2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)
July 6-11, 2014, Beijing, China

Fuzzy Community Detection in Social Networks Using a Genetic Algortihm

Jianhai Su
Department of Computer Science
Michigan Technological University
Houghton, Michigan 49931
Email: jianhais@mtu.edu

Timothy C. Havens
Department of Electrical and Computer Engineering
Department of Computer Science
Michigan Technological University
Houghton, Michigan 49931
Email: thavens@mtu.edu

***Abstract*—Community structure in a network usually is an indicator of some important hidden pattern, and thus can deepen our understandings of some phenomenon and also enable useful applications. Even though the introduction of Newman's modularity stimulates a magnitude of modularity-based community detection methods, only a few of them are designed for uncovering fuzzy overlapping communities in a network, which in practice are really common in social networks. Considering that modularity maximization is NP-hard and that a genetic algorithm's ability to find fairly good solutions of an NP-hard problem, an $O(n^2)$ genetic algorithm for fuzzy community detection (GAFCD) is proposed based on the generalized modularity, a one-step extension of Newman's modularity. Crossover in GAFCD works as an outer-layer search framework that statistically determines sub search spaces with which a novel mutation operator finds the best partition in each sub search space. We compare our proposed GAFCD method with an existing fuzzy community detection algorithm, MULTICUT spectral FCM (MSFCM), and GALS, one of the most effective disjoint (or crisp) community detection, on 10 real world data sets; it is observed that GAFCD outperforms the other algorithms in terms of finding max-modularity communities. Furthermore, GAFCD is able to find both fuzzy partitions and crisp partitions while MSFCM can only find fuzzy partitions and GALS can only find crisp partitions. This unique feature makes GAFCD the first genetic algorithm for finding truly fuzzy (i.e., inclusive of both fuzzy and crisp communities) max-modularity community structure in a network.**

I. Introduction

COMMUNITY Structure is an important pattern in a social networks, which has nourished many applications, e.g., online commodity recommendation system for online retail companies and dedicated service configuration of mirror servers. In the common view, community structure is groups of vertices such that the number of edges within each group is larger than the number of edges that connect the vertices within the group to the rest of the network. Uncovering community structure is called community detection. Under graph theory, community detection in a network is the process of finding a $c \times n$ partition U of a graph $G = (V, E, W)$, where each entry u_{ki} in U, $k = [c], i = [n]$, is the membership of vertex i in community k. Three types of partitions, possibilistic partitions, fuzzy partitions, and crisp/hard partitions, are respectively

TABLE I. Notations and Symbols

Symbol	Description
$[i:n]$	set of integers from i to n
$[n]$	set of integers from 1 to n
n	number of nodes (vertices) in network
c	number of communities
G	graph $G = (V, E, W)$
V	set of n vertices
E	set of edges, e_{ij} connects vertices v_i and v_j
W	adjacency matrix, $W \in \mathbb{R}^{n \times n}$, where w_{ij} is weight of edge e_{ij}
m_i	degree of vertex v_i
$\mathbf{m}$	degree vector $\mathbf{m} = (m_1, \ldots, m_n)^T$
B	modularity matrix $B = W - \mathbf{m}^t\mathbf{m}/\|W\|$
U	partition or cover matrix, $U = [u_{ki}]^{c \times n}$, $u_{ki} \in [0, 1]$
$\mathbf{u}_i$	ith column of U
$\mathbf{1}_d$	d-dimensional vector of all ones, $(1, 1, \ldots, 1)$
$\mathbf{0}_d$	d-dimensional vector of all zeros, $(0, 0, \ldots, 0)$

defined as [1]:

$$M_{pcn} = \left\{ U \in R^{c \times n}; 0 \le u_{ki} \le 1, \forall k, i; \sum_{k=1}^{c} u_{ki} \le c, \forall i; \sum_{i=1}^{n} u_{ki} < n, \forall k \right\}, \quad (1a)$$

$$M_{fcn} = \left\{ U \in M_{pcn}; \sum_{k=1}^{c} u_{ki} = 1, \forall i \right\}, \quad (1b)$$

$$M_{hcn} = \{ U \in M_{fcn}; u_{ki} \in \{0, 1\} \}. \quad (1c)$$

Note that $M_{hcn} \subset M_{fcn} \subset M_{pcn}$. Hence, possibilistic partitions include fuzzy partitions, which include crisp partitions.

The validity of community detection methods is based on the "goodness" of the found partitions, as evaluated by some quality function. The most popular quality function for community detection is modularity, first introduced by Newman and Girvan [2] to evaluate crisp partitions (or disjoint community structures) of a network. Based on Newman's modularity, a number of methods for disjoint community detection have sprung up, the the majority of which are categorized into greedy methods, extremal optimization, simulated annealing, and spectral optimization [3]. Many methods use heuristics to find good solutions because the modularity maximization problem is NP-complete [4].

Due to the effectiveness in approximating the solution to NP-hard problems, *genetic algorithms* (GAs) have attracted an increasing attention in community detection. In general, these GAs are divided into two groups according to their

978-1-4799-2072-3/14/$31.00 ©2014 IEEE

chromosome representations. The first group uses a string-of-group encoding, introduced by Tasgin and Bingol [5], to represent a chromosome. The second is based on the locus-based adjacency representation (LAR), which was first adopted to a GA by Pizzuti [6], together with traditional crossover and mutation operators. Following Pizzuti's work, a new and one-way like crossover operator was introduced by Shi et al. [7], which makes their GA the first to be appropriate for large networks. To further improve the clustering quality, Di et al. [8] proposed a new LAR-based GA called GALS. The main innovation of GALS is its mutation operator, which is a local search strategy that optimally assigns a community label for each marginal node.

All existing GAs for modularity-based community detection can only find disjoint communities in a network. But fuzzy communities are very common in real world networks. For example, a person can only partially dedicate herself to all her communities, e.g. school, friends, and clubs, but not spend equal energy in each of them. So, we are motivated to propose a GA for fuzzy community detection that is based on the generalized modularity (GM) [9], a soft extension of Newman's modularity. The rest of this paper is organized as follows: first, the generalized modularity is introduced in Section II; then, the design of our GA is discussed in Section III, followed by experiments and discussion in Section IV; lastly, Section V summarizes.

II. Generalized Modularity

Newman's modularity is built on the assumption that no community structure exists in a random graph. So, for a crisp c-partition U of a graph $G = (V, E, W)$, the modularity is computed by comparing the edges within communities with such edges in a null model that is a random graph with the same vertex degrees as that in the input network. Here, W is a positive and symmetric edge weight matrix of G, where w_{ij} is the weight of the edge that connects vertices v_i and v_j. Let the c crisp vertex subsets be $V_1, ..., V_c$. Then, the modularity of partition U for network G is

$$Q_h = \sum_{k=1}^{c} \left[\frac{S(V_k, V_k)}{S(V, V)} - \left(\frac{S(V_k, V)}{S(V, V)} \right)^2 \right], \tag{2}$$

where $S(V_a, V_b) = \sum_{i \in V_a, j \in v_b} w_{ij}$. By letting $m_i = S(i, V) = \sum_{j=1}^{n} w_{ij}$ and $\|W\| = \sum_{i,j=1}^{n} w_{ij}$, an equivalent equation of (2) is given in [3]:

$$Q_h = \frac{1}{\|W\|} \sum_{k=1}^{c} \sum_{i,j \in v_k} \left(w_{ij} - \frac{m_i m_j}{\|W\|} \right). \tag{3}$$

Adding a selection variable $u_{ki}u_{kj}$ to Newman's modularity, (3) is extended into the generalized modularity, shown at (4).

$$Q_g = \frac{1}{\|W\|} \sum_{k=1}^{c} \sum_{i,j \in V_k} \left(w_{ij} - \frac{m_i m_j}{\|W\|} \right) u_{ki} u_{kj}, \tag{4a}$$

$$= \frac{1}{\|W\|} \sum_{k=1}^{c} U_k B U_k^T, \tag{4b}$$

$$= \mathrm{tr}\left(UBU^T\right) / \|W\|. \tag{4c}$$

where $B = [W - (\mathbf{m}^T \mathbf{m} / \|W\|)]$ and $\mathbf{m} = (m_1, ..., n_n)^T$. As we can see, the selection variable $u_{ki}u_{kj}$ tells the degree of truth to which both v_i and v_j belong to community k. Q_g is a truly soft modularity such that it can be used to evaluate all three types of partitions. Furthermore, Q_g reduces to Q_h for crisp partitions. It also explicitly shows how a partition is involved in the computation of modularity.

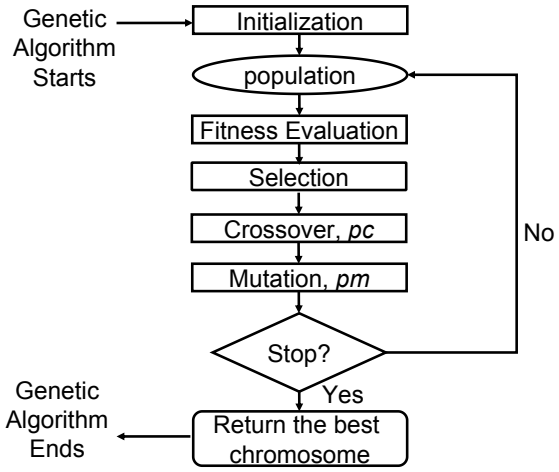

Fig. 1. The diagram of Standard Genetic Algorithm

Although Zhang et al. [10] and Liu [11] proposed other modified modularities for fuzzy community detection, their fuzzy modularities are not truly soft because both modularities require an input fuzzy partition to be first hardened. Their rules for hardening a fuzzy partition are different, but the resulting modularities are the same, which is proven in [9]. GM, together with a modified MULTICUT spectral fuzzy c-means method (MSFCM) [12], shows a better performance in finding community structure than each of the two modularities proposed by Zhang and Liu (with their corresponding community detection methods). MSFCM applies FCM to the resulting spectral features extracted from the graph G.

III. Genetic Algorithm for Fuzzy Community Detection

The standard genetic algorithm (SGA) proposed in [13] aims to solve optimization problems by mimicking *natural selection*, where competitive individuals have a higher chance to survive and reproduce. In SGAs, an individual (also named chromosome), as represented by a binary bit string, is a candidate solution of the targeted problem. Before evolution, a population of individuals are randomly initialized. At each iteration, the competitiveness of individuals are first evaluated based on some quality function that assigns a fitness score to each individual. Then, individuals are selected for crossover and mutation with predefined probabilities pc and pm respectively. The selection process guarantees that an individual with a higher fitness score will be chosen with a higher probability. After a new generation is produced, SGA terminates and returns the best individual of the current generation if some stopping conditions are satisfied. Otherwise, another iteration begins. The diagram of an SGA is illustrated in Fig. 1.

Existing GAs still adopt the population-based framework of SGA, but use an appropriate representation for a given problem solution and modify each operation accordingly.

Inspired by the idea of GALS, one of the most effective disjoint community detection methods, we design a genetic algorithm for fuzzy community detection called GAFCD. In GAFCD, a chromosome is represented by a fuzzy partition, while the fitness score of the individual is its GM. Two stopping conditions are applied in GAFCD: 1) a maximum number of generations g_{max} have been computed; and 2) the highest-ranking solution's fitness score has reached a plateau such that no more than ϵ_Q improvement is observed between two successive generations for a given number of iterations, occ_{max}. For population initialization, a given number, npc, of fuzzy partitions with the same community number c, $c_{min} \leq c \leq c_{max}$ are initialized by the one-step FCM initialization in Alg. 1. Thus, the total population size, $popSize$, is $npc \times (c_{max} - c_{min} + 1)$.

Algorithm 1: One-Step FCM for Random Cover Initialization

Input: Adjacency matrix W; number of communities c
Output: Initialized partition U
$m = 1.7$
I is a random permutation of $[n]$
$V = (\mathbf{v}_1, \ldots, \mathbf{v}_c) = (\mathbf{0}_n, \ldots, \mathbf{0}_n)$
for $k = [c]$ **do**
  $v_{k,I(k)} = 1$
  $d_{ki} = \|\mathbf{w}_i - \mathbf{v}_k\|^{2/(m-1)}$, $i = [n]$
$u_{ki} = \left(\sum_{j=1}^{c} \frac{d_{ki}}{d_{ji}} \right)^{-1}$, $i = [n]$

In each iteration, the current population is first sorted in descending order according to their fitness scores, then the top cp percent of the current population are directly selected as potential chromosomes in the next generation. This is called *elitism*. Next, all chromosomes are randomly paired for crossover and mutation with probabilities pc and pm, respectively. For each parent pair, two children are produced independently via crossover and mutation. Here, crossover acts like a guide that leads GAFCD to swiftly converge to an optimal (or good enough) solution; while mutation aims to find the best fuzzy partition in a search subspace. This subspace is all fuzzy partitions that have the same number of communities c as the mutated partition. Finally, all new children are grouped together with the kept *elite* chromosomes and are sorted in descending GM order; the top $popSize$ individuals are picked to be the new population. GAFCD iterates until either of the two stopping conditions is met and returns the best individuals in that generation. The diagram of GAFCD is shown in Fig. 2, while the algorithm is defined in Alg. 2. To our knowledge, GAFCD is the first GA that can uncover max-modularity fuzzy community structure of a network.

Algorithm 2: Genetic Algorithm For Fuzzy Community Detection (GAFCD)

Input: adjacency matrix, W; crossover probability, pc; mutation probability, pm; range of community numbers, $[c_{min}, c_{max}]$; number of initialized partitions with the same community number, npc; copy percentage in elitism strategy, cp; stopping criteria, $(\epsilon_Q, g_{max}, occ_{max}, t_{max})$.
Output: partition, U
$n \leftarrow$ the number of rows in W
$B = W - \mathbf{m}^T\mathbf{m}/\|W\|$
$popSize = npc \times (c_{max} - c_{min} + 1)$
Initialize each $population$ chromosome by Alg. 1 for $c \in [c_{min}, c_{max}]$
Compute initial best modularity $prebestQ$ of population
$g = 1$, $numOcc = 0$, $tempPop \leftarrow \emptyset$, $eliteSize = ceil(popSize * cp)$
while $g \leq g_{max}$ **do**
  $ecnss =$ average GM of $population$
  $R =$ random permutation of integers $[popSize]$
  for $curpopsize = 0 : 2 : popSize$ **do**
    Selection:
    $ch1 = population(R(curpopSize + 1))$
    $ch2 = population(R(curpopSize + 2))$
    Crossover:
    if $unifrnd(0, 1) <= pc$ **then**
      $[ch1, ch2] = crossover(n, ecnss, ch1, ch2)$
    Mutation:
    if $unifrnd(0, 1) <= pm$ **then**
      $ch1 = mutation(W, ch1, B, \|W\|, t_{max}, \epsilon_Q)$
    if $unifrnd(0, 1) <= pm$ **then**
      $ch2 = mutation(W, ch2, B, \|W\|, t_{max}, \epsilon_Q)$
    add $ch1$ and $ch2$ to $tempPop$
  $curPop \leftarrow$ {top $eliteSize$ chromosomes of $population$} $\cup$ $tempPop$
  $population \leftarrow$ top $popSize$ chromosomes of $curPop$
  $g = g + 1$, $bestQ \leftarrow \max\{Q_g(population)\}$, $tempPop \leftarrow \emptyset$
  if $bestQ - prebestQ < \epsilon_Q$ **then**
    $numOcc = numOcc + 1$
    if $numOcc > occ_{max}$ **then**
      return $U \leftarrow$ *best partition of population*
  else
    $numOcc = 0$
return $U \leftarrow$ *best partition of population*

A. Crossover

The GAFCD crossover operator is composed of two steps: community number determination and fuzzy partition formation. The work flow of crossover is shown in Fig. 3. First, the community number, c, of a child is chosen from the interval 2 to the sum of its parents' community numbers, which we call the legitimate range, and is decided by *roulette wheel selection* (RWS) such that the probability of choosing c is

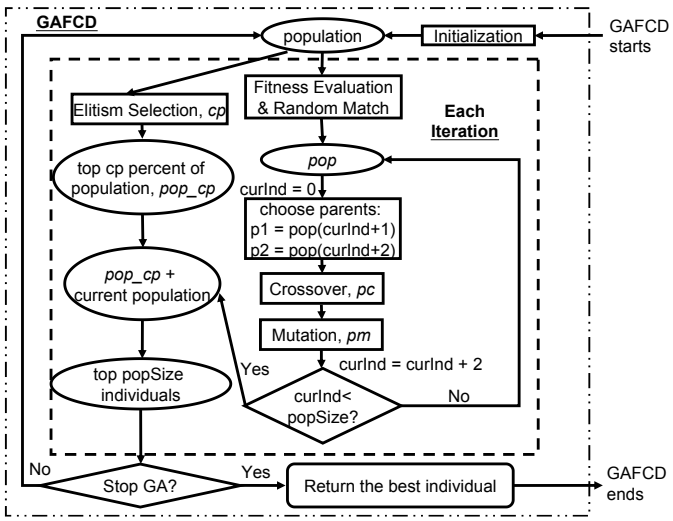

Fig. 2. GAFCD diagram

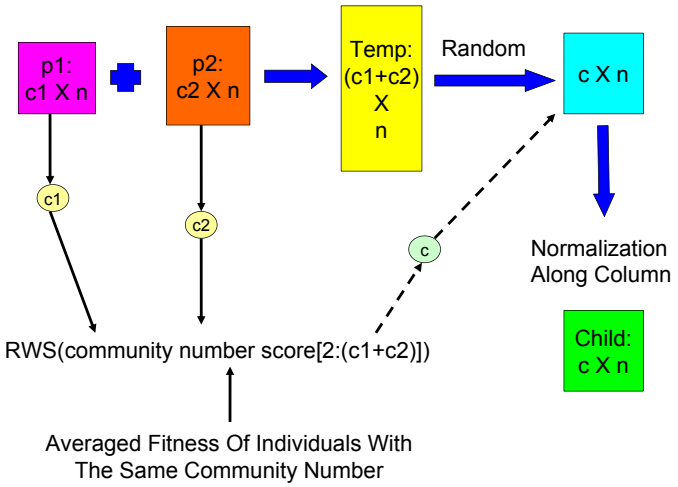

Fig. 3. GAFCD Crossover diagram

proportional to the average fitness score of chromosomes in the current generation, each of which has c communities in its representative partition. For example, if one parent has 3 communities and the other parent has 4 communities, then the number of communities of the child would be selected from the interval $[2:7]$, where each c in this interval has a probability of being chosen proportional to the fitness of current population members with each of those community numbers. This ensures that strong community structures are selected more often than weak ones. The averaged fitness score for a community number is denoted *community number score* (CNS). If a community number is in the legitimate range, then its corresponding CNS is denoted *legitimate community number score* (LCNS). All LCNSs are first scaled such that no negative LCNS exists (because modularity can be negative for very poor community structure), then normalized and fed into the RWS probability calculation. However, for some community numbers in the legitimate range, there might not exist corresponding partitions in the current population. The scores for these community numbers are denoted *non-existing legitimate community number scores* (NLCNSs). Then, for the supplementary set of these community numbers, their scores are named *existing legitimate community number scores* (ELCNSs). Because of the mechanism of RWS, non-existing legitimate community numbers would never be chosen and thus corresponding solution spaces would not be explored. That is, these solutions spaces are dead. To enable our crossover operator to explore the dead solution spaces, scores of all these legitimate but (currently) non-existing community numbers are set to be equal to the minimum ELCNSs divided by the total number of NLCNSs. After the community number c of a child is selected, a child is born by randomly selecting c communities (rows of U) from the parents and then normalizing each column of the child's partition so that the child represents a fuzzy partition. The whole crossover algorithm is described in Alg. 3.

Algorithm 3: Crossover

Input: number of nodes in network, n; existing community number scores, $ecnss$; two parents, $p1$ and $p2$
Output: two children, $child1$ and $child2$

$cp1, cp2 \leftarrow$ community numbers of $p1, p2$ respectively
if $(cp1 + cp2) < n$ **then**
  $maxC = cp1 + cp2$
else
  $maxC = n - 1$
legitimate community scores:
$elcnss \leftarrow$ legitimate scores in $ecnss$
$num_{nlcn} \leftarrow$ the total number of non-existing but legitimate community numbers
each entry of $nlcnss \leftarrow \frac{\min elcnss}{num_{nlcn}}$
legitimate community number scores $lcnss \leftarrow (elcnss, nlcnss)$
$selectionProbs \leftarrow$ scale and normalize $lcnss$
Child 1:
$c1 \leftarrow$ RWS($selectionProbs$)
$ch1 \leftarrow$ randomly pick $c1$ communities from $\{p1 \cup p2\}$
normalize U of $ch1$ along its columns
Child 2:
$c2 \leftarrow$ RWS($selectionProbs$)
$ch2 \leftarrow$ randomly pick $c2$ communities from $\{p1 \cup p2\}$
normalize U of $ch2$ along its columns

B. Mutation

GAFCD muttation aims to find an optimal partition whose community number is the same as that of its input partition. This optimization problem is solved by iteratively calling a local search strategy called *One-Step Modularity Maximization* (OSMM) [14]. OSMM determines the community memberships of a given node (i.e., the corresponding column of a partition U) without altering the memberships of all other nodes such that the modularity of U is maximized. For a detailed derivation of the OSMM process, please refer to [14]; we now give a brief introduction to OSMM, outlined in Alg. 4, and present the OSMM mutation operator, outlined in Alg. 5.

The OSMM process directly maximizes GM by operating upon the membership values of one node at a time, i.e., one column of U at a time. Based on the GM at (4), we can define the contribution of node i—that is, the partition elements $\mathbf{u}_i =$

Algorithm 4: One-Step Modularity Maximization (OSMM)

Input: partition matrix $U \in [0,1]^{c\times n}$; the number of communities, c; the index of the chosen node, i; the modularity matrix B; the degree of the adjacency matrix, $\|W\|$
Output: updated partition matrix, U'; improvement of modularity, ΔQ_i

Relaxed OSMM:
for $k = [c]$ **do**
  $u^o_{ki} = \frac{1}{-B_{ii}} \sum_{j=1,j\neq i}^{n} (B_{ij}u_{kj})$
Simple Quadratic Knapsack Problem:
$H \leftarrow$ a $c \times c$ identity matrix, $f = -u_i^T$
$A_{eq} = \mathbf{1}^{1\times c}, b_{eq} = 1$
$u^*_i \leftarrow$ apply SQKP on the problem (H, f, A_{eq}, b_{eq})
$U' \leftarrow$ replace the ith column of U with u^*_i
CIM:
$\Delta Q_i = \frac{B_{ii}}{\|W\|} \sum_{k=1}^{c} (u^*_{ki} - u_{ki})(u^*_{ki} + u_{ki} - 2u^o_{ki})$

Algorithm 5: Mutation

Input: partition, U; adjacency matrix, W; modularity matrix, B; community number, c; stopping criteria, (ϵ_Q, t_{max}).
Output: partition, U

$n \leftarrow$ number of rows in W;
$t \leftarrow 0$; $\epsilon = Q$, modularity of U; $occ = 0$
while $t < t_{max}$ *and* $\epsilon < \epsilon_Q$ **do**
  $\epsilon = 0$; $I =$ random permutation of $[n]$
  for $i = 1 : n$ **do**
    $[U, \Delta Q_i] =$ OSMM$(U, c, I_i, B, \|W\|)$
    $\epsilon = \epsilon + \Delta Q_i$
  Remove zero-rows in U and Return U

$(u_{1i}, \ldots, u_{ki})$—to Q_g as

$$Q_i = \frac{1}{\|W\|} \sum_{k=1}^{c} \left[2u_{ki} \sum_{j=1,j\neq i}^{n} (B_{ij}u_{kj}) + B_{ii}u_{ki}^2 \right]. \quad (5)$$

It can be shown that instituting an improvement in Q_i by some amount improves the overall GM Q_g by the same amount. Hence, if we can maximize Q_i, then we can iteratively maximize Q by applying the process on each $i \in [1:n]$. This is proven in [14].

The OSMM problem is thus formulated as iterated updates of

$$\mathbf{u}^*_i = \arg\max_{u_i} Q_i, \ \sum_{k=1}^{c} u_{ki} = 1, 0 \leq u_{ki} \leq 1, \ \forall i, k. \quad (6)$$

The solution of (6) turns out to be a two-step process. The first step is called *Relaxed* OSMM (ROSMM), which is accomplished by a closed-form solution of (6) ignoring the constraints $0 \leq u_{ki} \leq 1$. The solution of ROSMM is

$$u^o_{ki} = \frac{1}{-B_{ii}} \sum_{j=1,j\neq i}^{n} (B_{ij}u_{kj}). \quad (7)$$

The second step of OSMM is a projection of each $\mathbf{u}^o_i$ (i.e., the ith column of U^o) into the unit hypercube defined by the inequality constraint $0 \leq u^o_{ki} \leq 1$. It is shown in [14] that this second step is equivalent to solving the *simplified quadratic knapsack program* (SQKP), given by

$$\mathbf{u}^*_i = \arg\min_{\mathbf{u}} \left\{ \frac{\mathbf{u}^T\mathbf{u}}{2} - (\mathbf{u}^o_i)^T\mathbf{u} \right\}, \ \mathbf{1}^T\mathbf{u} = 1, \ \mathbf{0}_c \leq \mathbf{u} \leq \mathbf{1}_c, \quad (8)$$

which is essentially a search for the nearest vector to $\mathbf{u}^o_i$ that lies in the unit hypercube $\mathbf{0}_c \leq \mathbf{u} \leq \mathbf{1}_c$. The significant finding in [14] is that the application of ROSMM at (7) and then SQKP at (8) is a equivalent to finding the global optimum $\mathbf{u}^*_i$ at (6). Hence, OSMM essentially does a local search for the maximum modularity partition by constraining the optimization to the c variables that are the ith column of U.

The objective function of the SQKP at (8) is convex and can be solved in polynomial time by the ellipsoid method [15] and several kinds of interior point methods [16]–[19]. More importantly, the SQKP can be solved in a linear time with either a binary search method [20] or randomized algorithm [21]. In our problem, the number of variables in SQKP is c, the number of communities. Then whole algorithm for OSMM is given in Alg. 4.

By applying OSMM on a given column of U, we transform a given cover U to a new cover U' that is proven in [14] to be at least as good as, if not better than, the original in terms of modularity. So, we propose the mutation operator at Alg. 5 that iteratively applies OSMM on the input partition U until it is observes t_{max} iterations or no more than ϵ improvement in GM.

C. Time Complexity

The most time consuming part of GAFCD is the mutation algorithm, which is called at most $popSize$ times in each iteration. In mutation, OSMM is called at most $t_{max} \times n$ times. OSMM is solved by applying ROSMM to find the point $\mathbf{u}^o_i$ and then SQKP to find the optimal partition column $\mathbf{u}^*_i$. The time complexity of ROSMM is $O(cn)$, while the time complexity of SQPK is $O(c)$. Then, the improved modularity is calculated in a cost of c multiplications, which is denoted as CIM in Alg. 4. So, the time complexity of OSMM is deduced as

$$\begin{aligned} T(OSMM) &= T(ROSMM) + T(SQKP) + T(CIM), \\ &= O(cn) + O(c) + O(c), \\ &= O(cn). \end{aligned}$$

In real world networks, the number of communities c is much smaller than the number of nodes n, so c can be treated as a constant here. That is, the complexity of OSMM is $O(n)$. Thus, mutation and GAFCD run in $O(t_{max} \times n^2)$ and $O(g_{max} \times popSize \times t_{max} \times n^2)$. For significantly large n, all g_{max}, $popSize$ and t_{max} can be considered as constant. Thus, the time complexity of GAFCD becomes $O(n^2)$.

IV. Experimental Results and Discussion

We compare GAFCD with MSFCM and GALS on 10 real-world data sets that are described in Table II. Metabolic

TABLE II. REAL-WORLD NETWORKS USED IN THE EXPERIMENTS

Abbr.	Name	$\|V\|$	$\|E\|$	Network Description
K	Karate	34	78	Zachary's karate club [23]
D	Dolphin	62	159	Dolphin social network [24]
P	PolBooks	105	441	Books on US politics [25]
F	Football	115	613	American college football [2]
J	Jazz	198	2742	Jazz musicians network [26]
S	Sawmill	36	62	Sawmill communication network [27]
L	LesMis	77	254	Coappearances of characters in the musical Les Miserables [28]
W	Words	112	425	Adjacencies of adjectives and nouns [29]
M	Metabolic	453	2025	Metabolic Network of C.elegans [30]
E	Email	1133	5451	Network of e-mail interchanges [31]

TABLE III. RANGE OF COMMUNITY NUMBER FOR MSFCM AND GAFCD ON EACH DATA SET

Data set	K	D	P	F	J
Range of c	[2,10]	[2,10]	[2,10]	[5,15]	[2,10]
Data set	S	L	W	M	E
Range of c	[2,10]	[2,10]	[2,10]	[2,20]	[2,30]

Network is an undirected, weighted graph, but it has 15 loops or self-connections (none of the algorithms here can handle these loops). Here, we simply remove these loops to make Metabolic Network a simple graph. Karate and LesMis datasets are weighted and undirected, while all the other data sets are undirected and unweighted. For both GAFCD and MSFCM, a range of community numbers should be guessed beforehand for each data set, which is shown in Table III. For MSFCM, the community number is limited in this interval. But this is not the case for GAFCD, which can return community numbers in the interval $[2:n]$. The parameters of MSFCM are: $m = 1.7$, termination criterion $\epsilon = 10^{-5}$, and maximum number of iterations $t_{max} = 500$. While the parameters of GAFCD are: $g_{max} = 200$, $pc = 0.9$, $pm = 1.0$, $npc = 10$, $cp = 0.1$, $\epsilon_Q = 1e-5$, $occ_{max} = 10$ and $t_{max} = 100$. Due to the lack of available implementation of the linear SQKP algorithms [20], [21], we use the *qpip* solver [22], which is based on a primal-dual predictor-corrector algorithm.[1] For GALS, we run the code of Ding et al. [8] with their parameter settings except that the number of runs is changed from 50 to 100. Since GALS only works with unweighted graphs, we simply set the edge weights of a weighted network to 1 when the weight is > 0 and 0 else (this is *only* for the computation of the GALS partition; the weighted graph is used for the calculation of the final modularity value).

For each data set, each of the three methods is tested for 100 independent runs that are accomplished in a single dedicated core of the Superior Computing Cluster [32]. Experimental results are reported from three perspectives, shown in Table IV: 1) averaged modularity and standard deviation, where bold indicates statistically superior results under the 2-sample t-test; 2) the best modularity and the corresponding community number, where bold indicates the highest modularity; 3) running times in seconds, where bold indicates the fastest running time. As seen in these three tables, GAFCD is far superior to MSFCM and beats GALS in most tests in terms of modularity. Even though GALS is an $O(n)$ method—if the input graph is sparse—GALS runs slower than GAFCD on the 8 relatively small data sets. Possibly, some parameters in GALS, like population size and maximum generations, impair its efficiency.

[1]Note that the *qpip* solver is a generalized QP solver that is not specifically built for SQKP problems; hence, we believe that a dedicated SQKP solver could be used to further improve on the run-time results of GAFCD.

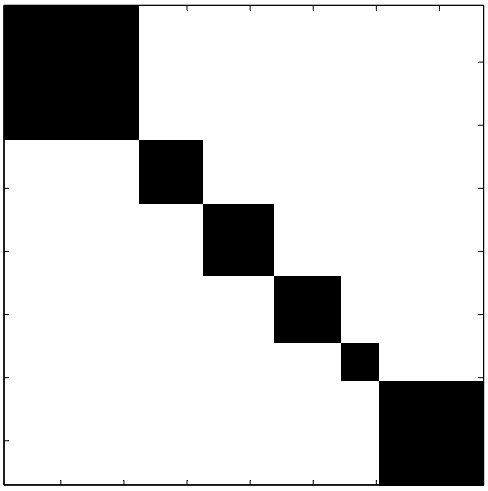

Fig. 4. LesMis, $Q = 0.5667$, $c = 6$, best partition found is crisp

TABLE V. METABOLIC: COMMUNITIES TO WHICH EACH FUZZY NODE BELONGS (SEE FIG. 7)

	FN1	FN2	FN3	FN4	FN5	FN6
Communities	2 , 3	3 , 4 , 5	2 , 5	3 , 6	4 , 6	4 , 6

To demonstrate that GAFCD is a truly fuzzy community detection method, we applied *Visual Assessment of Tendency* (VAT) [33] to visualize the best partition found by GAFCD on each of the 10 data sets. First, a dissimilarity matrix $R = 1 - U^T U$ is computed based the best partition U; then R is reordered into RV by VAT; the image of RV is our visualization. Using the visualization result, we observed that GAFCD uncovers fuzzy but not crisp community structures in the Jazz, Word, and Metabolic data sets, and finds crisp community structures in the remaining networks. Neither MSFCM nor GALS is able to do this; MSFCM *always* finds fuzzy communities and GALS *always* finds crisp communities. For brevity, we show only visualization results of Jazz, Les Miserables, Words, Metabolic, and Email in Figs. 4–8, the data sets for which GAFCD outperforms GALS (and MSFCM for that matter). The best partitions found by GAFCD of Les Miserables and Email are crisp, while the best partitions found of Jazz, Words and Metabolic are fuzzy.

To simplify our description, we use *FNi* to represent the ith fuzzy node in a fuzzy partition. For each of Jazz and Words, seen in Figs. 6 and 8, only one fuzzy node is found, which has membership in both community 1 and community 2 in the corresponding fuzzy partition. For Metabolic, shown in Fig. 7, 6 fuzzy nodes are found. The detail of belongingness of these nodes in the 9 communities is given in TableV.

V. CONCLUSIONS

In this paper, we propose an $O(n^2)$ genetic algorithm that detects fuzzy communities in network data by directly maximizing generalized modularity. The main operators of GAFCD, i.e., crossover and mutation, are designed as follows:

TABLE IV. COMPARED PERFORMANCE OF COMMUNITY DETECTION ALGORITHMS

	Algo.	K	D	P	F	J	S	L	W	M	E
mean Q	MSFCM	0.4129	0.3963	0.4596	0.5266	0.3980	0.3279	0.4897	0.0052	0.2588	0.3452
std Q		0.0001	0.0043	0.0009	0.0008	0.0200	0.0001	0.0108	0.0013	0.0118	0.0241
	GAFCD	**0.4449**	**0.5285**	**0.5272**	**0.6046**	**0.4452**	**0.5501**	**0.5667**	**0.3107**	**0.4261**	**0.5741**
		0.0000	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0009	0.0014	0.0015
	GALS	**0.4449**	0.5282	**0.5272**	0.6045	0.4448	**0.5501**	0.5313	0.3094	0.4153	0.5441
		0.0000	0.0004	0.0000	0.0003	0.0001	0.0000	0.0013	0.0020	0.0068	0.0091
Q_{best}	MSFCM	0.4132	0.3991	0.4601	0.5268	0.4078	0.3280	0.4971	0.0083	0.2876	0.3804
c		3	4	3	10	4	5	5	9	7	4
	GAFCD	**0.4449**	**0.5285**	**0.5272**	**0.6046**	**0.4452**	**0.5501**	**0.5667**	**0.3126**	**0.4287**	**0.5782**
		4	5	5	10	4	4	6	7	9	12
	GALS	**0.4449**	**0.5285**	**0.5272**	**0.6046**	0.4449	**0.5501**	0.5439	0.3121	0.4280	0.5575
		4	5	5	10	4	4	6	7	18	38
Time	GAFCD	**1295**	**3783**	**5505**	**7193**	**10061**	**1488**	**3593**	**19744**	180024	682141
(secs)	GALS	5853	9634	15555	18085	32295	4933	11990	19991	**78062**	**192533**

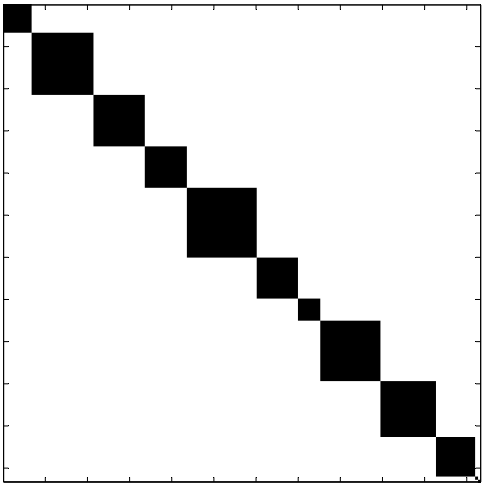

Fig. 5. Email, $Q = 0.5782$, $c = 12$, best partition found is crisp

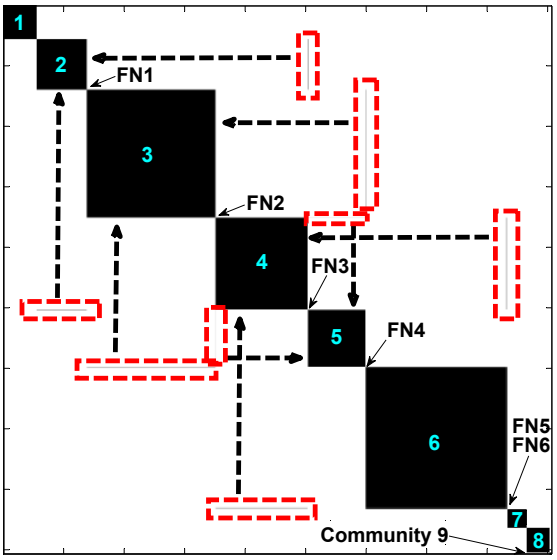

Fig. 7. Metabolic, $Q = 0.4287$, $c = 9$

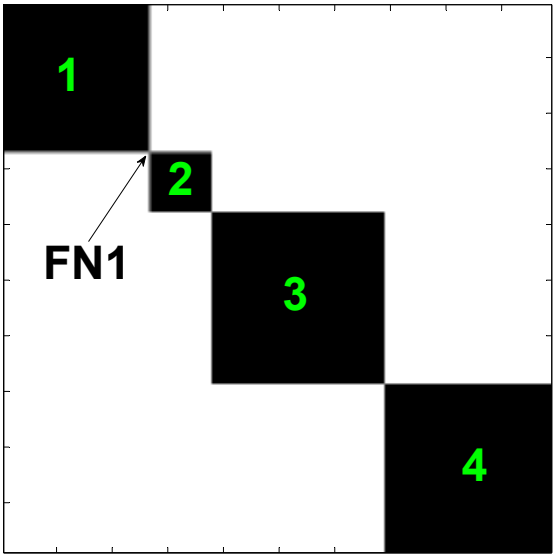

Fig. 6. Jazz, $Q = 0.4452$, $c = 4$

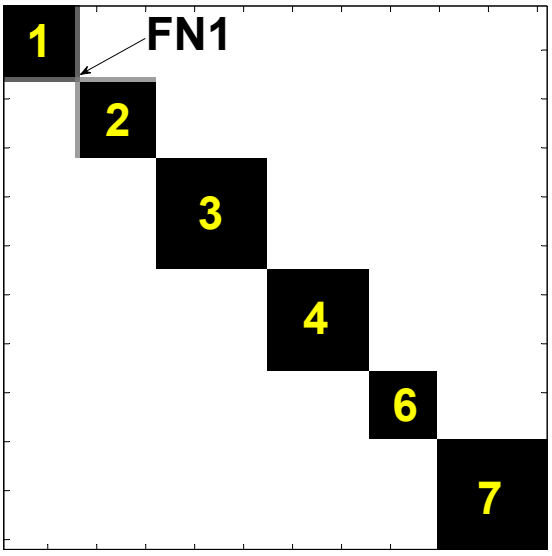

Fig. 8. Words, $Q = 0.3126$, $c = 7$

1) crossover is designed to make GAFCD converge quickly to a best (or fairly good) solution while effectively exploring the search space over all possible community numbers. This is accomplished by statistically determining the optimal community number for a child based on the available scores of different community numbers and community numbers of the two selected parents; 2) mutation aims to find the best partition whose community number is the same as that of the input partition. This is realized by iteratively calling the OSMM algorithm [14] on columns of the input partition. In other words, crossover behaves as a guide to lead mutation to optimally search the best solution in many search subspaces, where each subspace represents the partitions of a single community number. We compared the GAFCD algorithm with the MSFCM and GALS methods; GAFCD shows a better performance in finding the best partition of a network in terms of modularity in *all* tests. Also, GAFCD can reveal a fuzzy partition of a network when it is appropriate and find a crisp partition of a network when appropriate. But MSFCM or GALS can only uncover one of the two types of partitions—MSFCM finds fuzzy partitions and GALS finds crisp partitions. To our knowledge, GAFCD is the only modularity-based community detection algorithm that can return a community in the crisp sub-set of fuzzy partitions when a crisp partition is appropriate. In the future, we will put our efforts to enable our GAFCD workable for large social networks. With the assumption that large social networks are usually sparse graphs, we will attempt to reduce the time cost for computing Q_g for a fuzzy partition. Meanwhile, we will work towards a new effective and but more efficient algorithm to replace the current mutation operator.

REFERENCES

[1] J. C. Bezdek, J. Keller, R. Krisnapuram, and N. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[2] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, pp. 7821–7826, June 2002.

[3] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, pp. 75–174, 2010.

[4] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner, "Maximizing modularity is hard," *arXiv preprint physics/0608255*, 2006.

[5] M. Tasgin and A. Bingol, "Communities detection in complex networks using genetic algorithms," in *Proc. of the European Conference on Complex Systems (ECSS06)*, 2006.

[6] C. Pizzuti, "Community detection in social networks with genetic algorithms," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. ACM, 2008, pp. 1137–1138.

[7] C. Shi, Y. Wang, B. Wu, and C. Zhong, "A new genetic algorithm for community detection," in *Complex Sciences*. Springer, 2009, pp. 1298–1309.

[8] D. Jin, D. He, D. Liu, and C. Baquero, "Genetic algorithm with local search for community mining in complex networks," in *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, vol. 1. IEEE, 2010, pp. 105–112.

[9] T. Havens, J. Bezdek, C. Leckie, K. Ramamohanarao, and M. Palaniswami, "A soft modularity function for detecting fuzzy communities in social networks," *Fuzzy Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013.

[10] S. Zhang, R. Wang, and X. Zhang, "Identification of overlapping community structure in complex networks using fuzzy c-means clustering," *Physica A: Statistical Mechanics and its Applications*, vol. 374, pp. 483–490, 2007.

[11] J. Liu, "Fuzzy modularity and fuzzy community structure in networks," *The European Physical Journal B*, vol. 77, no. 4, pp. 547–557, October 2010.

[12] D. Verma and M. Meila, "A comparison of spectral clustering algorithms," 2003.

[13] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[14] J. Su and T. C. Havens, "Quadratic program-based modularity maximization for fuzzy community detection in social networks," *in review, IEEE Trans. Fuzzy Systems*, 2013.

[15] M. K. Kozlov, S. P. Tarasov, and L. G. Khachiyan, "The polynomial solvability of convex quadratic programming," *USSR Computational Mathematics and Mathematical Physics*, vol. 20, no. 5, pp. 223–228, 1980.

[16] D. Goldfarb and S. Liu, "An O(n^3L) primal interior point algorithm for convex quadratic programming," *Mathematical Programming*, vol. 49, no. 1-3, pp. 325–340, 1990.

[17] R. D. Monteiro, I. Adler, and M. G. Resende, "A polynomial-time primal-dual affine scaling algorithm for linear and convex quadratic programming and its power series extension," *Mathematics of Operations Research*, vol. 15, no. 2, pp. 191–214, 1990.

[18] M. Kojima, S. Mizuno, and A. Yoshise, "An O($\sqrt{n}L$) iteration potential reduction algorithm for linear complementarity problems," *Mathematical Programming*, vol. 50, no. 1-3, pp. 331–342, 1991.

[19] P. M. Pardalos, Y. Ye, and C.-G. Han, "Algorithms for the solution of quadratic knapsack problems," *Linear Algebra and its Applications*, vol. 152, no. 0, pp. 69 – 91, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/002437959190267Z

[20] P. Brucker, "An O(n) algorithm for quadratic knapsack problems," *Operations Research Letters*, vol. 3, no. 3, pp. 163–166, 1984.

[21] P. M. Pardalos and N. Kovoor, "An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds," *Mathematical Programming*, vol. 46, no. 1-3, pp. 321–328, 1990.

[22] A. Wills, "A primal-dual predictor-corrector algorithm based solvoer for general strictly convex quadratic programming problems," May 2007. [Online]. Available: http://sigpromu.org/quadprog/index.html

[23] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, vol. 33, pp. 452–473, 1977.

[24] 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*, vol. 54, pp. 396–405, 2003.

[25] V. Krebs, "Books about U.S.A. politics." [Online]. Available: http://www.orgnet.com/

[26] P. M. Gleiser and L. Danon, "Community structure in jazz," *Adv. Comlex System*, pp. 656–573, July 2003.

[27] J. H. Michael and J. G. Massey, "Modeling the communication network in a sawmill," *Forest Products*, vol. 47, pp. 25–30, 1997.

[28] D. E. Knuth, *The Stanford GraphBase: a platform for combinatorial computing*. Addison-Wesley Reading, 1993, vol. 4.

[29] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Phys. Rev. E*, vol. 74, p. 036104, Sep 2006. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.74.036104

[30] J. Duch and A. Arenas, "Community detection in complex networks using extremal optimization," *Phys. Rev. E*, vol. 72, p. 027104, Aug 2005. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.72.027104

[31] R. Guimerà, L. Danon, A. Díaz-Guilera, F. Giralt, and A. Arenas, "Self-similar community structure in a network of human interactions," *Phys. Rev. E*, vol. 68, p. 065103, Dec 2003. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.68.065103

[32] "Superior: A high performence computing cluster." [Online]. Available: http://superior.research.mtu.edu/

[33] J. Bezdek and R. Hathaway, "VAT: a tool for visual assessment of (cluster) tendency," in *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, vol. 3, 2002.