Crossover Operator for Frequent Subgraph Mining

Juan Beltrán

2012

Contents

List of Figures

List of Tables

Abstract

Graph-based data mining approaches have been mainly proposed to the task popularly known as frequent subgraph mining subject to a single user preference, like frequency, size, etc. In this work, I propose a new crossover operator for frequent subgraph mining problem, where a subgraph (or solution) is defined by a genetic algorithm through several iterations, reproductions and filtering. I have develop a standard genetic algorithm, which includes most of the used stages as selection, crossover (without mutation), evaluation and replacement. Evolutionary algorithm for Graph-base data mining approaches is a very recent field, and the genetic algorithms for frequent subgraph mining subject is introduced in this project, with the proposal of a new crossover operator. This project is based in the framework of Subdue algorithm for subgraph mining. The method is called optimization by genetic algorithms (GAOptimize) and has several advantages: (i) optimization from a Subdue's solutions stack in a single run (ii) selection of different constraints for substructure selection and reproduction (iii) search in the subgraphs lattice space and (iv) capability to deal with different isomorphic graph search algorithms. The good performance of GAOptimize is shown on two samples datasets from Subdue and two real-life datasets.

Keywords: Graph-based data mining, Frequent subgraph mining, Subdue, Genetic Algorithms, Evolutionary optimization, Crossover Operator.

Chapter 1

Introduction

Graph-based data mining (GBDM) has been prevalently used in a wide range of application domains, such as computing communities [1][2], subgraph discovery [3][4][5][6], topic detection [7], attack detection [8], computing the number of triangles [9], clustering [10][11], peta graph mining [12]. Recently GBDM has been recognized as one of the ten challenging problems in data mining research [13]. For the recent developments and comprehensive survey of this important and emerging topic, the reader is referred to [14][3].

GBDM approaches are characterized by the representation of multi-relational data in the form of graphs. They have been extensively applied to the task popularly known as frequent subgraph mining. These approaches can be categorized into mathematical graph theory-based approaches (such as, MoFa/MoSS [15], FSG [16], Gaston [17], gSpan [18], CloseGraph [19], gPrune [20]), greedy search-based approaches (like Subdue [21] and GBI [22]), and kernel function-based approaches [23]. All these approaches work by performing a search in the lattice of all possible subgraphs [24]. The underlying search process, which could either involve an exact exhaustive or approximate heuristic search, is usually guided by a single objective, which represents a unique and specific user preference.

The existing GBDM approaches applying such simple thresholds for frequent subgraph mining task have important limitations. For example, the number of mined subgraphs is large (respectively, few or nil) in the cases of weak (respectively, strict) thresholds [25]. Moreover, in real-life applications, a user is generally interested in mining a graph-based repository using several objectives that are actually meaningful to her/him, which are often conflicting in nature [25]. For example, users prefer obtaining subgraphs with both high frequency and size values. Nevertheless, these objectives are conflicting as simpler descriptions are usually the most frequent ones and vice-versa.

The Subdue framework [21] offers a substructure discovery algorithm, which performs data mining on databases represented as graphs. The system performs two key data-mining techniques: unsupervised pattern discovery and supervised concept learning from examples. Subdue discovers substructures searching (using a beam search algorithm) that compress the original data and represent structural concepts in the data.

The main idea of the new crossover operator proposed in this project, is to use a set of parents P to reproduce a set of children C considering the lattice space of P. This task is done merging the parent chromosomes and selecting consecutively a number of edges and vertices at random. As we show at final of this report, this new approach shows good results in the datasets used.

In the following section I will describe the related work with the respective objective of the project. Later, I will do a review of the methodology adopted to develop the method called GAOptimize. Finally, I will do a complete explanation of the algorithm, with some graphics, examples and results.

Chapter 2

Related work and contribution

Recent work in the data mining community has been focused on developing graph-based data approaches to discover subgraphs consisting of complex relationships between entities [14][21]. In this section, we briefly review some fundamental developments related to our work.

2.1 Related work

2.1.1 Gaston

GrAph Sequence Tree extractiON (GASTON) algorithm [17]. Given a database of graphs, a graph mining algorithm searches for substructures that satisfy constraints such as minimum frequency, minimum confidence, minimum interest and maximum frequency. Example of substructures include graphs, trees and paths, for these substructures many mining more efficient, GASTON is based on the “Quick-start principle”, exploiting the fact that the various substructures are contained in each other.

2.1.2 gSpan

Graph-based Substructure PAtterN mining (gSpan) [18], which discovers frequent substructures without candidate generation. gSpan builds a new lexicographic order among graphs, and maps each graph to a unique minimum DFS code as its canonical label. Based on this lexicographic order, gSpan adopts the depth-first search strategy to mine frequent connected subgraphs efficiently.

2.1.3 MoFa/MoSS

Mining MOlecular FrAgments (MoFa) [15] is an algorithm to find fragments sets of molecules that help to discriminate between different classes of, for instance, activity in a drug discovery context. Instead of carrying out a brute-force search, our method generates fragments by embedding them in all appropriate molecules in parallel and prunes the search tree based on a local order of the atoms and bonds, which results in substantially faster search by eliminating the need for frequent, computationally expensive re-embeddings and by suppressing redundant search.

2.1.4 FSG

Frequent SubGraph discovery algorithm (FSG) [16] for finding frequent subgraphs in large graph datasets. FSG starts by enumerating all frequent single and double edge subgraphs. Then, it enters its main computational phase, which consists of a main iteration loop. During each iteration, FSG first generates all candidate subgraphs whose size is greater than the previous frequent ones by one edge, and then counts the frequency for each of these candidates and prunes subgraphs that do no satisfy the support constraint. FSG stops when no frequent subgraphs are generated for a particular iteration.

2.1.5 gPrune

gPrune [20] algorithm proposes to incorporate all the constraints in such a way that they recursively reinforce each other through the entire mining process. Pattern-inseparable Data-antimonotonicity, is used to handle the structural constraints unique in the context of graph, which, combined with known pruning properties, provides a comprehensive and unified classification framework for structural constraints. The exploration of these antimonotonicities in the context of graph pattern mining is a significant extension to the known classification of constraints, and deepens our understanding of the pruning properties of structural graph constraints.

2.1.6 GBI

Graph-Based Induction (GBI) [22] extracts typical patterns from directed graph data by stepwise pair expansion, including tree structured data and multi-inputs/outputs nodes and loop structure which cannot be treated in the conventional way.

```
Subdue(Graph, BeamWidth, MaxBest, MaxSubSize, Limit)
    ParentList = {}
    ChildList = {}
    BestList = {}
    ProcessedSubs = 0
    Create a substructure from each unique vertex label and its single-vertex instances;
        insert the resulting substructures in ParentList
    while ProcessedSubs <= Limit and ParentList is not empty do
        while ParentList is not empty do
            Parent = RemoveHead(ParentList)
            Extend each instance of Parent in all possible ways
            Group the extended instances into Child substructures
            foreach Child do
                if SizeOf(Child) <= MaxSubSize then
                    Evaluate the Child
                    Insert Child in ChildList in order by value
                    if Length(ChildList) > BeamWidth then
                        Destroy the substructure at the end of ChildList
            ProcessedSubs = ProcessedSubs + 1
            Insert Parent in BestList in order by value
            if Length(BestList) > MaxBest then
                Destroy the substructure at the end of BestList
            Switch ParentList and ChildList
return BestList
```

Figure 2.1: Subdue's discovery algorithm [21]

2.1.7 Subdue

Subdue [21] is a GBDM method designed for different tasks as frequent subgraph mining, hierarchical clustering, and classification model building from relational data.

Subdue is an instance of greedy search-based approaches,which use heuristics to evaluate the subgraphs. It represents data in graph form and it can support either directed or unidrected edges. Input to Subdue is a single graph or a set of graphs. The framework of Subdue is explained in the figure 2.1.

In the algorithm, we have to distinguish a step very important for this project: `Evaluate the child`. The evaluation method is based on the MDL [26] principle. The MDL value of the subgraph p is given as:

$$MDL(G, p) = DL(p) + DL(G|p) \tag{2.1}$$

where $DL(p)$ is the description length of the subgraph p, and $DL(G|p)$ is the description length of the input graph G compressed by the subgraph p. Thus, we can conclude that the best subgraph has the smallest value of MDL measure.

```
Genetic Algorithm
    Input Paramteres (Pc, Pm, NumGen, PopSize, ... );
    t ← 0;
    Init(P(t));  // Initial Population
    Evaluate(P(t));  // Fitness funcion
    while ( t < NumGen) {
        t ← t+1;
        P'(t) = Select(P(t-1));  // Selection
        P''(t) = Alter(P'(t));  // Crossover and Mutation
        Evaluate( P''(t));  // Fitness function
        P(t) = Replace(P'(t), P''(t));  // Replacement
    }
end.
```

Figure 2.2: A conventional genetic algorithm [28]

2.1.8 Genetic Algorithms

Genetic Algorithms are search and optimization algorithms which are based on the mechanics of natural evolution, particularly on natural selection and genetic inheritance.

They were introduced by John Holland and his collaborators at the University of Michigan in the early 1970's [27]. A conventional genetic algorithm is shown in the figure 2.2.

2.1.9 GBDM and Evolutionary Algorithms

EP belogs to the same category of evoluonary computation as genetic algorithms (GAs) [28][29][30], the primary differences being (i) EP does not place any constraint on the representation, while GA usually requires the problem solutions to be encoded as strings, and (ii) EP does not use any sexual reproduction (or crossover), while crossover is one of the fundamental operators in GA. In EP only mutation is applied on the parent chromosomes to produce offspring [31]. The offspring are then evaluated in the same way as their parents. Subsequently the next generation is selected from collection of both the parents and the offspring. In this regard, EP is quite similar to evolutionary strategies (ES) [32], although the two approaches evolved independently.

So, the searching capability of evolutionary programming is utilized for discovering concepts or substructures that are often repeating in such structural data. So, the substructure discovery algorithm used by the EP-based technique, is shown in figure 2.3.

```
Algorithm EP-Substructure-Discovery(G,Pop-Size, Limit)
    ParentList = {}
    ChildList = {}
    ChromList = NewChromList = {}
    BestChromosome = Chromosome = {}
    Generations = 0
    Create a substructure from each unique vertex label and its single-vertex instances
    Insert the resulting substructures in ParentList
    while ParentList is not empty do
        Parent = RemoveHead( ParentList)
        Extend each instance of Parent in all possible ways
        Group extended instances into Child substructures
        foreach Child do
            Insert Child in ChildList
    for i = 1 to Pop-Size do
        Randomly select one substructure from ChildList
        Chromosome = pointer to this substructure
        Evaluate the substructure
        Assign the value to the fitness of Chromosome
        ChromList = ChromList + Chromosome
while Generations < Limit do
    foreach chromosome in ChromList do
        Mutate chromosome and append to ChromList
        Evaluate the mutated substructure
        Assign value to the fitness of chromosome
    Store the chromosome with maximum fitness in BestChromosome
    Introduce BestChromosome in NewChromList
    Select (Pop-Size - 1) chromosomes from ChromList and append to NewChromList
    Assign NewChromList to ChromList
    Generations = Generations + 1
return BestChromosome
```

Figure 2.3: EP-based discovery algorithm [31]

2.2 Own contribution

The crossover operator for frequent subgraph mining is a new concept, that I will try to introduce in this project. It begins from a binary representation of the parents to builds the offsprings, considering the lattice space of G. Each child reproduced will be reevaluated in the complete graph G, to reject it or admit it into the possible solutions of Subdue, by a fitness function called: **MDL** [26].

The original Subdue algorithm is wrote in ANSI C with a procedural paradigm of programming, and it offers a good performance for many different problems, that we can see in the user guide. The crossover operator and the genetic algorithm that I have designed, was attached as a new library into the main header *"subdue.h"*. In most cases the new implementation has good performance, as we show later.

2.3 Objective

Develop a Crossover Operator for Frequent Subgraph Mining. Given that there is not any research in Genetics Algorithms for Frequent Subgraph Mining, we will propose a new crossover operator which integrates the GA idea to frequent subgraph mining concept.

Chapter 3

Crossover Operator proposal

In this section, we will describe the application of the crossover operator into an evolutionary algorithm for optimal subgraph discovery, including its methodology description.

3.1 Genetic algorithms for Frequent Subgraph Mining

We have not found any specific reference of using Genetics Algorithms for Frequent Subgraph Mining, so, we have to apply the conventional genetic algorithm explained in the section 2.1.8 and the figure 2.2. Thus, we can distinguish two different phases: (i) Sentences before first cycle starts, and (ii) Sentences inside the while cycle. The inputs parameters, as the initial population will be all the solutions (substructures) of Subdue algorithm.

Every substructure of the initial population is evaluate by MDL (Minimun description length) index, described in section 2.1.7 and describes the level of information that contains the substructure from the initial graph. Thus, a greater MDL index means a greater level of information and a better quality of the chromosome.

The second phase includes the classic selection, crossover, mutation, evaluation and replacement operations. The selection method does the following steps:

k **separation** The classic GA requires chromosomes with the same size. Although, in our binary representation of the chromosome have the same size nxn, the graphical representation shows that two chromosomes does not need to have the same size, i.e., the same k number of nodes.

However, if the user wants to keep the same size for the parent chromosomes, then we will have to separate in l levels, the different substructures with the same k.

l **selection** If the user wants to keep the constraint mentioned in the preceding item, then we will have to select a specific level of k substructures. Given an uniform distribution such that $U\tilde{}(0,l)$, we select a level with a number of substructures greater than 2. This procedure will make a pool of similar substructures that we will use later, for select our parent chromosomes.

P **selection** Regardless of the user's decision, we will have to select the parent chromosomes for a pool of substructures, with the same k or not. Only the substructure which have instances that share at least one node between them, as the figure 3.1 shows, could be selected. So, if we have more than 2 substructures that have an instance which share at least one node, then we will have more than 2 parents chromosomes P for our later cross operation. This particular case we will denominate it as a coincidence. If we have m different coincidences, then we will select at random, under a uniform distribution $U\tilde{}(0,m)$ the couple of parent to use.

This selection process is just an adaptation of the classic GA to our project, to adjust it to the later crossover operator. Note that in this case we have preferred to let open the possibility of using more than 2 parents in any cross operation.

The transformation process includes the crossover and mutation operations, nevertheless, the crossover process requires a whole subsection to describe it, because it is our new proposal. The mutation operation will not cover in our project.

Once every offspring is generated, everyone are evaluated and inserted in the substructure list of the initial population. The evaluation and insertion are done in the same function, removing the worst results after $nSubs$' substructures. Therefore, we have always $nSubs$ substructures in every iteration of our model.

3.2 Crossover problem statement

The crossover operator problem is based on the idea of subgraph optimization, where a solution x is defined a subgraph C, a set of nodes and edges,

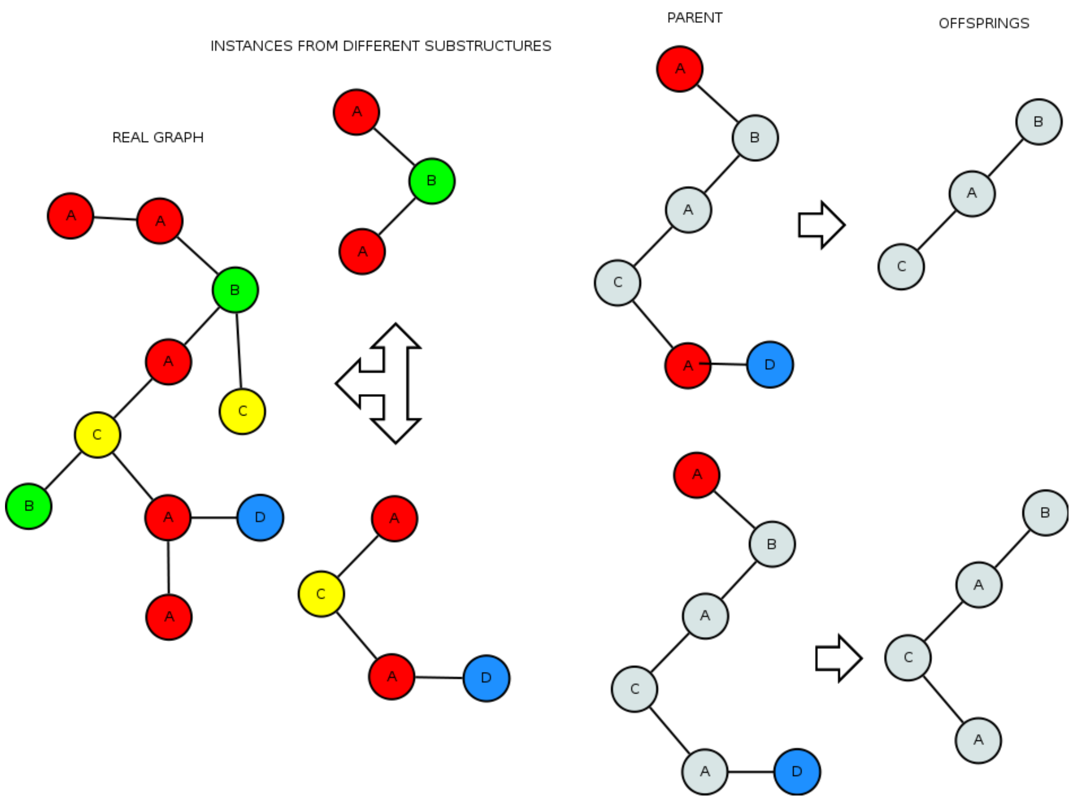

Figure 3.1: Example of the crossover operator.

the solution space is referred as the subgraph search space, i.e., the subgraph lattice.

The crossover method requires a special analysis, because we cannot only cross two chromosomes as the classic GA indicates. We have a set P of square binary matrices as bi-dimentional chromosomes, but if we make some offspring from the classic approach, just interchanging some percentage of everyone, then we may brake the structure of the initial graph G, making offspring that cannot correspond with the original graph.

To make children or offsprings that correspond with their original graph, we have to consider the lattice space of the initial graph G, which is built by the set of nodes and edges. This consideration is more easy to implement from a graphical representation of the problem, besides of the binary representation.

In this context, we cannot make chromosomes from the substructure, which is represented by a graph pattern, because we have to consider the local connections of every subgraph, represented by subdue algorithm as instances. This local connections is added to our problem as neigborhood problem, e.g., every instance to be crossed, from different substructure, must share at least one node as the figure 3.1 shows. Thus, a particular chromosome will be represented by just one instance, that represent, a particular subgraph of G.

3.3 Methodology

The current project is developed in the software engineer field, because we have to develop a computational algorithm enough stable to ensure the experiments showed at the final of the document. In this context, I have preferred an adaptation of the eXtreme Programming (XP) methodology [33], because it has good capabilities for developments in short times, with an acceptable quality of the solution. This methodology is explained in the following subsection.

3.3.1 Extreme Programming

Extreme Programming (XP)[33] is an "agile methodology" that some people advocate for the high-speed, volatile world of Internet and Web software development [34]. XP's critical underlying assumption is that developers can obviate the traditional high cost of change using technologies such as objects, patterns, and relational databases, resulting in a highly dynamic XP process.

The method that I have applied have the following basic elements (with some notes related to our decisions in this project):

Planning game Quickly determine the next release's scope. Thus, this documents describes our planning game, including the future work.

Small releases Put a simple system into production quickly. I have done continual implementations of GAOptimize method and its integration with Subdue.

Metaphor Guide all development with a simple, shared story of how the overall system works. Continually done, via e-mail (and resumed in this document).

Simple design Design as simply as possible at any given moment. I have try to do a model as simply as I can describe in this document.

Testing Developers continually write unit test that must run flawlessly.

Refactoring Restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility. Thus, although Subdues is development under an imperative programming paradigm, I have developed the GAOptimize method as a new module of subdue, very flexible, simple and with good quality.

Pair programming All production code is written by two programmers at one machine. I have omitted this topic.

Collective ownership Anyone can improve any system code anywhere at any time. I have omitted this topic, because It does not makes sense in this kind of project.

Continuous integration Integrate and build the system many times a day. This topic was strictly done every time of this project.

40-hour weeks Work no more than 40 hours per week whenever possible. I really does not apply this topic in our project.

On-site customer Have an actual user on the team full-time to answer questions. I have omitted this topic.

Coding standards Have rules that emphasize communication throughout the code. In this case, I have followed the same way of coding that Subdues uses.

3.4 Crossover operator method

Given a set of parents P which share at least one node, we proposes a new crossover operation, to make a set of offsprings C. The procedure is shown in the figure 3.1. The classic GA allows only 2 parents to make 2 offspring, but for our purposes, for taking advantage of the nature of the operation that we are proposing, we are letting open this possibility.

3.4.1 Chromosome representation

A substructure S is a graph pattern. Each substructure have its own isomorphic instances. Each instance is a unique sub-graph located into the initial graph. For the next purposes, we will consider the graph of this instance as a chromosome C.

A chromosome could have two basic representation. The first one is a graphic representation as the figure 3.1 and the second one is a binary square matrix of size n, where n is the number of nodes of the initial graph G.

3.4.2 Crossover operator proposal

In this section we will attempt to describe my contribution of a new crossover operator, listed in the figure 3.2. From the figure 2.2 I have called `Alter(P´(t))` to the method which will contain the core of the crossover operator. Thus, we can distinguish two different cycles to control the parent selection and the children reproduction. Before execute each second cycle, we have to covert to binary (subsection 3.4.1) each graph parent (as a result from the procedure described in subsection 2.1.8).

Afterward, inside the second cycle, we will reproduce our next offspring and we will convert it to the graph format of Subdue, later. The result will be added to the instance list of children. After all cycles, the children list is filtered to remove all repeated offspring.

As we know from the previous subsection, a substructure is a subgraph pattern into the lattice space of the initial set of graph G. To make all conversions we have to consider each instance as an independent pattern, e.g., a Substructure. Nevertheless, to estimate the physical representation, e.g., the instances of the new substructure, we will have to apply an isomorphic search into G to find them. Specifically, I have used the NP-hard method provided by Subdue called `FindInstances`.

```
Alter(P'(t))
    t ← 0;
    while ( t < NumMergedParents ) {
        t ← t+1;
        P'(t) = ConvertGraphToBinary(P'(t));
        c ← 0;
        while ( c < NumChildrenToReproduce ) {
            c ← c+1;
            offspring = ReproduceNewChild(P'(t));
            offspring = ConvertBinaryToGraph(offspring);
            InsertInstanceToList(listChild,offspring);
        }
    }
    listChild=TwinFilter(listChild);
    SubOffspring=InstanceToSubstructue(listChild);
return SubOffspring;
```

Figure 3.2: Overview crossover algorithm

Chapter 4

Experimental Study

In this chapter I will explain the experimental framework to evaluate the algorithm proposed.

Note: The GAOptimizer has been implemented in ANSI C, and all experimetns have been performed on an Intel Core i3 at 2.33 GHz (using only one core), with 4GB RAM, running Debian 6.0. with linux kernel 2.6.32-5-amd64.

4.1 Dataset description

The performance evaluation study has been conducted in our experiments on 4 datasets, which are summarized as follows:

Carbon

1. Name: Carbon
2. Type: Chemical
3. Number of vertices: 79
4. Number of edges: 90
5. Number of labels: 2
6. Author: Subdue Source Code

Subdue

1. Name: Sample
2. Type: Subdue Example
3. Number of vertices: 20
4. Number of edges: 19
5. Number of labels: 7
6. Author: Subdue Source Code

Pbd305d

1. Name: Pbd305d
2. Type: Protein Data Bank
3. Number of vertices: 159
4. Number of edges: 178
5. Number of labels: 5
6. Author: Protein Data Bank at http://www.rcsb.org/pdb

Pbd105d

1. Name: Pbd105d
2. Type: Protein Data Bank
3. Number of vertices: 364
4. Number of edges: 435
5. Number of labels: 9
6. Author: Protein Data Bank at http://www.rcsb.org/pdb

4.2 Parameter setting

Subdue and GAOptimizer methods have been run with three different values of *nsubs*=10, 20, and 40. Each of these methods have been run for 1000 generations. A single execution of Subdue and GAOptimizer has been carried out on the input graph datasets as a consequence of being deterministic methods. The parameters used in the testing framework are listed in table 4.1.

4.3 Performance evaluation

To evaluate the performance of the proposed Crossover Operator, we have used the following statistical measures: (i) Population mean (ii) Population variance, and (iii) Maximum value. Thus, Mean and Variance will help us to determine the behavior of the algorithm in our population, and the maximum value will determine the real effectiveness of the algorithm.

4.4 Analysis of results

We have executed Subdue with GAOptimize method using the previous parameters, and the results are shown in the table 4.2. .

Parameter	Value
Predefined substructure file	none
Output file	none
Beam width	4
Compress	false
Evaluation method	MDL
'e' edges directed	true
Incremental	false
Iterations	1
Limit	45
Minimum size of substructures	1
Maximum size of substructures	79
Number of best substructures	**(10,20,40)**
Output level	2
Allow overlapping instances	false
Prune	false
Threshold	0
Value-based queue	false
Recursion	false
gaOptimize	**true**
numIterations	**1000**
limitKNodes	false
numParents	2
lowerOffset	0

Table 4.1: Parameters of the testing framework

Dataset		Carbon.g					
Method		Subdue			GAOptimal		
nsubs	Generations	Mean	Var	Max	Mean	Var	Max
10	100	2.088357	0.141333	2.94186	2.262758	0.090199	2.94186
20	100	1.777807	0.169882	2.94186	1.960161	0.281191	3.644603
40	100	1.530214	0.148744	2.94186	1.729228	0.244242	3.644603
10	1000	2.088357	0.141333	2.94186	2.262758	0.090199	2.94186
20	1000	1.777807	0.169882	2.94186	2.059268	0.267798	3.644603
40	1000	1.530214	0.148744	2.94186	3.644603	0.244242	3.644603
Dataset		Sample.g					
Method		Subdue			GAOptimal		
nsubs	Generations	Mean	Var	Max	Mean	Var	Max
10	100	1.190461	0.073411	1.868188	1.190461	0.073411	1.868188
20	100	1.09348	0.056636	1.868188	1.09348	0.056636	1.868188
40	100	1.09348	0.056636	1.868188	1.09348	0.056636	1.868188
10	1000	1.190461	0.073411	1.868188	1.190461	0.073411	1.868188
20	1000	1.09348	0.056636	1.868188	1.09348	0.056636	1.868188
40	1000	1.09348	0.056636	1.868188	1.09348	0.056636	1.868188
Dataset		Pbd305d					
Method		Subdue			GAOptimal		
nsubs	Generations	Mean	Var	Max	Mean	Var	Max
10	100	2.068499	0.003686	2.168358	2.074885	0.003297	2.168358
20	100	1.894315	0.017469	2.168358	1.941247	0.008514	2.168358
40	100	1.808214	0.018276	2.168358	1.881699	0.009644	2.168358
10	1000	2.068499	0.003686	2.168358	2.086963	0.003376	2.168358
20	1000	1.894315	0.017469	2.168358	2.06169	0.003208	2.168358
40	1000	1.808214	0.018276	2.168358	2.036387	0.003433	2.219187
Dataset		Pbd105d					
Method		Subdue			GAOptimal		
nsubs	Generations	Mean	Var	Max	Mean	Var	Max
10	100	2.565378	0.029952	2.912358	2.565378	0.029952	2.912358
20	100	2.389825	0.046658	2.912358	2.392942	0.045401	2.912358
40	100	2.213252	0.054211	2.912358	2.270751	0.037985	2.912358
10	1000	2.565378	0.029952	2.912358	2.565378	0.029952	2.912358
20	1000	2.389825	0.046658	2.912358	2.399672	0.042944	2.912358
40	1000	2.213252	0.054211	2.912358	2.351047	0.022966	2.912358

Table 4.2: Summarized results comparison

In figures 4.1, 4.2, 4.3 and 4.4, we can observe an exponential increasing of the means of every graph, except in graph *Sample*, when the iterations or generations are also increasing. The graph *Sample* shows a semi-constant behavior, because there are a small number of edges and nodes, so the values when *nsubs* are 20 and 40 are equal, an very similar to *nsubs* 10. In general, we can see a good behavior through iterations increase.

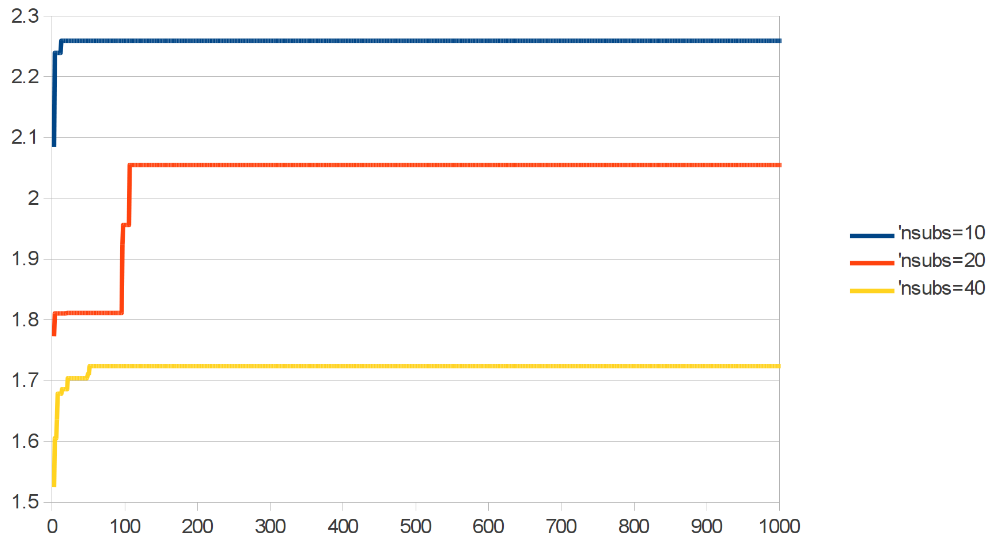

Figure 4.1: Curve of means in graph: *Carbon*.

In figures 4.5 4.6 4.7 and 4.8 we can see the variance behavior throughout the generations of the genetic algorithm. Thus, we can observe a good behavior with curves which are decreasing till get semi-constant. Of course, when *nsubs* is lower, then the variance is lower, because in each iterations the algorithm is considering only the best *nsubs* substructures. The figure 4.6 shows a semi-constant behavior, because the graph *Sample* has few nodes and edges to represent big variations in chromosomes throughout generations.

Around figures 4.9 4.10 4.11 and 4.12 we can observe the behavior of maximum values found throughout iterations of GA. In almost cases we have found a semi-constant curves, except in graphs *Carbon* and *pdb305d*, which show a slight increment. The lattice space of a graph limits the number of optimal solutions or substructues, so, we can consider this slight increment as a very good result.

In the previous figures, we can observe variations in the variance and the mean curves, when a newer best solution is found by the algorithm. In the

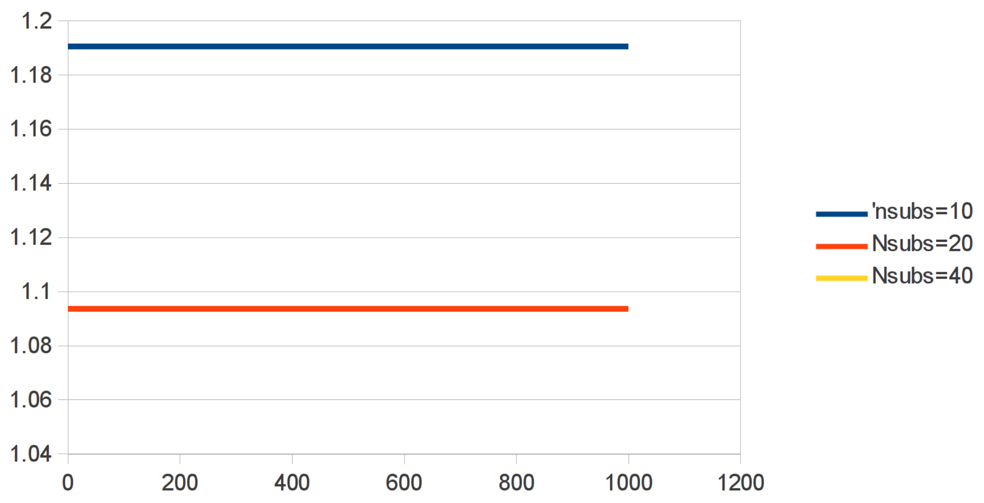

Figure 4.2: Curve of means in graph: *Sample*.

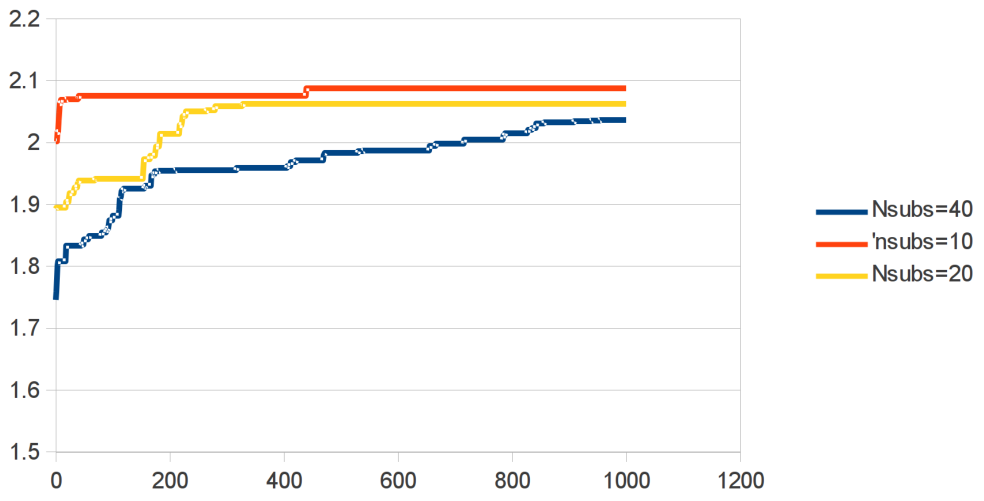

Figure 4.3: Curve of means in graph: *Pbd305d*.

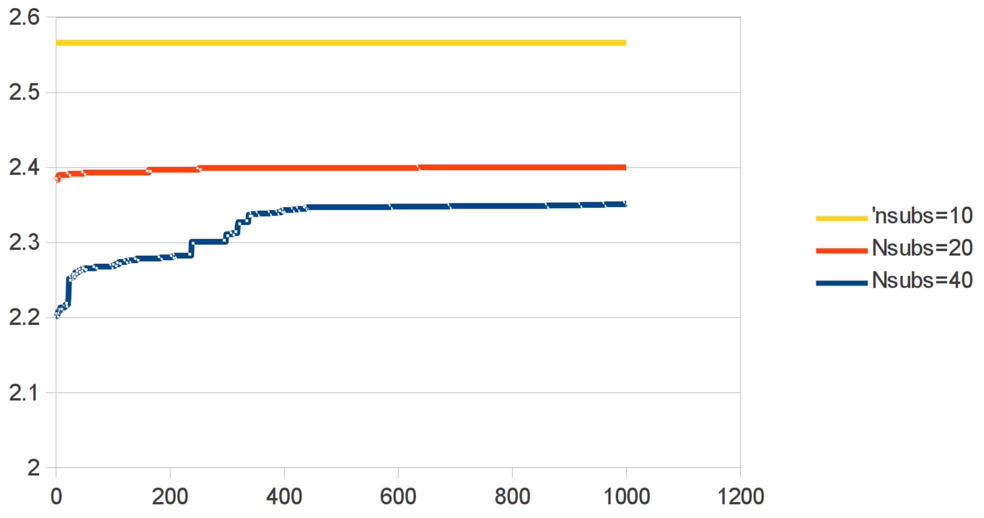

Figure 4.4: Curve of means in graph: *Pbd105d*.

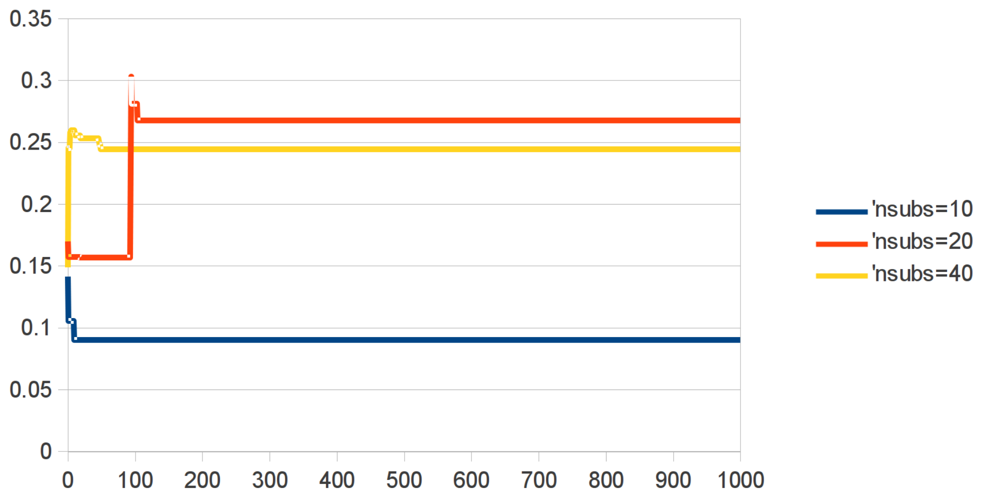

Figure 4.5: Curve of variances in graph: *Carbon*.

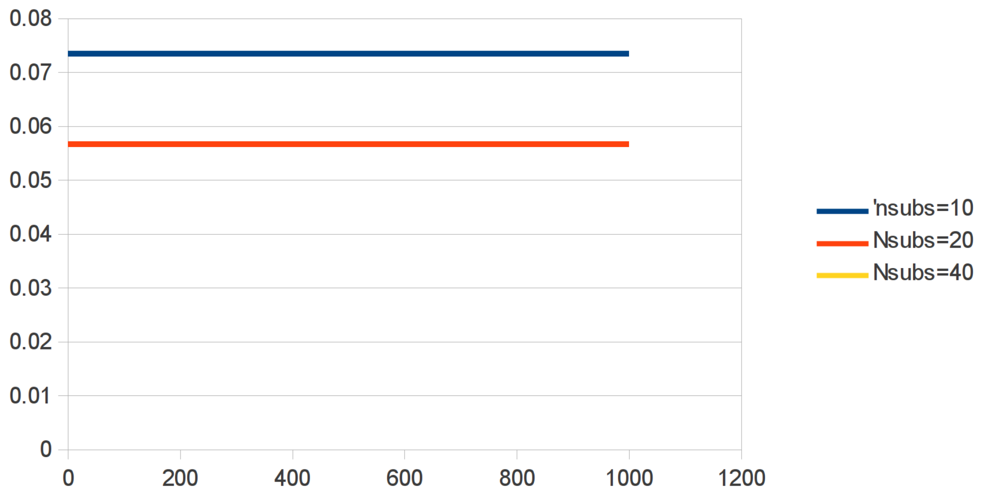

Figure 4.6: Curve of variances in graph: *Sample*.

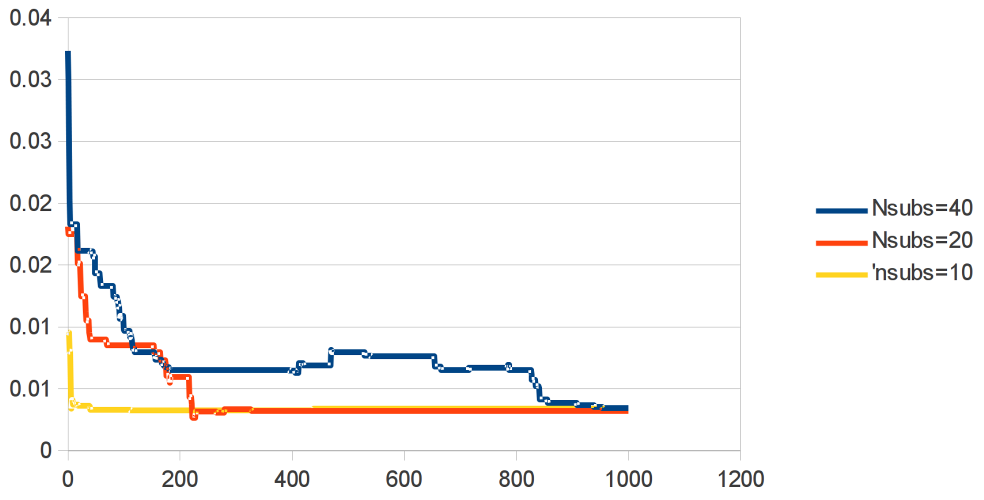

Figure 4.7: Curve of variances in graph: *Pbd305d*.

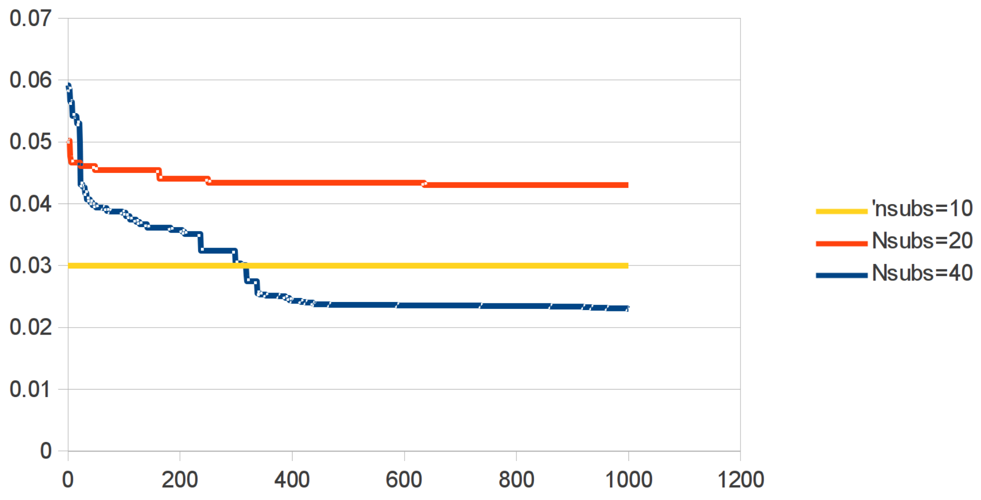

Figure 4.8: Curve of variances in graph: *Pbd105d*.

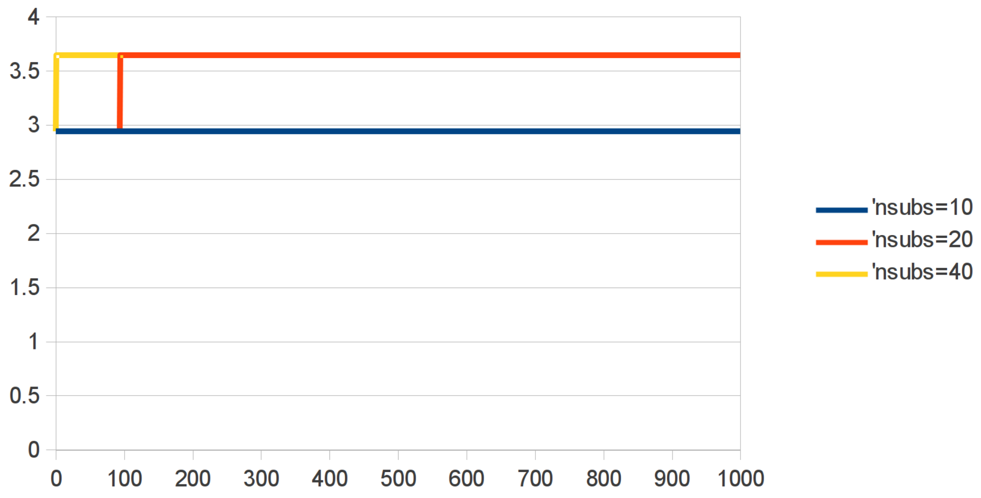

Figure 4.9: Curve of maximum values in graph: *Carbon*.

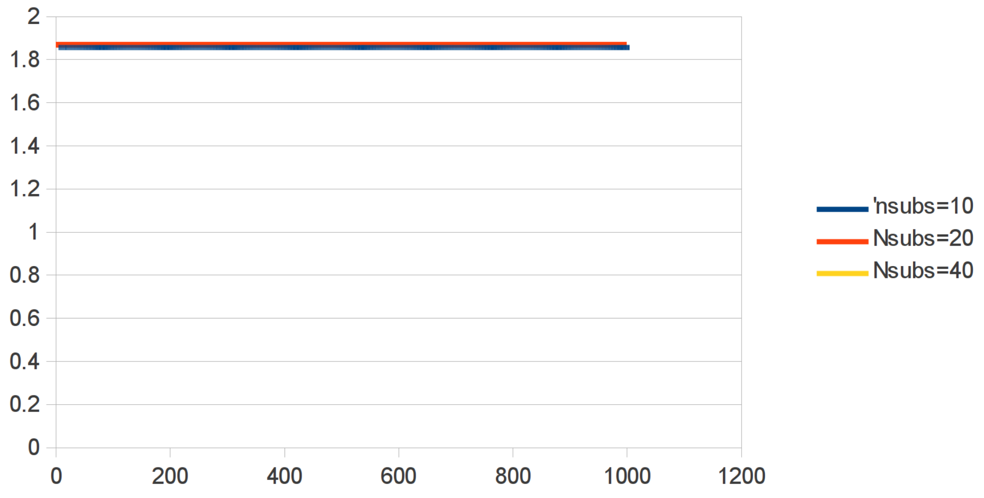

Figure 4.10: Curve of maximum values in graph: *Sample*.

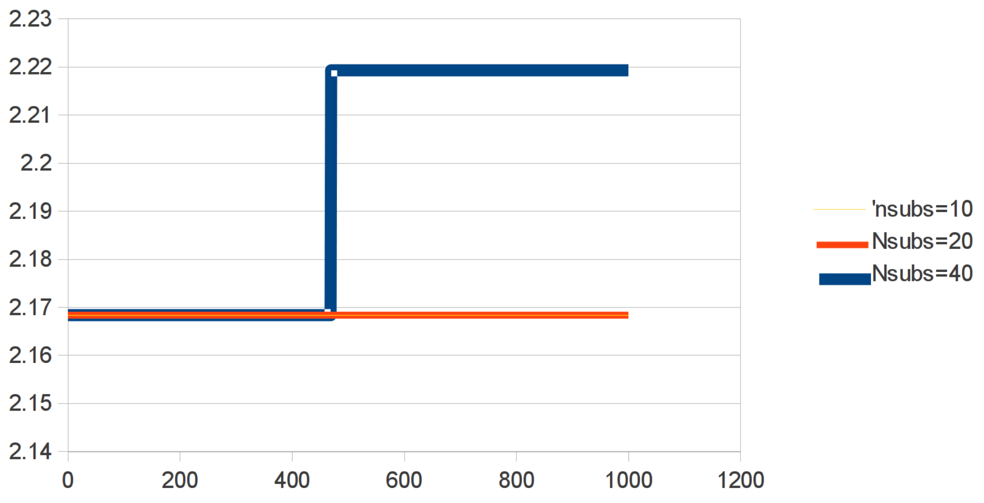

Figure 4.11: Curve of maximum values in graph: *Pbd305d*.

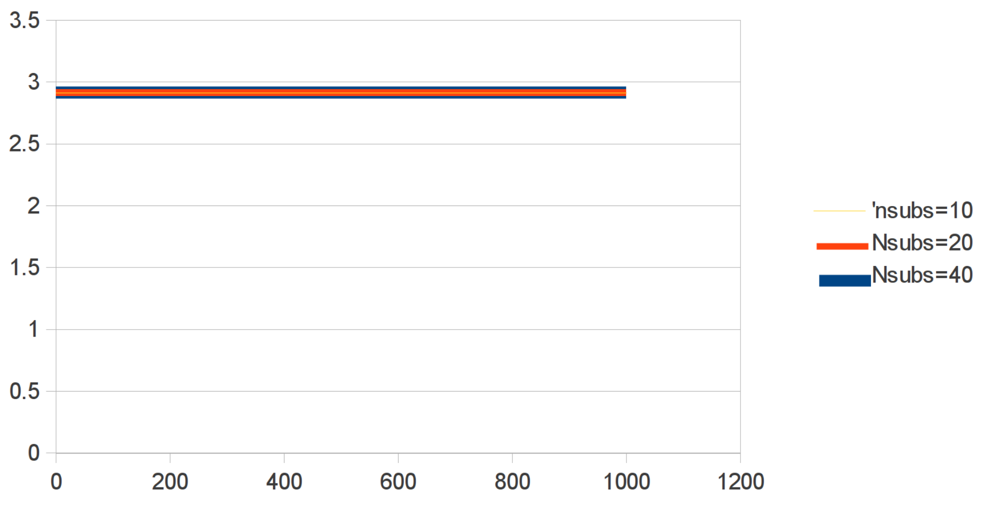

Figure 4.12: Curve of maximum values in graph: *Pbd105d*.

same way, we can observe in the real datasets a better convergence of the algorithm in the variance of population when *nsubs* is greater.

Chapter 5

Concluding remarks and future work

5.1 Conclusion

I have proposed the use of Genetics Algorithms for optimize the search strategy for subgraph mining in relational graph databases. The approach has been customized using the Subdue algorithm and has been called as GAOptimize (Genetic Algorithm Optimization). Two different variants on the selection of chromosomes also were proposed.

A development methodology was successfully applied to this project, making quickly the algorithm proposed. Although, we have proposed only a Cross Operator, we had to do special functions for selection and replacement, offering to the user options in order to improve the results in an specific problem.

The performance of GAOptimize has been analyzed using two real-world datasets and two sample datasets (from the source code of Subdue). From the obtained results, we have found that GAOptimize is able to **discover new optimal substructures** or solutions in a single run.

5.2 Future work

Several ideas for future developments arise from this work. On the one hand, at the final of the crossover operator, the algorithm uses the method of Subdue called *FindInstances* to do an isomorphic search of the chromosome reproduced into the set graph G to find the instances for the new substructure. This method use a NP-Hard method which use a modified version of the beam search of Subdue, to estimate any possible isomorphic subgraph equal to the chromosome. This search is computationally expensive for large graphs, and it would be very useful use other methods to allow use the good capabilities of GAOptimize for these kind of graphs.

The crossover operator was designed to be applied directly on instances of substructures that shares at least one node. So, we have two big future challenges. On the first hand, the crossover operator discards all non-overlapped instances. And, on the second hand, the crossover operator considers as parent chromosomes the instances objects, the real location of a graph pattern into G. So, a crossover operator in the abstraction level of substructures, has to consider the inclusion of all overlapped and non-overlapped instances.

The values used to evaluate the performance of the crossover operator only gives an idea of the progress of the algorithm, and some of them (like mean and variance) has not empirical application known, so, it could be interesting find any practical application for those evaluators.

In future realses of Subdue, would be very good see development in different paradigm of programming, like Object-Oriented Programming (in C++), because it would make easier the implementation of algorithms or modules, as we have developed.

Bibliography

[1] Falkowski T, Barth A, Spiliopoulou M *"Dengraph: a density-based community detection algorithm."*. 2007. IEEE/WIC/ACM Int Conf Web Intelligence, pp 112–115. IEEE Computer Society, Los Alamitos.

[2] Narasimhamurthy A, Greene D, Hurley N, Cunningham P *"Partitioning large networks without breaking communities."*. 2010. Know Inf Syst 25:345–369.

[3] Cook DJ, Holder LB *"Mining graph data."*. 2007. Wiley, London.

[4] Ranu S, Singh AK *"Graphsig: a scalable approach to mining significant subgraphs in large graph databases."*. 2009. Proceeding of 25th Int Conf Data Engg (ICDE'09), pp 844–855, IEEE.

[5] Yan X, Han J. *"gSpan: graph-based substructure pattern mining"*. 2002. Proceeding of IEEE Int Conf Data Min (ICDM'02), pp 721–724

[6] Zhu F, Yan X, Han J, Yu PS *"gPrune: a constraint pushing framework for graph pattern mining"*. 2007. Proceeding of PAKDD Conference, pp 388–400.

[7] Qian T, Srivastava J, Peng Z, Sheu P *"Simultaneously finding fundamental articles and new topics using a community tracking method"*. 2009. Thanaruk T, Boonserm K, Nick C, Tu-Bao H (eds) Advances in knowledge discovery and data mining, vol 5476 of Lecture Notes in Computer Science. Springer, Berlin, pp 796–803

[8] Shrivastava N, Majumder A, Rastogi R *"Mining (social) network graphs to detect random link attacks"*. 2008. IEEE 24th Int Conf Data Eng (ICDE'08), pp 486–495.

[9] Tsourakakis C *"Counting triangles in real-world networks using projections"*. 2011. Knowl Inf Syst 26(3):501–520.

[10] Long B, Zhang Z, Yu P *"A general framework for relation graph clustering"*. 2010. Knowl Inf Syst 24(3):393–413.

[11] Peng W, Li T *"Temporal relation co-clustering on directional social network and author topic evolution."*. 2011. Knowl Inf Syst 26(3):467–486.

[12] Kang U, Tsourakakis C, Faloutsos C *"Pegasus: mining peta-scale graphs"*. 2011. Know Inf Syst 27:303–325

[13] Yang Q, Wu X *"10 challenging problems in data mining research"*. 2006. Int J Inf Tech Decis 5:597–604.

[14] Aggarwal C, Wang H (eds) *"Managing and mining graph data series"*. 2010. Springer, Berlin.

[15] Borgelt C, Berthold MR *"Mining molecular fragments: finding relevant substructures of molecules"*. 2002. Proceeding of IEEE Int Conf Data Min (ICDM'02), pp 51–58.

[16] Kuramochi M, Karypis G *"An efficient algorithm for discovering frequent subgraphs"*. 2004. IEEE Trans Knowl Data Eng 16:1038–1051.

[17] Nijssen S, Kok JN *"A quickstart in frequent structure mining can make a difference"*. 2004. Proceeding of 10th ACM SIGKDD Int Conf Knowl Disc & Data Min (KDD'04), pp 647–652.

[18] Yan X, Han J *"gSpan: graph-based substructure pattern mining"*. 2002. Proceeding of IEEE Int Conf Data Min (ICDM'02), pp 721–724.

[19] Yan X, Han J *"CloseGraph: mining closed frequent graph patterns"*. 2003. Proceeding of 9th ACM SIGKDD Int Conf Knowl Disc & Data Min (KDD'03), pp 286–295.

[20] Zhu F, Yan X, Han J, Yu PS *"gPrune: a constraint pushing framework for graph pattern mining"*. 2007. Proceeding of PAKDD Conference, pp 388–400.

[21] Cook DJ, Holder LB *"Graph-based data mining"*. 2000. IEEE Intell Syst 15:32–41.

[22] Matsuda T, Horiuchi T, Motoda H, Washio T *"Extension of graph-based induction for general graph structured data"*. 2000. Terano T, Liu H, Chen ALP (eds) Proceeding of 4th Pacific-Asia Conf Know Dis Data Mining (PAKDD'00), volume 1805 of Lecture Notes in Computer Science, pp 420–431. Springer, Berlin.

[23] Kondor RI, Lafferty JD *"Diffusion kernels on graphs and other discrete input spaces"*. 2002. Proceeding of 19th Int Conf Machine Learning, (ICML'02), pp 315–322.

[24] Fischer I, Meinl T *"Graph based molecular data mining—an overview"*. 2004. Thissen W, Wieringa P, Pantic M, Ludema M (eds) IEEE Int Conf Syst Man Cy vol 76, pp 4578–4582.

[25] Papadopoulos AN, Lyritsis A, Manolopoulos Y *"SkyGraph: an algorithm for important subgraph discovery in relational graphs"*. 2008. Data Min Knowl Disc 17:57–76.

[26] Rissanen J *"Stochastic complexity in statistical inquiry theory"*. 1989. World Scientific Publishing Co Inc, River Edge.

[27] JH Holland *"Adaptation in natural and artificial systems"*. 1975. University of Michigan press.

[28] DE Goldberg *"Genetic algorithms in search, optimization, and machine learning"*. 1989. Addison-wesley.

[29] Michalewiez, Z. *"Genetic Algorithms + Data Structures = Evolution Programs"*. 1992. New York: Springer-Verlag.

[30] Mitchell, M. *"An Introduction to Genetic Algorithms"*. 1996. Cambridge, MA: MIT Press.

[31] Sanghamitra Bandyopadhyay, Ujjwal Maulik, Diane J. Cook, Lawrance B. Holder, and Yousuf Ajmerwala. *"Enhancing Structure Discovery for Data Mining in Graphical Databases Using Evolutionary Programming"*. 2002. Department of Computer Science Engineering. University of Texas at Arlington.

[32] Schwefel, H. P. *"Collective phaenomena in evolutionary systems"*. 1987. In 31st Annual Meeting of the International Society for General Systems Research. 1025–1033.

[33] K. Beck *"Extreme Programming Explained: Embrace Change"*. 1999. Addison-Wesley, Reading, Mass.

[34] Mark C. Paulk *"Extreme Programming from a CMM Perspective"*. 2001. Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA. 18 , Issue: 6.