UNIVERSITY OF BORDEAUX 1

Report for Study and Reseacch Course
MASTER OF SOFTWARE ENGINEERING (2011-2012)

Implementation of a polynomial time algorithm to determine relative clique-width of a graph

---

Author:
NGUYEN Huu Phat
LE Xuan Hoan

Supervisor:
Professor Bruno COURCELLE
Professor Maylis DELEST

Acknowledgements

We would like to say many thank to Professor Bruno COURCELLE who raise this project for us, a graph decomposition and its related terms does not similar for us before, but after this research project, we have some knowledge about it.

It is a pleasure thank to Professor Maylis DELEST- who teach us how to plan a research and write a professional report.

We also thank our classmates for their encouragements during our project.

Abstract

Many NP-hard problems are solvable in polynomial time for graphs having a bounded tree width, and correspondingly, having a bounded clique width [1]. Vadim Lozin and Dieter Rautenbach in [1] introduced a polynomial time factor 2 approximation algorithm to determine relative clique width of a graph. And more interestingly this algorithm determines exactly relative clique width with respect to a linear reduced term.

This report is about the implementation of this algorithm for the case of linear reduced terms, as well as about the program we developed to support a visual usage for users.

Contents

Chapter 1

Introduction

Clique width is an important graph parameter in the meaning that many NP-hard problems are solvable for graphs with bounded clique width. However calculating clique width of a graph is NP-Complete, it's related to determining the minimum number of labels used in constructing the graph by operations. [1] took a new approach, it dealt with another problem, determining relative clique width. In fact clique width of a graph is the minimum of its relative clique widths.

Although the author of [1] confirmed that the problem is NP-hard, an polynomial time approximation algorithm was introduced for the problem. And moreover the algorithm computes exactly the relative clique width in a special case.

For the next sections, we will have a brief consolidation about clique width and relative clique width, a detail presentation about the algorithm with a practical example. And then we will introduce about our implementation for this algorithm

Chapter 2

Definition and Objective

Contents

Hereafter we introduce necessary definitions and notations used in this report, then we will present the mentioned algorithm.

2.1 Clique width

Clique width is an important graph parameter in the meaning that many NP-hard proles are solvable for graphs with bounded clique width. Clique width of a graph G(V,E) is defined via labeling vertices of graph and constructing graph through operations.

Definition (labeling a vertex) C is a finite set, a labeled vertex $v \in V$ is denoted by i(v) with $i \in C$, we also call i(v) a-port

Then we can have a mapping:

$$\gamma : V \to C$$

There are 3 kinds of operations:

1. Relabeling: denoted by symbol $\rho_{a \to b}$: makes all vertices labeled by a become labeled by b
2. Edge addition: denoted by symbol $\eta_{a,b}$: creates edges between vertices labeled by a and vertices labeled by b
3. Disjoint union: denoted by symbol $\oplus$: disjoint union is commutative

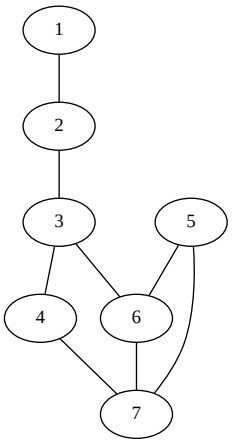
The process of constructing graph is denoted as an expression of labeled vertices and these three kinds of operations. We call such an expression a term.

Example We use an example from [1], we have a following term

$$t = \eta_{b,c}(((\rho_{b\to a}(\eta_{b,c}((\eta_{a,b}(a(1)\oplus b(2)))\oplus c(3))))\oplus b(4))\oplus(\eta_{a,c}(\eta_{a,b}(a(5)\oplus b(6)))\oplus c(7)))$$

This term denotes following graphs.

Figure 2.1:



Because there may be more than one way to construct a graph, therefore we can have many terms t that denote the same graph (two graphs are considered the same if they are isomorphic). For such a term t, we define $val(t) = (G, \gamma)$

Definition (Clique width) Clique width of a graph G cw(G) is defined as follows:

$$cw(G) = min\{|\ C\ ||\ \exists t : val(t) = (G, \gamma)\}$$

Calculating clique width of a graph is an NP-complete problem.

2.2 Relative clique width

Vadim Lozin and Dieter Rautenbach in [1] made a new approach to the problem of determining clique width by introducing the notion of relative clique width.

Definition (Reduced term) a reduced term r of a term t is a term acquired by replacing in t all symbols i(v) by v, and removing all symbols $\rho_{a\to b}$and $\eta_{a,b}$

Example for a term t as above, we have the reduced term r as follows:

$$r = red(t) = (((((((v_1\oplus v_2))\oplus v_3)))\oplus v_4)\oplus(((v_5\oplus v_6))\oplus v_7)) = (((1\oplus 2)\oplus 3)\oplus v_4)\oplus((v_5\oplus v_6)\oplus v_7)$$

A reduced term denoted a rooted binary tree

Definition (relative clique width) relative clique width of a graph G with respect to a reduced term r, cw(G,r) is defined as follow:

$$cw(G, r) = min\{|\ C\ ||\ \exists t : val(t) = (G, \gamma) \wedge red(t) = r\}$$

[1] introduced a factor 2 algorithm to approximate cw(G,r). And an interesting thing is the algorithm can determine exactly in a special case, the case of linear reduced terms

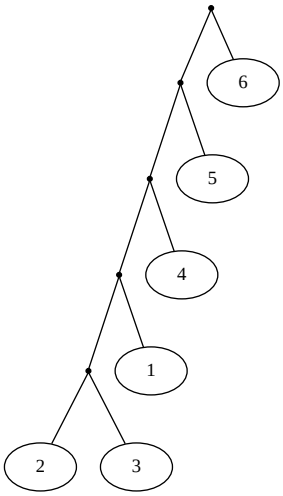
Definition (linear reduced term) a reduced term r is linear iff: $\forall$ sub-term $s_1 \oplus s_2$ of r, $\exists i \in \{1, 2\} : s_i$ has form of v

Example following is an example of a linear reduced term

$$r = (((1 \oplus (2 \oplus 3)) \oplus 4) \oplus 5) \oplus 6$$

This term denotes following rooted binary tree:

Figure 2.2:



2.3 $GraphH_s$

Following is a very important notation used in the algorithm, but firstly let us define $N_G(u)$ to be the neighborhood of vertex u in graph G

Definition For a sub-term s of r, let V_s denote the set of vertices in s. We define a graph $H_s = (V_s, E_s)$ with $uv \in E_s \iff N_G(u) \setminus V_s \neq N_G(v) \setminus V_s$

2.4 Algorithm

This algorithm determines cw(G, r) by recursively constructing a term t such that t satisfies definition of cw(G, r), after we have t, determining cw(G, r) is trivial. Let s is a

sub-term of r, we construct t(r) recursively as following Algorithm 1:

Algorithm 1 Relative clique determining algorithm

1. If $s = v \in V$, then $t(s) = i(v), i \in C$
2. If $s = s_1 \oplus s_2$, then $t(s) = \beta_l(\beta_{l-1}(...(\beta_1(\alpha_k(\alpha_{k-1}(...(\alpha_1(t(s_1) \oplus t(s_2)))))))))$, with $\alpha_1, ..., \alpha_k$ have form of $\eta_{i,j}$ and $\beta_1, ..., \beta_l$ have form of $\rho_{i \to j}$ such that following conditions must be satisfied
   (a) $val(t(s)) = (G[Vs], \gamma_{t(s)})$
   (b) red(t(s)) = s
   (c) $\forall u, v \in V_s, \gamma_{t(s)}(u) = \gamma_{t(s)}(v) \Leftrightarrow uv \notin E_s$, it means $\gamma_{t(s)}$assigns the same label to two vertices iff they stay in the same partite set of H_s
   (d) $\gamma_{t(s_1)} \cap \gamma_{t(s_2)} = \emptyset$

With condition (c), we need to determine partite sets of H_s. Fortunately, we can have an algorithm to do that from Lemma 2 in [1]:

> If $s = s_1 \oplus s_2$ is a sub-term of r, then every partite sets of $H_s$is a union of partite sets of $H_{s_1}$and H_{s_2}

Call $U_{s_1}$and $U_{s_2}$is sets of partite set of $H_{s_1}$and H_{s_2}, we have Algorithm 2 as follows:

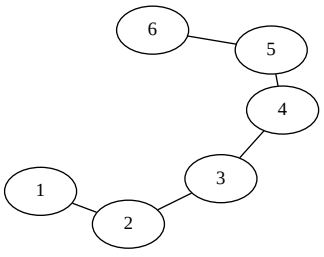
Algorithm 2 Partite sets determining algorithm

2.5 A practical example

To make an example, we will run manually the relative clique determining algorithm with following input:

- A Graph with 6 vertices

Figure 2.3:



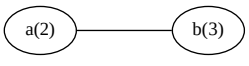
- Linear reduced term $r = (((1 \oplus (2 \oplus 3)) \oplus 4) \oplus 5) \oplus 6$

- Set of label {'a', ..., 'z'}

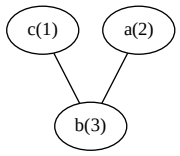
It's easy to see that actually the algorithm does the recursion on the structure of s (a tree-like structure). However to keep the illustration for this algorithm clear and simple, instead of going top-down from the root of the tree, we will describe the illustration directly, bottom up from the deepest leaves (see an example of a rooted binary tree above).

1. s = 2
   - We have: a(2)
   - V_s= {2}
   - H_s= 

   - t(s) = a(2)
   - U(H_s, 1) = {a(2)}
   - $\gamma_{t(s)}(V_s) = \{a\}$
2. s = 3
   - We have: b(3)
   - V_s= {3}
   - H_s= 

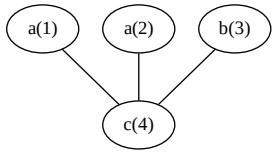
   - t(s) = b(3)
   - U(H_s, 1) = {b(3)}
   - $\gamma_{t(s)}(V_s) = \{b\}$
3. $s = 2 \oplus 3$
   - We have: $a(2) \oplus b(3)$
   - Apply (a) $\Longrightarrow \eta_{a,b}(a(2) \oplus b(3))$
   - V_s= {2, 3}
   - H_s= 

   - U(H_s, 1) = {a(2)}, U(H_s, 2) = {b(3)}
   - $t(s) = \eta_{a,b}(a(2) \oplus b(3))$
   - $\gamma_{t(s)}(V_s) = \{a, b\}$
4. $s = 1 \oplus (2 \oplus 3)$

- We have: $c(1) \oplus \eta_{a,b}(a(2) \oplus b(3))$
- Apply (2) $\Longrightarrow \eta_{c,a}(c(1) \oplus \eta_{a,b}(a(2) \oplus b(3)))$
- V_s= {1, 2, 3}
- H_s = 

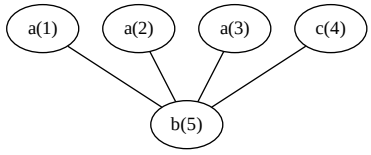
- U(H_s, 1) = {c(1), a(2)}, U(H_s, 2) = {b(3)}
- Apply (c) $\Longrightarrow t(s) = \rho_{c\to a}(\eta_{c,a}(c(1) \oplus \eta_{a,b}(a(2) \oplus b(3))))$
- $\gamma_{t(s)}(V_s) = \{a, b\}$

5. $s = (1 \oplus (2 \oplus 3)) \oplus 4$

- We have: $\rho_{c\to a}(\eta_{c,a}(c(1) \oplus \eta_{a,b}(a(2) \oplus b(3)))) \oplus c(4)$
- Apply (2) $\Longrightarrow \eta_{c,b}(\rho_{c\to a}(\eta_{c,a}(c(1) \oplus \eta_{a,b}(a(2) \oplus b(3)))) \oplus c(4))$
- V_s= {1, 2, 3, 4}
- H_s = 

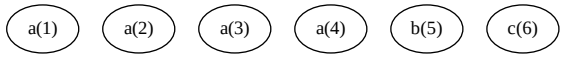
- U(H_s, 1) = {a(1), a(2), b(3)}, U(H_s, 2) = {c(4)}
- Apply (c) $\Longrightarrow t(s) = \rho_{b\to a}(\eta_{c,b}(\rho_{c\to a}(\eta_{c,a}(c(1) \oplus \eta_{a,b}(a(2) \oplus b(3)))) \oplus c(4)))$
- $\gamma_{t(s)}(V_s) = \{a, c\}$

6. $s = ((1 \oplus (2 \oplus 3)) \oplus 4) \oplus 5$

- We have: $\rho_{b\to a}(\eta_{c,b}(\rho_{c\to a}(\eta_{c,a}(c(1) \oplus \eta_{a,b}(a(2) \oplus b(3)))) \oplus c(4))) \oplus b(5)$
- Apply (a) $\Longrightarrow \eta_{b,c}(\rho_{b\to a}(\eta_{c,b}(\rho_{c\to a}(\eta_{c,a}(c(1) \oplus \eta_{a,b}(a(2) \oplus b(3)))) \oplus c(4))) \oplus b(5))$
- V_s= {1, 2, 3, 4, 5}
- H_s = 

- U(H_s, 1) = {a(1), a(2), a(3), c(4)}, U(H_s, 2) = {b(5)}
- Apply (c) $\Longrightarrow t(s) = \rho_{c\to a}(\eta_{b,c}(\rho_{b\to a}(\eta_{c,b}(\rho_{c\to a}(\eta_{c,a}(c(1) \oplus \eta_{a,b}(a(2) \oplus b(3)))) \oplus c(4))) \oplus b(5)))$
- $\gamma_{t(s)}(V_s) = \{a, b\}$

7. $s = (((1 \oplus (2 \oplus 3)) \oplus 4) \oplus 5) \oplus 6$

- We have: $\rho_{c\to a}(\eta_{b,c}(\rho_{b\to a}(\eta_{c,b}(\rho_{c\to a}(\eta_{c,a}(c(1)\oplus\eta_{a,b}(a(2)\oplus b(3))))\oplus c(4)))\oplus b(5)))\oplus c(6)$
- Apply (a) $\Longrightarrow \eta_{c,b}(\rho_{c\to a}(\eta_{b,c}(\rho_{b\to a}(\eta_{c,b}(\rho_{c\to a}(\eta_{c,a}(c(1)\oplus\eta_{a,b}(a(2)\oplus b(3))))\oplus c(4)))\oplus b(5)))\oplus c(6))$
- V_s= {1, 2, 3, 4, 5, 6}
- H_s = 

- U(H_s, 1) = {a(1), a(2), a(3), c(4), b(5), c(6)}
- Apply (c) $\Longrightarrow t(s) = \rho_{c\to a}(\rho_{b\to a}(\eta_{c,b}(\rho_{c\to a}(\eta_{b,c}(\rho_{b\to a}(\eta_{c,b}(\rho_{c\to a}(\eta_{c,a}(c(1)\oplus\eta_{a,b}(a(2)\oplus b(3))))\oplus c(4)))\oplus b(5)))\oplus c(6))))$
- $\gamma_{t(s)}(V_s) = \{a\}$

Chapter 3

Implementation of the algorithm

This section is an introduction about the work we have done: write a program to realize this algorithm. First of all we will summarize the requirements from the advisor, then introduce briefly the program, and finally point out pros and cons of this program

3.1 Requirement specification

The program takes following as input:

- The number of vertices of a graph
- The list of edges of the graph, for example: ((1, 2) , (2 , 3) , (3 , 4) , (4 , 5) , (5 , 6))
- A linear reduced term r, for example_oplus(oplus(oplus(oplus(1, oplus (2, 3)), 4) , 5), 6). To leverage the input readability, two types of parenthesis, () and [] are supported, it means someone can input oplus[oplus[oplus(oplus(1, oplus (2, 3)), 4), 5], 6]

The program will output a term t such that $val(t) = (G, \gamma)$ and red(t) = r and the number of labels used in t is minimum. For example, with the reduced term r as above, the program will output

```
rel_b_a(rel_c_a(add_c_b(oplus(rel_c_a(
add_b_c(oplus(rel_b_a(add_c_a(oplus(
rel_c_b(add_c_b(oplus(add_b_a(oplus(
a(3),b(2))),c(1)))),c(4)))),b(5)))),c(6)))))
```

3.2 Program specification

Our program is provided in two user interfaces: a console UI and a graphical UI

1. Console UI:

- The usage as follows:

  `rcw.(bat|sh) <input_file> [ouput_file]`

- In the case output file is missing, term t will be printed out to console

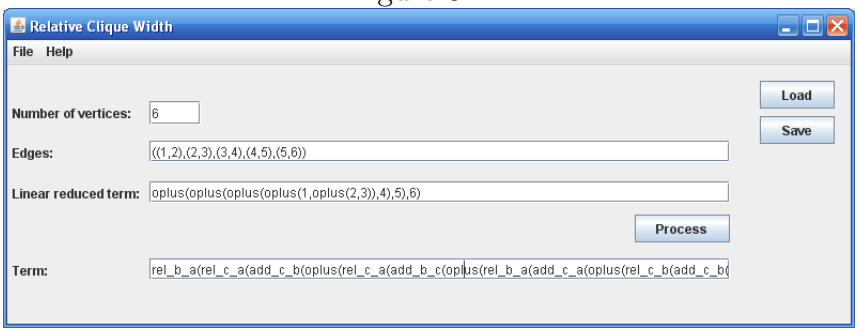
2. Graphical UI:

   - We provide a GUI version of the program, to start it, just run:

     `rcwui.(bin|sh)`

   - Following is a screen-shot of this GUI

Figure 3.1:



3.2.1 Advantage and disadvantage of the program

The program basically satisfies the requirements. However the user interface is not really user-friendly and the usability should be improved more. For example, to increase usability, there are many things the program can do:

- Support drawing graph
- Mapping from a sub-term of r to the corresponding sub-graph
- Output the result step-by-step like the example we did above, so user can easily verify the result

Chapter 4

Methods and used tools

Now is the time for us to look deeper into the details of implementing. Throughout this section, we will provide you information about the programming language, the processing of input data, and the data structure (or according to object oriented programming, the class diagram) we use

4.1 Programming language

Our choice of programming language is Java. Java is a platform-independent language, so our program can run seamlessly on any operation system that supports Java (Windows, Linux, Mac OS ...). Beside that Java has a very big user-community, so we can find a big support from community for common problems we can cope with during implementation process. For example, to process data input, which we are about to discuss right after, we utilize a lexer/parser generator from public domain

4.1.1 Processing of data input

As mentioned above, we have three types of input data

1. The number of vertices of a graph
2. The list of edges of the graph, for example: ((1,2),(2,3),(3,4),(4,5),(5,6))
3. A linear reduced term r, for example: oplus(oplus(oplus(oplus(1,oplus(2,3)),4),5),6). To leverage the input readability, two types of parenthesis, () and [] are supported, e.g. someone can input oplus[oplus[oplus(oplus(1,oplus(2,3)),4),5],6]

Processing of (1) is trivial, (2) is less easy, and (3) is a little difficult. Fortunately, we could overcome this problem quite quickly with so-called parser generator, and more fortunately, there are such generators for Java. The one we chose is JavaCC - stands for Java Compiler Compiler, a very popular parser generator for Java. JavaCC takes a grammar specification written in Extended Backus–Naur Form (EBNF), and then generates Java source code of the corresponding parser. JavaCC also comes with a specific tools JJTree, which helps

to build an Abstract Syntax Tree (AST) from data input to parser. So all the works we must do are just writing the context free grammar for (2) and (3)

- Edge list

```
COMMA = ",";
LP = "(";
RP = ")";
NUMBER = ['0'-'9']+;
VERTEX = NUMBER;
EDGE = LP VERTEX COMMA VERTEX RP;
EDGELIST = LP EDGE COMMA EDGE* RP;
```

- Linear reduced term

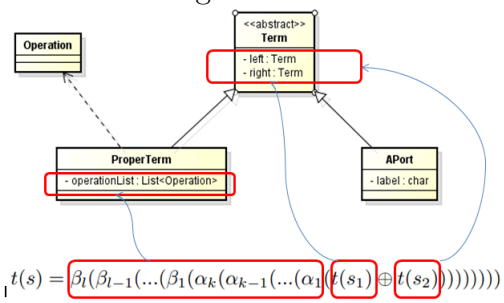
```
OPLUS = "oplus";
LP = "(";
RP = ")";
LSB = "[";
RSB = "]";
COMMA = ",";
NUMBER = (["0"-"9"])+;
VERTEX = NUMBER;
REDUCEDTERM = VERTEX |
        <OPLUS> <LP> VERTEX <COMMA> REDUCEDTERM <RP> |
        <OPLUS> <LP> REDUCEDTERM <COMMA> VERTEX <RP> |
        <OPLUS> <LSB> VERTEX <COMMA> REDUCEDTERM <RSB> |
        <OPLUS> <LSB> REDUCEDTERM <COMMA> VERTEX <RSB>;
```

4.2 Data structure

Java is an OOP language, so to install the algorithm in Java, we need to realize the incident notations to the algorithm (such as term, graph) as Java classes. Following are class diagrams related to these notations:

4.2.0.1 Class diagram for term-related notations

Figure 4.1:

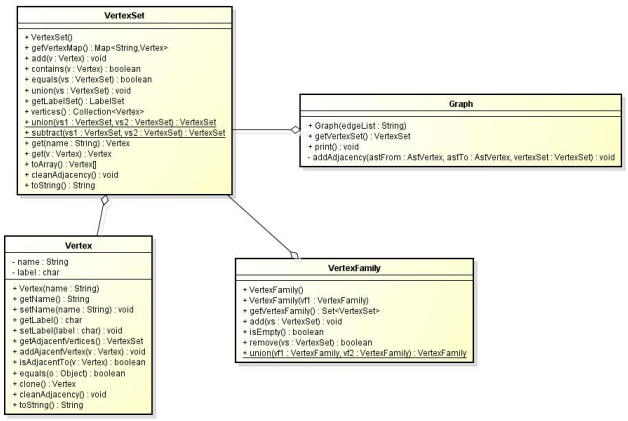


Some annotations:

- APort is class for terms t(s) with s = v
- ProperTerm is class for terms t(s) with $s = s_1 \oplus s_2$

4.2.0.2 Class diagram for graph-related notations

Figure 4.2:



Some annotations:

- Vertex is class to abstract a vertex, each vertex has a list of adjacent vertices
- VertexSet is a class to abstract a set of vertices, VertexSet supports some set operations such as subtract, union
- VertexFamily is a set of sets of vertices. We use VertexFamily to denote notations such as set of partite sets

4.2.1 Source code version control system

We also setup a central source code repository for development process. The version control system we use is Git, and our source code is hosted at github.

Chapter 5

Result

We tested this program with some data tests as follows, program works fine all cases

1. Test case 1
    - n = 6
    - r = oplus(oplus(oplus(oplus(1,oplus(2,3)),4),5),6)
    - Edge list = ((1,2),(2,3),(3,4),(4,5),(5,6))
2. Test case 2
3. Test case 3

Chapter 6

Conclusion

The program basically meets our advisor's requirements, it is able to determines exactly relative clique width of a graph with respect to a linear reduced term. However, as mentioned above, there are still many things waiting to improve, especially about usability of the program.

Clique width as well as other parameters of graph is an interesting subject. Researching in this field is quite valuable. During this project, we have been known that we have more effective algorithms for determining clique width. So we believe one of further works we could try is to investigate those algorithms.

Bibliography

[1] Vadim Lozin, Dieter Rautenbach, "The relative clique-width of a graph", *Journal of Combinatorial Theory, Series B*, pp. 846-858, 2007.

[2] Bruno Courcelle, Stephan Olariu, "Upper bounds to the clique width of graphs", *Discrete Applied Mathematics*, pp. 77-114, 2000.

[3] Bruno Courcelle, "Automata for monadic second-order model-checking", Bordeaux, 2011.