Graph Grammars

Marc Provost
McGill University marc.provost@mail.mcgill.ca

February 18, 2004

Abstract
This presentation introduces graph transformations.
Structure of the talk

- Graph Theory
- Subgraph isomorphism problem and algorithms
- Graph Rewriting: General Framework & Difficulties
- Double & Single pushout approaches
A \textit{(labelled, directed) graph} $G = (V, E, \text{source}, \text{target}, \text{label})$ consists of a finite nonempty set $V$ (vertices) and a finite set $E \subseteq V \times V$ (edges), along with two mappings $\text{source}$ and $\text{target}$ assigning a source and a target node to each edge, and a mapping $\text{label}$ assigning a labelling symbol from a given alphabet to each node and each edge.

- \text{order: } |V|, \text{ size: } |E|
- An arc $e = (v, w)$ is said to be incident with vertices $v$ (source) and $w$ (target).
- $G.\text{inc}(v)$, $G.\text{outc}(v)$, are two mapping assigning a given vertice $v$ to its in and out connections, respectively.

- Subgraph, induced subgraph. Given two graphs, $G = (V, E)$, $G' = \ldots$
(\(V', E'\)), \(G'\) is said to be a subgraph of \(G\) if \(E' \subseteq E\). The subgraph of \(G\) induced by \(G'\) is the graph \((V', E \cap V' \times V')\).

![Subgraph and Induced Subgraph](image)

**Figure 1: Subgraph / Induced subgraph (labels omitted)**

- **Homomorphism (mapping).** Given two graphs \(G = (V, E), G' = (V', E')\), a vertex mapping \(H : V \rightarrow V'\) is said to be an homomorphism from \(G\) to \(G'\) if it is edge preserving: Given two vertices \(u, v \in V\), \(H(u)\) is adjacent to \(H(v)\) whenever \(u\) is adjacent to \(v\).

- **Graph Isomorphism.** A bijective (one-to-one, surjective) homomorphism.
• *(Partial) Subgraph Isomorphism.* An injective (one-to-one) homomorphism. Also called *monomorphism* in the literature.

• *Induced (Complete) Subgraph Isomorphism.* A injective homomorphism reflecting in $G$ the structure of the induced subgraph of $G'$.

![Diagram of homomorphism](image)

**Figure 2: Homomorphism**
Figure 3: Complete, Partial subgraph isomorphisms
Subgraph Isomorphism Problem & Algorithms

- The first step to rewrite a host graph is to match the left-hand side of a rule.

- Subgraph Isomorphism is NP-Complete for general graphs.

- In general, the algorithms are based on the idea of backtracking: Extend a partial solution, one variable at a time, until a complete solution is reached or the partial solution cannot be extended anymore; this can be viewed by a backtracking tree.

- Most algorithms try to limit the time-space explosion by pruning the backtracking tree.
Backtracking tree matching $L$ in $G$.

Figure 4: Execution of a typical subgraph isomorphism algorithm.
• Pruning involves testing that the current partial solution cannot evolve toward a complete solution. Given a graph \( L \) to be matched into a graph \( G \), suppose we have the following partial solution \( M = ((l_1, g_1), (l_2, g_2), \ldots, (l_i, g_i)) \). Now, we are extending \( M \) with \((l_{i+1}, g_{i+1})\), and we want to test if this path can possibly lead to a complete solution. Possible pruning:

- If \( L.outc(l_{i+1}) > G.outc(g_{i+1}) \): prune the tree!
- If \( L.inc(l_{i+1}) > G.inc(g_{i+1}) \): prune the tree!
- **Ullmann’s algorithm** [4]: Check that, for all vertices \( l_v \) adjacent to \( l_{i+1} \), there is at least one vertex \( g_v \) adjacent to \( g_{i+1} \) such that \( M \cup \{(l_{i+1}, g_{i+1}), (l_v, g_v)\} \) is a valid partial solution. If not, prune the tree!

• Other approaches to subgraph isomorphism:

  - Is it possible to solve the problem in polynomial time if some
preprocessing is used? A very interesting paper [3] shows how to solve the problem by means of decision trees:

* A decision tree is constructed from a database of *model* graphs. (By generating the set of all permutations of the adjacency matrix of all graphs, and organizing it into a decision tree)
* At run-time, easily determine if there is a subgraph isomorphism between an unknown *input* graph and some of the model graphs.
* Run in time $O(n^2)$ if preprocessing is neglected
* But, the size of the decision tree contains an exponential number of nodes...
* Useful if the model graphs are small, and real time behavior is needed.
Graph Rewriting: General Idea [2]

- Basic Idea: Iteratively apply rules to transform a *host* graph $G$.

- General framework: a (graph transformation) rule $r = (L, R, K, \text{glue}, \text{emb}, \text{appl})$ consists of:
  - Two graphs: a left-hand side $L$ and a right-hand side $R$
  - A subgraph $K$ of $L$; the *interface* graph.
  - A homomorphism $\text{glue}$, relating $K$ to the right-hand side $R$
  - An embedding relation $\text{emb}$, relating nodes of $L$ to nodes of $R$.
  - A set $\text{appl}$ specifying the applications conditions for the rule
Figure 5: Example of a graph transformation rule
• Step 1: Match \( L \) in the host graph \( G \):

• Step 2: Check the applications conditions (Assume none in this example).
Step 3: Remove from $G$ the part of the isomorphic match that correspond to $L$ (i.e. keep the interface subgraph $K$), along with all dangling edges. This yields the context graph $D$. In short, $D = G - (L - K)$:
Step 4: Glue the context graph $D$ and the right-hand side $R$, according to the glue homomorphism. This yields the gluing graph $E$:
Step 5: Embed the right-hand side $R$ into $D$, according to the relation $emb$. This yields the derived graph $H$: 

![Diagram showing graph rewriting process](image)
Several Difficulties occur with graph grammars:

- Do we require an isomorphism between the LHS and the host graph?
- Do we delete dangling edges when replacing the LHS by the RHS? Or do we not allow such rule to execute?
- How do we organize the rules?
Double pushout approach (DPO) [1]

- The first *algebraic* approach

- *Algebraic*: Basic algebraic constructions (from *category* theory) were used to define the transformation.
  - In DPO: Two pushouts (glues) are performed in the transformation. The first pushout correspond to the deletion of the LHS in the host graph. The second pushout correspond to the insertion of the RHS.

- No embedding function.

- The homomorphisms between K and L; K and R must be injective.
• Does not require an isomorphism between $L$ and the host graph $G$. Instead, a less restrictive approach is used. Two conditions are added for the application of a rule. (also known as the gluing conditions):

  - *Dangling condition*: If a vertex is deleted, then the production must specify the deletion of the edges incident to that node. $G$

---

$G$

$K$

$L$

$R$

$K$

---

Typeset by FoilTEX
Marc Provost

Double pushout approach

- **Identification condition**: Every element that should be deleted in \( G \) has only one pre-image in \( L \)

\[
\begin{array}{c}
\text{L} \\
\text{K} \\
\text{G} \\
\text{R} \\
\text{K}
\end{array}
\]

– Typeset by FoilTEX –

19
Figure 6: General graph transformation rule
Figure 7: DPO graph transformation rule
Single pushout approach (SPO) [1]

- The SPO approach was introduced to add expressiveness to the derivations.
- No gluing conditions.
- No embedding function.
- Deletion has priority over preservation.
- The interface graph is expressed as a partial homomorphism between L and R.
Figure 8: SPO graph transformation rule
References

