Interactive Graph Grammar Induction

Luka Fürst

1University of Ljubljana, Faculty of Computer and Information Science, Tržaška cesta 25, SI-1000 Ljubljana, Slovenia
luka.fuerst@fri.uni-lj.si

Abstract. A graph grammar is a concise generative representation of a possibly infinite graph set. Since it is often difficult to build a graph grammar by hand, it is desirable to induce a grammar automatically from a finite set of sample graphs. However, the graph set represented by the induced grammar might differ from the target graph set and might be highly dependent on the input set. Therefore, we propose that graph grammar induction be treated as an interactive process and present a tool that supports a typical interaction scenario.

Keywords. graph grammar; grammar induction; human-computer interaction

1 Introduction

Graph grammars [6], a generalization of well-known string grammars, can be used in a variety of domains, including visual languages [5], pattern recognition [4], computer graphics [1], etc. A graph grammar is composed of a set of initial graphs (axioms) and a set of graph replacement rules (productions). The language of a graph grammar is a set comprising the axioms and all graphs derived from the axioms via an arbitrary number of production applications. A graph grammar can thus be viewed as a concise representation of a graph set, such as the set of graphs representing valid programs of a visual language.

A graph grammar for a given target graph set is often difficult to create by hand. It is therefore of interest to strive for automatic induction of a grammar from sample graphs. The graph grammar induction problem can be stated as follows: Given a set of sample graphs, each of which is labeled 'positive' or 'negative', construct a grammar whose language contains all of the positive and none of the negative input graphs. Such a grammar is consistent with the input set. In addition, the induced grammar should represent a meaningful generalization of the input set. For example, imagine inducing a grammar for arbitrary flowcharts given a few labeled samples of legal and illegal flowcharts. We [2] formulate induction as constrained specific-to-general search for the smallest1 grammar consistent with the input set.

The language of the induced grammar might not coincide with the target graph set and may be highly dependent on the input set. The reason is that the input set does not necessarily capture the diversity of the entire target set. We therefore propose an

1 Inspired by Ockham's razor (simple hypotheses are preferable).
interactive approach to graph grammar induction and present a tool that supports such an approach. This is the contribution of our paper.

The rest of the paper is structured as follows: Section 2 introduces graph grammars and their induction. Section 3 presents an interactive approach to grammar induction and a tool to support such an approach. Section 4 concludes the paper.

2 Graph grammars and graph grammar induction

We induce grammars from a subclass of context-sensitive graph grammars [5]. A context-sensitive graph grammar is a quadruple \((N, T, P, A)\), where \(N\) and \(T\) are sets of terminal and nonterminal graph labels, respectively, \(P\) is a set of productions, and \(A\) is a set of axioms. A production is a rule of the form \(L ::= R\), where \(L\) and \(R\) are graphs with a possibly nonempty common subgraph \(L \cap R\), called the context graph. To apply a production \(L ::= R\) to a graph \(G\), an image of \(L\) in \(G\) has to be replaced with an image of \(R\). The image of \(L \cap R\) in \(G\) is preserved, since it serves as a fixed context of the replacement. Figure 1 displays a grammar for cycle graphs. The elements of the context graphs are grayed. The label ‘\#1’ is nonterminal.

![Graph grammar for cycle graphs](image)

Fig. 1. A graph grammar for cycle graphs.

We shall now give a brief overview of our induction algorithm; a detailed description can be found in [2]. The algorithm is restricted to build grammars with productions that specify the replacement of a single edge with a graph or with another edge. The grammar of Fig. 1, for example, satisfies this requirement. The induction algorithm searches for the smallest grammar in the space of candidate grammars, i.e., grammars consistent with the input set. The algorithm maintains a set of current candidate grammars. At the beginning, this set contains only the most specific candidate grammar. In each subsequent iteration, the algorithm constructs a new set by applying elementary generalizations to the grammars in the current set. An elementary generalization of the first type generalizes a given grammar by introducing a new production and reverse-applying that production to all axioms. The second type simplifies the grammar by merging a set of similar productions. The induction algorithm interacts with a graph grammar parser [3, 5], i.e., an algorithm that determines whether a given graph belongs to the language of a given grammar. The parser is used to ensure consistency with the input set; a grammar whose language contains any negative input graph is immediately discarded.
3 Interactive graph grammar induction

As we already mentioned, the language of the induced grammar might not coincide with the target graph set. If the grammar is overly specific, the input graph set should include more positive samples; if the grammar is too general, the input set should be augmented with additional negative samples. Based on this observation, we can construct a typical human-computer interaction scenario for grammar induction:

1. The user prepares a set of positive and (optionally) negative input graphs.
2. The computer induces a grammar based on the input graph set.
3. If the user is satisfied with the induced grammar, the interaction terminates with success.
4. Otherwise, the user tells the computer to generate some graphs from the induced grammar's language.
5. If the generated set contains any graphs that do not belong to the target graph set, the user adds these graphs to the set of negative input graphs.
6. The user tells the computer to check whether a not-yet-considered graph from the target graph set also belongs to the induced grammar's language. If it does not, the user adds it to the set of positive input graphs.
7. Return to step 2.

By enlarging the input graph set, the likelihood that the induced grammar meets the user's expectations generally increases. The above scenario guides the user through the input set construction process; rather than having to guess a suitable input graph set to obtain a desirable grammar, the user is able to augment the input set in a purposeful step-by-step fashion.

Fig. 2. Screenshot of our interactive graph grammar induction tool
Figure 2 displays a screenshot of a tool for interactive graph grammar induction. The tool is written in Java using the JUNG library\(^2\). The two upper panels of the main window show the positive and negative input graphs. The bottom-left panel displays the grammar induced from the current input set. The tabs labeled induced-bi show the grammar’s axioms, and those labeled induced-pi show the productions, with colors indicating which left-hand-side and right-hand-side elements coincide. (Such elements form the context graphs.) The bottom-right panel shows random graphs belonging to the induced grammar’s language. The user is able to instruct the program to generate several groups of random graphs. For each group, the user specifies its size and the maximum number of production applications to be used to generate individual graphs. If the user finds out that a random graph does not belong to the target set, they can immediately transfer it to the set of negative input graphs.

Step 6 will be supported once the parser is integrated into the tool. This is planned for the near future.

### 4 Conclusion

We have proposed an interactive approach to graph grammar induction and presented a tool to support a typical interaction scenario. The tool visualizes the input graph set and the induced grammar. Using the integrated random graph generator, the user may assess how well the grammar’s language matches the target graph set. If necessary, they can easily augment the input set and re-run the induction process.

The induction tool can be enhanced in several ways. Besides adding support for step 6 of the typical interaction scenario, providing an integrated graph editor would undoubtedly improve the usability of the tool. A further improvement would be to prevent the generation of the same random graphs in successive runs of the algorithm.

### References
