Context-free graph grammars as a language-bias mechanism for graph pattern mining

Christophe Costa Florêncio CHRISTOPHE.COSTAFLORENCIO@CS.KULEUVEN.BE
Jan Ramon JAN.RAMON@CS.KULEUVEN.BE
Katholieke Universiteit Leuven, Dept. of Computer Science, Celestijnenlaan 200A, 3001 Leuven, Belgium

Jonny Daenen JONNY.DAENEN@STUDENT.UHASSELT.BE
Jan Van den Bussche JAN.VANDENBUSSCHE@UHASSELT.BE
Dries Van Dyck DRIES.VANDYCK@UHASSELT.BE
Hasselt University, Transnational University of Limburg, Agoralaan Building D, 3590 Diepenbeek, Belgium

1. Introduction

Many graph mining tasks involve search over languages of graphs. Generating all elements of a graph language is a non-trivial task, and there are often additional constraints, e.g. to allow for efficient pruning. There are several approaches to generate graphs (Kuramochi & Karypis, 2004; Kuramochi & Karypis, 2005; Gudes et al., 2006; Yan & Han, 2002; Ramon & Nijssen, 2009) For some applications, however, it is desirable to constrain the search and only consider a subclass of all graphs. For this, the use of language bias has been considered, e.g. in the field of inductive logic programming (Nédellec et al., 1996).

In the context of graph mining, one would expect of a similar general language bias mechanism that it allows one to specify many different classes of graphs in a declarative way. In the present paper, we propose the use of context-free graph grammars for this purpose. Graph grammars are well known (Rozenberg, 1997), and a wide variety of classes of graphs can be specified using graph grammars. This makes them suitable for use as an expressive and flexible mechanism for graph pattern language bias. Examples of context-free graph languages include: all trees; all cycles; all outerplanar graphs; all graphs of treewidth at most k.

A particularly elegant approach to graph grammars has been developed by Bauderon and Courcelle (1987). As has been pointed out before (Bodlaender, 1998), Courcelle's writings might be somewhat inaccessible for readers with little algebraic background. This paper has three purposes: (1) review Courcelle's graph grammars (more precisely: the HR-equation systems) in a simple and, hopefully, accessible manner, (2) define a canonical form for bounded treewidth graphs based on Arnborg's treewidth checking algorithm (Arnborg et al., 1987) and (3) show how these can be used as a tractable formalism for the generation of graph patterns (Theorem 3).

2. Graph expressions

Basically we assume a finite set $\mathcal{A}$ of *edge labels* and a sufficiently large universe $\mathcal{U}$ of *nodes*. By a *graph* in this paper we mean a triple $G = (V, E, S)$, where V is a finite set of nodes, E is a subset of $V \times \mathcal{A} \times V$, and $S \subseteq V$. The elements of E are called the *edges* of G, and the nodes in S are called the *sources* of G. For any graph G, we will use $V(G)$, $E(G)$, and $S(G)$ to refer to the node set, the edge set, and the set of sources of G, respectively.[1]

A graph G is called *plain* if it has no sources, i.e., $S(G) = \emptyset$. We are primarily interested in plain graphs; the sources are an auxiliary feature that allows us to represent graphs using *graph expressions*, as introduced by Bauderon and Courcelle, and independently by Corradini, Montanari, and Rossi (1994). We use here a syntax suggested by Cardelli, Gardner and Ghelli (2002).

The syntax of graph expressions e is defined by the following grammar, in which lc is short for local. (Here, a ranges over $\mathcal{A}$ and x and y range over $\mathcal{U}$.)

$$e \to a(x, y); \quad e \to e \mid e; \quad e \to (\mathrm{lc}\, x)\, e$$

The semantics of graph expressions will be defined using the two graph operations *merge* and *project*:

Merge: For two graphs G_1 and G_2 we say that $G_1 \mid G_2$ *is allowed* if $V(G_1) \cap V(G_2) \subseteq S(G_1) \cap S(G_2)$. When allowed, a *merge* is defined as: $G_1 \mid G_2 = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2), S(G_1) \cup S(G_2))$.

Project: Let G be a graph and let $x \in S(G)$. Then the *projection* is defined as

[1] We consider only graphs without *isolated nodes*; these are nodes that do not participate in any edge.

$(\mathrm{lc}\, x)\, G = (V(G), E(G), S(G) \setminus \{x\})$.

Graph expressions will represent graphs up to isomorphism. So, as a final step towards the definition of the semantics of graph expressions, we need to define our notion of isomorphism formally.

To that end, let G and G' be two graphs such that $S(G) = S(G')$ (graphs with different sets of sources can never be isomorphic in our formalism). Then an *isomorphism from G to G'* is a bijection $\pi : V(G) \to V(G')$ that is the identity on $S(G)$, and such that the set $\{(\pi(u), a, \pi(v)) \mid (u, a, v) \in E(G)\}$ equals $E(G')$. If such an isomorphism exists, we call G and G' isomorphic; we denote this by $G' \cong G$. The set $\{G' \mid G' \cong G\}$ of all graphs isomorphic to some graph G is called the *isomorphism class of G* or the *the abstract graph G*.

We would like to apply merge and project also to isomorphism classes of graphs. This is easy to define, and we note the following lemma in this respect:

Lemma 1. *For any graphs G_1 and G_2, there exist graphs $G_1' \cong G_1$ and $G_2' \cong G_2$ such that $G_1' \mid G_2'$ is allowed.*

So, for abstract graphs G_1 and G_2, we can always assume without loss of generality that the representatives G_1 and G_2 are chosen such that $G_1 \mid G_2$ is allowed, so that we can indeed consider the abstract graph $G_1 \mid G_2$. For the project operation there is no problem, and given any abstract graph G, we can also consider the abstract graph $(\mathrm{lc}\, x)\, G$.

We are finally ready to define the semantics of graph expressions. For every expression e, we define an abstract graph $\|e\|$ as follows:

$$\begin{aligned} \|a(x,y)\| &= (\{x,y\}, \{(x,a,y)\}, \{x,y\})\,; \\ \|e_1 \mid e_2\| &= \|e_1\| \mid \|e_2\|\,; \\ \|(\mathrm{lc}\, x)\, e\| &= (\mathrm{lc}\, x)\, \|e\|\,. \end{aligned}$$

We note the following easy to prove property:

Proposition 1. *Every graph can be defined by a graph expression, i.e., for every abstract graph G there exists a graph expression e such that $\|e\| = G$.*

Proof. Let $G = (V, E, S)$. The desired expression e consists of the merge of all expressions $a(x, y)$ with $(x, a, y) \in E$, followed by all projections $(\mathrm{lc}\, x)$ where $x \notin S$. □

Note that the proof of the above proposition "wastes" a lot of sources: as many sources are used as there are nodes in the graph. A considerable strenghtening of the above proposition is the following theorem, essentially due to Courcelle (1992). In order to formulate the theorem, let us define the *width* of an expression as the number of *different* sources occurring in the expression, minus 1. We then define the *expression width* of a graph G as the minimal width of any expression defining G. We now have the following theorem and immediate corollary:

Theorem 1. *For any graph $G = (V, E, S)$, the following two statements are equivalent:*

(1) The plain graph (V, E) underlying G has a tree decomposition of width k, such that S is included in at least one bag.[2]

(2) G can be defined by an expression of width k.

Corollary 1. *A graph expression of width k for a plain graph G can be transformed in linear time into a tree decomposition of width k for G and vice versa.*

The key insight for the above corollary is that the bags of a tree decomposition of width k describe which vertices must be sources at the same time to construct all edges of the graph using only $k+1$ sources. In other words, graph expressions and tree decompositions of width k are essentially equivalent.

However, because graph expressions are strings, they have a natural lexicographical ordering which can be used to define a polynomial time computable canonical form for graphs of treewidth at most k. The key idea is to associate a *lexicographical minimal* graph expression with a given tree decomposition of width k by considering all possible injective mappings φ from the vertices of the graph to $k + 1$ sources such that φ is bijective when restricted to a bag. A *canonical graph expression* can then be obtained by taking the lexicographically minimal expression over all tree decompositions of a particular form which can be enumerated in polynomial time by adapting the well known algorithm of Arnborg, Corneil and Proskurowski (1987).

Theorem 2. *A canonical graph expression of width k for a graph of treewidth at most k can be computed in polynomial time.*

3. Context-free graph grammars

In order to generate sets of graphs, one can use an elegant version of context-free graph grammars due to Bauderon and Courcelle, which we introduce here.

Recall that a standard context-free grammar, as classically used to generate strings over some finite alphabet Σ, consists of a finite set $\mathcal{N}$ of *nonterminal symbols*, together with a set of productions, each of the form $X \to s$, where X is a nonterminal and s is a string over the alphabet $\Sigma \cup N$. One of the nonterminals is singled out as the *start symbol*.

The notion of context-free graph grammar now is exactly the same, except that instead of strings as right-hand sides of productions, we will use graph ex-

[2] The X_t's are called the *bags* of a tree decomposition $(T, (X_t)_{t \in V(t)})$ of G and X_t is called the *bag* of t.

pressions. To this end, we must extend our notion of graph expression to allow for nonterminals to occur in them. Let $\mathcal{N}$ be a finite set of nonterminals. Then the syntax of graph expressions e *over* $\mathcal{N}$ is a simple extension of the syntax we already had, as follows. (Here, X ranges over nonterminals in $\mathcal{N}$.)

$e \to X; \quad e \to a(x,y) \quad e \to e \mid e; \quad e \to (\mathrm{lc}\, x)\, e$

Graph expressions in which no nonterminals occur are called *terminal*.

Given a context-free graph grammar, an expression e over $\mathcal{N}$ can be rewritten in a single step in several possible ways, depending on the productions that are present in the grammar. More formally, for expressions e and e' over $\mathcal{N}$, we say that e *rewrites to* e' *in one step*, denoted by $e \to e'$, if

(1) There is a grammar production $X \to f$;
(2) We have obtained e' from e by replacing one occurrence of X in e by f.

Note that if e is terminal, no rewritings are possible, since no nonterminals occur in e. The transitive closure of the relation $\to$ is denoted by $\to^*$. So, $e \to^* e'$ if e can be rewritten into e' in several steps.

Now the *graph language* generated by the grammar is the (typically infinite) set consisting of all terminal expressions e such that $A \to^* e$, where A is the start symbol of the grammar. So, the grammar generates graph expressions rather than actual graphs, but we can easily obtain these by taking the semantics of the produced graph expressions. Indeed, having an expression for each generated graph is an advantage as an expression provides us with the detailed structure of the graph.

Let us now see an example. The following graph grammar, with start symbol C, generates all simple cycles by closing all simple paths from source x to source y, generated by P, and projecting out the sources:

$C \to (\mathrm{lc}\, x, y)\, (P \mid a(y,x));$
$P \to (\mathrm{lc}\, z)\, (Z \mid a(z,y)); \qquad P \to a(x,y);$
$Z \to (\mathrm{lc}\, y)\, (P \mid a(y,z)); \qquad Z \to a(x,z)$

As a more advanced example, we consider a grammar for undirected graphs by dropping the edge directions. Starting from the start symbol G, it generates all biconnected outerplanar graphs[3]:

$G \to (\mathrm{lc}\, x, y)\, (XY \mid a(y,x)); \quad G \to (\mathrm{lc}\, x, z)\, (XZ \mid a(z,x));$
$G \to (\mathrm{lc}\, y, z)\, (ZY \mid a(y,z));$
$XY \to XY \mid a(y,x); \qquad XY \to a(x,y);$
$XY \to (\mathrm{lc}\, z)\, (XZ \mid a(z,y)); \quad XY \to (\mathrm{lc}\, z)\, (a(x,z) \mid ZY);$
$XZ \to XZ \mid a(z,x); \qquad XZ \to a(x,z);$
$XZ \to (\mathrm{lc}\, y)\, (XY \mid a(y,z)); \quad XZ \to (\mathrm{lc}\, y)\, (ZY \mid a(y,z));$
$ZY \to ZY \mid a(y,z); \qquad ZY \to a(z,y);$
$ZY \to (\mathrm{lc}\, x)\, (a(z,x) \mid XY); \quad ZY \to (\mathrm{lc}\, x)\, (a(y,x) \mid XZ)$

[3]Outerplanar graphs occur abundantly in graph datasets representing molecular structures (such as NCI).

4. Tractable pattern generation

A language bias on the possible graph patterns can be nicely and declaratively specified in the form of a graph grammar. Moreover, under reasonable assumptions[4] , patterns belonging to the language generated by the grammar can be enumerated with polynomial delay (proof ommitted due to space limitations).

Theorem 3. *Let k, Δ be constants, $\mathcal{G}$ a graph grammar generating graphs with treewidth at most k and degree at most Δ and n an integer. Then, there is an algorithm enumerating all elements of $\mathcal{G}$ that have at most n nodes with polynomial delay, i.e. the time needed to list each next element of $\mathcal{G}$ is bounded by a polynomial in the size of $\mathcal{G}$ and n.*

References

Arnborg, S., Corneil, D. G., & Proskurowski, A. (1987). Complexity of finding an embedding in a k-tree. *SIAM J. Alg. Disc. Meth.*, *8*, 277–284.

Bauderon, M., & Courcelle, B. (1987). Graph expressions and graph rewritings. *Math. Syst. Theory*, *20*, 83–127.

Bodlaender, H. L. (1998). A partial k-arboretum of graphs with bounded treewidth. *Theor. Compu. Sci.*, *209*, 1–45.

Cardelli, L., Gardner, P., & Ghelli, G. (2002). A spatial logic for querying graphs. *ICALP* (pp. 597–610). Springer.

Corradini, A., Montanari, U., & Rossi, F. (1994). An abstract machine for concurrent modular systems: CHARM. *Theor. Comput. Sci.*, *122*, 165–200.

Courcelle, B. (1992). The monadic second-order logic of graphs. III. Tree-decompositions, minors and complexity issues. *RAIRO Inform. Théor. Appl.*, *26*, 257–286.

Gudes, E., Shimony, S. E., & Vanetik, N. (2006). Discovering frequent graph patterns using disjoint paths. *IEEE Trans. Knowl. Data Eng.*, *18*, 1441–1456.

Kuramochi, M., & Karypis, G. (2004). An efficient algorithm for discovering frequent subgraphs. *IEEE Trans. Knowl. Data Eng.*, *16*, 1038–1051.

Kuramochi, M., & Karypis, G. (2005). Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.*, *11*, 243–271.

Nédellec, C., Adé, H., Bergadano, F., & Tausend, B. (1996). Declarative bias in ILP. In L. De Raedt (Ed.), *Advances in inductive logic programming*, vol. 32 of *Frontiers in Artificial Intelligence and Applications*, 82–103. IOS Press.

Ramon, J., & Nijssen, S. (2009). Polynomial-delay enumeration of monotonic graph classes. *JMLR*. To appear.

Rozenberg, G. (1997). *Handbook of graph grammars and computing by graph transformation, volumes 1–3*. World Scientific Pub Co Inc.

Yan, X., & Han, J. (2002). gSpan: Graph-based substructure pattern mining. *ICDM* (pp. 721–724). IEEE Computer Society.

[4]E.g., molecular structures inherently have bounded degree and almost all have treewidth at most 3 (NCI: 98%).