KULeuven

Department of
Computer Science

ADVANCED CAPITA SELECTA ARTIFICIAL INTELLIGENCE (H02A8A)

Metaheuristics - Report

Jeroen Craps (r0292642)
Jorik De Waen (r0303087)

Academic year 2016–2017

Contents

1 introduction

Networks are ever so present in the world, due to the rise of social media and the emergence of Big Data[1] over the last decade. The detection of communities[2] in these large networks has grown in importance. Communities can be seen as fairly independent parts of the graph. Certain elements can be present in multiple communities. This concept can be compared to a person being part of multiple groups of friends on social networks at the same time. A community implies important information about relationships between topology and network functionality. The information contained in these graphs can be of utmost importance to understanding the data that is being dealt with. The problem has been proven to be NP-hard [**?**].

Seeing that this problem is NP-hard the most interesting approach would be to use a meta-heuristic. A lot of work has been done with genetic algorithms INSERT CITATIONS. In this scientific report a method is presented to help tackle this problem. An attempt is made to decrease the amount of nodes and with that the length of chromosones in the genetic algorithm. With this we hope to improve the speed of the exsisting algorithms without decreasing their accuracy.

2 Background

This chapter will explain some elements to be able to understand the contents of this scientific report. The focus will be put on graph theory and genetic algorithms. For both of these the basics will be shortly mentioned and the reasoning behind why they are important for this work.

2.1 Graph Theory

Graph theory is the mathematical theory of the properties and applications of graphs. A graph $G = [V, E[$ is a collection of vertices (points) and edges (lines). An edge connects two vertices or creates a loop for one vertice. Graphs can have a certain properties, e.g. directed, undirected, weighted or complete. Certain combinations of these properties are also possible.

In directed graphs an edge has an orientation, e.g. $A \rightarrow B$. This says that A has a connection to B, but B does not have a connection to A. Normally in an undirected graph an edge might look like $A - B$, which implies a connection in both ways.

Edges in the graph can contain a certain scoring-function, if this is the case then it's called a weighted graph. An common example of this is the euclidean distance between the two vertices the edge connects together. A graph is complete when all nodes are connected to eachother. A graph $G' = [V', E'[$ is a subgraph of G if $V' \subset V$, $E' \subset E$ and E' only including edges between the vertices of V'. A complete subgraph is called a **clique**. There are other properties for graphs, but they are not required for the understanding of this report.

2.2 Genetic Algorithm

In this subsection the concept of genetic algorithms will be expanded upon. First, the basic concept will be explained. Afterwards a more concrete explanation will be given about the elements that have been used in this research. Important to note is that if a problem can be represented and the proper operators can be designed any problem can be solved by a genetic algorithm if the purpose of it is to optimize a certain scoring function.

[1] Big data is a term for data sets that are so large or complex that traditional data processing applications are inadequate to deal with them.

[2] A network is said to have community structure if the nodes of the network can be easily grouped into (potentially overlapping) sets of nodes such that each set of nodes is densely connected internally.

Position	1	2	3	4	5	6	7	8	9	10	11
Genotype	3	5	7	10	4	11	9	11	3	2	8

Figure 1: The locus representation of a graph.

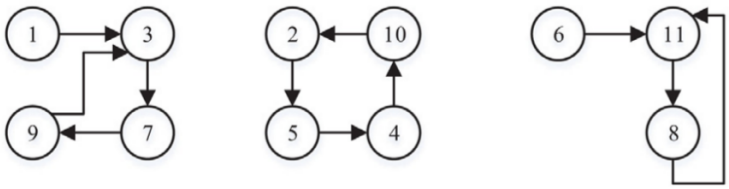

Figure 2: The visualisation of Fig. 1.

2.2.1 Basics

A genetic algorithm is a metaheuristic inspired by natural evolution and selection. The goal of the metaheuristic is to mimic biological evolution, which strives to improve on itself. It is a method for solving optimization problems which can be constrained or unconstrained. A population of individuals is generated each generated each iteration. The strongest individual, according to the fitness function, approaches an optimal solution for the given problem.

A typical genetic algorithm contains a genetic representation of the solution domain and a fitness function to evaluate the solution in the problem domain. The chromosome[3] is the set parameters which define a solution for the problem. A set of chromosomes is called a population. By iteratively updating the population by using certain operators, the algorithm attempts to find the optimal solution for the specified problem.

2.2.2 Representation

The most commonly used representation in genetic algorithms is the binary representation. But this isn't always a good choice, because the translation to a representation that we can easily understand (phenotype) might be very difficult. That is why for the problem that will be discussed in this report we will be using a locus based representation. An example of this can be seen in Fig. 1.

An individual is represented as an array of numbers. The index represents the identification number of a vertice in the graph. The value in the array stands for the vertice it is connected to. While decoding the representation we say that a set of vertices are in a group if there is a link between any of the vertices.

2.2.3 Operators

The goal of the algorithm is to improve on the initial population which can be initiated randomly or with a heuristic. To do so some operators are required. Exploring the search space in usually done by using mutation which will randomly change an individual. Exploitation is done by combining information of two (or more) individuals to one (or more) new individual(s). For both of these a short explanation will be given.

Crossover

Crossover is the operator that will take care of find the optimum in the current population by combining high scoring individuals with eachother. By doing this an attempt to select the superior parameters will be

[3]Also known as the genotype.

taken from both and used to form an even better individual to the problem. Combining two good solutions doesn't always lead to a better one, but combining one good solution with a bad one might do so.

Some example of these on binary representations are the following:

- One-point crossover
-

Mutation

Mutation normally allows for new sections of the search space to be explored by creating a new value for a parameter that previously wasn't present in the population. An expample of this is a bit flip on a binary representation for a possible solution. By doing this a bit that is always 0 in the population can be changed to 1 and thus introducing new "DNA" into the population. This is necessary to keep the population from going into a local optima instead of exploring the entire search space and potentially missing out on a better solution.

3 Relevant research

A large amount of research has gone into community detection in the last several decades. Finding a good way to approach this issue has been particularly hard because it is not trivial to come up with an exact definition of a community and a metric to compare different methods of partitioning. As the amount of data being gathered grows at an incredible pace, so has the size of the networks which need to be analysed. After a short introduction for both domains that are being combined, some relevant papers are mentioned and discussed.

3.1 Genetic Algorithms

Traditional approaches simply do not scale to many thousands, let alone millions, of nodes. Because the problem is NP-hard, calculating the optimal solution for large networks is an unreachable goal. Genetic algorithms provide a way to still find solutions in such large networks. One such algorithm was introduced in a recent paper by Li et al. [**?**], using a multi-agent approach.

3.1.1 MAGA-Net

In the paper by Li et al. [**?**] every agent is a candidate solution for the community detection problem. Each agent "lives" in a lattice structure. In this lattice, candidate solutions compete with their direct neighbours. Due to this reasoning, agents can perceive and react to their direct environment. All agents work together to achieve the common goal, which is to optimize the fitness function. A candidate solution is in this case a division of the network in communities.

The solution proposed in the paper avoids getting trapped in a local optima by increasing modularity as much as possible. Mainly done by the implementation of their operators: split and merging based neighborhood competition operator, hybrid neighborhood crossover, adaptive mutation and self-learning operator. The experiments show that the algorithm is able to find the global optima and can solve large-scale social networks.

3.1.2 Overlapping communities

Most algorithms for community detection assume that each node can only be a part of a single community. In many cases, this is simply not true. Overlapping community detection algorithms can be divided into two groups: node-based algorithms and link-based algorithms [**?**].

The node-based algorithms focus directly on the nodes and try to detect communities by looking at how nodes are related. The link-based algorithms are built with the assumption that the links between nodes are actually more important than the nodes themselves. Not the individuals, but the relations between the individuals define the community. The links are divided into communities, and only afterwards is that translated to the nodes. Generally, link-based algorithms have been shown to yield superior results, but at a much higher computational cost. Ding et al. [**?**] have proposed a new approach which attempts to improve on the computational cost typically associated with a link-based algorithm using network decomposition. This algorithm is not genetic, but others have proposed several different genetic overlapping community detection algorithms [**?**, **?**, **?**].

3.2 Graph Theory

4 Hypothesis

4.1 Hypothesis

The main focus of this research is to see if equal accuracy can be achieved with a pre-processed representation of the original communtity graph using a combination of previously exsisting algorithms. These algorithms will be further explained in Chapter [**?**].

4.2 Goal

The goal of this research is to see if we can achieve similar performance with a reduced version of the original graph.

4.3 Research questions

Main research question

> Does the reduction of a graph have any influences on the performance of current state of the art, mainly on the time that is required to reach reasonably good results?

Other research questions

To reduce the complexity of the main research question, some smaller research questions have been chosen.

> What kind of influence does the replacement of a small cliques by a single node have on the information in the graph?
>
> Can all of the information of the graph be retained while still reducing the size?
>
> Is it worth to preprocess a graph before applying the meta-heuristic?
>
> What differences are there between the algorithm with and without the preprocessing?

5 Method

In this section

5.1 Preprocessing

5.2 Genetic Algorithm

6 Implementation

This section outlines how the algorithms are implemented. All the code is written in Java 8.

The first algorithm reads a graph from a text file and generates useful data structures for this graph. Afterwards, it searches for cliques in the graph and combines the vertices in those cliques into a single vertex. This reduces the amount of vertices and edges in the graph.

The second algorithm takes a graph and divides it into communities using a genetic algorithm. It uses several genetic operators, including a special self-learning operator.

6.1 Clique algorithm

Before we can describe the clique algorithm, we will describe the data structures used by this algorithm. A Graph object is a collection of Vertex objects and Edge objects. An edge connects two vertices and also contains a weight value. When a graph is imported, these weights are usually equal to one. However, when the algorithm replaces vertices with cliques, edges with higher weights will be constructed.

When the algorithm finds a clique, the vertices in that clique will be replaced by a Clique object. All edges to members of that clique are replaced by edges to the clique as a whole. This often leads to duplicate edges, which is modelled using the edge weight. This allows the algorithm to significantly reduce the amount of edges and vertices in the graph, while preserving all relevant information about the structure of the graph.

The Clique algorithm randomly samples vertices from the graph. From this sample point, the largest clique containing it is generated by following the edges to its neighbors.

6.2 Genetic Algorithm

6.2.1 Core

6.2.2 Split/Merge Neighborhood Competition Operator

6.2.3 Hybrid Neighborhood Crossover Operator

6.2.4 Adaptive Mutation Operator

6.2.5 Self-Learning Operator

7 Results and reflection

bespreking van performance met verschillende datasets bespreking van lessen die we geleerd hebben en hoe we bepaalde dingen anders zouden aanpakken