Shifting Niches for Community Structure Detection

Corrado Grappiolo, Julian Togelius
Center for Computer Games Research
IT University of Copenhagen, Denmark
Email: cogr@itu.dk, juto@itu.dk

Georgios N. Yannakakis
Institute of Digital Games
University of Malta, Malta
Email: georgios.yannakakis@um.edu.mt

***Abstract*—We present a new evolutionary algorithm for community structure detection in both undirected and unweighted (sparse) graphs and fully connected weighted digraphs (complete networks). Previous investigations have found that, although evolutionary computation can identify community structure in complete networks, this approach seems to scale badly due to solutions with the wrong number of communities dominating the population. The new algorithm is based on a niching model, where separate compartments of the population contain candidate solutions with different numbers of communities. We experimentally compare the new algorithm to the well-known algorithms of Pizzuti and Tasgin, and find that we outperform those algorithms for sparse graphs under some conditions, and drastically outperform them on complete networks under all tested conditions.**

Keywords*—*Evolutionary Computation, Niching, Community Structures, Complete Weighted Networks, Sparse Graphs.

I. Introduction

The problem of community structure detection (CSD) in graphs consists in identifying groups of nodes that are similar, based on the information held by the edges connecting them. The most common approach for CSD consists into forming partitions so that the edges connecting the nodes *within* the communities are more numerous or heavier weighted than the edges connecting the nodes *between* the communities. Applications abound in areas where networks/graphs need to analysed, e.g. nature [1], social networks [2], or even multi-player games [3]. More formally, given a graph $G = (V, E)$, where V is the set of nodes $v_i \in V$ and E is the set of edges connecting two different nodes, $e_{i,j} \in E, v_i, v_j \in V$, the CSD task consists in partitioning V into m subsets or communities, based on the structure of E, in order to optimise a quantitative measure function of E. CSD has been extensively studies in various graph types, both directed and undirected [4]–[6], and weighted and unweighted [7]–[9]. However, the task of optimal graph partitioning into community structures is computationally hard, known to be NP-complete over the set of all graphs of a given size [5], [8], meaning that approximate solutions to the problem are needed. Many are the techniques and algorithms used, e.g. spectral graph partitioning [8], [10], greedy approaches [11], or even Ant-Colony Optimisation [12].

In this paper we address the problem of CSD via evolutionary computation. We investigate two graph types: sparse undirected and unweighted ones (sparse graphs), which are extensively used to represent e.g. social and biological networks [1], and fully connected, weighted and directed graphs (complete networks), which constitutes a more complex problem, and are used to represent e.g. geographic functional regions [8], and the level of cooperation in artificial simulations [9] or in computer games [3]. We use Pizzuti's *GaNet* [13] and Tasgin et al.'s algorithm [14] as benchmarks for sparse graphs, being them developed and specially tuned for such condition. For the complete network condition, on the other hand, the benchmark is a genetic algorithm we had previously designed for CSD in complete networks [9]. This *monolithic* algorithm combines solutions of different community sizes in the same population. The novel aspect of this paper comes from the definition of a new algorithm, *NicheShift*, which divides the population into shifting niches depending on the number of community structures in individual solutions. As GaNet and Tasgin's algorithm are specialised for sparse graphs, we convert the complete networks to this format before applying them (*reduced* approach). As our own algorithms are devised for complete networks, we can use them without any preprocessing (*direct* approach). While the reduced approach entails some information loss, it also leads to a smaller search space. Three main research questions are investigated in this paper: (1) will the reduced approach allow existing algorithms for sparse graphs to successfully identify communities in complete networks? (2) will NicheShift provide tangible performance benefits compared to the monolithic algorithm in complete networks? (3) how well will NicheShift perform on sparse graphs, compared to algorithms specifically designed for this condition? We conducted several CSD experiments using synthetic graphs/networks according to the procedure initially proposed by Girvan and Newman [1]. We considered three scenarios, based on the same size of $|V| = 128$ nodes, of four, eight and sixteen communities. Moreover, the scenarios incorporated noise, modelled as the number and weights of the *out-edges*. The results gathered highlight the fact that (1) the reduced approach fails to detect the *true* community structures in complete networks independently of the scenario under investigation; (2) NicheShift provides promising results in both sparse graphs and complete networks, although its performance could still conceivably be enhanced; (3) the monolithic approach does not scale well with respect to the size of the networks in any of the conditions investigated.

II. Related Work

As a an exhaustive review of work in community structure detection (CSD) is beyond the scope of this paper, this section will discuss those studies which we consider to be the most relevant to the research presented in this paper.

Unarguably, Newman and colleagues are among the key contributors: their seminal work in detection of community structures in unweighted and undirected graphs [1] led to the investigation of weighted networks [7] and furthermore to the

investigation of directed networks [5]. The last two studies have inspired work on CSD for directed weighted networks via spectral partitioning [8] and studies on the detection of community structures — based on a complete weighted and directed collaboration networks — of the levels of altruism existent among complex socially driven artificial agents [9].

Genetic search via evolutionary computation has proved successful on undirected and unweighed networks (see [15]–[21] among others). Among scenarios similar to those examined in our study, Gog et al. [22] use an algorithm based on an information sharing mechanism between individuals in a population, whilst Liu et al. [23], on the other hand, decide to adopt an arguably more controlled genetic approach in which a genetic algorithm repeatedly partitions a subset of nodes into 2 subsets. Similarly to Tasgin et al. [14], both studies share the same fitness function and chromosome representation; both algorithms are intended to be applied to sparse graphs.

Finally, with respect to research aims and approaches similar to the one adopted in our paper, Lancichinetti and Fortunato [24] performed a thorough comparison of several algorithms for CSD, though none of them based on evolutionary computation, on a benchmark graph model which is a special case of the planted *l-partition model*. Another benchmark model was proposed for weighted directed networks with overlapping communities [6]. While the benchmark problem adopted there is rather different from the problems addressed here, it would be interesting future work to apply the algorithms presented here to that problem.

III. Synthetic Graph Generation

This Section describes, in details, the steps needed in order to build the sparse graph (see Subsection III-A), and the counterpart complete networks (see Subsection III-B), given a partition of nodes into *true* community structures, for the experimental setups conducted in our research.

A. Sparse Graph Generation

This phase of the algorithm is implemented in accordance to Girvan and Newman [1]. The 3-tuple

$$G_u = (V_u, E, K_u) \tag{1}$$

represents a sparse graph, where V_u is the set of $|V_u| = n$ nodes, E is the set of edges, and K_u is the partition of V_u in $|K_u| = m$ community structures. We denote v_i the i-th node in V_u, $e_{i,j} \in E$ the edge between v_i and v_j, and $k_i \in K_u$ the community structure identity of node v_i. Given n and m, the algorithm first generates K_u, by partitioning V_u into communities of equal size n/m, assigns, to each node v_i, its community structure identity k_i, and sets $E = \emptyset$. Then, given the parameters, z ($1 \leq z \leq n-1$) and $z - out$ ($0 \leq z - out \leq z$), the algorithm stochastically generates z edges for each node, of which $z - out$ edges connect nodes belonging to different community structure identities. Intuitively, the higher the $z - out$ value, the more difficult is the detection of K_u [1], [13], [14]. In our experiments, we will consider $0 \leq z - out \leq z/2$, since higher values for $z - out$ will lead to the generation of graphs in which the *within* community connectivity is lower than the *between* community connectivity, thus contradicting the assumptions made for the solution of the CSD problem.

Figure 1(a) and Fig. 1(b) depict as a graph and as a matrix, respectively, the representation of a sparse graph where $n = 8$, $m = 2$, $z = 3$ and $z - out = 1$. The colours of the nodes represent the partition K_u.

B. Complete Network Generation

The conversion of G_u (1) into a complete network is organised in three sub-steps. First, the complete network, represented by the following 3-tuple:

$$G_w = (V_w, W, K_w) \tag{2}$$

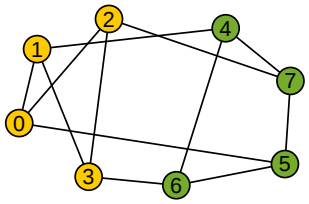
is initialised as follows: $V_w = V_u$, $K_w = K_u$, and $W = \emptyset$. Each weighted directed edge $w_{i,j} \in W$, connecting node v_i to node v_j, will be generated via Normal distribution, with mean values dependent on E and K_w, and unique standard deviation:

$$w_{i,j} = \begin{cases} N(\mu_{in,1}, \sigma) & \text{if } k_i = k_j, e_{i,j} \in E \\ N(\mu_{out,1}, \sigma) & \text{if } k_i \neq k_j, e_{i,j} \in E \\ N(\mu_{in,0}, \sigma) & \text{if } k_i = k_j, e_{i,j} \notin E \\ N(\mu_{out,0}, \sigma) & \text{if } k_i \neq k_j, e_{i,j} \notin E \end{cases} \tag{3}$$

with $w_{i,j}$ bounded to the $[0, 1]$ interval. In our experiments, we will consider $\mu_{in,1} = 0.8$, $\mu_{out,1} = 0.6$, $\mu_{in,0} = 0.4$, $\mu_{out,0} = 0.2$ and $\sigma = 0.2$. The choices of setting the $\mu_{x,y}$ values equidistant from each other, and with values which prioritise first the existence of edges in E, and second the community structure identities, together with the choice of setting σ equal across the four distributions, are motivated by the willingness to generate complete networks for which the underlying sparse graph can still be hinted. Figure 2(a) and Fig. 2(b) depict a complete network (in both graph and matrix form) obtained from the previous sparse graph (see Figure 1).

IV. Community Structure Detection via Genetic Algorithms

Given a sparse graph $G_u = (V_u, E)$, or a complete network $G_w = (V_w, W)$, the solution of the CSD problem is a partition $\tilde{K}$ of the node set V_u (V_w), generally based on a metrics function of E (W), which would correspond to the unknown partition K_u (K_w), see Eq. (1) and (2). In this paper we consider four algorithms: Pizzuti's GaNet [13] and Tasgin's et al. algorithm (Tasgin) [14], which were conceived to solve the CSD problem for sparse graphs, a genetic algorithm mechanism we developed previously to detect community structures in complete networks (Monolithic) [9], and finally NicheShift, a modified version of the Monolithic algorithm based on the concept of shifting niches. A quick recap of the differences and similarities among the algorithms is presented in Table IV. Subsections IV-B, IV-C and IV-D respectively describe the main characteristics of GaNet, Tasgin and the Monolithic algorithm, whilst subsection IV-E details NicheShift.

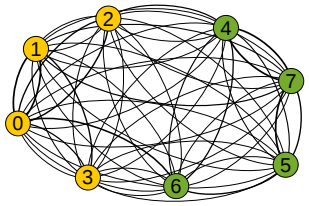


(a) Sparse Graph — Graphical Representation

	0	1	2	3	4	5	6	7
0		1	1	0	0	1	0	0
1	1		0	1	1	0	0	0
2	1	0		1	0	0	0	1
3	0	1	1		0	0	1	0
4	0	1	0	0		0	1	1
5	1	0	0	0	0		1	1
6	0	0	0	1	1	1		0
7	0	0	2	0	1	1	0	

(b) Sparse Graph — Matrix Representation

Fig. 1. The graphical representation of a generated sparse graph for $n = 8$ nodes, $m = 2$ communities, $z = 3$ edges, and $z - out = 1$ *out-edges*.



(a) Complete Network — Graphical Representation

	0	1	2	3	4	5	6	7
0		1.00	0.80	0.73	0.00	0.26	0.90	0.09
1	1.00		0.24	0.80	0.57	0.00	0.45	0.64
2	0.86	0.28		0.36	0.73	0.13	0.20	0.45
3	0.49	0.62	0.60		0.08	0.43	0.08	0.26
4	0.16	0.00	0.49	0.05		0.65	0.75	1.00
5	0.35	0.12	0.01	0.43	0.62		0.79	1.00
6	0.48	0.19	0.12	0.11	0.67	0.45		0.50
7	0.17	0.70	0.20	0.14	1.00	0.79	0.50	

(b) Complete Network — Matrix Representation

Fig. 2. The conversion of the sparse graph depicted in Figures 1(a) and 1(b) into a complete network.

TABLE I. OVERVIEW OF THE MAIN CHARACTERISTICS OF THE FOUR ALGORITHMS CONSIDERED IN THIS STUDY

	GaNet	Tasgin	Monolithic	NicheShift
Requires reduced approach for complete networks	Yes		No	
Genetic Representation	Subnetworks	Community Structure Identities		
Detection Approach	n/a	Agglomerative	Divisive	
Dependent on the graph structure	Yes		No	
Termination Condition	Fixed number of generations		Un-improvement of fitness values	
Optimisation	Fitness Maximisation			

A. Reduced Approach to CSD for Complete Networks

Given that GaNet and Tasgin are designed for sparse graphs [13], [14], in order to be applied to complete networks without any internal algorithmic changes, a reverse transformation from the complete network into a sparse graph is required. We will hereafter refer to this transformation process as the *reduced approach*. The assumption made by this approach is that G_w represents a noised version of the original G_u; if it were possible to filter out such noise, then the complexity of the problem would be reduced, eventually making the correct detection of K a simpler task as well. Hence, given G_w, the output of the re-conversion is the following graph:

$$\hat{G}_u = (\hat{V}_u, \hat{E}) \tag{4}$$

where $\hat{V}_u = V_w$ is given, and $\hat{E}$ is the result of the transformation of W, here described. The first step transforms G_w into an undirected weighted graph; the result is an approximate weighted edge set $\hat{W}$, in which each edge $\hat{w}_{i,j}$ is calculated as follows:

$$\hat{w}_{i,j} = \hat{w}_{j,i} = \frac{1}{2}\left(w_{i,j} + w_{j,i}\right) \tag{5}$$

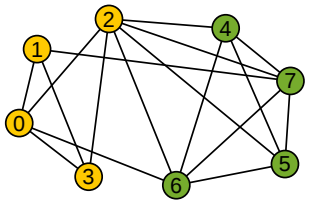
finally, by defining $\mu_{\hat{W}}$ as the average weight of $\hat{W}$, each approximated edge $\hat{e}_{i,j} \in \hat{E}$ is built as follows:

$$\hat{e}_{i,j} = \begin{cases} 1 & \text{if } \hat{w}_{i,j} > \mu_{\hat{W}} \\ 0 & \text{if } \hat{w}_{i,j} < \mu_{\hat{W}} \\ U(0,1) & \text{if } \hat{w}_{i,j} = \mu_{\hat{W}} \end{cases} \tag{6}$$

Clearly, the re-conversion introduces an approximation error. Figure 3(a) and Fig. 3(b) depict, respectively as graph and as matrix, the re-converted complete network depicted in Fig. 2.

B. GaNet Algorithm

Beside its reported success on the sparse graph CSD problem [13], the choice of GaNet as a baseline mechanism for our study was mainly driven by the different genetic representation held by its chromosomes. GaNet relies on the locus-based adjacency representation: each chromosome has length of n genes, and the allele value j of gene i represents the edge $e_{i,j}$. Its initialisation process takes in account the effective connections of the nodes in E: allele value l of genes i will be *repaired*, if $e_{i,l} \notin E$, by replacing it with an allele value j for which $e_{i,j} \in E$. This guided initialisation of *safe* individuals biases the algorithm towards a decomposition of the network in connected groups of nodes [13]. The algorithm makes use of standard elitism, roulette wheel selection and uniform crossover mechanisms, whilst its mutation operation is dependent on the structure of E, similarly to the initialisation process. GaNet identifies the subnetworks represented by the



(a) Converted Graph — Network Representation

	0	1	2	3	4	5	6	7
0		1	1	1	0	0	1	0
1	1		0	1	0	0	0	1
2	1	0		1	1	1	1	1
3	1	1	1		0	0	0	0
4	0	0	1	0		1	1	1
5	0	0	1	0	1		1	1
6	1	0	1	0	1	1		1
7	0	1	1	0	1	1	1	

(b) Reconverted Undirected Unweighted Graph — Matrix Representation

Fig. 3. An example of re-conversion of the complete weighted digraph depicted in Fig. 2(a) and Fig. 2(b). Compared to the original sparse graph depicted in Fig. 1(a) and Fig. 1(b), the reconversion created 8 new and omitted 3 old edges.

chromosomes, the subnetworks are then interpreted as community structures, and the fitness value of such subnetwork, called *community score*, is an edge density measure, based on volume and row/column means, of the subnetwork's edges extracted from E. The evolutionary process aims to maximise the *community score* and is terminated after a fixed number of generations.

C. Tasgin Algorithm

The choice of Tasgin et al.'s algorithm as baseline mechanism for our study was driven by its reported efficiency on sparse network CSD [14] and the semantic differences of the approach compared to the algorithms we propose. Tasgin considers a population of chromosomes with length n; the allele value j of gene i represents the community structure identity $k_i = j$ of node v_i. Similarly to GaNet, Tasgin leverages on the given edge set E to initialise its genetic population. Specifically, Tasgin first initialises each gene i with allele value i, i.e. it starts by considering n communities; subsequently, a refinement process reduces the number of communities, by iteratively selecting a gene i and assigning its allele value i to all the genes j for which $e_{i,j} \in E$. In other words, Tasgin can be interpreted as an agglomerative algorithm. Tasgin implements elitism — i.e. the g-th best chromosome mates with the g+1-th best chromosome realising the *one-way crossing over* operation — which transfers entire community structures from the first to the second parent, and mutation defined as the swapping of the allele values of two uniformly selected genes. The fitness function used is the modularity measure initially proposed by Newman and Girvan [4]. Evolution aims to maximise that modularity measure and terminates after a fixed number of generations.

D. Monolithic Algorithm

This algorithm, which was designed to solve the CSD problem directly on complete networks, is motivated by our ongoing research aiming to computationally infer the existence of collaborative group structures in complex artificial societies [9]. The algorithm provided promising results, though for small scenarios, i.e. 20 and 50 nodes, which allowed us to conclude that it is possible to solve the CSD problem without reducing its complexity (*direct* approach).

Similarly to Tasgin, the Monolithic GA maintains a genetic population of n-gene chromosomes, in which the allele value j of gene i represents the community structure identity $k_i = j$ of node $v_i \in V$. The algorithm implements elitism, rank selection, uniform crossover and standard mutation operations. The peculiarity of the Monolithic algorithm resides on the way the possible allele values are defined. More specifically, the algorithm maintains an alphabet C, which holds the labels of possible community structure identities and from which the allele values are sampled. At the beginning of the evolutionary run, C contains only two symbols and its genetic population is initialised by sampling allele values uniformly within C. If for l consecutive generations the average fitness of the elite chromosomes does not improve then, (a) a new symbol is added to C, (b) all but the most fit chromosomes are thrown away, and (c) the non-elite population is re-initialised by uniformly sampling allele values from the new alphabet C. If, for g consecutive generations, the average fitness of the elite population does not improve, independently of the augmentation of C, the evolutionary process ends. The fitness function under maximisation is defined as follows:

$$f(x) = \frac{1}{m} \sum_{i,j} \left(W_{i,j} - \frac{w_i^{\text{in}}\ w_j^{\text{out}}}{w} \right) \delta(k_i, k_j) \qquad (7)$$

where $m = \sum_{i,j} W_{i,j}$ is the total sum the weights of the given edge set W; w_i^{in} is the in-degree of gene/node i; w_j^{out} is the out-degree of gene/node j; k_i and k_j are the allele values/community structure identities of gene/nodes i and j, respectively; and $\delta(k_i, k_j)$ is the Kronecker delta symbol, for which $\delta(k_i, k_j) = 1$ if $k_i = k_j$ and $\delta(k_i, k_j) = 0$ otherwise [8], [9]. While the algorithm relies on its ability of finding sub-optimal solutions based on current C alphabet, once a local optima is considered to be found — i.e. the limit of l generations of fitness un-improvement is reached —, the algorithm proceeds with deepening its search by adding a new symbol to C — i.e. by considering $|C| + 1$ community structures. Thus, opposed to Tasgin, the Monolithic approach can be interpreted as a divisive algorithm.

E. NicheShift Algorithm

Although earlier studies have provided promising results for 20-node complete networks, the Monolithic algorithm showed some weaknesses in networks of 50 nodes [9]. A thorough analysis of the generated evolutionary landscapes and genetic population highlighted that despite Monolithic was augmenting its alphabet C — and hence promoting the exploration of the search space — eventually the elite

chromosomes were almost always filled with the same sub-optimal solutions which were considering a lower number of community structures with respect to the *true* ones. The main reason for this unwanted behaviour appears to be that Monolithic does not allow for a fair competition between new unfit offspring — which could potentially lead to better community structure partitions — and old fit individuals — which already converged to suboptimal solutions. NicheShift aims to overcome this drawback.

NicheShift is composed of h niches $H_1, H_2, \dots H_h$ of equal population size of p chromosomes of length n; similarly to Monolithic and Tasgin, the allele value j of gene i represents the community structure $k_i = j$ of node v_i. Similarly to Monolithic, NicheShift evaluates its chromosomes in order to maximise the fitness function as defined in Eq. (7). Each niche H_k is based on its own alphabet C_k; the peculiarity of NicheShift resides on the existence of an incremental ordering of the niches and thus their alphabets. We could imagine the niches being horizontally aligned, as depicted in Figure 4: the leftmost niche H_1 has alphabet $C_1 = \{x^1\}$, the subsequent niche H_2 has alphabet $C_2 = C_1 \cup \{x^2\}$, and so on until the rightmost niche, H_h, of alphabet $C_h = C_{h-1} \cup \{x^h\}$.

Algorithm 1 presents the pseudocode of NicheShift's main loop. The initialisation of the algorithm (line 1) sets C_1 to two symbols, the other alphabets are initialised accordingly, and each chromosome of each niche H_i is initialised by uniformly sampling allele values based on its related alphabet C_i. Each lim_{mig} generations (line 7) NicheShift performs chromosome migrations (line 8). This operation — similar to the migration of chromosomes adopted by the Island Models approach [25] — aims to transfer, among the most fit chromosomes of niche H_i, those who have not migrated yet, unidirectionally, to the niche at its right H_{i+1}. Clearly, the unidirectional migration policy has a direct relationship to Monolithic's increase of its alphabet; moreover, NicheShift's strict migration to unique chromosomes aims to tackle the stagnation of the population diversity observed in Monolithic [9]. After the eventual migration operation, NicheShift performs the evolution of each niche (row 11), by means of elitism, rank selection, uniform crossover and standard mutation operations, similarly to the Monolithic algorithm.

The most distinct and important feature of NicheShift is the *shift and merge* policy. Each lim_{snm} generations (line 13), the algorithm retrieves the list of niches which score the highest fitness (line 14, and line 4 for initialisation purposes); if no changes occur since the last check (status quo, line 15), and this is repeated (line 16) for $maxStatusQuo$ times (line 17), then NicheShift terminates its execution (line 18). On the other hand, in case of new most fit niches, the *shift and merge* operation is performed (line 21).

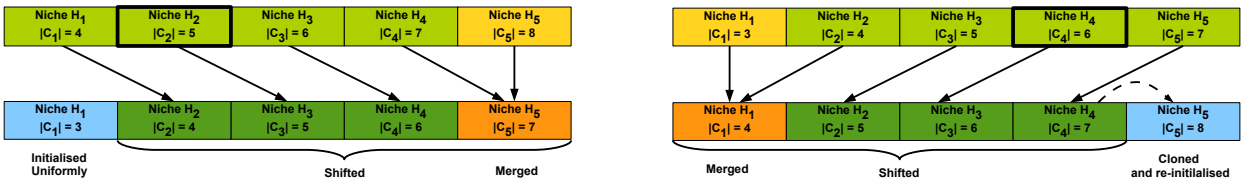
The purpose of the *shift and merge* operation is, similarly to the approach adopted by the Monolithic algorithm, to gradually move from candidate solutions with a low number of community structures towards solutions with a higher number of detected communities. The pseudocode for this operation is listed in Algorithm 2. Given the list of most fit niches, the algorithm starts by retrieving their related alphabets (line 1); then it determines the median alphabet of this set, C_{med} (line 2). If $|C_{med}|$ is bigger than the size of the current central alphabet $C_{h/2}$ (line 3), then we will assist to a *leftward shift and merge* (line 4), otherwise we will assist to a *rightward shift and merge* (line 6). The *shift* operation is the same for both directions and has the duty to re-centre the niches and relative alphabets with respect to C_med. In other words, C_{med} and its relative niche H_{med} will be *shifted* in order to become the new, most central niches. This shifting operation is applied to as many niches and relative alphabets as possible. The shifting will lead, eventually, to a set of consecutive niches (alphabets), either on the very right (for the leftward case) or very left (for the rightward case); these will be *merged*, together with the last (for the leftward case) or the first niche (for the rightward case). Finally, a set of *new* niches and relative alphabets, either at the left (for leftward operation) or at the right (for rightward operation) will be required to be initialised. In case of leftward *shiftAndMerge*, the chromosomes of the new niches will be initialised by sampling allele values uniformly within their new alphabet. In case of rightward *shiftAndMerge*, instead, each new niche H_i will first *clone* the niche on its left H_{i-1}, then its chromosomes will be re-initialised: each gene has now probability 50% to change into the niche's new alphabet symbol x^i. Figure 4 depict an example for leftward and rightward *shift and merge* operations for $h = 5$ niches.

Algorithm 1 NicheShift Main Loop

```
1:  initialisePopulation()
2:  g ← 1
3:  nStatusQuo ← 0
4:  oldBest_H ← getBestNiches()
5:  loop ← true
6:  while loop do
7:    if g mod lim_mig = 0 then
8:      performMigration()
9:    end if
10:   for all niches do
11:     performEvolution()
12:   end for
13:   if g mod lim_snm = 0 then
14:     newBest_H ← getBestNiches()
15:     if newBest_H = oldBest_H then
16:       nStatusQuo ← nStatusQuo + 1
17:       if nStatusQuo = maxStatusQuo then
18:         loop ← false
19:       end if
20:     else
21:       shiftAndMerge(newBest_H)
22:       nStatusQuo ← 0
23:       oldBest_H ← newBest_H
24:     end if
25:   end if
26:   g ← g + 1
27: end while
```

Algorithm 2 shiftAndMerge($best\_H$)

```
1: best_C ← getAlphabets(best_H)
2: C_med ← getMedianAlphabetIsland(best_C)
3: if |C_{h/2}| > |C_med| then
4:   leftwardShiftAndMerge(...)
5: else
6:   rightwardShiftAndMerge(...)
7: end if
```



(a) Leftward *shift and merge*

(b) Righttward *shift and merge*

Fig. 4. An example of the Leftward (Fig. 4(a)) and Rightward (Fig. 4(b)) *shift and merge* operation. In Fig. 4(a), niche H_2 scored the highest fitness; the *shift* operation transfers it into the central position, i.e. $H_3 \leftarrow H_2$; its adjacent niches are shifted accordingly: $H_2 \leftarrow H_1$, $H_4 \leftarrow H_3$. The remaining niches H_5 and H_4 are *merged* in the new the last niche, $H_5 \leftarrow H_4 + H_5$. Finally, the new niche H_1 is initialised by assigning, to each gene of each chromosome of its population, a symbol uniformly drawn within the new C_1 alphabet. In Fig. 4(b), niche H_4 scored the highest fitness; the *shift* operation transfers it into the central position, i.e. $H_3 \leftarrow H_4$; its adjacent niches are shifted accordingly: $H_2 \leftarrow H_3$ and $H_4 \leftarrow H_5$. The remaining niches H_1 and H_2 are *merged* in the new the last niche $H_1 \leftarrow H_1 + H_2$. Finally, the new niche H_5 is initialised: first, it clones the whole niche at its left, i.e. H_4; then, for each chromosome it re-initialises each of its gene, setting allele value x^5, i.e. the new symbol of the alphabet C_5, with probability 50%.

V. Experiments and Results

GaNet, Tasgin, Monolithic and NicheShift were tested across the sparse graph and complete network cases. For the latter, GaNet and Tasgin were applied to the approximated graphs obtained by means of the Reduced Approach. For each case $n = 128$ and three scenarios were considered, namely $m = \{4, 8, 16\}$ communities; moreover, for each scenario, $z = n/2k$ and $z - out \in [0, z/2]$. In total, this paper presents the results of $2 * (9 + 5 + 3) = 34$ experimental setups, each of which is executed 30 times for a better approximation of the performance of stochastic algorithms.

The performance measure we consider in this study is the normalised mismatch error $nme(K, \hat{K})$ [9], between the true community structures K, and the detected ones $\hat{K}$, calculated as follows:

$$nme(K, \hat{K}) = \frac{n - h(K, \hat{K})}{n} \tag{8}$$

where $h(K, \hat{K})$ is the maximum assignment score obtained by running Kuhn's Hungarian algorithm [9].

After a quick fine parameter tuning experimental phase, based on complete networks with parameters $n = 128$, $m = 4$, $z = 16$, $z - out = 0$, the following parameters were setup for NicheShift: population size is 200; $h = 5$ islands; migration occurs each $lim_{mig} = 30$ generations only for unique individuals selected among the 1% of fittest individuals; end condition check occurs each $lim_{snm} = 30$ generations with termination occurring after $maxStatusQuo = 4$ status quo; finally, the mutation probability is 0.8 and elite population is composed by the first 50% fittest chromosomes. On the other hand, GaNet, Tasgin, and Monolithic algorithms were executed, across each of the 42 experimental setup, with their standard parameters as those are retrieved from the corresponding studies [9], [13], [14]. Even though GaNet and Monolithic present, with respect to NicheShift, different parameter setups which ultimately lead to a lower number of fitness evaluations, extensive parameter tuning experiments suggested that the algorithm performances are not sensitive to parameter change. We decided not to include the results of these experiments due to space constraints.

A. Analysis of Sparse Graphs

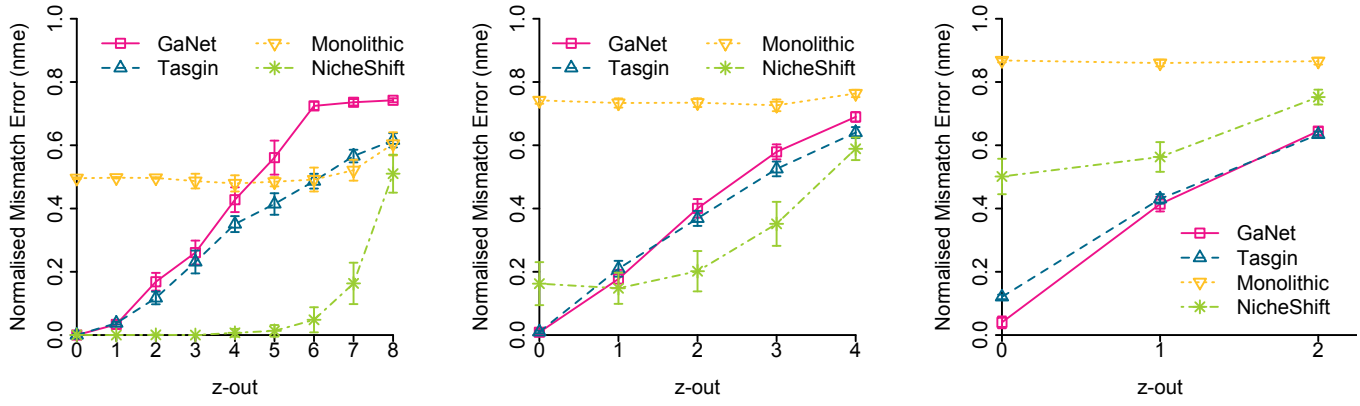
Figure 5 depicts the average performance and standard deviation values of 30 experimental runs across three different community structures ($m \in \{4, 8, 16\}$) for sparse graphs of $n = 128$ nodes, $z = \{16, 8, 4\}$ and $z - out \in [0, z/2]$. The first, remarkable finding, is that NicheShift appears to be the most robust algorithm scoring the lowest average errors excluding the $z - out = 0$ scenario in both $k = 4$ and $k = 8$ setups. What we thus conclude that NicheShift manages to centre its islands around the alphabet of exact size $|C_{h/2}| = k$. Figure 5(a) further supports this as NicheShift makes use of $k = 5$ niches; since the initial alphabet for H_1 is composed of two symbols, then the third, central niche will have alphabet size of $C_3 = |C_1| + 2 = 4$ symbols. This also means that its modularity measure (7) can be applied to discrete networks.

On the other hand, the Monolithic algorithm performs worst. This result is not unexpected, considering the algorithm's apparent difficulties in solving the CSD problem for smaller networks [9]. The poor performance is probably due to quick stagnation of the elites, as previously observed.

For the $m = 4$ scenario, GaNet and Tasgin manifest less robustness with respect to increasing $z - out$ values, which aligns well with the findings highlighted in earlier studies [13], [14]. Their performance is almost linearly dependent on the $z - out$ values for $m = 8$, whilst they outperform Monolithic and NicheShift for $m = 16$. This result is not surprising, considering that these graphs have $n * z/2 = 128 * 4/2 = 256$ out of $n(n-1)/2 = 8128$ possible edges: GaNet and Tasgin exploit their knowledge about the graph structure, which allows them to focus on more effective candidate solutions. In contrast, NicheShift and Monolithic do not rely on any knowledge about the edge set since they are originally devised to work on complete networks.

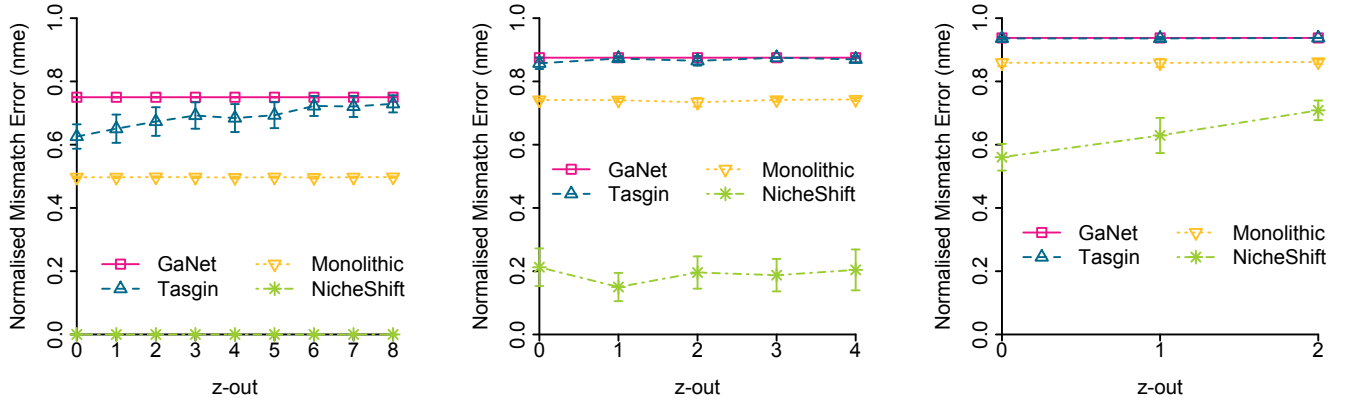
B. Analysis of Complete Weighted Digraphs

Similarly to Figure 5, Figure 6 depicts the average performance and standard deviation values of 30 experimental runs across three different community structures ($m \in \{4, 8, 16\}$) for complete networks of $n = 128$ nodes, $z = \{16, 8, 4\}$ and $z - out \in [0, z/2]$. As expected, GaNet and Tasgin score the worst performance across all 17 setups. This highlights the drastic impact the re-conversion process (implemented in the reduced approach) has on the performance of these algorithms.



(a) $m = 4$ Community Structures (b) $m = 8$ Community Structures (c) $m = 16$ Community Structures

Fig. 5. Sparse Graphs: Average performance and standard deviation values (30 runs) for GaNet, Tasgin, Monolithic and NicheShift, for sparse graphs of $n = 128$ nodes, $k \in \{4, 8, 16\}$) community structures, $z = n/2k$, and $z - out \in \{0, z/2\}$ edges.



(a) $m = 4$ Community Structures (b) $m = 8$ Community Structures (c) $m = 16$ Community Structures

Fig. 6. Complete Networks: Average performance and standard deviation values (30 runs) for GaNet, Tasgin, Monolithic and NicheShift, for complete networks of $n = 128$ nodes, $k \in \{4, 8, 16\}$) community structures, $z = n/2k$, and $z - out \in \{0, z/2\}$ edges.

Strikingly, the NicheShift approach yields the lowest nme errors. NicheShift appears to leverage on the information relative to the edges, connecting nodes within the same community, for which the original undirected network would have not registered an edge — i.e. most of the edges built with $N(\mu_{in,0}, \sigma)$, see Eq. (3) —. These edges manage to empower the connection among edges of the same community, and to weaken the noise represented by the *out-edges*.

Finally, we observe that the four algorithms change their nme trend, from being directly influenced by increasing values of $z-out$, to their nearly complete independence. Although at first hint the generation of normally distributed weights might have an impact on such behaviour — e.g. by making the total number of $n(n-1) = 16256$ weights uniformly distributed — an analysis of the weight distribution did not show any significant changes across the different $z-out$ configurations.

VI. Discussion and Future Work

When faced with a CSD problem on complete networks, at least two possible approaches could be taken: trying to solve them directly or reduce them to a simpler problem. This paper has attempted both approached.

While the reduced approach allows algorithms such as GaNet and Tasgin to work on a wider range of problems, it has a dramatic impact on the efficacy of these algorithms. It would therefore seem advisable to choose the direct approach wherever possible. NicheShift shows that it is possible to solve CSD problems without reducing their complexity. It also shows very good results for sparse graphs, which suggest that NicheShift would also work well for the intermediate cases such as incomplete networks.

The drastic performance increase of NicheShift over the Monolithic algorithm prompts us to ask what the core features that allows this improvement are, and whether they can be further improved. The key invention of NicheShift is the *shift and merge* operation. Here, the correct number of niches and the policy for when to shift and merge are important. Remarkably, when $m = 4$ in both sparse and complete directed digraph scenarios, NicheShift achieved $nme = 0$. This seems to be because the central niche already has the right number of

groups. A faster adaptation of niches in order to replicate such ideal conditions for other m might lead to even better results under those conditions. Thus, future work will investigate other policies for when to shift and merge, e.g. using techniques from reinforcement learning. We also plan to analyse and find ways of reducing the resources required by NicheShift to compute its solutions, i.e. a memory space $O(hmn^2)$ quadratic with respect to the problem size, for $m, h \ll n$, and its related fitness evaluations for an undefined number of generations.

Future work will also consider the application of NicheShift to dynamic networks [19], [26], being them related to the original motivation for the current line of research [9]. We will even further investigate the performance of other evolutionary approaches, e.g. those listed in the related work section, and compare NicheShift with other non-evolution algorithms, e.g. Ant Colony Optimisation [12] and spectral partitioning [8]. Another interesting line of research would be the extension of NicheShift to multi-objective optimisation [15], possibly based on multiplex networks [27], or even for the detection of overlapping communities [6], [16], [28].

VII. Conclusions

We have investigated the possibility to solve the community structure detection (CSD) problem, undirected and unweighed (sparse) graphs, and in complete networks, by means of evolutionary computation. We have considered four Genetic Algorithms (GAs), two of them initially designed to work for undirected and unweighted (sparse) graphs (GaNet and Tasgin's algorithm [13], [14]), and two specifically designed for complete networks: one previously designed for another line of research of ours [9], and another one, based on shifting niches (NicheShift), which is first presented in this paper.

Experiments conducted on both sparse graphs and complete network highlight the fact that NicheShift is extremely robust across both sparse graph and complete network cases, under three different scenarios (four, eight and sixteen communities), and noise represented by the edges connecting nodes belonging to different communities. On the other hand, the need for the conversion from complete networks to sparse graphs, in order to apply GaNet and Tasgin's algorithm (reduced approach), showed the potential drawbacks arising from the reduction of complexity of the problem. This suggest thats if the reduced approach is to be taken in further research, extensive investigations should be made for the network-to-graph conversion phase.

Acknowledgments

This work has been supported, in part, by the FP7 ICT project SIREN (project no: 258453).

References

[1] M. Girvan and M. E. Newman, "Community Structure in Social and Biological Networks," *Proc. of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[2] C. Pizzuti, "Community Detection in Social Networks with Genetic Algorithms," in *Proc. of GECCO*. ACM, 2008, pp. 1137–1138.

[3] N. Ducheneaut, N. Yee, E. Nickell, and R. J. Moore, ""Alone Together?": Exploring the Social Dynamics of Massively Multiplayer Online Games," in *Proc. of CHI*, 2006, pp. 407–416.

[4] M. E. Newman and M. Girvan, "Finding and Evaluating Community Structure in Networks," *Physical review E*, vol. 69, no. 2, 2004.

[5] E. A. Leicht and M. E. J. Newman, "Community Structure in Directed Networks," *Physical Review Letters*, vol. 100, no. 11, 2008.

[6] A. Lancichinetti and S. Fortunato, "Benchmarks for Testing Community Detection Algorithms on Directed and Weighted Graphs with Overlapping Communities," *Physical Review E*, vol. 80, no. 1, 2009.

[7] M. E. Newman, "Analysis of Weighted Networks," *Physical Review E*, vol. 70, no. 5, 2004.

[8] C. Farmer and A. Fotheringham, "Network-based Functional Regions," *Environment and Planning A*, vol. 43, no. 11, pp. 2723–2741, 2011.

[9] C. Grappiolo, J. Togelius, and G. N. Yannakakis, "Artificial Evolution for the Detection of Group Identities in Complex Artificial Societies," in *Proc. of IEEE SSCI ALife*, 2013, pp. 126–133.

[10] M. E. Newman, "Finding Community Structure in Networks Using the Eigenvectors of Matrices," *Physical review E*, vol. 74, no. 3, 2006.

[11] ——, "Fast Algorithm for Detecting Community Structure in Networks," *Physical Review E*, vol. 69, no. 6, 2004.

[12] S. R. Mandala, S. R. Kumara, C. R. Rao, and R. Albert, "Clustering Social Networks Using Ant Colony Optimization," *Operational Research*, pp. 1–19, 2011.

[13] C. Pizzuti, "GA-Net: A genetic Algorithm for Community Detection in Social Networks," *Parallel Problem Solving from Nature*, vol. 5199, pp. 1081–1090, 2008.

[14] M. Tasgin, A. Herdagdelen, and H. Bingol, "Community Detection in Complex Networks Using Genetic Algorithms," *arXiv:0711.0491*, 2007.

[15] C. Pizzuti, "A Multiobjective Genetic Algorithm to Find Communities in Complex Networks," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 3, pp. 418–430, 2012.

[16] J. Liu, W. Zhong, H. A. Abbass, and D. G. Green, "Separated and Overlapping Community Detection in Complex Networks Using Multiobjective Evolutionary Algorithms," in *Proc. of IEEE CEC*, 2010, pp. 1–7.

[17] C. Pizzuti, "Overlapped Community Detection in Complex Networks," in *Proc. of GECCO*, 2009, pp. 859–866.

[18] D. Jin, D. He, D. Liu, and C. Baquero, "Genetic Algorithm with Local Search for Community Mining in Complex Networks," in *Proc. of IEEE ICTAI*, 2010, pp. 105–112.

[19] F. Folino and C. Pizzuti, "Multiobjective Evolutionary Community Detection for Dynamic Networks," in *Proc. of GECCO*, 2010, pp. 535–536.

[20] M. Lipczak and E. Milios, "Agglomerative Genetic Algorithm for Clustering in Social Networks," in *Proc. of GECCO*, 2009, pp. 1243–1250.

[21] C. Shi, Z. Yan, Y. Wang, Y. Cai, and W. BIN, "A Genetic Algorithm for Detecting Communities in Large-scale Complex Networks," *Advances in Complex Systems*, vol. 13, no. 01, pp. 3–17, 2010.

[22] A. Gog, D. Dumitrescu, and B. Hirsbrunner, "Community Detection in Complex Networks Using Collaborative Evolutionary Algorithms," *Advances in Artificial Life*, pp. 886–894, 2007.

[23] X. Liu, D. Li, S. Wang, and Z. Tao, "Effective Algorithm for Detecting Community Structure in Complex Networks Based on GA and Clustering," *Computational Science–ICCS*, pp. 657–664, 2007.

[24] A. Lancichinetti and S. Fortunato, "Community Detection Algorithms: a Comparative Analysis," *Physical Review E*, vol. 80, no. 5, 2009.

[25] W. N. Martin, J. Lienig, and J. P. Cohoon, "Island (migration) models: Evolutionary algorithms based on punctuated equilibria," *Handbook of Evolutionary Computation*, vol. 6, no. 3, 1997.

[26] C. Grappiolo, J. Togelius, and G. N. Yannakakis, "Interaction-based Group Identity Detection via Reinforcement Learning and Artificial Evolution," in *Proc. of GECCO (to appear)*, 2013.

[27] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela, "Community Structure in Time-dependent, Multiscale, and Multiplex Networks," *Science*, vol. 328, no. 5980, pp. 876–878, 2010.

[28] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society," *Nature*, vol. 435, no. 7043, pp. 814–818, 2005.