An evolved Memetic algorithm applied to Clustering

A genetical engineering endeavour

José A. Magaña Mesa jose_a_magana@hotmail.com

***Abstract*— The current report is an endeavor to apply to a Memetic Algorithm a further evolution following the parallelism of Genetics Engineering in Biology, where the best genes are selected for combination. An algorithm has been developed, inspired in two existing ones and the combination method has been modified to apply domain knowledge information. Also the LocalSearch has been modified to become stochastic and incorporate a mutation probability as regular GA. During algorithm implementation some evaluation of the performance of the implemented options has been required to drive evolution of the implementation. Two real-world datasets have been used to measure the performance of the algorithm and benchmark against the reference successful implementations. The results, while far from optimal, are encouraging and manage to obtain optimal results for the simplest case and a quick approximate solution in both cases. The algorithm requires though a large number of iterations even for simple problems but as a virtue can take a non-fitted initial population as starting point.**

***Keywords*— Clustering, Community Detection, Memetic Algorithms, Local Search, Genetics Engineering**

I. Problem statement and goals

Clustering of complex networks, also known as Community Detection, is a technique that has been approached from plenty of different perspectives and different methods have been developed [1]: hierarchical, graph, partitional, spectral,… Many of these methods are based on mathematical models that explode physical and mathematical properties and metrics of the network and hence, the results are heavily supported by theoretical grounds. The methods can be considered "exact" in the sense that they produce a result that is the deterministic and mathematical outcome of applying a certain model. These techniques are though quite often computationally expensive.

On the other hand, Genetic Algorithms (GA) can generate, for many problems and specifically for Clustering problems, solutions that are reasonably good and in many cases match or improve the Best Known Results (BKR) for some well-known datasets. GA is though a wide category with different variations being Memetic Algorithms (MA) [3] one of them. MA is a "mutation" of GA in which the mutation procedure that is applied to the result of combination of existing solutions is replaced by a local search procedure that applies domain knowledge in a local search procedure to modify the individual to improve its fitness value. If GA can be compared to natural reproduction, MA through this local search incorporates to the individuals the capacity to learn, each of this improvements obtained through the local search is a 'meme' [2], something that makes the individual better during his lifetime as it learns (imitates in the original: " *so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation*"). Local search only incorporates these memes when they contribute to improve the individual; it is a selective evolution process.

This use of selective evolution and in general the local search procedure to select the changes that the individual takes is not usually applied in the recombination operator and the combination is done following much more stochastic techniques that while they contribute to keep the diversity of the individuals in the MA population do not foster the quick improvement of the population and hence, the discovery of good solutions.

In the same way that Genetics Engineering[4] (GE) (referred to the medical and biology field) is able to introduce changes in the genes of an embryo so that parents can choose its gender or guarantee that will not suffer specific genetic diseases, in the computing homologous, for problems, as Clustering, where there is a large knowledge about the problem, if the genes are encoded in an appropriate way, a recombination algorithm that explodes domain knowledge can be used to extract from the parents the best genes and combine them to produce an offspring whose fitness is better than its parents with a much higher probability than stochastic recombination would provide. Same as GE is considered by some a Pandora box of humans playing God without being aware of the side effects of gene manipulation, there are also risks on the approach of GE in computing field, as some of the design criteria well established in GA or MA may not hold for GE.

Keeping this idea in mind and taking as starting point two papers recently published regarding the use of MA in Clustering, my objective has been to design an algorithm that maintaining the basic scheme of MA algorithms incorporates in the recombination operator domain knowledge criteria regarding networks structure. The implementation has been done from scratch using Python and the NetworkX library that has been used to store the graph structure and retrieve neighbor nodes and edges as required by the algorithm, [11]. Everything else has been coded for the project, apart from some other modules for common calculations (numpy, pyparsing).

The design of the algorithm has followed an iterative approach on which changes have been introduced to compensate or fix the drawbacks detected during the design phase. For the design phase, a simple well-known network, Zachary Karate Club [5], has been used as testing field. From [10]: “The karate club network was constructed by Zachary, who observed 34members of a karateclub over a period of 2 years [34]. During the course of the study, a disagreement developed between the administrator of the club and the club’s instructor, which ultimately resulted in the instructor’s leaving and starting a new club, taking about half of the original club’s members with him.” The split matches a K=2 clustering for the dataset although a better modularity can be obtained for K=5, that maintains the partition for K=2 (members in different clusters remain in different clusters). It is a simple network compared to other datasets in use but large and complex enough for not being a trivial problem: 38 nodes and 78 edges, with an optimal modularity of 0.4198 (range [-0.5,1])

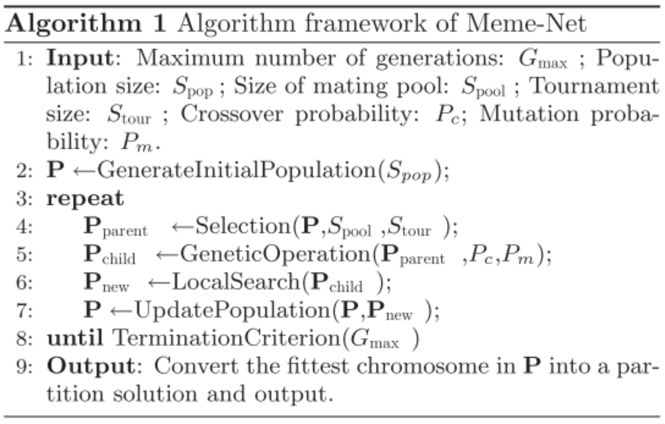
Once the algorithm has been considered to overcome reasonably well the common issues to consider in this kind of problems the resulting algorithm has been used with the same network and a second network to proof its performance. The second network is the Dolphins network [6]. From [10]: “The network of 62 bottlenose dolphins, living in Doubtful Sound, New Zealand, was compiled by Lusseau from the observation of dolphin behavior for 7 years. A tie between two dolphins was established by their statistically significant frequent association.” Again the network splits in two large groups but better modularity can be obtained for a five group partition. This is a larger network than the previous one but still small to keep computational costs limited and guarantee that any experiment can be executed in the available time for this project: 62 nodes and 159 edges, with a maximum modularity of 0.5285.

Algorithm profiling of the population and its main parameters in terms of its evolution has been required in order to analyze, diagnose and provide insights for changes in the algorithm.

II. Previous work

A. A little bit of background

To hold common ground and future reference, let’s remind what the traditional structure and parameters of a MA is:



Algorithm 1 Algorithm framework of Meme-Net

```
1: Input: Maximum number of generations: $G_{\max}$ ; Population size: $S_{\text{pop}}$ ; Size of mating pool: $S_{\text{pool}}$ ; Tournament size: $S_{\text{tour}}$ ; Crossover probability: $P_c$; Mutation probability: $P_m$.
2: $\mathbf{P} \leftarrow$GenerateInitialPopulation($S_{pop}$);
3: repeat
4:     $\mathbf{P}_{\text{parent}} \leftarrow$Selection($\mathbf{P}$,$S_{\text{pool}}$ ,$S_{\text{tour}}$ );
5:     $\mathbf{P}_{\text{child}} \leftarrow$GeneticOperation($\mathbf{P}_{\text{parent}}$ ,$P_c$,$P_m$);
6:     $\mathbf{P}_{\text{new}} \leftarrow$LocalSearch($\mathbf{P}_{\text{child}}$ );
7:     $\mathbf{P} \leftarrow$UpdatePopulation($\mathbf{P}$,$\mathbf{P}_{\text{new}}$ );
8: until TerminationCriterion($G_{\max}$ )
9: Output: Convert the fittest chromosome in P into a partition solution and output.
```

Figure 1 MA algorithm

A commonly used recombination algorithm is Partition Crossover (PX) [12]. The algorithm generates from two different complete partitions of the same network a new complete partition for the network. For doing so the different clusters (genes) in the two networks are combined in a single list and the list traversed. In this configuration each node will be in the list in two clusters, one proceeding from each of the original networks. As the list of clusters is traversed, the cluster is added to the new partition being created and the nodes part of this cluster are removed from the clusters still on the list. In a greedy version of the algorithm, GPX, the clusters are ordered so that at every iteration the cluster with higher cardinality is chosen.

An example, extracted from [9] is shown on the application of the PX operator for two partitions and a certain partition order (priority):

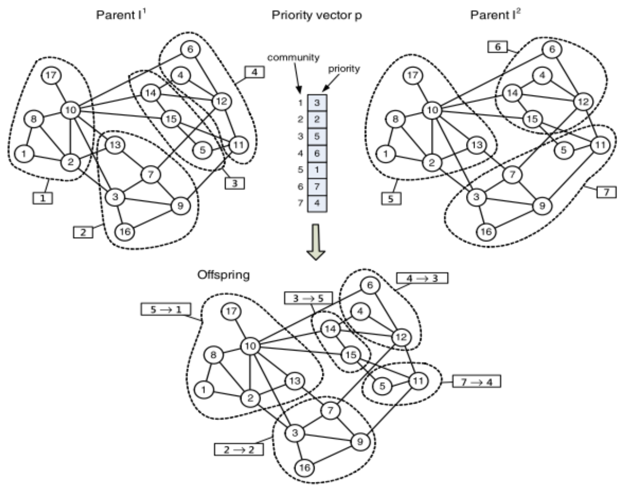


Figure 2 The PPX algorithm graphically

Our driver is to use domain knowledge information in our problem. We will require then a metric to measure the clustering quality that we will use as fitness function. The metric must be calculated for the whole partition but also for each of the clusters in the partition. The problem here is not to find one but to choose one, as the literature [1] refers or provides many different Modularity (Q) functions that measure the cohesion of a clustering. To limit our search, benchmark and do not deviate prematurely much from a known good solution for the purpose of the project the modularity function from [9] has been chosen:

$$Q(I) = \sum_{i=1}^{K} \left[\frac{W(C_i, C_i)}{W(V, V)} - \left(\frac{d_i}{W(V, V)} \right)^2 \right]$$

(Equation 1: Modularity)

A second option could have been [10], but it is more parametrical and attacks multi-resolution clustering while its λ-free version does not reach the BKR for the problems being used:

$$D_\lambda = \sum_{i=1}^{m} \frac{2\lambda L(V_i, V_i) - 2(1-\lambda) L(V_i, \overline{V_i})}{|V_i|}.$$

In this kind of problems, the possible knowledge of the best modularity Q is not incorporated to the problem and must be learnt by the method; neither is it the number of clusters that lead to the optimal Q. For the latter, there are families of methods that require it but it has not been the case for our implementation.

An important factor in any GA algorithm is population diversity. The UpdatePopulation() method must consider this in addition to individual fitness. For this, we require a distance function that can be used to compare the similarity of two individuals beyond its fitness value.

A common distance function is the Edge Rand Index (ERI) that measures how many coincidences are there for edges of the graph whose ends are in the same cluster in both partitions or none of the partitions. The definition can be seen in [9] as:

$$d(X, Y) = \frac{\sum_{e \in E} d_e(X, Y)}{m}$$

$$d_e(X, Y) = \begin{cases} 0 \text{ if } \exists X_i \in X, \exists Y_j \in Y \text{ s.t. } e \in X_i \text{ and } e \in Y_j \text{ OR} \\ \quad \text{if } \forall X_i \in X, \neg(e \in X_i) \text{ and } \forall Y_i \in Y, \neg(e \in Y_j) \\ 1 \text{ otherwise.} \end{cases}$$

(Equation 2: Edge Rand Index)

ERI being a ratio is bounded [0,1] while Q takes values in the range [-0.5,1].

Normalized Mutual Information (NMI) defined in [10] is another indicator of partition similarity. It is a measure of the entropy of the confusion matrix of the two partitions: the number and distribution of the disagreements among partitions.

$$I(A, B) = \frac{-2 \sum_{i=1}^{c_A} \sum_{j=1}^{c_B} C_{ij} \log(C_{ij} N / C_{i.} C_{.j})}{\sum_{i=1}^{c_A} C_{i.} \log(C_{i.}/N) + \sum_{j=1}^{c_B} C_{.j} \log(C_{.j}/N)},$$

B. *Loans and inspirations*

The first, and main algorithm used as reference, is MA-COM [9]. This MA algorithm uses in its Genetic Operation (Recombination) a Priority-based Partition Crossover (PPX) operator in which the genes that correspond to clusters in the solution are sorted randomly.

A variation of PPX has been implemented in the current algorithm. Clusters priority is given according to its individual modularity calculated using Equation 1.

The initialization of the population is, in my humble opinion, the weak point of the algorithm. The BGLL [13] algorithm, that is able to generate optimal or close to optimal solutions for many of the datasets reported, is used what really makes MA-COM not a Clustering algorithm but more an “optimizer” for existing good clustering approximations.

The second algorithm is MEME-NET [10]. MEME-NET instead uses a more reasonable initialization procedure where initially each node is a cluster and randomly a certain number of nodes are chosen and all its neighbors in the graph assigned to the cluster the node belongs to. This has been used as initialization in the implemented algorithm. This method requires deciding how many times a node is going to be chosen. The paper uses a factor of 0.2 times the number of nodes in the graph that has also been fixed for the implementation done.

The implementation for the UpdatePopulation() while inspired in the MEME-NET has been modified significantly. MEME-NET uses a parameter δ to define a minimum distance between individuals that is set arbitrarily by the authors and that has not been considered in my implementation.

For the two networks used in the experiments, it is required not only to obtain the network but also to know the optimal partition. This permits to benchmark our results. The ground truth for the Zachary Karate Club has been obtained from [10] while the ground truth for Dolphins has been obtained from [8] and a correction/clarification provided by Dr. Gach (coauthor of [10])

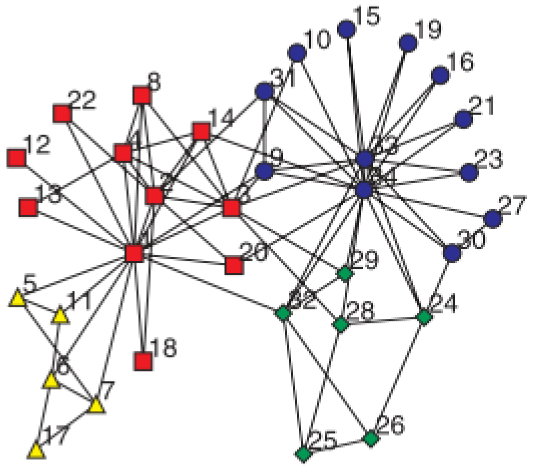


Figure 3 Zachary Karate Club Ground Truth

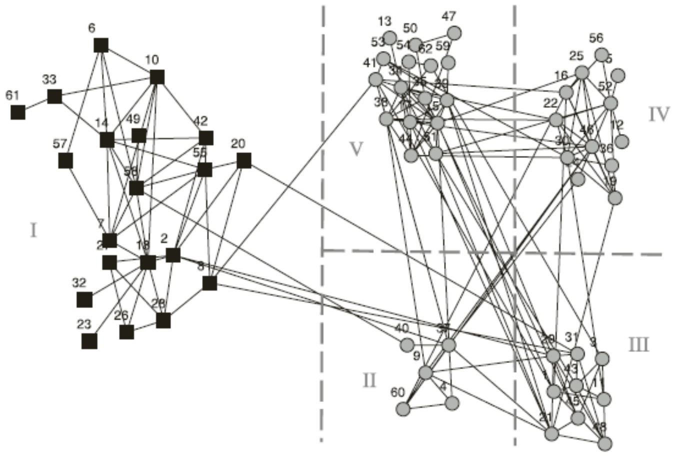


Figure 4 Dolphins

III. THE CI METHODS: ALGORITHM DESIGN

A. *PX algorithm analysis*

The PX algorithm is a key part of the implementation. Analyzing how it behaves will bring us to some conclusions that are interesting for the design of our algorithm:

1. The order in which the clusters are considered changes the result.
2. Nodes that belong to the same cluster in both input partitions will also be in the same partition in the offspring individual. The algorithm never separates what both parents agree to be together.
3. Nodes in different clusters in the two parents will be added in different steps and will remain in different clusters. The algorithm never puts together what both parents keep separated.
4. The number of clusters in the offspring will not be less than the minimum number of clusters in the parents.
5. The resulting number of clusters can be greater than the maximum number of clusters in the parents
6. The combination of one element with itself generates the same element: A * A = A

B. *Chromosome coding*

Individuals in the population must be coded in a way that enables the operators for combination and mutation to act efficiently and guarantee that the result of the operations is also a solution for the problem. To achieve this, the coding chosen uses a tuple with the following components:

- Modularity of the solution, in the range 0 to 1.5 (after applying an offset as explained in C Selection)
- Dictionary mapping nodes to clusters, 1 to 1. For the Dolphins ground truth:

  {0: 4, 1: 1, 2: 4, 3: 5, 4: 3, 5: 1, 6: 1, 7: 1, 8: 5, 9: 1, 10: 4, 11: 3, 12: 2, 13: 1, 14: 2, 15: 3, 16: 2, 17: 1, 18: 3, 19: 1, 20: 4, 21: 3, 22: 1, 23: 3, 24: 3, 25: 1, 26: 1, 27: 1, 28: 4, 29: 3, 30: 4, 31: 1, 32: 1, 33: 2, 34: 2, 35: 3, 36: 5, 37: 2, 38: 2, 39: 5, 40: 2, 41: 1, 42: 4, 43: 2, 44: 4, 45: 3, 46: 2, 47: 4, 48: 1, 49: 2, 50: 2, 51: 3, 52: 2, 53: 2, 54: 1, 55: 3, 56: 1, 57: 1, 58: 2, 59: 5, 60: 1, 61: 2}

  where the first integer (key dictionary) in each pair refers to the node identifier and the second to the cluster number.
- Dictionary mapping clusters to nodes, 1 to N. For the Dolphins ground truth:

  {1: [1, 5, 6, 7, 9, 13, 17, 19, 22, 25, 26, 27, 31, 32, 41, 48, 54, 56, 57, 60],

  2: [12, 14, 16, 33, 34, 37, 38, 40, 43, 46, 49, 50, 52, 53, 58, 61],

  3: [4, 11, 15, 18, 21, 23, 24, 29, 35, 45, 51, 55],

  4: [0, 2, 10, 20, 28, 30, 42, 44, 47],

  5: [3, 8, 36, 39, 59]}

  Where the first integer is the cluster number, not necessarily consecutive, for construction reasons, and the list corresponds to the nodes in that cluster.
- List of the modularity of each of the clusters in the partition in the same order than the clusters to nodes mapping.

C. *Selection*

None of the two referenced algorithms describe the implementation used for the Selection method by them. The implementation used selects two members of the population based on a probability that is proportional to the modularity of the individual. For this being possible an offset (+0.5) has been applied to the modularity to make it non-negative. All the modularities in the population have been added (total_q) and a random number generated between 0 and total_q to choose two parents.

The reason for using this procedure instead of a ranking based, for example, is that the modularity range (0 to 1.5 after transformation) and the values observed for the initial population guarantee that even very poor partitions (typically with modularity 0 in the original scale and 0.5 in our modified scale) will have a high probability of being selected, 1/3 of the optimal modularity and ½ of the optimal value for the examples used. The criterion is not strictly elitist although it does not offer many opportunities to close to -0.5 modularity partitions.

The two parents cannot be the same element so the selection must be done without replacement. Using the same individual twice would generate the same individual again (as explained in PX analysis) and would cause then a “lost generation” effect as no better solution would be generated an in addition if the offspring is added to the population it would add to reduce the Take Over time of the algorithm.

D. *Combination*

As explained in the previous section, the Combination algorithm used is a variation of the PPX. The clusters in the parents are sorted according to its modularity. The list is then traversed and the PX algorithm is applied. The calculation of the modularity is static; the modularity is not recalculated for the clusters when nodes are removed because the nodes are part of a cluster already added to the output partition. This is a design decision motivated by the intuition that it is better to incorporate a part of a good cluster existing in the parents than a not so good whole cluster whose incorporation would continue fragment the former one.

From the operation, it is obtained a partition that is made up of the common denominator of the parents and a list of small clusters that includes the remains of this common denominator extraction that the PX performs. In particular, a list of clusters with one node exists. In the initial population this list is long because the initialization method applies the merge as many as 20% of the number of nodes. As the population evolves, the number of single nodes gets reduced. Also single nodes appear as a result of combining different parents.

The existence of these single nodes will be exploded in the LocalSearch method of the algorithm.

E. *LocalSearch*

The LocalSearch algorithm implemented is indeed a LocalSearch algorithm in the traditional MA approach but with some stochasticity to provide a non-greedy approach in the Search and help the algorithm escape local minima. Clustering techniques usually have a tendency to create a large number of small communities that this algorithm avoids in the generated individuals.

The algorithm consists on looking for all the clusters with only one node on them in the offspring. These single nodes are then added to the cluster, with size larger than one, with whom the node has more edges. If the change of community improves the modularity, the change is applied, if the change does not improve the node remains single. Against what intuition could indicate, the change does not always cause a modularity improvement.

To introduce some stochasticity in the method, two considerations are made in the implementation:

1. The communities are shuffled so that the trial and error is not always in the same order, as a free parametric mechanism to break ties among communities with whom a node may have the same number of edges.
2. Single nodes only go through this process with a certain probability, 1-Pm (mutation), meaning that with Pm probability the node is NOT tried to be incorporated to a community and remains single.

The algorithm while keeping the characteristics of LocalSearch introduces stochasticity with the idea of avoiding greediness that could take the individuals to local minima.

A variation of the algorithm has been implemented in which the single node is always added to the existing cluster with whom more edges exist with the idea that when some other nodes change from a community to other, the change now applied will contribute to the modularity of the individual. This implementation is a tweak that although strongly supported by theoretical complex networks grounds cannot be fitted in the existing model for a GA/MA algorithm as the algorithm is defined. The role of Pm can be justified as a hybridization of LocalSearch and mutation but changing the individual heuristically (hoping for a latter improvement) breaks the paradigm.

Nevertheless, a quick exploration test of the idea has been done to validate, with negative results, if the change could improve the algorithm performance. The experiment and results are detailed in section IV.A LocalSearch.

The variability for the Pm parameter, including not considering the parameter (Pm=0), has been analyzed as part of the tune of the best parameters for the algorithm.

F. *UpdatePopulation*

COMB-MA updates its population following a double criterion. The offspring is compared with all the members of the population and the closer and further ERI-wise individuals selected. If the distance to the closer is smaller than a δ_{min} defined for the algorithm, fixed to 0.01 (1% of edges), the offspring replaces the closer member if is fitter than him. If the two conditions are not met (δ and Q) the offspring will replace the further member if fitter.

Initially, this same method was implemented but, quickly, it was observed that it was not working as it was causing premature Take Over of the population by non-optimal individuals. An experiment was designed to confirm and diagnose the symptoms and help in finding a better strategy as well as confirming that the solution strategy was not suffering the same or other problems.

The experiment results are detailed in IV.B UpdatePopulation design. As a conclusion, it was decided to use a single criterion and the offspring to replace always the closer individual whenever it was fitter, without considering any δ. The decisions are in addition supported by the analysis done of the PX algorithm in III.A that becomes useful for designing the UpdatePopulation method.

The first useful ‘meme’ from the analysis is that if the PX algorithm uses two identical individuals as inputs, the output will be identical to them too. So, the algorithm will not allow an individual to be added to the population if the individual already exists in the population. With this we prevent the risk of premature Take Over and help to keep the population diverse. As a consequence, the population size can remain small.

The algorithm requires single nodes in both parents, or originated from disjoints in the two parents, so that the offspring tuning can use them to continue improving the individual. It is important then that the evolution of the population is slow and that individuals in the initial population have a chance to survive for a long time contributing their randomly generated small clusters to the diversity and maintaining a wide range of individuals that when inter-combined maintain exploration and when combined with fit individuals enable exploitation.

The best found individual is always added to the population. This decision has been made even being aware that a higher Q does not guarantee a lower ERI respect to the optimal solution.

G. TerminationCriterion

MEME-NET uses a termination criteria of a fixed number of generations, being the number G_{max}=50, a quite low number for this type of algorithms, in my opinion. COMB-MA instead uses a stability criterion: 500 generations without a modularity improvement greater than 10^{-4}.

For the algorithm, and given the approach used for selecting other parameters, a static number of iterations have been chosen.

The COMB-MA criteria was considered to cause very long runs but some executions using the same criteria have shown that in the algorithm it causes the algorithm to finish after less than 1000 generations and doing so without reaching the optimal solution.

H. Parameters

The designed algorithm has then the following parameters that could affect its performance:

- Population Size
- Number of Generations
- Mutation Probability

The topic of parameter tuning for Genetic Algorithms is a broad and complex topic that could be a research topic on itself. In order to limit the effort a ‘shortcut’ has been taken. From [14], a set of different configurations have been taken and adapted to the needs of the current algorithm, as they specify other parameters not needed in our case.

The settings chosen are shown in TABLE I. . All configurations are different from each other in at least two parameters. It has been surprising (for a novice) the very low value of mutation probability. ‘Default’ configuration is the configuration used during development of the algorithms. It was maintained as the results were not worse than for some of the other configurations and was very different.

TABLE I. GA PARAMETERS

Configuration Name	**Parameters**		
	Population Size	***Number of Generations***	***Mutation Probability***
Default	10	200	0.2
Dejong	50	1000	0.001
Grefenstette	30	1000	0.01
MicroGA 1	5	100	0.04
MicroGA 2	5	100	0.02

These parameters have been used in several execution rounds of 20 repetitions each in order to find the configuration that produces better results. After the best configuration has been found, some additional executions have been found modifying one parameter at a time in order to try to further optimize the results.

Finally, some of the best configurations for the Zachary Karate Club network (used so far in all the experiments) have been used in the Dolphins network to measure performance and validate the reusability of the parameters found.

IV. Results and Discussion

A. *LocalSearch design*

To validate the introduction of a change in the algorithm that permits a single node to be added to the cluster with more common edges without requiring the change to increase the modularity, both algorithms have been compared for the Parameters described in previous section.

As observed in the diagrams, the Mean Modularity is better (right axis) in the non-heuristic version of the algorithm (MEAN) than for the heuristic (MEAN_R) for all the cases. The STDEV of the mean modularity is lower also in all cases. The ERI respect to the ground truth is better for the non-heuristic (ERI) than heuristic (ERI_R) in the three cases when modularity gets a better value while for the microga cases the situation is the opposite. The STDEV follows the same pattern than the corresponding metric providing better (lower) value for the three cases with better ERI for the non-heuristic variation (STDEV_ERI).

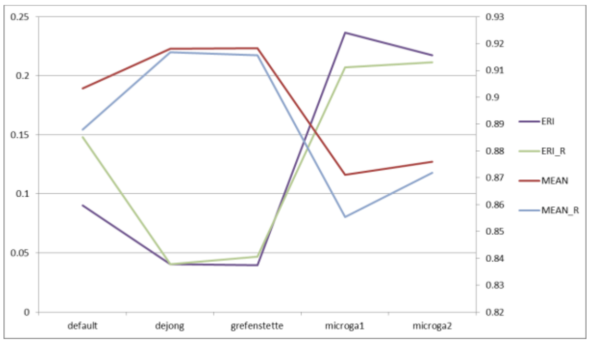


Figure 5 Metrics for non-heuristic variation

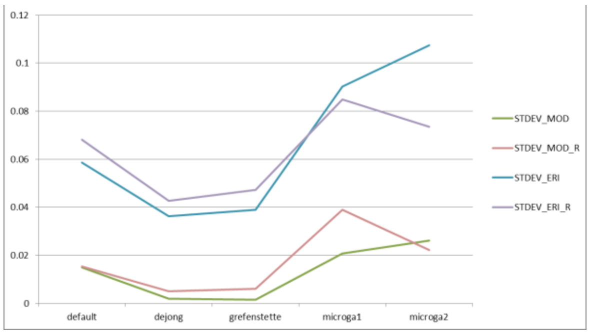


Figure 6 Metrics for heuristic variation

B. *UpdatePopulation design*

As mentioned earlier, the implementation of the COMB-MA method for the algorithm was causing premature Take Over. In order to diagnose the still unknown problem an experiment was designed in order to analyze the situation. For a set of parameters (Dejong), the algorithm was executed and the modularity ad ERI minimum and maximum evolution was measured along the whole execution. Also the number of times that the offspring was replacing the closer (crepl) and further (wrepl) members was captured.

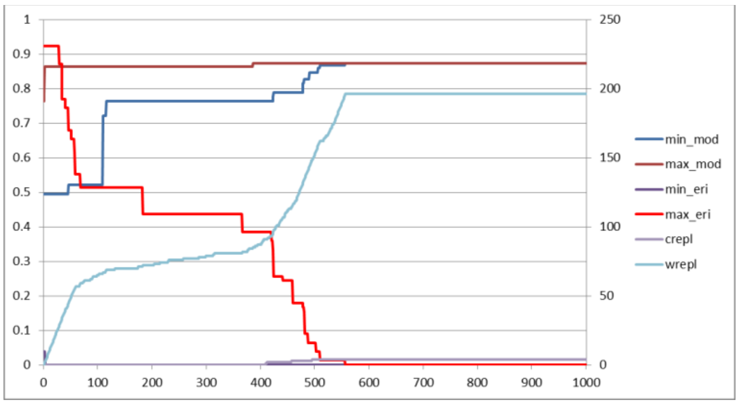


Figure 7 COMB-MA updatePopulation

From the diagram, it can be observed that few replacements (right axis scale) of the closer member happen along the whole execution while most of the replacements were for the further member. This was causing a clear decay of the maximum ERI inter-member and also a quick increase of the minimum modularity that was causing the population to become homogeneous (max ERI=0 and min_mod=max_mod) after less than 600 generations (population size 50).

To overcome the difficulty a first modification was introduced to only update the population if the offspring was not already a member. The same metrics were calculated, obtaining the following diagram:

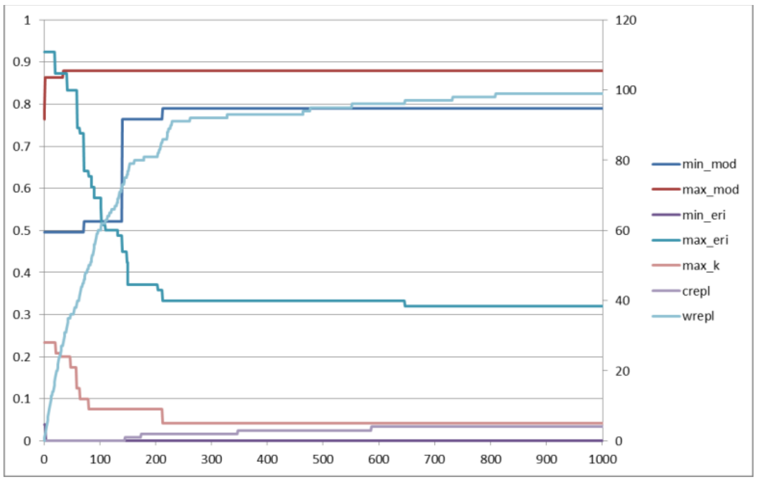


Figure 8 COMB-MA updatePopulation without duplicates

In this case, after a transitory phase in which the modularity and ERI ranges narrow, around 200 generations, the situation becomes steady maintaining a constant range for both values. During this transitory, the number of further replacements is high. Only after this transitory has been reached replacements of the closer members start to happen. The number of further replacements is much lower than for the previous situation.

From this, it can be inferred, that the population is more diverse but that the algorithm fails to explode the diversity on it, or the diversity is not enough despite the ranges of modularity and ERI latent on it.

A third implementation was introduced in which only replacements of the closer member of the population were considered independently of the distance δ among them. In this way, the best individual is always incorporated and the modularity and ERI ranges are wider. In addition, evolution while slower is able to reach the optimal solutions.

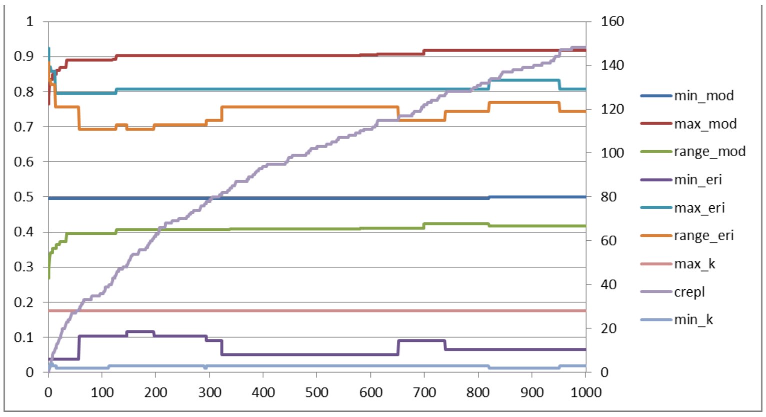


Figure 9 Diversity of updatePopulation implemented

For a 20 execution round of each algorithm the averaged results are:

TABLE II. UPDATE POPULATION VARIATIONS RESULTS

Algorithm	*Optimal Solution*	*Mean Modularity*	*Stdev Modularity*	*Mean ERI*	*Stdev ERI*
COMB-MA	0 / 20	0.89189	0.01568	0.12756	0.0576
COMB-MA no duplicates	0 / 20	0.89660	0.01559	0.14551	0.0668
Closer replacement	9 / 20	0.91763	0.00388	0.03461	0.0397

Both Modularity and ERI, as metric, share an important characteristic; their optimal value does not allow deviations towards one direction (ERI cannot be negative, 0 is the optimal and Modularity has an upper bound that is the optimal). This makes it is not required to calculate any absolute or square error to measure the accuracy, as deviations do not compensate.

Based on these results, the Closer Replacement implementation was incorporated to the algorithm. No further analysis has been done as both metrics obtain less variant (smaller stdev) results and better mean.

C. *Parameter tuning*

For the described configurations the results obtained averaging 20 executions are:

TABLE III. RESULTS FOR ALL CONFIGURATIONS

Algorithm	***Optimal Solution***	***Mean Modularity***	***Stdev Modularity***	***Mean ERI***	***Stdev ERI***
Default	2 /20	0.90335	0.01494	0.09038	0.05853
Dejong	6 /20	0.91803	0.00197	0.04038	0.03632
Grefenstette	9 /20	0.91831	0.00150	0.03974	0.03886
MicroGA 1	0 /20	0.87105	0.02066	0.23653	0.09024
MicroGA 2	0 /20	0.87598	0.02606	0.21730	0.10745

Both Dejong and Grefenstette have very close results in Modularity and ERI although Grefenstette settings obtain the optimal solution more times. If computational resources and time would have allowed both settings should have been selected as except for the number of hits there is no statistically significant difference among the two results.

Given that time has become a limitation, Grefenstette settings have been selected based on its much higher optimal solutions found.

Fixing these parameters, three different experiments have been run trying to find a more optimal configuration. It is well known, as mentioned in class, that the tuning of the parameters cannot be done individually as they are closely interconnected, as clearly stated in [17]:

1. Different parameter settings may result in large difference in performance on a single problem.
2. A parameter setting that is good for one test problem may not be suitable for another one.
3. Dependences exist among GA parameters, making it inappropriate to tune parameters independently.

But in absence of a simple methodology this simple approach has been chosen albeit the expected results are far from optimal.

For the number of generations, experiments have been run for 2000, 4000, 8000, 16000, 32000, 64000 and 128000 generations. The results have been as shown in the following figure, where the red line corresponds to the number of times the optimal solution has been found (right axis). NORM_MOD is the normalized difference to the optimal modularity (norm_mod=opt-mod/opt).

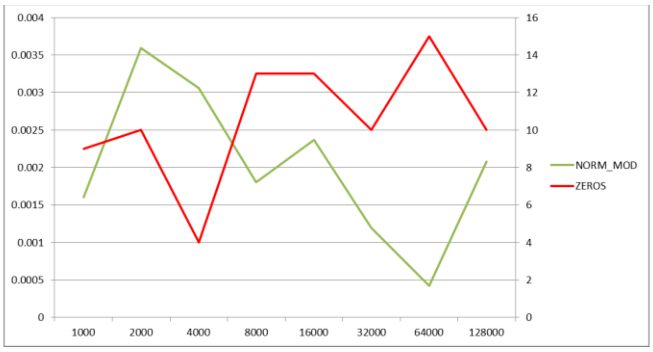


Figure 10 Best results for different maximum generations

The graph shows that the tendency is not monotonic. Given again that computing resources have become a limitation a number of 8000 iterations have been selected as the number of zeroes (13) is close to the maximum obtained (15) but this for an 8x number of generations while the difference in modularity is below 0.3%.

For population size, the experiment has been repeated for 10, 20, 30, 50, 100, 150, 200 and 300 individuals obtaining the following results. Note the logarithmic scale in the left axis (for NORM_MOD and STDEV):

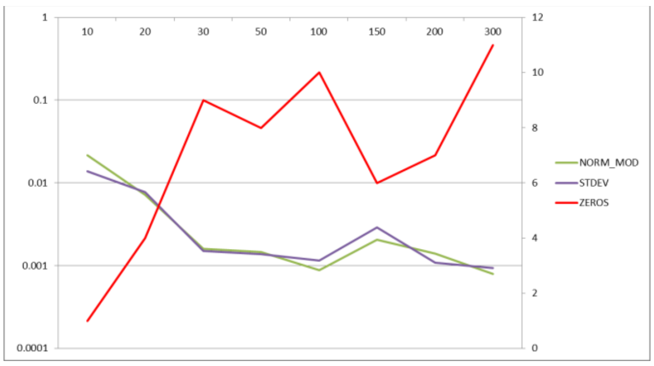


Figure 11 Average results and variance for different population sizes

The best results are then obtained for 100 and 300 individuals, without having a monotonic tendency, although there is a high correlation between average normalized modularity and zeroes. As the original value was 100, the other value is selected (300). From the traces of the problem it can be seen that the increase in the population size generates a much fitter population having individuals whose modularity is up to 80% of the maximum value. This makes the population starts closer to the optimal solution maintaining a large and diverse mass population to help the best individuals continue improving the fitness of the future generations.

For Mutation Probability, the experiment is repeated for probability values of 0, 0.001, 0.01, 0.05, 0.1, 0.2, 0.4, 0.6 and 0.9:

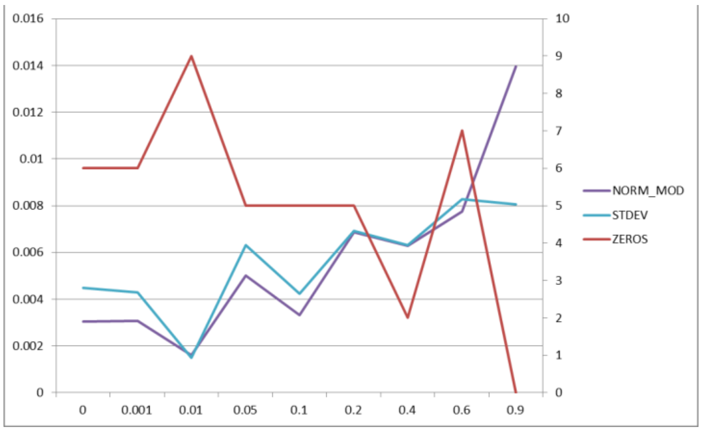


Figure 12 Average results and variance for different Pm

Again the results show a clear anti-correlation of the number of zeroes and the average normalized modularity. Based on this, the probabilities of 0.01 and 0.6 are selected, as they are very different and both generate very good results compared to tested values around them.

Based on the settings for each value that have produced the best results, two new configurations have been evaluated.

TABLE IV. BEST PARAMS CONFIGURATIONS

Configuration Name	**Parameters**		
	Population Size	***Number of Generations***	***Mutation Probability***
Best Params 1	300	8000	0.01
Best Params 2	300	8000	0.6

In the first case, the optimal solution has been found for the 20 executions, while in the second one, it has been for 16.

TABLE V. RESULTS FOR BEST PARAMS 2 CONFIGURATION

Algorithm	***Optimal Solution***	***Mean Modularity***	***Stdev Modularity***	***Mean ERI***	***Stdev ERI***
Best Params 2	16 / 20	0.919592	0.000394	0.005128	0.010256

If instead of 8000 generations, 1000 generations are allowed, the results are still satisfactory for the first set (Pm=0.01):

TABLE VI. RESULTS FOR BEST PARAMS CONFIGURATIONS WITH 1000 ITERATIONS

Algorithm	*Optimal Solution*	*Modularity*	*ERI*	*NMI*
Best Params 1 (1000 generations)	11/20	0.919058	0.02051	0.9630
Best Params 2 (1000 generations)	0/20	0.9079511	0.08461	0.855383

So, even when the optimal solution is not found the best solution for the execution is very close to it.

Supported by the improvement observed with the different configuration settings, some of the configurations have been evaluated against a second network, the Dolphins. The results obtained are far from good as in none of the configurations the optimal solution has been found not even once. The different metrics calculating do not indicate good results.

TABLE VII. AVERAGE RESULTS FOR DOLPHINS PROBLEM

Configuration Name	**Parameters**			**Results (Average values)**		
	Population Size	*Number of Generations*	*Mutation Probability*	*Modularity*	*ERI*	*NMI*
Grefenstette	30	1000	0.01	1.004813892	0.148113	0.770945
Best population size	50	1000	0.01	1.016592	0.11195	0.818573
Best max. generations	30	16000	0.01	1.016864	0.116981	0.805675
Best mutation Probability	30	1000	0.05	1.015225	0.122327	0.809444
Best Params 1 (for ZKC)	300	8000	0.01	1.02650	0.10440	0.82608
Best Params 2 (for ZKC)	300	8000	0.6	1.02359	0.09968	0.83424
Best Params 1 with Reduced Generations	300	1000	0.01	1.00654	0.13176	0.79909
Best Params 2 with Reduced Generations	300	1000	0.6	0.98932	0.16603	0.75366

If instead of considering the average results, we consider the best results, we can observe that quite often as many as 16 out of 20 times, the solution gets stacked in a different value very close to the optimal:

TABLE VIII. BEST RESULTS FOR DOLPHINS PROBLEM

Configuration Name	**Results (Best values)**		
	Modularity	*ERI*	*NMI*
Grefenstette	1.02646	0.10691	0.82221
Best population size	1.02646	0.10691	0.82221
Best max. generations	1.02679	0.12578	0.76503
Best mutation Probability	1.02679	0.12578	0.76503
Best Params 1 (for ZKC)	1.0277	0.05031	0.94074
Best Params 2 (for ZKC)	1.02646	0.10691	0.82221
Best Params 1 with Reduced Generations	1.01380	0.08805	0.86431
Best Params 2 with Reduced Generations	1.01420	0.08176	0.83719

D. *Benchmark*

If the results obtained are compared with the two reference papers, we obtain the following table:

TABLE IX. BENCHMARK RESULTS FOR THE ZACHARY KARATE CLUB

Metric Average	**Zachary Karate Club (K=4)**			
	My algorithm (Best Params Gen=8000)	***My algorithm (Best Params Gen=1000)***	***COMB-MA***	***BGLL***
NMI	1	0.9630	1	1
ERI	0	0.02051	0	0
Modularity [-0.5,1]	0.4198	0.4190	0.4198	0.4198

TABLE X. BENCHMARK BEST RESULTS FOR THE DOLPHINS

Metric		**Dolphins (K=5)**		
		My algorithm (best settings)	***COMB-MA***	***BGLL***
Average	NMI	0.8185	0.976	n/a
	ERI	11.1%	1.9%	n/a
	Modularity [-0.5,1]	0.516864	0.5286	0.5281
Best	NMI	0.94074	1	n/a
	ERI	5.03%	0	n/a
	Modularity [-0.5,1]	0.5277	0.5286	0.5286

For Meme-NET the results cannot be directly compared but from indirect comparison of the results provided it can be observed that the optimal solution is not obtained since the NMI claimed for the best K=5 partition does not match the NMI between the optimal solution for K=5 and K=2. Instead the available results for BGLL (initialization used for BGLL) have been included in the comparison.

V. EXTENSIONS, STRENGTHS AND WEAKNESSES

The algorithm developed has been able to find the optimal solution with a high confidence (16 out of 20 times for the worst of the two best parameters selected and 20 out of 20 for the best parameters) in the case of the Zachary Karate Club problem. To do so, it has required a very high number of iterations, 8000.

Nevertheless good results were also obtained when the number of iterations was set to 1000, 11 out of 20, and this could be used to increase the likelihood of the algorithm finding the best solution. Some methods restart the algorithm when convergence is detected (MicroGA settings). If this approach was used the algorithm would find the optimal solution even with 1000 generations and less than 3 restarts (4 executions) with a probability of 95.8%: (1-11/20)^4. That would be half the iterations for an 8000 round execution.

In order to continue improving the algorithm, several ideas should be further analyzed to detect the origin of the problems in the algorithm. Some clues based on the analysis done in the project timeframe are clear.

The LocalSearch algorithm only handles the single nodes in the partition, while this is contributing to improve the modularity it is not exhaustive enough and in many cases some clusters with few nodes exist that could be evaluated to migrate to other cluster and further improve the results while reducing the number of generations required. This can affect also to the number of clusters in the solution if very often some groups of nodes are put together.

The UpdatePopulation algorithm, that only replaces the closer element to the offspring, while has succeeded on maintaining a wide range of modularity and ERI in the population, it is suspicious of failing to maintain a wide middle class that keeps evolution ongoing for a longer period. My assumption, without further time to analyze, is that the population is made up of many poor partitions (present in the initial population) and a few very fitted but suboptimal individuals that do not find good genes in the mass population that can contribute to their evolution. This would cause the elite to be trapped in local minima.

Following, with the parallelism between algorithms and natural systems, middle class is the power engine of a society. An additional test should be implemented to see the modularity distribution of the population and the ERI distances among elements with similar modularity. For low modularity the ERI distance will be high while for high modularity the ERI distance will be much closer. Hence, the best elements are replacing each other maintaining the elite group small (fratricide genocide). Plotting the modularity of the replaced individual and the offspring as well as the parents' modularity would help to diagnose the problem whose solution may lay more in the field of optimization problems and complex networks that GA/MA.

On the Combination algorithm, we have used a static measure of the modularity. An option would be to apply a dynamic measurement as the clusters are modified by node removal of clusters incorporated to the solution.

Given that one of the problems of the algorithm is that it requires a huge number of generations, some preliminary work has been done and some papers on how to improve convergence in GA has been found, in particular [15] and [16] describe, respectively, how to reduce convergence time and how to avoid Take Over time, although the latter refers to parallel GA's. In [17], some guidelines on how to tune the parameters of a GA algorithm are introduced requiring the implementation of a meta-problem (Meta-EA) to find the best values for the GA parameters.

The algorithm has been successful to not require a fitted initial population as was the case for COMB-MA and was our main complain about the algorithm. In addition, the algorithm has shown that is able to obtain very good solutions in the first generations for Zachary problem.

Despite not being able to find the optimal solution for the Dolphins problems, the solutions obtained are close to the optimal what somehow supports, given that the development effort has been limited, that this Genetics Engineering inspiration to look for the best genes in the parents can generate good results in problems where domain knowledge is available. Also, this result indicates that additional tuning of the LocalSearch (moving nodes in non-single clusters) or the Combination could optimize the algorithm and allow it to continue improving what needs to take it to be able to find the optimal solution, with high probability, as NMI=0.94.

The algorithm has not required additional parameters to those in any regular GA algorithm and has not used the δ parameter in COMB-MA whose use should be reconsidered in a review of the UpdatePopulation method given that our results are much worse in both overall results and (presumably) number of generations required for convergence.

The technique used to fine tune the parameters of the GA algorithm has provided good results in the Karate Club problem. Despite this, no conclusion can be extrapolated. It may be, as the lack of monotonic behavior in all three parameters graphics indicates, a lucky coincidence or an evil influence trying to mislead the efforts put in the project. It is interesting though that Pm=0.6 (less than half mutations allowed) also has good results and it adds to the idea that the link between GA and exact methods is bidirectional.

Another aspect to consider would be to use the NMI distance to replace the ERI distance in the updatePopulation method and not only to benchmark the results. The behavior of the two is very different when the solutions are close (ERI=15%) to the optimal with NMI being below 25% in some cases. If NMI is used to measure the quality of the solutions, it could be also useful to consider it to measure the distance among population individuals.

VI. CONCLUSIONS

The algorithm implemented includes all essential parts of a MA algorithm without breaking the general structure of these algorithms even in the combination method that is the one that in essence is more different as the genes combined are selected following domain knowledge criteria.

As explained in previous section, the implementation has been able to prove some merits as not requiring a fitted initial population, being able to obtain close to optimal solutions in the first generations, as can be seen in Figure 9 and being able to solve the Zachary Karate Club with high confidence, using though a very high number of generations. The ability to find optimal solutions is still high if the number of generations is reduced and it can be fostered applying some restart techniques common in the field.

The attempt to fine tune the GA parameters has confirmed the hypothesis about the dependency, no linearity and (apparently) no problem independency.

An original method to profile the behavior of the main algorithm, based on analysis of metrics range, has been put in place to detect problems in early phases of the algorithm design and implementation and evaluate the impact of the solution. A second generation of this profiling method has been designed in previous section but lack of time has prevented the implementation of it and the possible alternatives it could have stemmed. Part of this extended analysis could also be the analysis of number of useful mutations and population replacements used in the updatePopulation() implementation.

As was discussed in person during the problem definition, some of the heuristics (modularity based priority, movement of one vertex at a time …) are quite straightforward for any person working in the field. When nothing has been published about the application of these techniques the reason it is likely to be the techniques are not able to cope with the complexity of the problem. Still, the project for an introductory course in Computational Intelligence is a perfect opportunity to attempt an endeavor of the problem, find problems and try to apply some techniques, related to the field, that help to improve the results even if the outcome is far from the desired.

The results obtained have been compared to the extent allowed by the information available applying relevant metrics in the field (modularity, ERI, NMI) that have been implemented from scratch and validated in the three cases. A deeper statistical evaluation has not been done as the criteria used, to select best parameters, has been computational cost or diversity of the settings chosen (as in the Pm case). Also the results obtained didn't justified applying more rigorous methods (that have been indicated where could be useful) as the focus was required to be put on improving the algorithm.

It has also been clear, when trying to collect the results obtained in previous work that many papers fail to provide the material needed to replicate the results. In my case, because of the quick and efficient collaboration of Dr. Gach, I have been able to obtain a minimum dataset that has been the basis for my experiments. If I ever manage to publish something, I will make sure that I don't fall in the same tendency.

REFERENCES

[1] Fortunato, S. (2010). "Community detection in graphs." Physics Reports 486(3): 75-174.

[2] Moscato, P. and C. Cotta (2003). A Gentle Introduction to Memetic Algorithms. Handbook of Metaheuristics. F. Glover and G. Kochenberger, Springer US. **57:** 105-144.

[3] Neri, Ferrante, Carlos Cotta, and Pablo Moscato, eds. "Handbook of memetic algorithms." (2011).

[4] http://en.wikipedia.org/wiki/Genetic_engineering

[5] W. W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33, 452-473 (1977).

[6] A.Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, Behavioral Ecology and Sociobiology 54, 396-405 (2003).

[7] Wu, Qin, et al. "“Follow the Leader”: A Centrality Guided Clustering and Its Application to Social Network Analysis." The Scientific World Journal 2013 (2013).

[8] Hao, J.-K. (2012). Memetic algorithms in discrete optimization. Handbook of Memetic Algorithms, Springer**:** 73-94.

[9] Gach, O. and J.-K. Hao (2012). A memetic algorithm for community detection in complex networks. Parallel Problem Solving from Nature-PPSN XII, Springer**:** 327-336.

[10] Gong, M., et al. (2011). "Memetic algorithm for community detection in networks." Physical review E **84**(5): 056101.

[11] NetworkX library: https://pypi.python.org/pypi/networkx/

[12] P. Galinier and J.K. Hao. Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization, 3(4):379–397, 1999.

[13] Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech.: Theory Exp., P10008 (October 2008), doi:10.1088/1742-5468/2008/10/P10008

[14] http://eislab.gatech.edu/people/scholand/gapara.htm

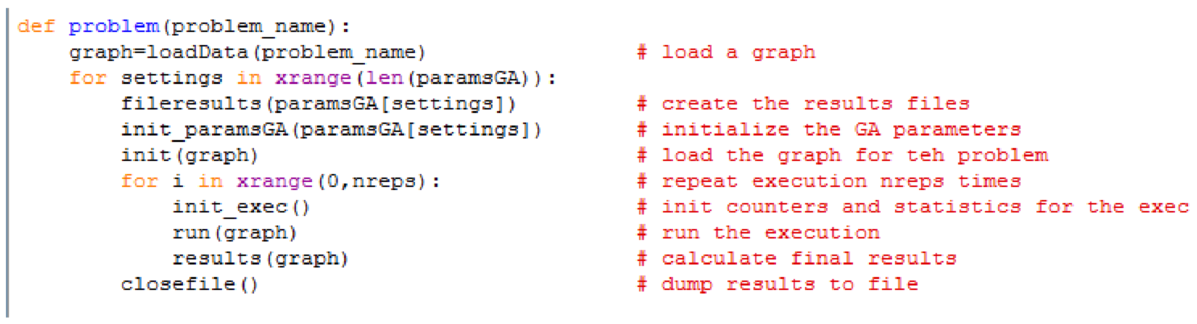
[15] Maria Angelova and Tania Pencheva, “Tuning Genetic Algorithm Parameters to Improve Convergence Time,” International Journal of Chemical Engineering, vol. 2011, Article ID 646917, 7 pages, 2011. doi:10.1155/2011/646917

[16] Cantu-Paz, E. (1999). Migration policies and takeover times in genetic algorithms. Proc. Genetic and Evolutionary Computation Conference (Joint meeting of the 8 th International Conference on Genetic Algorithms and the 4 th Annual Genetic Programming Conference), Morgan Kaufmann (ISBN 1558606114.

[17] Yuan, Bo, and Marcus Gallagher. "A hybrid approach to parameter tuning in genetic algorithms." Evolutionary Computation, 2005. The 2005 IEEE Congress on. Vol. 2. IEEE, 2005.

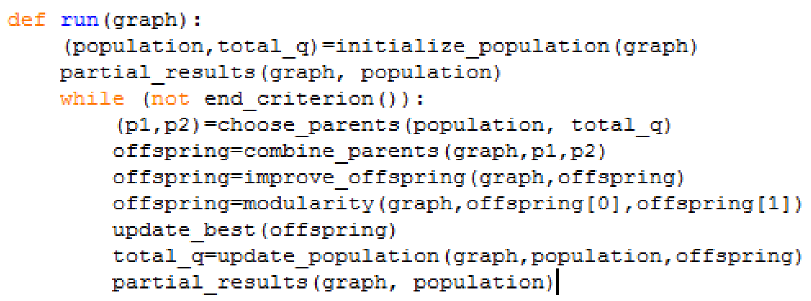
APPENDIX A. IMPLEMENTATION DETAILS AND SOURCE CODE

A. *General structure*



```
def problem(problem_name):
    graph=loadData(problem_name)                # load a graph
    for settings in xrange(len(paramsGA)):
        fileresults(paramsGA[settings])         # create the results files
        init_paramsGA(paramsGA[settings])       # initialize the GA parameters
        init(graph)                             # load the graph for teh problem
        for i in xrange(0,nreps):               # repeat execution nreps times
            init_exec()                         # init counters and statistics for the exec
            run(graph)                          # run the execution
            results(graph)                      # calculate final results
        closefile()                             # dump results to file
```

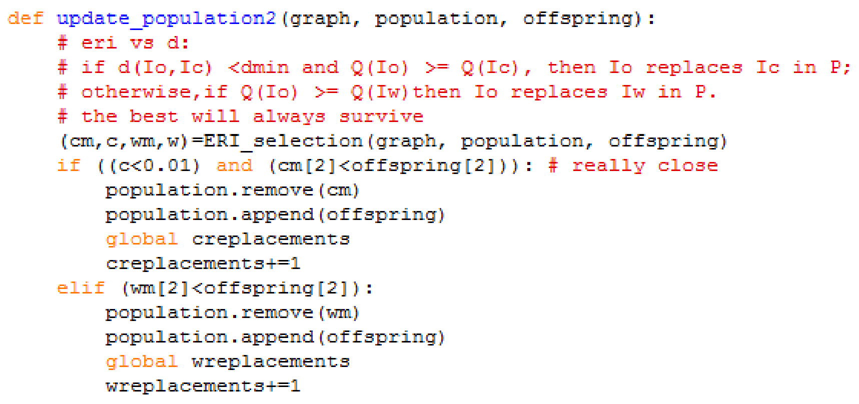
B. *Memetic Algorithm structure*



```
def run(graph):
    (population,total_q)=initialize_population(graph)
    partial_results(graph, population)
    while (not end_criterion()):
        (p1,p2)=choose_parents(population, total_q)
        offspring=combine_parents(graph,p1,p2)
        offspring=improve_offspring(graph,offspring)
        offspring=modularity(graph,offspring[0],offspring[1])
        update_best(offspring)
        total_q=update_population(graph,population,offspring)
        partial_results(graph, population)
```

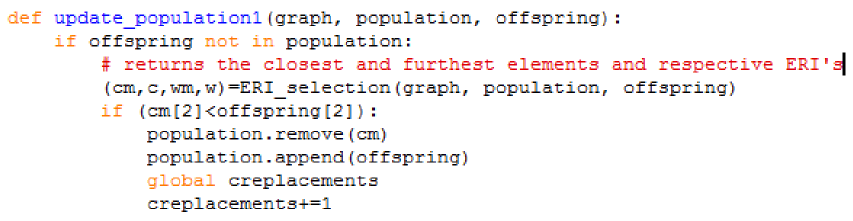
C. *Update Population*

1) *In COMB-MA*



```
def update_population2(graph, population, offspring):
    # eri vs d:
    # if d(Io,Ic) <dmin and Q(Io) >= Q(Ic), then Io replaces Ic in P;
    # otherwise,if Q(Io) >= Q(Iw)then Io replaces Iw in P.
    # the best will always survive
    (cm,c,wm,w)=ERI_selection(graph, population, offspring)
    if ((c<0.01) and (cm[2]<offspring[2])): # really close
        population.remove(cm)
        population.append(offspring)
        global creplacements
        creplacements+=1
    elif (wm[2]<offspring[2]):
        population.remove(wm)
        population.append(offspring)
        global wreplacements
        wreplacements+=1
```

2) *In my implementation*

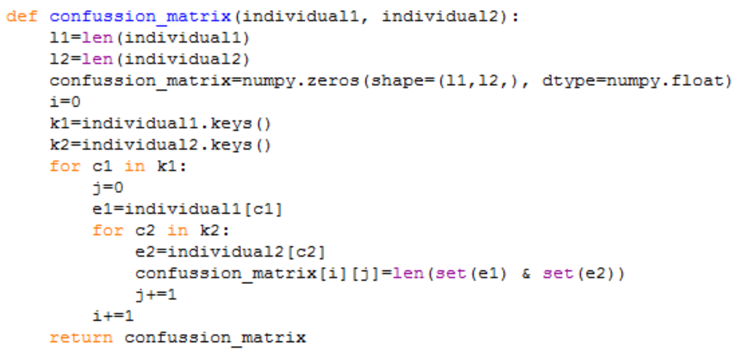


```
def update_population1(graph, population, offspring):
    if offspring not in population:
        # returns the closest and furthest elements and respective ERI's
        (cm,c,wm,w)=ERI_selection(graph, population, offspring)
        if (cm[2]<offspring[2]):
            population.remove(cm)
            population.append(offspring)
            global creplacements
            creplacements+=1
```

D. *Metrics*

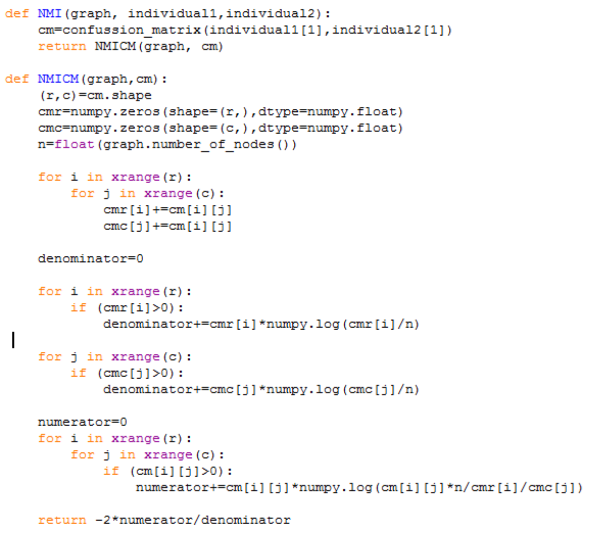
1) *NMI:*

a) *Confusion matrix:*



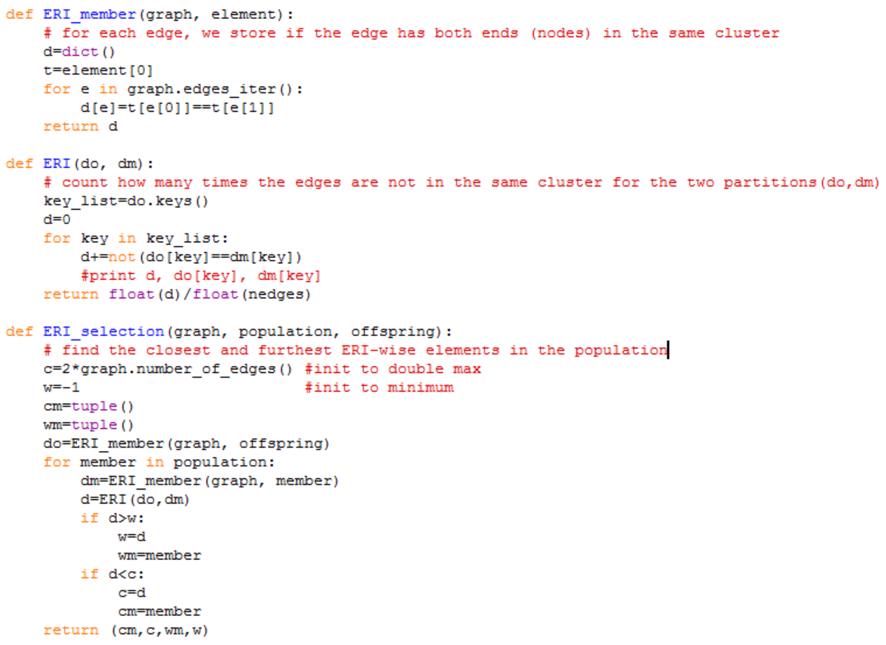
```
def confussion_matrix(individual1, individual2):
    l1=len(individual1)
    l2=len(individual2)
    confussion_matrix=numpy.zeros(shape=(l1,l2,), dtype=numpy.float)
    i=0
    k1=individual1.keys()
    k2=individual2.keys()
    for c1 in k1:
        j=0
        e1=individual1[c1]
        for c2 in k2:
            e2=individual2[c2]
            confussion_matrix[i][j]=len(set(e1) & set(e2))
            j+=1
        i+=1
    return confussion_matrix
```

b) *NMI calculation:*



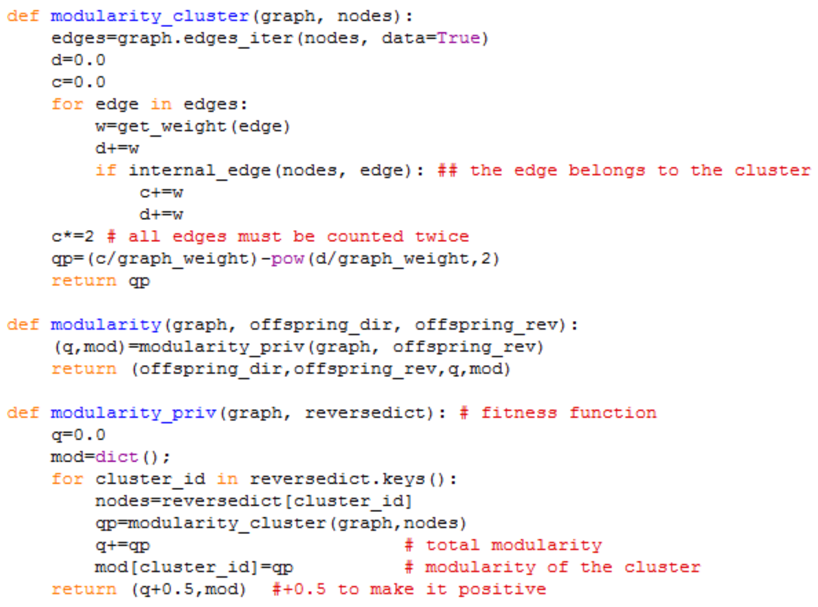
```
def NMI(graph, individual1,individual2):
    cm=confussion_matrix(individual1[1],individual2[1])
    return NMICM(graph, cm)

def NMICM(graph,cm):
    (r,c)=cm.shape
    cmr=numpy.zeros(shape=(r,),dtype=numpy.float)
    cmc=numpy.zeros(shape=(c,),dtype=numpy.float)
    n=float(graph.number_of_nodes())

    for i in xrange(r):
        for j in xrange(c):
            cmr[i]+=cm[i][j]
            cmc[j]+=cm[i][j]

    denominator=0

    for i in xrange(r):
        if (cmr[i]>0):
            denominator+=cmr[i]*numpy.log(cmr[i]/n)

    for j in xrange(c):
        if (cmc[j]>0):
            denominator+=cmc[j]*numpy.log(cmc[j]/n)

    numerator=0
    for i in xrange(r):
        for j in xrange(c):
            if (cm[i][j]>0):
                numerator+=cm[i][j]*numpy.log(cm[i][j]*n/cmr[i]/cmc[j])

    return -2*numerator/denominator
```

2) *ERI:*



```
def ERI_member(graph, element):
    # for each edge, we store if the edge has both ends (nodes) in the same cluster
    d=dict()
    t=element[0]
    for e in graph.edges_iter():
        d[e]=t[e[0]]==t[e[1]]
    return d

def ERI(do, dm):
    # count how many times the edges are not in the same cluster for the two partitions(do,dm)
    key_list=do.keys()
    d=0
    for key in key_list:
        d+=not(do[key]==dm[key])
        #print d, do[key], dm[key]
    return float(d)/float(nedges)

def ERI_selection(graph, population, offspring):
    # find the closest and furthest ERI-wise elements in the population
    c=2*graph.number_of_edges() #init to double max
    w=-1                        #init to minimum
    cm=tuple()
    wm=tuple()
    do=ERI_member(graph, offspring)
    for member in population:
        dm=ERI_member(graph, member)
        d=ERI(do,dm)
        if d>w:
            w=d
            wm=member
        if d<c:
            c=d
            cm=member
    return (cm,c,wm,w)
```

3) *Modularity:*



```
def modularity_cluster(graph, nodes):
    edges=graph.edges_iter(nodes, data=True)
    d=0.0
    c=0.0
    for edge in edges:
        w=get_weight(edge)
        d+=w
        if internal_edge(nodes, edge): ## the edge belongs to the cluster
            c+=w
            d+=w
    c*=2 # all edges must be counted twice
    qp=(c/graph_weight)-pow(d/graph_weight,2)
    return qp

def modularity(graph, offspring_dir, offspring_rev):
    (q,mod)=modularity_priv(graph, offspring_rev)
    return (offspring_dir,offspring_rev,q,mod)

def modularity_priv(graph, reversedict): # fitness function
    q=0.0
    mod=dict();
    for cluster_id in reversedict.keys():
        nodes=reversedict[cluster_id]
        qp=modularity_cluster(graph,nodes)
        q+=qp                          # total modularity
        mod[cluster_id]=qp             # modularity of the cluster
    return (q+0.5,mod)  #+0.5 to make it positive
```