Bypass Strong V-Structures and Find an Isomorphic Labelled Subgraph in Linear Time

Heiko Dörr

Institut für Informatik
Freie Universität Berlin
Report B-94-08
7/13/94

This paper identifies a condition for which the existence of an isomorphic subgraph can be decided in linear time. The condition is evaluated in two steps. First the host graph is analysed to determine its strong V-structures. Then the guest graph must be appropriately represented. If this representation exists, the given algorithm constructively decides the subgraph isomorphism problem for the guest and the host graph in linear time.
The results applies especially to the implementation of graph rewriting systems. An isomorphic subgraph must be determined automatically in each rewriting step. Thus the efficient solution presented in this paper is an important progress for any implementation project.

Institut für Informatik
Freie Universität Berlin
Takustraße 9
D-14195 Berlin
doerr@inf.fu-berlin.de

Report B-94-08
July 13, 1994

1 Introduction

A main effort in any implementation of rule-based graph rewriting systems is directed to the solution of the subgraph isomorphism problem for labelled graphs. A graph rewriting rule is applicable to a host graph only if there is a subgraph isomorphic to the left-hand side of the rule. Hence the labelled subgraph problem must be solved in each rewriting step. Current implementations follow two strategies to deal with the problem. They either keep the problem open to the user who determines an isomorphic subgraph interactively [LöBe93], [Him89] or in advance [KiMa92], or they perform a full search [Zün92]. Neither strategy is satisfying with respect to a fast execution which is a key condition for a broader use of graph rewriting.

In general the subgraph isomorphism problem lies in *NP* for undirected and directed labelled graphs [GaJo79]. When considering labelled graphs more information on the problem instances is available. Vertices and edges become distinguishable by their labels. Our approach to an efficient algorithm for the labelled subgraph isomorphism makes heavy use of labels.

Let Σ_V, Σ_E be finite alphabets. A finite non-empty set V is the set of vertices, and $E \subseteq V \times \Sigma_E \times V$ is the set of *edges*. A total function $l\colon V \to \Sigma_V$ is a *vertex labelling*. The triple $G = (V, E, l)$ is a *directed labelled graph* over Σ_V, Σ_E, or just a *graph*. The numbers $p = |V|$ and $q = |E|$ denote the *order* and *size* of G. The set of *unique vertex labels of G* is defined as $uv(G) = \{ul \in \Sigma_V | \exists! v \in V \text{ such that } l(v) = ul\}$.

Let $G = (V, E, l)$ and $G' = (V', E', l')$ be graphs. The graph G' is a *subgraph* of G, $G' \subseteq G$ iff $V' \subseteq V$, $E' \subseteq E$ and $l' = l|_{V'}$. Let $h_V\colon V' \to V$ be a bijective vertex map such that for the corresponding edge map $h_E\colon E' \to E$, defined as $h_E(E') = \{(h_V(s), el, h_V(t)) | (s, el, t) \in E'\}$, holds $h_E(E') = E$ and $l = l' \circ h_V^{-1}$. Based on the vertex bijection h_V the *graph isomorphism* $\hat{h}(G') = (h_V(V'), h_E(E'), l' \circ h_V^{-1})$ is defined. The graph G contains a *subgraph isomorphic* to G' iff there is a graph isomorphism $\hat{h}$ such that $\hat{h}(G') \subseteq G$. The graph G is the *host*, whereas G' is the *guest graph*. When clear from context we omit the indices of the vertex and edge maps h_V and h_E respectively.

The subgraph isomorphism problem for labelled graph is stated as follows: Given two graphs G and G', is there a graph isomorphism $\hat{h}$ such that $\hat{h}(G') \subseteq G$?

Let G and G' be two graphs. Any graph $G_p \subseteq G$ for which there exists a partial graph isomorphism $\hat{h}\colon G' \to G_p$ is a *partial handle*. If $\hat{h}$ is total, then G_p is a *full handle*. A partial handle $G_p = (\{v\}, \emptyset, l)$ with $v \in V$ is an *initial handle*.

Let $\hat{h}\colon G' \to G$ be a partial graph morphism. The graph morphism $\hat{h}'$ *extends* $\hat{h}$ by a vertex $v \in V'$ iff $\{v\} = dom(h') \backslash dom(h)$ and there is an edge $e \in E'$ incident to v such that $\{e\} = dom(h') \backslash dom(h)$ where *dom* determines the domains of the vertex and edge maps h induced by $\hat{h}$.

2 The Basic Algorithm

The labelled subgraph isomorphism can be constructed by traversing the guest graph starting at an arbitrary initial vertex. The first step of the procedure determines the set of initial partial handles according to the initial vertex. In each subsequent step the algorithm tests whether an edge can be mapped to an edge incident to a current partial handle. If possible, current partial handles and the induced vertex maps are extended and comprise the set of handles current to the next step. Thus all handles current to an iteration have an identical inverse image.

A so-called connected enumeration represents a traversal of the guest graph. The elements of the enumeration are successively drawn by the algorithm. The enumeration must ensure that for all edges there is one adjacent edge preceding it in the enumeration because the main principle of our algorithm is the extension of partial handles. Otherwise an extension would be impossible. The initial edge deserves special treatment. The root of a connected enumeration is one vertex incident to the initial edge.

Definition 1 Let $G = (V, E, l)$ be a graph with size q. A sequence (e_i) of edges $e_i \in E$, $i = 1 \ldots q$ is an *enumeration* of E iff $E = \{e_i | i = 1 \ldots q\}$. An enumeration (e_i), $i = 1 \ldots q$ of E is *connected* iff for all e_i there is an adjacent edge e_j with $j < i$. The *root vertex* of (e_i), $i = 1 \ldots q$ is a vertex incident to e_1.

The connected enumeration of a guest graph G' and the host graph G are the input to the algorithm for labelled subgraph matching. The algorithm performs a breadth first search for a graph isomorphism and decides the existence of a graph isomorphism by construction.

Algorithm labelled isomorphic subgraph
Let G and G' be graphs.
INPUT: G and a connected enumeration (e_i), $i = 1 \ldots q$ of E' with root vertex $v \in V'$
OUTPUT: a set of graph isomorphisms $\hat{h}$ with $\hat{h}(G') \subseteq G$.

1. INITIALIZE $A_0 := \{\hat{h} | \exists v' \in V, l'(v) = l(v'), h(v) = v'\}$
2. SET $W_0 := \{v\}$
3. FOR $i = 1 \ldots q$ DO
   LET $(s, el, t) = e_i$
   CASE $s, t \in W_{i-1}$: $A_i := \{\hat{h} \in A_{i-1} | h(e_i) \in E\}$ (* check *)
   $s \notin W_{i-1}$: $A_i := \{\hat{h}_i | \hat{h}_i \text{extends } \hat{h}_{i-1} \in A_{i-1} \text{by } s\}$ (* extension *)
   $t \notin W_{i-1}$: $A_i := \{\hat{h}_i | \hat{h}_i \text{extends } \hat{h}_{i-1} \in A_{i-1} \text{by } t\}$
   SET $W_i := W_{i-1} \cup \{s, t\}$
4. OUTPUT A_q

We prove the correctness of the algorithm.

Lemma 1 *Let G and G' be two graphs. Let* (e_i), $i = 1 \ldots q$ *be a connected enumeration of* E' *with root vertex* $v \in V'$. *Let A be the output of the labelled subgraph algorithm for the input G,* (e_i), $i = 1 \ldots q$, *and v. For all* $\hat{h} \in A$ *it holds that* $\hat{h}(G') \subseteq G$.

Proof. We adopt the terminology of the labelled subgraph algorithm and prove $\hat{h}(G') \subseteq G$ by induction over *i*. By definition of A_0 it holds that $\hat{h}_0(G_0) \subseteq G$ for all $\hat{h}_0 \in A_0$ and initial handles $G_0 = (\{v\}, \emptyset, l')$. Let $i \in \{0, \ldots, q-1\}$, $\hat{h}_i \in A_i$, and $(s, el, t) = e_{i+1}$ be the current edge. Assume now that $\hat{h}_i(G_i) \subseteq G$ for the partial handle $G_i = (W_i, \{e_j | j = 1 \ldots i\}, l')$ and let $\hat{h}_{i+1} \in A_{i+1}$ be an extension of $\hat{h}_i$.
If both endpoints are already found, i.e. $s, t \in W_i$, then it holds for all $\hat{h}_{i+1} \in A_{i+1}$ that $h_{i+1}(e_{i+1}) \in E$. Since $\hat{h}_{i+1}$ extends a proceeding $\hat{h}_i$ by the mapping of e_{i+1} it follows for the induced vertex maps that $dom(h_i) = dom(h_{i+1}) = W_i$. The same argument holds for the induced edges maps $dom(h_i) = dom(h_{i+1}) \backslash \{e_{i+1}\}$. With $\hat{h}_i(G_i) \subseteq G$ it follows that $\hat{h}_{i+1}(G_{i+1}) \subseteq G$.
Let now without loss of generality $s \notin W_i$. Since $\hat{h}_{i+1}$ extends $\hat{h}_i$ by s it holds that $dom(h_i) = (dom(h_{i+1}) \backslash \{s\} = W_i)$ and $dom(h_i) = dom(h_{i+1}) \backslash \{e_{i+1}\}$.
The extended graph morphism maps *s* into *V* and e_{i+1} into *E*. Hence $\hat{h}_{i+1}(G_{i+1}) \subseteq G$ and the proof is complete. ■

For the analysis of the algorithm, we assume that the host graph *G* is given in a frame-based data structure. Each frame stores the direct neighbourhood of a vertex. Each slot of a frame contains a list of isomorphic edges incident to the frame vertex. With this data structure we can directly address incident edges by their labels and direction.

The algorithm performs two major operations depending on the current edge, either a simple check or an extension. A check is executed when both endpoints of the edge are in the domain of the morphisms found already. It selects those morphisms which are defined for that edge, i.e. the corresponding partial handles must contain vertices which are incident to the image of this edge. When one endpoint is missing the algorithm extends the current handles based on the entries of the slot. We assume that a single check and a single extension take one unit of time.

The complexity of the algorithm depends strongly on the number of slot entries. In case of multiple entries the algorithm must scan the list of entries to find the possible images of the given edge. Consequently the check takes time dependent on the number of entries. If we have to extend a match and the respective slot has multiple entries, the extension is performed for each entry and each current handle. Thus multiple extensions must be constructed.

The run time of the algorithm is determined by steps 1 and 3. The analysis of step 3 depends on the edge which is being processed. Let us determine the run time for the *i*-th iteration with edge $e_i = (s, el, t)$ for $i \in \{1, \ldots, q\}$. If both endpoints are included in the current match one check is performed for each handle. Let $n_{h(t), i}$ and $n_{h(s), i}$ be the number of entries in the respective slot of vertices $h(t)$ and $h(s)$ for morphisms $\hat{h} \in A_{i-1}$. Hence the number of tests is

$$\sum_{\hat{h} \in A_{i-1}} n_{h(t), i} \qquad \text{or} \qquad \sum_{\hat{h} \in A_{i-1}} n_{h(s), i}$$

depending on the vertex at which the existence of the edge is checked.

When an extension is performed, assume that we extend the current handles by possible images of the source s, i.e. $s \notin W_{i-1}$. Let again $n_{h(t),i}$ be the number of entries in the respective slot of vertices $\{h(t) | h \in A_{i-1}\}$. When the slot in the image of t is not empty, i.e. $n_{h(t),i} > 0$, the algorithm extends all handles of A_{i-1}. For each item of a slot an extension must be defined. The number of extensions in the i-th step is then

$$\sum_{\hat{h} \in A_{i-1}} n_{h(t),i}$$

To determine the overall run time, let p and q be the order and the size of the guest graph G'. In step 1, $|V|$ comparisons are necessary to find all initial handles. Then the algorithm must map $p-1$ vertices onto images in the graph G. The mappings are constructed by extensions of current handles. As a consequence, the existence of $q-(p-1)$ remaining edges must be checked. Without loss of generality, assume a connected enumeration which drives the isomorphism algorithm such that all extensions are performed before any edge existence is checked. Assume further that only source vertices must be found. Let $n_{h(t),i}$ be defined as above. The overall run time of the algorithm then is

$$|V| + \sum_{i=1}^{p-1} \left(\sum_{\hat{h} \in A_{i-1}} n_{h(t),i}\right) \text{ extensions} + \sum_{i=p}^{q} \left(\sum_{\hat{h} \in A_{i-1}} n_{h(t),i}\right) \text{ checks}.$$

An execution of the algorithm does not branch if the number of handles does not increase in any step. This is a restriction particular for the extension steps since checks per definition do not increase the number of morphisms. In case the set of initial morphisms has only one element, the non-branching execution of the algorithm computes a unique full handle if there is a match at all.

Definition 2 Let G and G' be two graphs and (e_i), $i = 1 \ldots q$ be a connected enumeration of E' with root vertex $v \in V'$. The labelled subgraph isomorphism algorithm *executes without branches* iff for all $i \in \{1, \ldots, q-1\}$ and any $\hat{h}_i \in A_i$ there exists at most one extension $\hat{h}_{i+1} \in A_{i+1}$.

For an execution without branches we can derive the number of occurring extensions.

Lemma 2 *Let G and G' be two graphs with $p = |V'|$ and $q = |E'|$. Let (e_i), $i = 1 \ldots q$ be a connected enumeration of E' with root vertex $v \in V'$. Let $m = |A_0|$ be the number of initial handles. If the labelled subgraph isomorphism algorithm executes non-branching it performs $m(p-1)$ extensions.*

Proof. From the premises it follows for all $i = 1 \ldots q$ and any $\hat{h}_i \in A_i$ that there exists at most one extension $\hat{h}_{i+1} \in A_{i+1}$. Hence the algorithm performs extensions only on single entry slots and either $n_{h(s),i} \leq 1$ or $n_{h(t),i} \leq 1$ depending on the direction of the extension. Furthermore it holds that $|A_{i+1}| \leq |A_i|$ for all $i = 0 \ldots q-1$; hence $|A_i| \leq |A_0| = m$ for $i = 1 \ldots q$. Let us assume that the elements of the enumeration are such that the mapping algorithm extends by the source vertex. For the number of extensions follows $\sum_{i=1}^{p-1} \left(\sum_{\hat{h} \in A_{i-1}} n_{h(t),i}\right) \leq \sum_{i=1}^{p-1} (|A_i|) \leq \sum_{i=1}^{p-1} (|A_0|) \leq m(p-1)$. ■

Theorem 3 *The algorithm takes time linear to the size of the guest graph if the extensions and checks are performed on single entry slots and there is exactly one initial handle.*

Proof. Lemma 2 states that if the algorithm executes without branches the number of extensions is bound by $m(p-1)$. If the root vertex of the input enumeration has a unique image in the host graph, the number of extensions equals $p-1$. The run time of the whole algorithm now depends still on the checks for the existence of the remaining $q-(p\text{-}1)$ edges. If these checks are performed on vertices which have at most one entry in the slot for the respective edge, the algorithm performs $q-(p\text{-}1)$ checks. ■

As a consequence we must determine single entry slots of the host graph. Furthermore if we can give an appropriate enumeration of the guest graph we have proven the statement given as the title of the paper.

3 Strong V-Structures

There are two main factors for the combinatorial explosion of the subgraph algorithm. Firstly there may be several initial handles for the root vertex; secondly the connected enumeration of the guest graph may admit multiple extensions of partial handles. In that case the algorithm must process multiple partial handles in the next iteration.

There are mostly several enumerations for a graph. Why should it not be possible to determine connected enumerations which do not lead to multiple extensions? The major question would then be: how can we determine that enumeration, if it exists?

Figure 1 gives an example for a host graph for which a multiplying and a non-multiplying connected enumeration exist. The study of that graph provides a first impression of the problem for a guest graph G' and a host G. Assume that the algorithm has in an intermediate state determined the subgraph G_1. Only edges incident with G_1 are candidates for the next extension step. At this stage we have two alternatives. We could test the existence of an image either of edge `(2,b,3)` or of edge `(1,e,3)`. The selection of the first alternative causes three extensions and thus three partial handles which are analysed further. Two of them are misleading. This fact will be observed in the next iteration. If we try to extend G_1 following the second alternative, we are lucky because we find a singular extension. Furthermore we can complete the handle in one additional step.

Obviously we do not want to rely on fortune when we select a connected enumeration for a given rule. A closer analysis of the example given in Figure 1 provides a hint for this distinction. The host graph G has a characteristic property causing the multiplication of partial handles. It contains three edges for which the extension of G_1 by the edge `(2,b,3)` is possible. Pairs of these potential images form automorphic semipaths of length 2. Whenever a host graph contains such automorphism, the isomorphism algorithm may take non-linear time.

The example also includes the key property for the solution of the "multiplication problem". There is one enumeration initiated in vertex `1` which performs a non-multiplying search. When we choose the enumeration `((1,e,3),(2,b,3),(1,a,2))`, we do not fall into the automorphism trap. Thus the occurrence of non-trivial automorphisms on semipaths of length 2 is not crucial, but an important information for the selection of a non-multiplying connected enumeration.

Strong V-structures characterize automorphic semipaths of length 2 by their labels. Since only the labels of vertices and not their identity is of interest for the algorithm, this characterization is sufficient. We have chosen the adjective "strong" because the labels of the automorphic outer vertices will be significant also.

Definition 3 The set of all *strong V-structures* over alphabets Σ_V, Σ_E is defined as $SVS = \Sigma_V \times \Sigma_V \times \Sigma_E \times \{in, out\}$. Let $G = (V, E, l)$ be a graph. A pair of edges $(e, e') \in E^2$ is an *instance* of a strong V-structure $vs = (vl_1, vl_2, el, d)$ iff there are vertices $x, y, z \in V$, $y \neq z$ such that $l(x) = vl_1$, $l(y) = l(z) = vl_2$ and for $d = out$ $e = (x, el, z)$, $e' = (x, el, y)$ or otherwise $e = (z, el, x)$, $e' = (y, el, x)$. The set of *strong V-structures of a graph G* is given by

$$svs(G) = \{vs \in SVS | \exists (e, e') \in E^2 \text{ instance of } vs\}.$$

Obviously the component vl_2 of a strong V-structure (vl_1, vl_2, el, d) must not be a unique label. The edges `((12,b,13) (12,b,14))` of graph G in Figure 1 form an instance of the strong V-structure `(B,C,b,`*out*`)`.

We can determine *svs(G)* by sorting the incident edges and checking the occurrence of duplicates.

The strong V-structures of a graph are closely related to the data structure which was proposed for the implementation. We distinguished single and multiple entry slots depending on the number of entries. For each vertex of a graph G in the frame representation and from the knowledge of *svs(G)* it follows whether a slot has definitely at most one entry or not. For all instances (e, e') of a strong V-structure (vl_1, vl_2, el, d) it holds that e and e' are isomorphic with respect to their center vertex. Thus the slot of the center vertex determined by vl_2, el, and d contains at least e and e', and thus it is a multiple entry slot.

As a consequence of Lemma 2 on the number of extensions, and from the additional knowledge of *svs(G)*, we can decide whether the labelled subgraph isomorphism executes

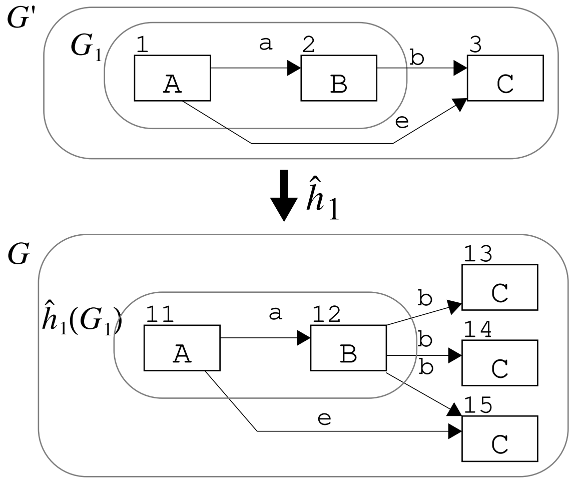

Figure 1 Multiplying and non-multiplying connected enumerations

without branching. We therefore consider connected enumerations which bypass a set of strong V-structures.

Definition 4 Let G be a graph and $q = |E|$. Let (e_i), $i = 1 \ldots q$ be a connected enumeration of E with root vertex v. Let W_i be the set of vertices incident to an edge which is part of the prefix of length i for $i = 1 \ldots q$ and $W_0 = \{v\}$. Let *svs* be a set of strong V-structures. The *connected enumeration* (e_i), $i = 1 \ldots q$ *bypasses svs* iff for all $e_i = (s, el, t)$,

if $s \notin W_{i-1}$: $(l(t), l(s), el, in) \notin svs$

if $t \notin W_{i-1}$: $(l(s), l(t), el, out) \notin svs$, and

if $s, t \in W_{i-1}$: either $(l(t), l(s), el, in) \notin svs$ or $(l(s), l(t), el, out) \notin svs$.

Lemma 4 *Let G and G' be two graphs and let* (e_i), $i = 1 \ldots q$ *be a connected enumeration of* E' *with root vertex* $v \in V'$. *If* (e_i), $i = 1 \ldots q$ *bypasses* $svs(G)$ *then the algorithm for a labelled subgraph isomorphism executes without branching.*

Proof. We adopt the terminology of the algorithm. Let $i \in \{1, \ldots, q\}$ and $e_i = (s, el, t)$ be the current edge. When both endpoints are already found, i.e. $s, t \in W_{i-1}$, the algorithm does not branch by definition. Otherwise let $s \notin W_{i-1}$ and $\hat{h}', \hat{h}'' \in A_i$ be two extensions of $\hat{h}_i \in A_{i-1}$. They must be identical to their predecessor on its domain, i.e. $\hat{h}_i = \hat{h}'|_{dom(h_i)} = \hat{h}''|_{dom(h_i)}$, and $h'(s), h''(s) \in V \backslash h_i(V')$ with $h'(e_i), h''(e_i) \in E$. Since the connected enumeration bypasses *svs(G)* it follows that $(l(t), l(s), el, in) \notin svs(G)$. Hence $h'(s) = h''(s)$ and the extensions are identical: $\hat{h}' = \hat{h}''$. ■

Lemma 4 gives a sufficient condition for the non-branching execution of the labelled subgraph isomorphism. The remaining question is, how can we perform the existence checks in constant time? There is a small gap between the definition of bypassing and the algorithm in the case that both endpoints are found and an existence check is to be performed. The algorithm has no means to determine the endpoint on which the check is performed. For all edges $e_i = (s, el, t)$ of a bypassing enumeration with $s, t \in W_{i-1}$ either s or t has a single entry slot. Thus we refine the elements of the connected enumeration by this additional information. Consequently we modify the isomorphism algorithm to take that additional information into account. The modified isomorphism algorithm takes linear time for an refined enumeration if it bypasses the set of strong V-structures of the host graph.

Theorem 5 *Let G be a graph. If for G' there exists an refined connected enumeration bypassing svs(G) and rooted in a uniquely labelled vertex v, then the algorithm for a labelled subgraph isomorphism takes* $O(|E'|)$ *time.*

Proof. Let G and G' be graphs with $p = |V'|$. Let (e_i), $i = 1 \ldots q$ be a connected enumeration of E' bypassing *svs(G)* and rooted in a uniquely labelled vertex. Thus from Lemma 5 it follows that the algorithm executes without branching. This property also holds for the modified algorithm. Since the enumeration is rooted in a uniquely labelled vertex the algorithm builds $p - 1$ extensions, each extension consuming constant time. By the refinement of (e_i), $i = 1 \ldots q$ we can drive the modified isomorphism such that each check is performed on a single entry slot. Thus each of the $q - (p-1)$ checks takes constant time. Hence the whole isomorphism needs time linear to q. ■

4 Determination of Bypassing Connected Enumerations

One open problem must be solved: How can we construct an appropriate connected enumeration? We give a transformation to the rooted spanning tree problem for a guest graph and a set of strong V-structures. The transformation inspects first the symmetric guest graph. The shift to a symmetric graph reflects the assumed ability to search for an adjacent vertex independent of the direction of the joining edge. The algorithm decides whether it should extend the current partial handles by the source or the target of the current edge. All edges of the symmetric graph have a common interpretation: try to extend a current handle by the target vertex! As a consequence each connected enumeration must contain one edge for each pair of symmetric edges. This holds except for symmetric edges already included in the left-hand side. In this case both edges must be part of the connected enumeration by definition.

Secondly, after modelling all possible enumerations, the transformation implements the information on strong V-structures in the symmetric graph. Some edges of the symmetric graph may be part of an instance of a strong V-structure. They must not be included in a bypassing connected enumeration. Thus they are removed from the symmetric graph. All remaining edges can be traversed without trapping into an instance of a strong V-structure during the algorithm's execution. In this transformation bypassing is equivalent to the existence of a directed spanning tree rooted in a uniquely labelled vertex. The edges of that tree form the first part of the bypassing connected enumeration. The remaining edges are put in the second part, still with respect to the set of strong V-structures. Hence the algorithm maps firstly the vertices of the guest graph. Afterwards, it checks the existence of the remaining edges. The overall transformation can be done in time linear to the size of the guest graph.

Theorem 6 *Let G and G' be graphs. Let* $p = |V'|$ *and* $q = |E'|$. *Let*

$$\overline{E'} = \{ (t, el, s) | (s, el, t) \in E' \} \setminus E'.$$

The elements of $\overline{E'}$ complete G' to a symmetric graph. Let

$$F' = \{ (s, el, t) \in E' | (l'(s), l'(t), el, out) \notin svs(G) \} \text{ and}$$

$$\overline{F'} = \{ (t, el, s) \in \overline{E} | (l'(t), l'(s), el, in) \notin svs(G) \}$$

be the edge sets of the symmetric graph cleared with respect to *svs(G)*.
If there are

- a uniquely labelled vertex $u \in V'$ with $l'(u) \in uv(G) \cap uv(G')$,
- a directed spanning tree $S(G^*, u)$ of $G^* = (V', F' \cup \overline{F'}, l')$ rooted in *u*, and
- for all $\{ (s, el, t) \in E' | (s, el, t)$ is not in $S(G^*, u)\}$ it holds that either $(l'(s), l'(t), el, out) \notin svs(G)$ or $(l'(t), l'(s), el, in) \notin svs(G)$,

then there exists a connected enumeration of E' bypassing *svs(G)*.

Proof. Let (t_i), $i = 1 \ldots p-1$ be a connected enumeration of the spanning tree of G^* rooted in *u*. Let W_i be the set of vertices incident to the edges (t_j), $j = 1 \ldots i$ for $i = 1 \ldots q$ and

$W_0 = \{u\}$. The enumeration (t_i), $i = 1\ldots p-1$ is defined such that for $t_i = (s, el, t)$ it holds that $t \notin W_{i-1}$. Let (e_i), $i = 1\ldots p-1$ be a connected enumeration with $e_i = t_i$ if $t_i \in E'$ and $e_i = (s, el, t)$ with $t_i = (t, el, s)$ otherwise.
The enumeration (e_i), $i = 1\ldots p-1$ bypasses *svs(G)*: let $i \in \{1, \ldots, p-1\}$ be fixed. Since the endpoints of e_i and t_i are identical it follows that $W_i = \bigcup_{j=1\ldots i} inc_{G'}(e_j)$. Let further be $e_i = (s, el, t)$. In case $e_i = t_i$ it holds that $t \notin W_{i-1}$ and it follows by definition that $t_i \in F'$. Hence $(l'(s), l'(t), el, out) \notin svs(G)$. In case $e_i \neq t_i$ it follows that $s \notin W_{i-1}$. By definition it holds that $t_i = (t, el, s) \in \overline{F}'$ and $(l'(t), l'(s), el, in) \notin svs(G)$. As a consequence it follows that (e_i), $i = 1\ldots p-1$ is bypassing *svs(G)*.
For all remaining edges $e \in E' \setminus \{e_i | i = 1\ldots p-1\}$ it holds that all vertices incident with e are member of W_{p-1}. From the premises it follows that any of these edges can be checked bypassing *svs(G)*. Hence we can add these edges to (e_i), $i = 1\ldots p-1$ and receive a connected enumeration of E' bypassing *svs(G)*. ■

5 Related Work

Our approach to reduce the complexity of the labelled subgraph isomorphism analyses the input to the algorithm. Corneil and Gotlieb take a similar approach for an efficient solution of the graph isomorphism problem [CoGo70]. In a preprocessing step a representative and a reordered representation of both input graphs is computed, and the problem is decided based on the transformed graphs. In contrast to our algorithm their procedure only gives an incomplete answer to the isomorphism problem. For some inputs it cannot decide whether the two input graphs are isomorphic.

The RETE-algorithm proposed by Bunke et al. [BGT91] addresses the labelled subgraph isomorphism problem in the context of graph rewriting systems. The algorithm is based on the observation that each rewriting step performs only local changes on the host graph. In a preprocessing step the rewriting system is analysed and the RETE-network created. Its topology represents the left-hand sides of the rewriting rules. It is supposed to carry all full handles. The network is initialized by input of the initial graph. In each rewriting step an appropriate isomorphic subgraph can be selected by inspection of the network. After execution of the rewriting step the information in the network is updated. This approach is not static since at runtime the network must be updated. We on the contrary precompute the bypassing enumeration only once and apply the rewriting rule without auxiliary updates.

We share our interest in V-structures with Witt who studied locally unique graphs [Wit81]. He shows that, by extension of the edge label alphabet, it is possible to create a homomorphic and locally unique image for any graph with bounded degree. Furthermore he proves the existence of a linearizable hull for each locally unique graph. In his context a graph is linearizable iff each vertex of the graph has a unique address given as a list of edge labels. A connected enumeration which bypasses the set of strong V-structures can serve as a linear addressing scheme for a subset of its vertices. But that enumeration contains more information to solve the isomorphic subgraph problem efficiently. The notion of local uniqueness is not sufficient for that purpose because it cannot distinguish as much edges as strong V-structures can.

Unique vertex labels are a property which is already mentioned by Nagl and Göttler [Nag79], [Gött88]. Nagl introduces the “statische Verankerungsstruktur” (static anchor), Göttler uses a “Fixknoten” (fixed vertex) to define an application area of a rewriting rule and to force the application of subsequent rules to that area. The static anchor is used to program graph rewriting systems by means of graph rewriting systems only. None of the authors and even none of the recent publications on graph rewriting systems [Schü91], [KlMa92] have given a formal definition not to mention a criterion for that static anchor.

6 Conclusions

We can apply our results to graph rewriting systems, if we infer information for all sentential forms. Then we can we decide whether a connected enumeration drives a non-multiplying subgraph isomorphism for an arbitrary host graph. Thus we must analyse all graphs generated by a given graph rewriting system. An appropriate analysis technique is abstract interpretation. The major result of our analysis is an upper bound for a set of strong V-structures present in any sentential form. These results are of major importance for an implementation of graph rewriting systems [Dö94].

The strong V-structures of a host graph determine single entry slots. This information on the host graph enables us to select a connected enumeration as input for the labelled subgraph isomorphism algorithm. We proved that the algorithm takes time linear in the size of the guest graph if two conditions hold for a connected enumeration: first, its root must be a uniquely labelled vertex and, second, it must bypass the set of strong V-structures of the host graph. We gave a construction for bypassing enumerations. Based on our approach we can implement graph rewriting systems which perform the application test in linear time.

References

[BGT91] Bunke, H.; Glauser, T.; Tran, T.-H.: ‘An efficient implementation of graph grammars based on the RETE matching algorithm’, [EKR91], pp.174-189.

[CoGo70] Corneil, D.G.; Gotlieb, C.C.: ‘An Efficient Algorithm for Graph Isomorphism’, *Journal of the Association for Computing Machinery,* 17 (**1**) 51-64 (1970).

[Dö94] Dörr, Heiko: ‘An Abstract Machine for the Execution of Graph Grammars’, to appear.

[EKR91] Ehrig, Hartmut; Kreowski, Hans-Jörg; Rozenberg, Grzegorz (ed.): *Graph-Grammars and Their Application to Computer Science*, *4th Int. Workshop*, Bremen, March 5-9, 1990, *LNCS 532*, Springer, Berlin, 1991.

[GaJo79] Garey, Michael R.; Johnson, David S.: ‘*Computers and Intractability*’, W.H. Freeman and Co., New York, 1979.

[Gött88] Göttler, Herbert: ‘Graphgrammatiken in der Softwaretechnik’, *Informatik-Fachberichte 178*, Springer, Berlin, 1988.

[Him89] Himsolt, Michael: ‘Graphed: An interactive Graph Editor’, in *STACS 89*, *LNCS 349* Spinger Verlag, Berlin, 1989.

[KlMa92] Klauck, Christoph; Mauss, Jakob: 'A Heuristic Driven Chart-Parser for Attributed Node Labelled Graph Grammars and its Application to Feature Recognition in CIM', *Research Report*, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern/Saarbrücken, **DFKI-RR-92-43**, 1992.

[LöBe93] Löwe, Michael; Beyer, Martin: 'AGG — An Implementation of Algebraic Graph Rewriting' in Kirchner, Claude (ed.) *Rewriting Techniques and Applications,* Montreal, Canada, June 16-18, 1993, *LNCS 690*, Springer, Berlin, 1993, pp.451-456.

[Nag79] Nagl, Manfred: *'Graph-Grammatiken, Theorie, Implementierung, Anwendungen'*; Vieweg, Braunschweig, 1979.

[Schü91] Schürr, Andreas: '*Operationales Spezifizieren mit programmierten Graphersetzungssystemen*', Deutscher Universitäts-Verlag, Wiesbaden, 1991.

[Wit81] Witt, Kurt-Ulrich: 'On linearizing graphs', in Noltemeier, Hartmut (ed.) *Graphtheoretic Concepts on Computer Science*, *WG '80*, Bad Honnef, *LNCS 100*, Springer, Berlin, 1981, pp.32-41.

[Zün92] Zündorf, Albert: 'Implementation of the imperative/rule based language PROGRES', *Aachener Informatik-Berichte Nr. 92-38*, RWTH Fachgruppe Informatik, Aachen, 1992.