Graph partitioning – comparison of naive and evolutionary approach

Graphpartitionierung – Vergleich zwischen naiver und evolutionärer Ansatz

Vanesa Georgieva

MSc Student at FDIBA, Technical University Sofia
Sofia, Bulgaria, vanesa.georgieva@fdiba.tu-sofia.bg

***Abstract* —** In this paper is examined the problem for graph partitioning. An innovative solution of this problem is presented, which uses an evolutionary genetic approach combined with one of the first algorithms for graph partitioning – Fiduccia-Mattheyses. This solution is theoretically explained and practically implemented. A comparative analysis between the evolutionary and the naive approach is also described.

***Zusammenfassung* —** In diesem Papier wird das Problem für die Graphpartitionierung untersucht. Eine innovative Lösung dieses Problem wird vorgestellt, die einen evolütionären genetischen Ansatz mit einem der ersten Algorithmen für die Graphpartitionierung - Fiduccia-Mattheyses kombiniert. Diese Lösung wird theoretisch erklärt und praktisch umgesetzt. Eine Vergleichsanalyse zwischen dem evolutionären und dem naiven Ansatz wird ebenfalls beschrieben.

I. Introduction

A. Motivation

The graphs are fundamental data structure in computer science. When we want to represent our data as a graph and the graph become too large to be processed on one machine we should divide it into smaller pieces. Here comes the graph partitioning. Graph partitioning is used today for example in Facebook for friendship recommendation, where the graph vertices represent the users and the graph edges the personal relationships between them or in Google for PageRank computations, where websites are represented by graph vertices and the hyperlinks between the websites are represented by the graph edge.

Graphs that have billions of vertices can become too large and this brings a lot of issues. For example the computing of the desired partition on a single machine can be difficult due to memory or the processing and the finding of the best partition can take very long, because of the large amount of different partitions combinations. For solving such problems, we should determine such graph partitioning algorithm, which improves the run-time performance by partitioning, uses less memory and has the possibility to process a large amount of data. The algorithm should also be competitive with the naive methods like random graph partitioning.

In the interest of finding such an algorithm the main goal of this paper is to study the application of evolutionary approach for graph partitioning and to compare it performance with the naive approach for graph partitioning.

B. Related works

In 1970 Kerninghan and Lin were maybe the first who defined and investigated the graph partitioning problem [2]. They invented an algorithm that was improved from Fiduccia and Mattheyses in 1982 [3]. Today there are a lot of studies and different solutions of the graph partitioning problem. For example Shopov and Markova presented the applicability of some k-medoids algorithms for graph clustering in the Social Network Analysis [8]. A clustering is also partition of vertices, but there is no balance constraint and the number of partitions k is not given.

In the following part the graph concept and the graph partitioning problem will be briefly described and related works for graph partitioning technics will be overviewed. In part three the main methods and algorithms, which will be compared in this paper will be presented: the naive approach and the evolutionary approach using the Fiduccia-Mattheyses algorithm. In the last part some experiments will be described, which will are used to compare the evolutionary and the naive approaches for graph partitioning.

II. Graph partitioning problem

There are a lot of approaches solving the graph partitioning problem such as local search algorithms, obtaining partitions, multilevel approach, evolutionary algorithms etc. For the achievement of the main goal of this work were used algorithms from the area of the local search algorithms and the evolutionary algorithms and this is why they will be examined in this part.

According to [1] the main goal of the local search algorithms is to partition a graph by improving the objective function (the count of edges, moving between the partitions) by moving vertices between the partitions. The main goal of the evolutionary algorithms is to find a population of candidate solutions (individuals) and to evolve them toward better solutions.

A. Graph theory

The graph theory is the study of graphs and their properties. It is found from the Swiss mathematician Leonhard Paul Euler. [4]

A graph is a collection of vertices (points) that are connected by edges (lines). A simple graph G consists of a

nonempty finite set V(G) of elements called vertices (or nodes), and a finite set E(G) of distinct unordered pairs of distinct elements of V(G) called edges. We call V(G) the vertex set and E(G) the edge set of G. An edge {v, w} is said to join the vertices v and w, and is usually abbreviated to vw. For example simple graph [4] G whose vertex set V(G) is {w, v, w, z} has edge set E(G), consisting of the edges uv, uw, vw and wz. In any simple graph there is at most one edge joining a given pair of vertices. However, many results that hold for simple graphs can be extended to more general objects in which two vertices may have several edges joining them. In addition, we may remove the restriction that an edge joins two distinct vertices, and allow loops - edges joining a vertex to itself. The resulting object, in which loops and multiple edges are allowed, is called a general graph - or, simply, a graph. Thus every simple graph is a graph, but not every graph is a simple graph.

There are different types of graphs depending on the number of edges, number of vertices, interconnectivity, and their overall structure. In this work we will talk about connected weighted undirected graphs. This means that there should be a path between each pair vertices, to each edge should be assigned a positive number (weight) and the edges should have no direction, they should be bidirectional. (Fig.1)

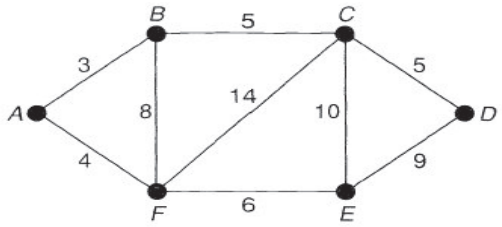

Fig. 1 Undirected weighted connected graph [4]

B. Graph partitioning

The data in the Graph partitioning problem (GPP) is presented as a graph G = (V, E). A set of n vertices is indicated with V and E indicates a set of n edges, so the graph can be partitioned into k partitions under specific criteria. The criteria is that each partition should be of about the same size and there should be a few connections between them. By fulfilling the criteria the capacity of the edges between the separate subgraphs should be minimized [5]. This problem is hard and its practical solutions are based on different approaches, called heuristics, which should solve, learn or discover problem, their practical methods are not optimal or perfect, but have immediate goals.

C. Kernighan-Lin algorithm

Probably the first, which investigated and defined the graph partitioning problem, Kernighan and Lin invented an algorithm [2], proposing simple heuristic found to produce globally optimal partition by moving vertices from one partition to another and to reduce its cost. The simplest version of this classic algorithm is to partition the graph into two partitions. They start with randomly chosen initial partition. Each partition should have about the same number of vertices. The vertices between these two partitions are exchanged by determining the cut size of each cut stage and the cut, which has maximum cut size, is saved. After all necessary exchanges are performed, the best cut, which is saved, is used as the output of the algorithm.

This idea was improved by Fidducia and Mattheyses in 1984 and is briefly described in the following subpart.

D. Fidducia-Mattheyses algorithm

Fiduccia and Mattheyses proposed improvements to Kernighan-Lin algorithm such that one pass of their version can be proved to run in linear time [3]. Like the Kernighan-Lin method, the Fiduccia-Mattheyses method performs passes in which each vertex is cut and moved at most once from one partition to another, and the best bisection observed during an iteration (if the corresponding reduction in the number of edges cut is positive) is used as input for the next iteration. However, instead of selecting pairs of vertices, the Fiduccia-Mattheyses method selects single vertex for movement.

E. Evolutionary algorithms

Evolutionary algorithms are stochastic search and optimization heuristics derived from the classic evolution theory, which are implemented on computers in the majority of cases [7]. The main goal of these algorithms is to select only the best individuals from the population, which cover specific criteria, and to reproduce only them. In this way the descendants will inherit properties only from the best individuals and the individuals, which not cover the criteria, will disappear. These population dynamics follow the basic rule of the Darvin's theory of evolution, which can be described in short as the "survival of the fittest".

III. Methods and algorithms

In this part will be explained the main two approaches for solving the graph partitioning problem, which will be compared in this paper: the naive approach and the evolutionary approach. We will begin with the naive approach.

A. Naive approach

The naive approach is a technique for estimation in which the last periods are used as current period's forecast. This method is usually used only for comparison with the forecasts, which are generated by better, developed to a high degree of complexity, techniques. In this paper it will be compared with evolutionary approach for GP.

1) Algorithm

The first part of the naive approach is dividing the vertices V of the graph G into all possible even k partitions with partition size q.

$$q = V/k \quad (q \in N) \tag{1}$$

The number of all possible partitions p of the graph is equal to the combination of all graph vertices n, which should be set into q positions, without repetitions. The order of the vertices in the partitions is not important.

$$p = C(n,q) = n!/q!(n-q)! \tag{2}$$

After the partitioning the graph into p partitions, we should remove the inappropriate ones. Inappropriate partition is partition, which contains disconnected subgraphs. We should check, if there is a path between each pair vertices in each partition and if there is no path, we should remove this partition.

After this check, if there is only one partition left that contains only connected vertices, we accept it as a final

partition of graph G and save the result as an output of the algorithm. Such a result can exist, if there is only one edge (v_n, v_m) that connects the subgraphs in the best partition. All other partitions will contain disconnected subgraphs.

If there is more than one partition left after the check for disconnected graph, we should find the best partition. The first requirement for best partition is fulfilled and the next requirement is the partition with the least connections between the subgraphs in it is to be found. For this purpose we need to calculate the gain g for each partition. The partition gain represents the count of vertices neighbors' nbc, which are distributed in the opposite partition. The smaller this count is, the less the connection between the subgraphs in the partitions are. The goal is to find the partition with the max gain:

$$g_{max} = nbc_{min} \quad (3)$$

When we find the partition with the highest gain, it is set as the best partition and will be saved as an end result and output of the partitioning of graph G.

```
Find the number p of all possible partitions of graph G such that p = n!/q!(n-q)!
For (i := 1 to p)
    Determine a balanced partition ci of the nodes into sets Ai and Bi with partition
    size = |V|/k = q
End For
For (i := to number c)
   Check connectivity of all vertices in subgraph Ai and subgraph Bi
   If (Ai OR Bi contains disconnected vertices) remove ci
End For
Let gv is empty list
For (i := 1 to size of c)
   Compute gain value g of ci
   Add g to gv[i]
End For
Best partition = the partition with max gain from gv
```

Fig. 2 Naive approach- Pseudo code

B. *Evolutionary genetic approach based on the Fiduccia-Mattheyses algorithm*

The main goal of this work is to present and compare with the naive approach, a solution of the GPP using evolutionary genetic approach, which processed large amount of data and improve the run-time performance and memory usage. In this subpart will be presented such a solution: the Fidducia-Mattheyses algorithm used together with evolutionary genetic approach.

When we use an evolutionary approach, we will need some terms, coming from the biology, which will be briefly described.

1) Evolutionary representations and operators

As described in [6] population is the number of all of the individuals of the same groups, which exist in a particular area, and have the capability of interbreeding. It is presented by a set of individuals, each of them represented graph *k*-partition. Each of these individuals will be created by random distribution of the graph vertices, which will create the chromosome of the individuals. The best individual from the population will be saved by each new generation. In general generation means aggregate of the functions and phenomena which attend reproduction.

As in the nature, as so in the evolutionary algorithm, there are constantly new populations and generations. That is why we need to set a start parameter for the maximum generation, which we want to reach. For example we can set the value of the maximum generation equals 100, so we want to find the best individual in the period of 100 epochs. When we choose the maximum generation that should be reached, we should find how big our partition can be. The dividing of the vertices V of the graph G into even k partitions should have partition size pSize (Eq.1)

2) Algorithm

To get the initial population, which we want to improve, we will use the algorithm of Fiduccia-Mattheyses. We start with adding random individuals to the first population, created by combinations from all graph vertices. The population is an array of individuals, containing k-partitions with pSize. Than for each individual in this population we want to find the best partition. The best partition for each individual is found in the following way. Firstly we execute the Fiduccia-Mattheyses algorithm for the current individual and calculate its fitness value. The fitness value is the cut size - the number of edges, which are removed from the graph, so it can be divided into k parts. The cut size should be as less as possible. If the fitness value of the individual, derived from the Fiduccia-Mattheyses (FM) is bigger than the fitness value of the current individual, we replaced the genes in the chromosome of the current individual with these from the chromosome of the individual derived from FM. We iterate the FM for each individual until no fitness improvement is found.

Now when we have initialized the initial population, we can start with its evolution. We will repeat the cycle of evolution until maximum generation is reached. Each cycle of evolution contains some steps, which will be described in the following paragraphs.

The first step is the step of the random selection. Here we select two random individuals from the population and called them parents. From these two parents we want to generate an offspring that can evolve. The generation of the offspring happens with the usage of the uniform crossover between the selected parents, which is the next step in the evolution.

During the step of the uniform crossover we want to create an individual with mixing ratio between the given parents, which enables the parent chromosomes to contribute the gene level of the offspring. First we compared the two parents and if they had similar genes, they are added to the new genes of

```
Initialize population with populationSize = vertices count/k
   For (i := to populationSize)
      Create individual
      Create FM individual as iterate Fiduccia Mattheyses algorithm until the best
      fitness value for the current individual is found
      While (fitness value of the current individual < fitness value of FM individual)
          Iterate Fiducciia Mattheyses algorithm until the best fitness value for the
          current individual is found
       End While
  End For
Start creating generations
Generation = 0
While (Generation < MaxGenerations)
  Select random parents from the current populatin
  Create FM individual as iterate Fiduccia Mattheyses algorithm until the best fitness
  value for the offspring is found
  While (fitness value of the offspring < fitness value of FM individual)
      Iterate Fiduccia Mattheyses algorithm until the best fitness value for the
      offspring is found
   End While
  Replace the worst individual from the old population with the offspring
  Increase Generation with 1
 End While
 Best partition = the last individual from the population
```

Fig. 3 Evolutionary approach- Pseudo code

the offspring. The mismatched genes are saved as their order is permutated. The first half of the mismatched genes goes to the first partition and the other half go to the other. In the end we get the new offspring. After that we execute the Fiduccia-

Mattheyses algorithm to the new offspring to find its best partition. In the end of the iteration we replaced the worst individual from the population (with the lowest fitness value) with the offspring. Than we receive the next generation.

We repeat this steps until the maximum generation is reached. After that we receive the best individual, derived from the population, as the optimal graph partition of graph G into k partitions.

IV. Experiments and Results

The evolutionary and the naive approaches for solving the graph partitioning problem was also practically implemented and the programming language that was used is Java. The implementation of the algorithms is used for comparison between them, because this is the main goal of this paper.

As an input value for the algorithms were used undirected, weighted, connected graphs with different number of vertices and edges between them. The conducted experiments are repeated several times for each approach and the differences between the execution time are insignificant. The execution time in the experiments will be presented in seconds and the memory usage in megabytes. The experiments will measure the coverage and the efficiency of the two approaches with the goal to prove, that the evolutionary approach can work with significantly larger graphs than the naive approach, works faster than it and uses less memory.

1) Experiment 1: The first experiment was done with relatively small graph that contains 17 vertices and 20 edges. This experiment has goal to show that there is a significant difference in the execution time and memory usage of the two approaches. The results are shown in the following graphic:

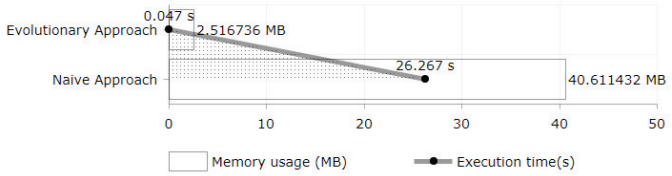

Fig. 4 Graph partitioning of graph with small number of vertices

As result of this experiment the both approaches output right partitions- there are no disconnected vertex in each partition. As we can see in the Fig.4 the execution time of the evolutionary approach takes an advantage over the naive approach with 28.22 seconds, which is 99% faster. The difference in memory usage is 38.09 megabytes-the evolutionary approach uses ~94% less memory than the naive approach. With this experiment we proved, that the evolutionary approach is more efficient than the naive approach: it is faster and uses less memory.

2) Experiment 2: The second experiment was done with a graph with large number of vertices and edges: 102 vertices and 101 edges and represents central cities in Russia. The goal of this experiment is to prove that the evolutionary approach can work with significantly larger graphs, but the naive approach cannot. The results are shown in the following graphic:

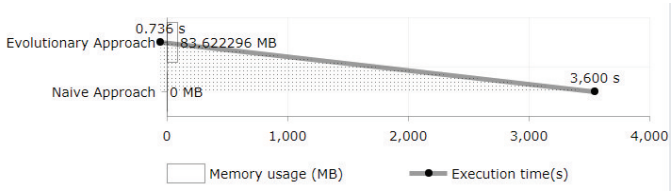

Fig. 5 Graph partitioning of graph with large number of vertices

In Fig.5 we can see that the evolutionary approach partition graph with large number of vertices for 0.736 seconds and uses 83 megabytes memory. It also outputs the right partitions- there are no disconnected vertex in each partition. The naive approach gives no results in this experiment. It was running for more than an hour (3600s) and there were no results. The reason is that there are a large number of combinations from graph vertices, which should be processed, so the naive approach can give appropriate results, and this will take a lot of time.

The experiments, which are described in this part, proved that the goal of this work is achieved. It was proved, that the evolutionary approach can work with significantly larger graphs than the naive approach, works faster than it and uses less memory.

V. Conclusion

The evolutionary approach can be easily adjusted to the graph partitioning problem. It has the advantages to change and customize any aspect of any given algorithm, which makes it more flexible and the progress is measured more easily. In this paper was presented a solution of the GPP using an evolutionary genetic approach combined with one of the first algorithms for GP-Fiduccia-Mattheyses. Different experiments were done, so it was proved that this approach is more efficient and has more coverage than the naive approach

Acknowledgements

I express my gratitude to Dr. Ventseslav Shopov and Dr. Vanya Markova for many valuable comments and help on a preliminary draft of this paper.

References

[1] C. Schulz, "*High quality graph partitioning",* Karlsruher Instituts für Technologie, 2013, pp 25-31.

[2] B. Kernighan and S. Lin. "*An efficient heuristic procedure for partitioning graphs*", The Bell System Tech J., The Bell System Technical Journal, February 1970.

[3] Fiduccia, C. M, Mattheyses. R. M., "*A linear-time heuristic for improving network partitions*." 19th Design Automation Conference, 14-16 June 1982.

[4] R. Wilson., *Introduction to graph theory*, Fourth edition, Addison Wesley Longman Limited, 1996, pp 8-25

[5] A. Buluç , H. Meyerhenke, I. Safro, P. Sanders, C. Schulz, "*Recent advances in graph partitioning*", Lawrence Berkeley National Laboratory, November 2016, pp 2-3.

[6] X. Yu, M. Gen, *Introduction to evolutionary algorithms*, Springer London Dordrecht Heidelberg New York, 2010, pp 8-26

[7] F. Streichert, "*Introduction to evolutionary algorithms*", University of Tuebingen, April 2002

[8] V. Markova, V. Shopov, "*Graph partitioning methods in social network analysis*", Bulgarian Academy of Sciences, Proceedings of the International Conference on Information Technologies (InfoTech-2016), 20-21 September 2016, Bulgaria