ant colony optimization routing algorithm for mobile ad hoc network. *Ad Hoc Networks*, 7(4):690–705, 2009.

D. J. Watts and S. H. Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440–442, 1998.

W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, pages 452–473, 1977.

H. Zhou. Distance, dissimilarity index, and network community structure. *Physical review e*, 67(6):061901, 2003.

E. Ziv, M. Middendorf, and C. H. Wiggins. Information-theoretic approach to network modularity. *Physical Review E*, 71(4):046117, 2005.