A Graph-Based Rule-Mining Framework for Natural Language Learning and Understanding

Lukas Molzberger
Fraunhofer Institut Autonome Intelligente Systeme
lukas.molzberger@ais.fraunhofer.de

Abstract

Learning and understanding natural languages are usually considered as independent tasks in natural language processing. These two tasks, however, are strongly interrelated and are presumably unsolvable as separate problems. In this paper, we present an algorithm called Frequent Rule Graph Miner (FRGM) that tackles these problems by alternately improving on the language model and the example interpretations. FRGM is based on an effective graph-mining algorithm adapted for enumerating frequent rule-graphs and is applicable to different layers of natural language processing such as morphology, syntax, semantics and pragmatics.

1 Introduction

Learning and understanding natural languages are two of the most challenging problems in artificial intelligence. One reason why these problems are so hard is that they can not be solved independent from one another [3]. Before deeper linguistic knowledge can be learned from an example sentence, an interpretation of the meaning of this sentence is required. But the construction of such an interpretation requires that a sufficiently good language model has been learned before. In this paper, we propose a rule-based approach able to integrate both the learning and the understanding steps.

Another problem in natural language processing is that the different layers of language such as morphology, syntax, semantics and pragmatics can not be processed separately from one another. There is also no predefined order in which rules acting on these layers have to be processed. Consider, for instance, the problem that grammar is in many cases syntactically ambiguous, i.e. there are often multiple possible parse trees for a given sentence. Choosing the most appropriate one usually requires semantic and contextual information [5]. One possible way to overcome this problem is to provide a common representation scheme in which all the information needed for a useful interpretation of text can be expressed. In our approach we chose a powerful class of labeled graphs, called *text-graphs* which meets these demands. Text-graphs allow to represent words, semantic annotations and concepts as vertices and to put them in relation to each other by the means of edges.

To represent knowledge in our language model, we introduce *rule-graphs*. A rule-graph is a pair $A \rightarrow B$, where A and B are labeled graphs. In this rule-graph, A is the rule body which is required to occur in a text-graph before the rule-graph can be applied. When that happens, B the rule head will be added to the text graph. Since these rules are based on graphs they are able not only to recognize a relational pattern, but also to provide a resulting relational pattern that references the original pattern. The ability to process relational data is an important trait that distinguishes rule-graphs from more conventional propositional rules.

In our approach the task of understanding a given sentence is performed by inferring an interpretation from this sentence using a previously learned language model. This is done by iteratively applying the rules in the model on a text-graph representation of the sentence. Hereby, each rule can rely on the information that has been added during the previous round so that more and more information is accumulated in each round. Ideally, the final interpretation assigns each word a unique meaning and also describes the relations between the phrases of the sentence. The rules are applied in a forward chaining fashion, meaning that we start from the data, and successively apply the rules. We note that this contrasts the approach taken by most parsing algorithms, where the starting point is a predefined goal that these algorithms try to "explain" by chaining the rules backwards until a complete parse tree is found. The other task that we try to tackle in our approach is to learn the language model from a set of example text-graphs. To do so, we formulate the learning task as a problem of finding all rule-graphs that occur frequent as positive embeddings and infrequent as negative embeddings in these examples. In other words we search for rule-graphs that are as general as possible, but at the same time do not make too many mistakes. The assumption is that rule-graphs that occur frequent in the training examples, will accurately predict the target values on unknown data, too. To determine the frequency threshold we use a linear function that requires higher frequency for larger rule-graphs, because those are less likely to be correct according to Occam's razor. If the quality of the training data is good enough, it makes sense to set the maximum threshold for negative embeddings to zero in order to disallow false embeddings. The forward-chaining property of our rule system allows to facilitate false parses as negative training examples.

The idea behind our approach is to go through sev-

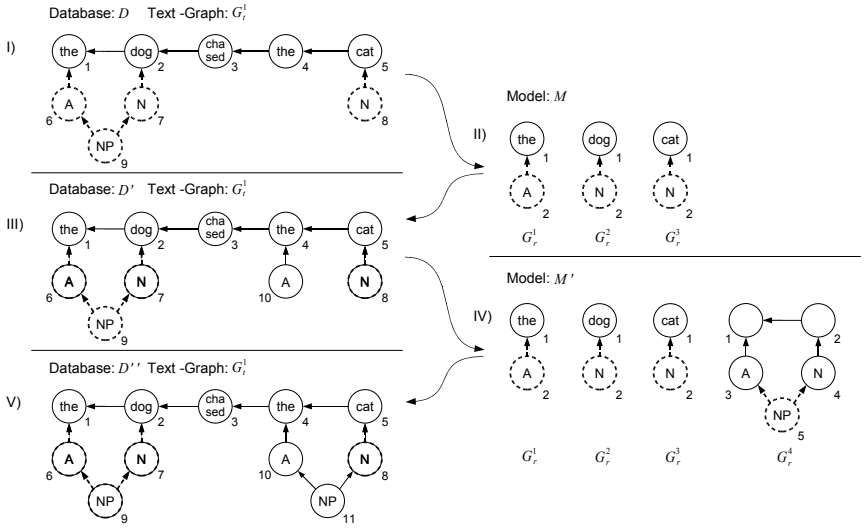

Figure 1: The running example

eral cycles of mining (learning) and inference (understanding) steps, so that alternately the language model and the training database are improved (Fig.2). During the first round only very basic rules are learned from the example database, but those rules are then applied to the database so that the next mining round can learn from a richer, better understood set of examples. Thus, complex training examples can profit from rules that have been previously learned from simpler examples. Once the model has been sufficiently trained it can be used to process a new previously unknown text. The output of this processing step is solely determined by the training examples that have been used to train the model.

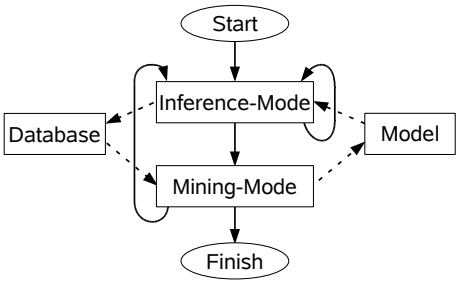

Figure 2: The main routine

For efficiency reasons we store the model in a generalized version space tree (GVST) as proposed by Rückert and Kramer [4]. But that means, that we have to define a unique or canonical form in which to store the rule-graphs.

Another application area where forward-chaining based inference has been quite successfully used is expert systems. Here, however, rules are usually based on a first order logic representation instead of a graph based one. The most popular algorithm for matching such rules is RETE [1].

In production rule systems runtime efficiency has always been a critical issue. Fortunately, new advances in the field of graph-mining such as GVSTs, allow to improve the performance not only of the rule-mining step but also for the rule-application step. The FRGM algorithm itself is based on a basic graph-mining algorithm similar to the gSpan [6] algorithm. The task of such an algorithm is to enumerate all frequent subgraphs in a database of graphs. A new technique that we employ in our algorithm are the *refinement candidates*, which simplify the generation of rule-graph refinements. Another approach concerned with mining for rule-graphs is the Subdue system [2]. Their graph grammar rules however replace (compress) parts of the instance graphs instead of adding new subgraphs.

2 A running Example

Before we go into further details, we want to illustrate our algorithm on a simple example sentence. We start the example by transforming the sentence "the dog chased the cat" into a text-graph where each word has a corresponding vertex. The correct sequence of words in the text-graph is maintained by edges specifying the previous word relations between the vertices. In this step, additional edges and vertices can be introduced to represent morphological features or long ranging word relations. In addition to the input data (marked by continuous lines), we need to specify the learning target (marked by dotted lines) as a set of additional vertices and edges. The learning target are those elements that we hope to predict on unknown data. According to Fig.2 processing would start with the rule application (inference) step, but since the model is still empty we can directly jump to the mining step. In mining mode the algorithm attempts to find all rule-graphs that occur frequently as positive example embeddings and at the same time infrequent as negative example embeddings. Positive example embeddings are those where the complete rule, body and head, can be embedded into the example text-graph such that the edges, labels and roles (INPUT $\Leftrightarrow$ BODY, TARGET $\Leftrightarrow$ HEAD) match together. A positive example embedding in Fig.1 is for instance the embedding $\{(1 \mapsto 1), (2 \mapsto 6)\}$ of rule-graph G_r^1 in text-graph G_t^1. Example embeddings where only an embedding of the rule-body exists, but not of the rule-

head are counted as either negative or neutral. This depends on whether the rule-head and the target part of the text-graph have common labels. Consider for instance the embedding $\{(1 \mapsto 4)\}$ of the rule-body of rule-graph G_r^1 into the text-graph G_t^1. This embedding is negative, because the label A is present as the label of a target vertex (e.g. Vertex 6) but there is no target vertex corresponding to the head of the rule-graph. The intention behind this scheme is to implicitly define the negative training examples. This scheme, however, demands that the target vertices of a certain label have always to be completely provided for a given text-graph since otherwise valid occurrences of a rule would be counted as negative ones.

Now, let us assume that the rule-graphs shown in model $\mathcal{M}$ meet the threshold demands and are therefore the results of the first mining round. These rules are then used in the inference step to create a first interpretation of the example sentence. This is done by applying all rules in the model on all occurrences of their bodies in the database. The interpretation, though not perfect yet, already delivers some additional information to the original sentence (see $\mathcal{D}'$, G_t^1). These informations further the next mining round so that more interesting and complex rules can be found (see $\mathcal{M}'$). The rule-graph G_r^4 can not be found earlier because the vertices labeled A (article) and N (noun) have not been there before and the patterns "the dog" and "the cat" were too infrequent.

3 The Text- and Rule-Graphs

In this section, we define the text- and rule-graphs as well as some necessary notations related to them.

A *text-graph* G_t is a directed acyclic graph (DAG) consisting of a 5-tuple $(V_t, E_t, \Sigma, \lambda_t, \alpha_t)$, where $V_t = \{v_1, ..., v_n\}$ is a set of vertices, $E_t \subseteq V_t \times V_t$ is a set of edges, Σ is an alphabet, $\lambda_t : V_t \cup E_t \to \Sigma$ is a labeling function mapping the vertices and edges to Σ, and α_t is a function that assigns to each vertex and edge either $\{INPUT\}$, or $\{TARGET\}$, or $\{INPUT, TARGET\}$.

The target elements in text-graphs play an important role in that they determine the future structure and labeling of the learned rules. Therefore, they also determine the final processing results on new sentences.

An important feature of text-graphs is that during inference they can be enriched by additional information represented by new vertices and edges. In order to avoid redundancy (i.e. the problem that two vertices represent the same information) we define a merging scheme that allows to eliminate those redundant vertices. Such a merging scheme requires that all vertices except the initial ones can uniquely be identified by their labels and their outgoing edges.

Vertices of a text-graph can be interpreted in a dual way as objects and as predicates relating to other objects. Therefore, the merging scheme allows to uniquely identify objects even if multiple rule-graphs recognize the same object.

Let $G_t = (V_t, E_t, \Sigma, \lambda_t, \alpha_t)$ be a text-graph. Two vertices $v_i, v_j \in V_t$ are *mergeable* if and only if (iff) (i) both have the same label, i.e., $\lambda_t(v_i) = \lambda_t(v_j)$, (ii) there are edges $e_i = (v_i, v), e_j = (v_j, v) \in E_t$ pointing to some common vertex $v \in V_t$ such that $\lambda_t(e_i) = \lambda_t(e_j)$, (iii) there are no edges $e_i = (v_i, v_x), e_j = (v_j, v_y) \in E_t$ pointing to different vertices v_x and v_y such that $\lambda_t(e_i) = \lambda_t(e_j)$, and (iv) there is no edge (v_i, v_j) or (v_j, v_i) connecting these two vertices. To illustrate this, consider the vertices A_1 and A_2 in Fig.3. Here, A_1 and A_2 are mergeable in text-graph ii), but not in the text-graphs i), iii) and iv).

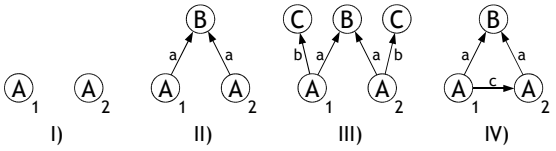

Figure 3: Mergeable (ii) and non-mergeable (i,iii,iv) vertices.

Let $v_i, v_j \in V_t$ be two mergeable vertices of a text-graph $G_t = (V_t, E_t, \Sigma, \lambda_t, \alpha_t)$. The graph derived from G_t by merging the vertices $v_i, v_j \in V_t$, denoted $\beta(G_t, v_i, v_j)$, is a text-graph obtained from G_t containing a new vertex v_k, that replaces the vertices v_i and v_j. Edges that were either connected to v_i or v_j are now connected to v_k. The same applies to labels. The roles of v_i and v_j are combined by a set union.

A *canonical text-graph* $G_t = (V_t, E_t, \Sigma, \lambda_t, \alpha_t)$ is a text-graph containing no two vertices $v_i, v_j \in V_t$ that are mergeable.

Any non-canonical text-graph G_t can be transformed into a canonical text-graph by successively applying the merge operator β on all mergeable vertices in G_t. We note that the order of the merge operator leading to canonical text-graphs is arbitrary. This is stated by the following proposition.

Proposition 1 For each text-graph it holds that its canonical text-graph is unique.

A *rule-graph* G_r is a connected DAG consisting of a 5-tuple $(V_r, E_r, \Sigma, \lambda_r, \alpha_r)$, where $V_r = \{v_1, ..., v_n\}$ is a set of vertices, $E_r \subseteq V_r \times V_r$ is a set of edges, $\lambda_r : V_r \cup E_r \hookrightarrow \Sigma$ is a partial labeling function, and α_r is a function that assigns to each vertex and edge either $\{HEAD\}$ or $\{BODY\}$.

In Fig. 2:G_r^1,... ,G_r^4 we see that in a rule-graph all vertices and edges have a role label that assigns them a role as rule head (marked by a dotted line) or rule body (marked by a continuous line).

Given a rule-graph $G_r = (V_r, E_r, \Sigma, \lambda_r, \alpha_r)$, the *rule body* G_r^{body} of G_r is the graph $G_r^{body} = (\{v_r \in V_r \mid BODY \in \alpha_r(v_r)\}, \{e_r \in E_r \mid BODY \in \alpha_r(e_r)\}, \Sigma, \lambda_r, \{V_r \cup E_r \to \{BODY\}\})$ of G_r whose vertices and edges have the role $BODY$ assigned to them. In a similar way, the *rule head* G_r^{head} of G_r is the graph $G_r^{head} = (\{v \in V_r \mid \alpha_r(v) \subseteq \{HEAD\}\}, \{e \in E_r : \alpha_r(e) \subseteq \{HEAD\}\}, \Sigma, \lambda_r, \{V_r \cup E_r \to \{HEAD\}\})$ of G_r whose vertices and edges have only the role $HEAD$ assigned to them.

The *role mapping function* δ is a bijection, mapping the rule-graph roles onto the text-graph roles in the following way: $\{(\emptyset \mapsto \emptyset), (\{BODY\} \mapsto \{INPUT\}), (\{HEAD\} \mapsto \{TARGET\}), (\{BODY, HEAD\} \mapsto \{INPUT, TARGET\})\}$

Let $G_t = (V_t, E_t, \Sigma, \lambda_t, \alpha_t)$ be a text-graph and $G_r = (V_r, E_r, \Sigma, \lambda_r, \alpha_r)$ be a rule-graph. An *isomorphic embedding* of G_r into G_t is a bijection $\varphi : V_r \to V_t$ preserving the edges, labels, and roles with respect to δ, i.e.,

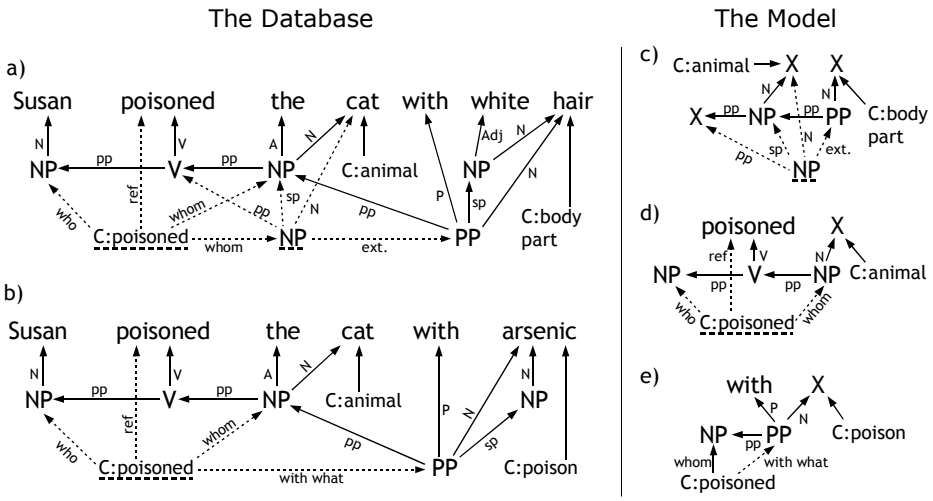

Figure 4: A syntactic disambiguation example [2]

- for every $v_i, v_j \in V_r, (v_i, v_j) \in E_r$ iff $(\varphi(v_i), \varphi(v_j)) \in E_t$,
- $\lambda_r(v) = \begin{cases} \lambda_t(\varphi(v)) & \text{if } \lambda_r(v) \text{ is defined} \\ \varnothing & \text{otherwise} \end{cases}$ for every $v \in V_r$,
- $\lambda_r((v_i, v_j)) = \begin{cases} \lambda_t((\varphi(v_i), \varphi(v_j))) & \text{if } \lambda_r((v_i, v_j)) \text{ is defined} \\ \varnothing & \text{otherwise} \end{cases}$ for every $(v_i, v_j) \in E_r$,
- $\delta(\alpha_r(v)) \subseteq \alpha_t(\varphi(v))$ for every $v \in V_r$, and
- $\delta(\alpha_r((v_i, v_j))) \subseteq \alpha_t((\varphi(v_i), \varphi(v_j)))$ for every $(v_i, v_j) \in E_r$.

The isomorphic embedding describes the occurrence of either a rule body or a complete rule-graph, in a text-graph. Take for instance the rule-graph G_r^4 in model $\mathcal{M}'$ and the text-graph G_t^1 in database $\mathcal{D}'$ in our running example, then there exists an isomorphic embedding $\{(1 \mapsto 1), (2 \mapsto 2), (3 \mapsto 6), (4 \mapsto 7), (5 \mapsto 9)\}$.

A *database* is a set of example text-graphs and a *model* a set of rule-graphs.

4 The Graph Inference and the Graph Mining Problem

Before we explain the graph inference problem in detail we would like to give a more complex example (Fig.4) that shows how a syntactic disambiguation can be realized with graph inference. Given are two example sentences where the prepositional phrase (i.e. "with ...") relates either to the noun phrase "the rat" or to the verb "poisoned". These sentences have already been partially processed so that some additional information is already given. The example now shows how new information (marked by dotted lines) is added to the text-graphs by applying the rule-graphs c), d), and e). In the example, rule-graph c) adds another noun phrase vertex to the text-graph a), which refers to the phrase "the rat with white hair". This rule-graph is not a pure grammar rule, since it also takes the semantic information into account that a rat is an animal and that hair are a body part. Without these information the rule would not be able conclude that the noun phrase "the rat" together with the prepositional phrase "with white hair" form a larger noun phrase. It is obvious that the phrase "the rat with arsenic" in b) does not form a meaningful noun phrase. The next rule-graph adds the concept "C:poisoned" to both text-graphs[1] which represent the respective predicate of these sentences. The last rule-graph adds only a single edge, describing that arsenic has been used for poisoning.

Now, we formally define the graph inference problem, i.e. the problem of how rule-graphs are used to derive interpretations from sentences.

Let $G_t = (V_t, E_t, \Sigma, \lambda_t, \alpha_t)$ be a text-graph and $G_r = (V_r, E_r, \Sigma, \lambda_r, \alpha_r)$ be a rule-graph containing the subgraphs $G_r^{head} = (V_r^{head}, E_r^{head}, \Sigma, \lambda_r^{head}, \alpha_r^{head})$ and G_r^{body} such that G_r^{body} can be embedded into G_t via subgraph isomorphism. Let $V_t^{'}$ be a new set of vertices such that $V_t \cap V_t^{'} = \emptyset$ and $|V_r^{head}| = |V_t^{'}|$. Let further $\varphi^{body} : V_r^{body} \rightarrow V_t$ be a bijection corresponding to such an embedding of G_r^{body}, $\varphi^{head} : V_r^{head} \rightarrow V_t^{'}$ be a bijection, and $\varphi = \varphi^{head} \cup \varphi^{body}$ be the combined bijection from V_r to $V_t \cup V_t^{'}$. The graph derived from G_t by G_r and φ^{body}, denoted $\tau(G_r, G_t, \varphi^{body})$, is a text-graph $G_t^\tau = (V_t^\tau, E_t^\tau, \Sigma, \lambda_t^\tau, \alpha_t^\tau)$, where

- $V_t^\tau = V_t \cup V_t^{'}$,
- $E_t^\tau = E_t \cup E_t^{'}$ such that $E_t^{'} = \{(\varphi(v_i), \varphi(v_j)) \mid \forall (v_i, v_j) \in E_r^{head}\}$,
- $\lambda_t^\tau(x) = \begin{cases} \lambda_t(x) & \text{if } x \in V_t \cup E_t \\ \lambda_r^{head}(\varphi^{-1}(x)) & \text{if } x \in V_t^{'} \\ \lambda_r^{head}((\varphi^{-1}(v_i), \varphi^{-1}(v_i))) & \text{if } x = (v_i, v_j) \in E_t^{'} \end{cases}$
- $\alpha_t^\tau(x) = \begin{cases} \alpha_t(x) & \text{if } x \in V_t \cup E_t \\ \{INPUT\} & \text{if } x \in V_t^{'} \cup E_t^{'} \end{cases}$

Given an occurrence of a rule body in a text-graph, the rule inference operator τ is used to apply this rule on the text-graph. That means, a copy of the head of this rule will be added to the text-graph. Head edges linking to the body of the rule will be transfered using

[1] Note that rule-graph d) can be applied twice on text-graph a).

[2] X: unlabeled vertex; NP: noun phrase; PP: prepositional phrase; V: verb; pp: previous phrase; sp: subphrase; N: main noun; A: article. We note that edges determining the range of phrases and the relations between words have been omitted.

the embedding of the rule body. To illustrate this, consider the text-graph G_t^1 in database $\mathcal{D}'$ and the rule-graph G_r^4 in model $\mathcal{M}'$ in our running example (Fig.2). Let $\{(1 \mapsto 4), (2 \mapsto 5), (3 \mapsto 10), (4 \mapsto 8)\}$ be an embedding of this rules body in the text-graph. The result of the inference operation is shown in v), that is, a new vertex labeled NP has been added.

Using the above notation of rule inference operator τ, we are now ready to define the *model inference operator*. This operator, denoted π is used to apply all rule-graphs of a model on a text-graph and to combine the results afterwards. Multiple additions of the same information are prevented by the text-graph merging scheme. For a canonical text-graph $G_t = (V_t, E_t, \Sigma, \lambda_t, \alpha_t)$ we define $G_t^{x,y} = \tau(G_r^x, G_t, \varphi_y^{body})$ for all $G_r^x \in \mathcal{M}$ [1] and all isomorphic embeddings φ_y^{body} of G_r^{body} into G_t. The text-graph $\pi(G_t, \mathcal{M})$ derived from the canonical text-graph G_t by $\mathcal{M}$ is $\pi(G_t, \mathcal{M}) = (V_t \cup V_t^{1,1} \cup V_t^{1,2} \cup ... \cup V_t^{n,m}, E_t \cup E_t^{1,1} \cup E_t^{1,2} \cup ... \cup E_t^{n,m}, \Sigma, \lambda_t \cup \lambda_t^{1,1} \cup \lambda_t^{1,2} \cup ... \cup \lambda_t^{n,m}, \alpha_t \cup \alpha_t^{1,1} \cup \alpha_t^{1,2} \cup ... \cup \alpha_t^{n,m})$.

Given a text-graph G_t and a model $\mathcal{M}$, the *graph inference problem* is to find a new text-graph G_t' such that G_t' can be derived by a sequence of model inference operator applications.

The basic assumption underlying the learning process in our approach is that predictive rules will occur often as positive embeddings in the training database and at the same time seldom as negative embeddings. In contrast to other supervised learning systems we use the notation of positive and negative embeddings instead of examples, because the target we are trying to predict is a subgraph within an example rather than a label for the whole example. Therefore, a single text-graph can contain several positive and negative embeddings of a rule-graph.

Before formulating the graph-mining problem let us first define the *threshold function* $\mu(G_r)$. Given a rule-graph G_r and real numbers a and b, the threshold function is a linear function of the form $\mu(G_r) = |G_r| * a + b$.

Let $\mathcal{D}$ be a database of example text-graphs. Let $count^+(G_r)$ be the number of embeddings of the complete rule-graph G_r (i.e. HEAD + BODY). Let $count^-(G_r)$ be the number of negative embeddings of G_r, i.e. the number of rule body embeddings in those examples that contain the labels of all head vertices of G_r as labels of target vertices minus the number of complete rule embeddings in those examples. The *rule-graph mining problem* is to find a model $\mathcal{M} = \{G_r \mid count^+(G_r) \geq \mu(G_r) \wedge count^-(G_r) < freq_{min}\}$ containing all positively frequent rule-graphs that occur in $\mathcal{D}$.

5 Generalized Version Space Trees and the Canonical Form of Rule Graphs

As already mentioned in the introduction, we use the Generalized Version Space Tree (GVST) structure to efficiently store our rule-graphs in a model. The main idea behind the GVST is to decompose each rule-graph into a sequence of primitive graph refinements. The original rule-graph can then be reconstructed by following the path from the root node, and adding each refinement on the path. A rule-graph that is not in canonical form needs to be transformed to a canonical form prior to storing it in the GVST. This is necessary to prevent that duplicates of the same rule-graph are stored in the GVST.

Basically, the canonical form of a graph is a unique sequential ordering of its vertices. In general the vertices of any graph could be represented in $n!$ different sequences whereby n is the number of vertices. By using a canonical form we can prevent that we have to consider all those different orderings of what is essentially the same graph. Designing a canonical form definition, however, implies several constraints that we need to take care of. Firstly, it must be possible to sequentially construct a connected canonical graph without having to rely on unconnected intermediate graphs. Secondly, an order among the syntactic variants of a rule-graph must be defined. And thirdly, rule-graphs differ from usual graphs in that they consist of two parts, the head and the body. For our purposes, we need to calculate the embeddings not only for the entire rule-graph, but also for the body part only. For this reason it is necessary to separate the head and body vertices in two ranges. Therefore, any canonical rule-graph begins with vertices of the body part, followed by the vertices of the head part.

The following definitions describe the canonical form of rule-graphs by defining a set of interdependent compare operators. The comparison starts with the edges and their labels, goes over the vertices and finally compares whole rule-graphs. The least rule-graph according to this order is the canonical one.

Given two elements x_i and x_j, *the relation* $x_i \prec_R x_j$ is true iff: $(BODY \in \alpha_r(x_i) \wedge BODY \notin \alpha_r(x_j))$

Given two elements x_i and x_j, *the relation* $x_i \prec_L x_j$ is true iff the label $\lambda(x_i)$ of element x_i has a lexicographically lower value than the label $\lambda(x_j)$ of x_j.

Given two elements x_i and x_j, *the relation* $x_i \prec_N x_j$ is true iff: $((x_i = \varnothing) \wedge (x_j \neq \varnothing))$

In the following we use the notation $(a_1 \prec_{X_1} b_1) \rhd (a_2 \prec_{X_2} b_2) \rhd ... \rhd (a_n \prec_{X_n} b_n)$ as shorthand for:

if $(a_1 \prec_{X_1} b_1) \vee (b_1 \prec_{X_1} a_1)$ **then return** $(a_1 \prec_{X_1} b_1)$
else if $(a_2 \prec_{X_2} b_2) \vee (b_2 \prec_{X_2} a_2)$ **then return** $(a_2 \prec_{X_2} b_2)$
...
else if $(a_n \prec_{X_n} b_n) \vee (b_n \prec_{X_n} a_n)$ **then return** $(a_n \prec_{X_n} b_n)$

The $e^a \prec_E e^b$ *relation* defines an order on the edges $e^a = (v_i, v_x)$ and $e^b = (v_j, v_y)$ [1], where $x = y$. The $e^i \prec_E e^j$ *relation* is true iff: $(e^a \prec_R e^b) \rhd (i < j) \rhd (e^a \prec_L e^b)$

The $v_i \prec_{VE} v_j$ *relation* defines an order on the vertices v^i and v^j with regard to their adjacent edges. The edges of these vertices are represented by two sequences $T_i = (e_1^i, e_2^i, ..., e_m^i)$ and $T_j = (e_1^j, e_2^j, ..., e_n^j)$. $T_i = (e^i = (v_x, v_y) \in E_r \mid e^i \preceq_R v_i, (x < i \vee y < i), (x = i \vee y = i))$ These sequences are ordered according to $\prec_E$. The $v^i \prec_{VE} v^j$ relation is true iff: $(e_1^i \prec_E e_1^j) \rhd (e_2^i \prec_E e_2^j) \rhd ... \rhd (e_n^i \prec_E e_n^j) \rhd (|T_i| < |T_j|)$ where $n = min(|T_i|, |T_j|)$

[1] Note that x here uniquely identifies a rule-graph of $\mathcal{M}$

[1] Note that the vertices v_x and v_y may belong to two different variants of the structurally same rule-graph

Given two vertices v_i and v_j, *the relation* $v_i \prec_V v_j$ is true iff: $(v_i \prec_R v_j) \rhd (v_i \prec_{VE} v_j) \rhd (v_i \prec_L v_j)$

Let $v_i, v_j \in V_r$ be two vertices in a rule-graph $G_r = (V_r, E_r, \Sigma, \lambda_r, \alpha_r)$. The rule-graph G_r is *ordered* iff: $\forall v_i, v_j \in V_r : (v_i \prec_V v_j) \leftrightarrow (i \leq j)$

Let G_r^i and G_r^j be two rule-graphs that are ordered. Let further be v_x^i a vertex of rule-graph G_r^i and v_x^j be a vertex of rule-graph G_r^j respectively.

The $G_r^i \prec_G G_r^j$ *relation* is true iff: $((v_1^i \prec_V v_1^j) \rhd (v_2^i \prec_V v_2^j) \rhd ... \rhd (v_n^i \prec_V v_n^j))$ where $n = |V_r|$.

Let L be the set of all ordered syntactic variants (i.e. all possible sequential orderings of vertices in a graph that are ordered) $\{G_r^1, G_r^2, ..., G_r^n\}$ of a rule-graph G_r. The least entry in L according to $\prec_G$ is the *canonical form*[1] of rule-graph G_r. $G_r^{canonical} = \{G_r^i \in L \mid \forall G_r^j \in L : G_r^i \prec_G G_r^j\}$

6 The Algorithm

The main routine of the FRGM algorithm, consists of just two loops which alternately call the core routine in inference and mining mode (see Fig.2).

Input: $\mathcal{D}$: A database of canonical text-graphs
$\mathcal{M}$: The initial model

Output: $\mathcal{D}'$: The result database
$\mathcal{M}$: The result model

```
1:  procedure FRGM(D, D', M)
2:    do
3:      D' ← D, D'' ← D
4:      do
5:        FRGMCore(D', D'', Root(M), INFERE)
6:        D' ← Canonize all text-graphs in D''
7:      while D' changed and not max iter.
8:      FRGMCore(D', ∅, Root(M), MINE)
9:    while M changed and not max iter.
10: end procedure
```

In rule inference mode the FRGM core routine (see appendix) takes a database $\mathcal{D}$ and a model $\mathcal{M}$ as input and computes a new database $\mathcal{D}'$ by applying all the rules in $\mathcal{M}$ on all text-graphs in $\mathcal{D}$. In rule mining mode the algorithm only takes a database $\mathcal{D}$ with example text-graphs as input and computes a new or improved model $\mathcal{M}$ containing all rules that meet the required frequency thresholds.

The FRGM is designed on the basis of three data structures. The first one is the GVST which is used to efficiently store the rule-graphs and their associated refinements. The second one is the embedding tree. A root path in this embedding tree describes the embedding of a rule-graph in the same way as a root path in the GVST describes the rule-graph itself. Each rule-graph node in the GVST can have several associated embedding nodes describing all occurrences of this rule-graph in the database. The third data structure are the *refinement candidates* (RCs). A RC describes a possible extension to an embedded rule-graph and is stored in an ordered set associated with the respective embedding. A RC $\{INPUT, TARGET\} \times \{INPUT, TARGET\} \times V_r \times E_t \times V_t$ is determined by the vertex roles, the starting point within the rule-graph, and an edge and a vertex in the text-graph. The order $\prec_C$ on RCs is defined as follows. Given two RCs $c_i = (R_v^i, R_e^i, v_r^i, e_t^i, v_t^i)$ and $c_j = (R_v^j, R_e^j, v_r^j, e_t^j, v_t^j)$, $c_i \prec_C c_j$ is true iff:

if $(v_r^i = \varnothing) \vee (v_r^j = \varnothing)$ **return** $(R^i \prec_R R^j) \rhd (v_r^i \prec_N v_r^j) \rhd (v_t^i \prec_L v_t^j)$
else return
$(R_v^i \prec_R R_v^j) \rhd (R_e^i \prec_R R_e^j) \rhd (v_r^i \prec_V v_r^j) \rhd (e_t^i \prec_L e_t^j) \rhd (v_t^i \prec_L v_t^j)$

By using RCs we avoid that refinements have to be considered that would lead to unordered (see definition in section 5) and therefore non-canonical rule-graphs. Of coarse, there may be more than one ordered variant of any given rule-graph. Thus, we still need to perform a complete canonical verification.

The FRGMCore routine starts with an initial database scan (ln:2-3) where for each text-graph in $\mathcal{D}$ an empty root embedding node is created. In the next step the algorithm calls the RC generation subroutine, where for all embeddings associated with the current rule-graph G_r the RCs are computed. These RCs are then converted to actual refinements (ln:5-7). The next section (ln:8-13) is the recursive step of the algorithm leading to the next level in the GVST. In line:9 the frequency threshold is checked. In inference mode a frequency of at least 1 is required while iterating the body parts of the rule-graphs. No embeddings are required while iterating the head parts of the rule-graphs, since those are the parts of the rules that we are going to add to the text-graphs. In inference mode the last step is to apply the rule-graph G_r on all occurrences in $\mathcal{D}$ (ln:14-15).

The routine GenerateRCs(G_r) is responsible for the creation of RCs for the embeddings. For the root level only initial RCs are created that do not yet have an edge or a starting point in the rule. All later RCs start from an existing rule-graph and describe a possible extension by an edge and a vertex. RCs for edges that already exist in the rule-graph are filtered out (ln:8).

The refinement operator converts the RCs into new refinements and embeddings for these refinements. Furthermore, the RCs are partially relayed (ln:13) to this next level of embedding nodes. That is, only those RCs are relayed that are greater than the current RC according to $\prec_C$ because all other RCs would lead to non-canonical rule-graphs, anyway.

7 Conclusions and Future Work

In this paper we have introduced text-graphs as a new way to represent text interpretations consisting of syntactic and semantic annotations as well as interrelating concepts. Furthermore, we have presented the FRGM algorithm which is able to extract knowledge from text-graphs and to apply it to new text-graphs. Our first experimental results suggest that the expressiveness of text- and rule-graphs allows to process effectively even complex linguistic tasks.

At the moment, we have only some preliminary experimental results. Using a generative grammar, in our first experiments we have created a test corpus consisting of 320 example datasets. On this corpus we were able to enumerate the 312378 frequent (given $\mu(G_r) = |G_r| * 2 + 6$) rule-graphs in 56 seconds (on an AMD Athlon XP 2000+). For this test we have limited the number of unlabeled rule-graph vertices to 3 because otherwise the number of frequent rule-graphs grows too large. We are going to perform experiments on large, real-world text datasets as well.

One interesting extension to the FRGM algorithm would be to introduce non-monotonic inference. A dis-

[1]The implementation of the canonical verification algorithm can be found at: http://www.ais.fhg.de/~lmolz

advantage of the threshold based rule-graph evaluation is that large rule-graphs require high frequencies and therefore many training examples. One way to overcome this problem could be the use of learning patterns. A learning pattern is a subgraph of both a text-graph and a candidate rule-graph which itself is embedded in the text-graph, too. Here, the learning pattern acts as a kind of template for this candidate rule-graph and allows to reduce the required minimum frequency threshold for the rule-graph.

Acknowledgments

I would like to thank Markus Ackermann, Thomas Gärtner, Tamas Horvath and Codrina Lauth, for their valuable comments and support.

References

[1] C.L. Forgy. RETE: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, pages 17-37, 19, 1982.

[2] I. Jonyer, L. Holder, and D. Cook. Concept formation using graph grammars. In *Proceedings of the KDD Workshop on, Multi-Relational Data Mining*, 2002.

[3] J. Lyons (1968): Introduction to Theoretical Linguistics. Cambridge [dt. (1995): Einführung in die moderne Linguistik. 8. Aufl., München]

[4] U. Rückert, S. Kramer: Generalized Version Space Trees. KDID 2003: 119-129

[5] Wikipedia: http://en.wikipedia.org/wiki/Natural_language_processing

[6] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining* (ICDM 2002), pages 721-724, 2002.

Appendix - The Algorithm

Input: $\mathcal{D}$: A database of text-graphs
G_r : The root rule of $\mathcal{M}$
$mode$: The mode $\{INFERE, MINE\}$
Inference Mode Output: $\mathcal{D}^{'}$: The result database
Mining Mode Output: The improved model $\mathcal{M}$

procedure FRGMCore($\mathcal{D}, \mathcal{D}^{'}, G_r, mode$)
  if G_r is the root node of the GVST **then**
    $Emb(G_r) \leftarrow \{(\varnothing, G_t, (\varnothing, \varnothing)) \mid \forall G_t \in \mathcal{D}\}$
  GenerateRCs(G_r)
  for all embedding nodes $emb = (emb^{parent}, G_t, (v_r, v_t)) \in Emb(G_r)$ **do**
    for all refinement candidates $c_{it} \in RefCand(emb)$
      RefinementOperator($G_t, G_r, emb, c_{it}, mode$)
  for all GVST nodes $(ref = (v_r, v_r^{'}, l_v, l_e, r_v, r_e), G_r^{'}) \in Ref(G_r)$ **do**
    if $(mode = MINE \wedge (count^{+}(G_r^{'}) \geq \mu(G_r^{'})))\vee$
      $(mode = INFERE \wedge ((count^{+}(G_r^{'}) > 0) \vee (r_v = HEAD \wedge (v_r = \varnothing \vee r_e = HEAD))))$ **then**
      if $count^{-}(G_r^{'}) \leq freq_{min}$ **then** activate $G_r^{'}$
      Fill the refined rule-graph $G_r^{'}$ based on its parent G_r and the refinement ref.
      if $G_r^{'}$ is in its canonical form **then** FRGMCore($\mathcal{D}, \mathcal{D}^{'}, G_r^{'}, mode$)
    end if
  end for
  if $mode = INFERE$ and G_r is active **then**
    Apply the rule G_r on all embeddings in $Emb(G_r)$ and add the results to $\mathcal{D}^{'}$.
end procedure

procedure GenerateRCs(G_r)
  if $mode = INFERE$ **then** $R \leftarrow \{\{INPUT\}\}$ **else** $R \leftarrow \{\{INPUT\}, \{TARGET\}\}$
  for all $(emb^{parent}, G_t, (v_r, v_t)) = emb \in Emb(G_r)$ **do**
    if G_r is the root node in the GVST **then**
      $C \leftarrow \{(R_v, \emptyset, \varnothing, \varnothing, v_t^{'}) \mid \forall v_t^{'} \in V_t(G_t), \forall R_v \in R : R_v \subseteq \alpha_t(v_t^{'})\}$
    else
      $C \leftarrow \{(R_v, R_e, v_r, e_t, v_t^{'}) \mid \forall e_t = (v_t, v_t^{'}) \in E_t(G_t), R_v \in R, R_e \in R : R_v \subseteq \alpha_t(v_t^{'}), R_e \subseteq \alpha_t(e_t)\}$
      $C \leftarrow \{(R_v, R_e, v_r, e_t, v_t^{'}) \in C \mid \neg\exists e_r = (v_r, v_r^{'}) \in E_r(G_r) : v_t^{'} = \varphi(v_r^{'})\}$
    end if
    $RefCand(emb) \leftarrow RefCand(emb) \cup C$
  end for
end procedure

φ : Is an isomorphic embedding derived from the root path of emb. (See section 3)
δ : Is the role mapping function (See section 3)
$c_{it} = (R_v, R_e, v_r, e_t, v_t)$

procedure RefinementOperator($G_t, G_r, emb, c_{it}, mode$)
  $v_r^{'} \leftarrow \varphi^{-1}(v_t)$
  if $v_r^{'} = \varnothing$ **then** $v_r^{'} \leftarrow$ **new** $Vertex$
  else if $(v_r \succ_V v_r^{'})$ **then return**
  $ref \leftarrow (v_r, v_r^{'}, \lambda_t(v_t), \lambda_t(e_t), \delta^{-1}(R_v), \delta^{-1}(R_e))$
  $G_r^{'} \leftarrow \{G_r^{'} \mid \exists (ref, G_r^{'}) \in Ref(G_r)\}$
  if $(mode = MINE) \wedge (G_r^{'} = \varnothing)$ **then**
    $G_r^{'} \leftarrow$ **new** $RuleGraph$
    $Ref(G_r) \leftarrow Ref(G_r) \cup \{(ref, G_r^{'})\}$
  end if
  $emb^{'} \leftarrow (emb, G_t, (v_r^{'}, v_t))$
  $Emb(G_r^{'}) \leftarrow Emb(G_r^{'}) \cup \{emb^{'}\}$
  if $v_r \neq \varnothing$ **then** $RefCand(emb^{'}) \leftarrow \{c \in RefCand(emb) \mid c_{it} \prec_C c\}$
  Count the embedding $emb^{'}$ as occurrence of $G_r^{'}$.
end procedure