On Generalization / Specialization for Conceptual Graphs

M.L. Mugnier
LIRMM-CNRS
161, rue Ada, 34392 Montpellier Cedex 5, France,
tel:(33)67418549, fax: (33)67418500, email: mugnier@lirmm.fr

Research Report LIRMM 93-003 (January 93)

Abstract

This paper centers on generalization/specialization relation in the framework of conceptual graphs (this relation corresponds to the logical subsumption when considering the logical formulas associated with conceptual graphs). Results given here apply more generally to any model where knowledge is described by labelled graphs and reasoning is based on graph subsumption, like in semantic networks or in structural machine learning. The generalization/specialization relation, as defined by Sowa, is first precisely analyzed, especially its links with a graph morphism, called projection. Besides Sowa's specialization relation (which is a preorder), another one is actually used in some practical applications (which is an order). Both are comparatively studied. The second topic of this paper is the design of efficient algorithms for computing these specialization relations. Since the associated problems are NP-hard, the form of the graphs is restricted in order to come to polynomial algorithms. In particular, polynomial algorithms are presented for computing a projection from a conceptual "tree" to any conceptual graph, and for counting the number of such projections. The algorithms are also described in a generic way, replacing the projection by a parametrized graph morphism, and conceptual graphs by directed labelled graphs.

This report is to appear in the Journal of Experimental and Theoretical Artificial Intelligence, vol.7, pp. 325-344, (1995).

1 Introduction

Conceptual graphs (Sowa 1984) form a knowledge representation model, that can be viewed as a kind of semantic network (see (Lehman 1992) for a synthesis of the main semantic network families). In this paper, we only consider a subset of this model. A conceptual graph is then a connected bipartite graph, one vertex class (called "concepts") representing entities, the other one (called "relations") representing relationships between these entities. Vertices are labelled by types. Concept types are partially ordered by an Is-a-kind-of link, which is coded separately from the conceptual graphs. In some applications, relation types are also ordered, which will not be considered here. With each conceptual graph, one can associate a first order logic formula, which has a simple form: it is a conjunctive formula without functions and existentially closed. Roughly said, predicates stand for relation vertices, and variables or constants stand for concept vertices.

When restricting relations to binary relations, one obtains the simplest form of a semantic network: a directed graph, provided with type labels on vertices (concepts) and edges (relations). Results on conceptual graphs can be translated to semantic networks through this correspondence.

The key computational notion in every reasoning concerning conceptual graphs is the *generalization / specialization* relation defined on the conceptual graph set. When considering the logical interpretation of conceptual graphs, it corresponds to logical subsumption. When considering graph operations, it can be computed by a particular graph morphism. This relation is the root of most sophisticated tools in conceptual graph based systems, e.g. hierarchical structuring of a conceptual graph database (Ellis 1991) (Levinson and Ellis 1991), or question-answering systems (Berard-Dugourd *et al.* 1988) (Levinson 1992).

This paper is devoted to the analysis of (part 2), and algorithms for (part 3) generalization / specialization. Results given here apply to any framework where knowledge is described by labelled graphs, and the subsumption operation can be translated into graph morphism, e.g. this is typically the case for structural machine learning (for instance (Mitchell 1982) (Haussler 1989)).

One original aspect of Sowa's model is that the generalization / specialization relation can be defined in terms of three kinds of operations: a set of *graph formation rules*, a set of *logical rules* (applying to the logical formulas associated with conceptual graphs), and a particular graph morphism, called *projection*.

Formation rules are elementary graph transformations. Given two conceptual graphs, G and H, G is said to be a specialization of H, if G can be obtained from H, and possibly other graphs, by a composition of formation rule applications. These graph operations ensure the same reasoning power as logical deduction. Indeed, when considering the logical interpretation of conceptual graphs, formation rules can be interpreted as specialization rules, or refutation rules (Sowa 1984), that is, if G is a specialization of H, then the interpretation of G logically implies the interpretation of H. Then the dual of specialization rules, say generalization rules, are inference rules. (Chein and Mugnier 1992) proves that these generalization rules constitute a *complete* set of inference rules on the set of logical formulas associated with conceptual graphs.

Projection can be seen as a global view of a specialization operation sequence: G is a specialization of H *iff* there is a projection from H to G (as proved in part 2.3). Nevertheless, there is a certain ambiguity around projection in practical applications. Indeed, many conceptual graph systems use projection in a restricted form, by enforcing an injectivity constraint. Roughly said, when using injective projection, one implicitly assumes that distinct vertices of a graph necessarily represent distinct individuals. Such an *injective projection* may define a second specialization relation. These two specialization relations are studied by pointing out some consequences of the choice of one or the other (parts 2.4 and 2.5). The specialization induced by injective projection partially orders conceptual graphs. The specialization defined by projection is not a (partial) order, because it is not antisymmetric. In other words, non-identical conceptual graphs can be equivalent. A normal form is introduced for a conceptual graph, called an *irredundant* graph, so that the specialization relation becomes an order on the set of irredundant conceptual graphs.

From an algorithmic viewpoint (part 3), all interesting problems associated with both kinds of projections are NP-hard. Some polynomial cases obtained by restricting the *form* of the graphs are explored. These polynomial cases are based on the *tree* notion (a conceptual tree is roughly a conceptual graph without cycles).

Algorithms are described for computing a projection from a conceptual tree to a general conceptual graph, for counting or enumerating these projections (part 3.1). Injective projection is then focused on (part 3.2).

A practical interest of this study, is that trees seem to be very frequent in conceptual graph applications. Furthermore, presented algorithms can be used as a preprocessing part for a general projection algorithm (part 3.3).

Algorithms are also described as applied to directed labelled graphs, which, as previously noted, is a specific case of conceptual graphs, and in a generic fashion with a parametrized morphism, projection being a specific "instance".

2 Analysis on specialization relations

Parts 2.1 and 2.2 introduce the framework, namely *well-formed* conceptual graphs related to a *support*, *derivation* of a conceptual graph, and *canonical graphs*. These definitions extend the work of (Sowa 1984).

2.1 Well-formed conceptual graphs

At a very elementary level, a conceptual graph is defined in relation to a *support*, which provides background knowledge on a specific domain application.

A support consists of (Figure 1):

- T_c, a set of *concept types*, structured in a lattice, with $\top$ as supremum (the universal type) and $\perp$ as infimum (the absurd type),

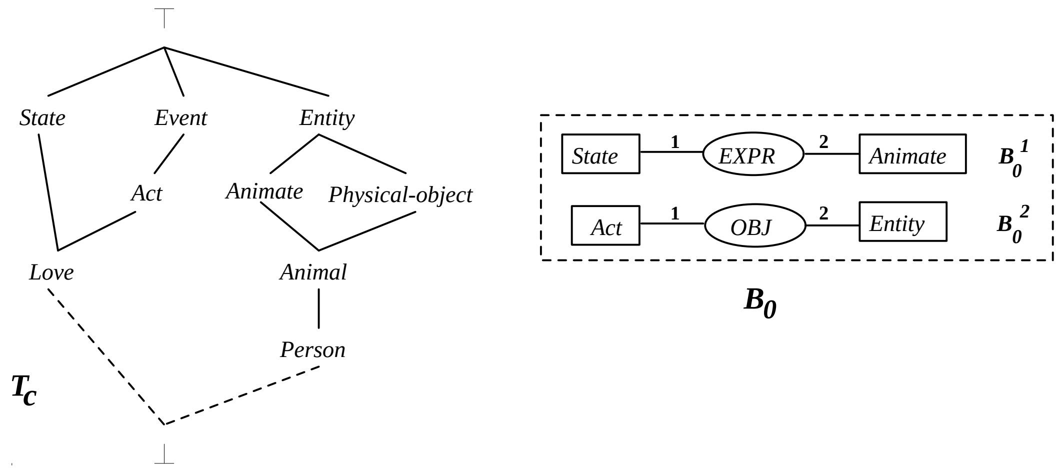
- T_r, a set of *relation types*, which is not structured — but could be partially ordered, see for example (Kocura and Kwong Ho 1991) or (Sabat and Vilnat 1991),
- a set of *markers* for concept vertices: one generic marker $*$ and a set M of individual markers which allow naming of distinct entities; $M \cup \{*\}$ is provided with the order, such that $*$ is greater than all markers, and any two individual markers are non-comparable,
- a signature for each relation type, represented with a *star graph*; each star graph fixes the arity of a relation type and shows the greatest concept types this relation type can link. The set of all star graphs is called the *basis* of the support, and denoted by $\mathcal{B}_0$.

A *conceptual graph* $G = (R, C, U, lab)$ is a bipartite, connected, finite graph. R and C denote the two classes of relation and concept vertices, also called r-vertices and c-vertices. U is the set of edges, and the edges adjacent to each r-vertex are totally ordered. Every vertex has a label defined by the mapping lab. If $c \in C$, then $lab(c) \in T_c \times M \cup \{*\}$; if $r \in R$, then $lab(r) \in T_r$ (Figure 2).

G is said to be *well-formed* relative to a support S if it conforms to the constraints defined by the star graph set of S.

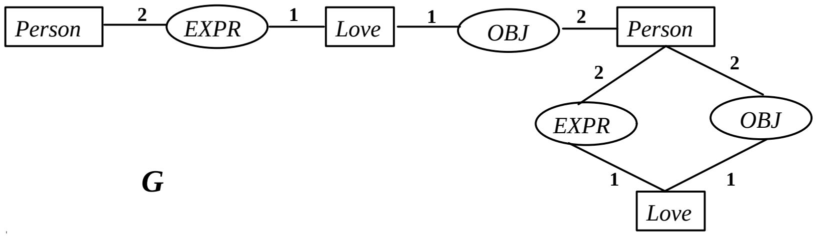
The set of all possible labels for c-vertices is partially ordered by the product of the orders on T_c and $M \cup \{*\}$ — *i.e.* given two labels $l = (t, m)$ and $l' = (t', m')$, $l \leq l'$ if $t \leq t'$ and $m \leq m'$.

Any star graph of $\mathcal{B}_0$ can be identified with the naturally associated conceptual graph: B_{t_r}, the star graph of the relation type t_r, is a well-formed conceptual graph, having a unique r-vertex labelled t_r, whose neighbours are generic c-vertices.



This partially defined support is built from the lexicon of (Sowa 1984).

Figure 1: Partial definition of a support



By default, all markers are generic ones. This graph could represent the sentence "a person x loves a person y, which is an egoist person" (where an egoist person is someone that loves him/herself).
Its translation into first order logic is
$(\exists x \exists y \exists z \exists t Person(x) \wedge Person(y) \wedge Love(z) \wedge Love(t) \wedge Expr(z, x) \wedge Obj(z, y) \wedge Expr(t, y) \wedge Obj(t, y))$

Figure 2: Well-formed graph relative to the support of Figure 1

2.2 The specialization relation ($\leq$)

2.2.1 Derivation of conceptual graphs

Let us now recall Sowa's formation elementary rules, which are operations on the well-formed conceptual graph set ((Sowa 1984), 3.4.3).

- *Simplify* — deleting a duplicate r-vertex (*i.e.* an r-vertex r such that there is another r-vertex with the same type and exactly the same i-ith neighbour, for all $i \in \{1, \ldots, degree(r)\}$).
- *Restrict* — decreasing a c-vertex label,
- *Join* — merging two c-vertices with the same label.

  The join is decomposed in two operations, depending on whether the joined c-vertices belong to the same graph (*Internal* join) or distinct graphs (*External* join).

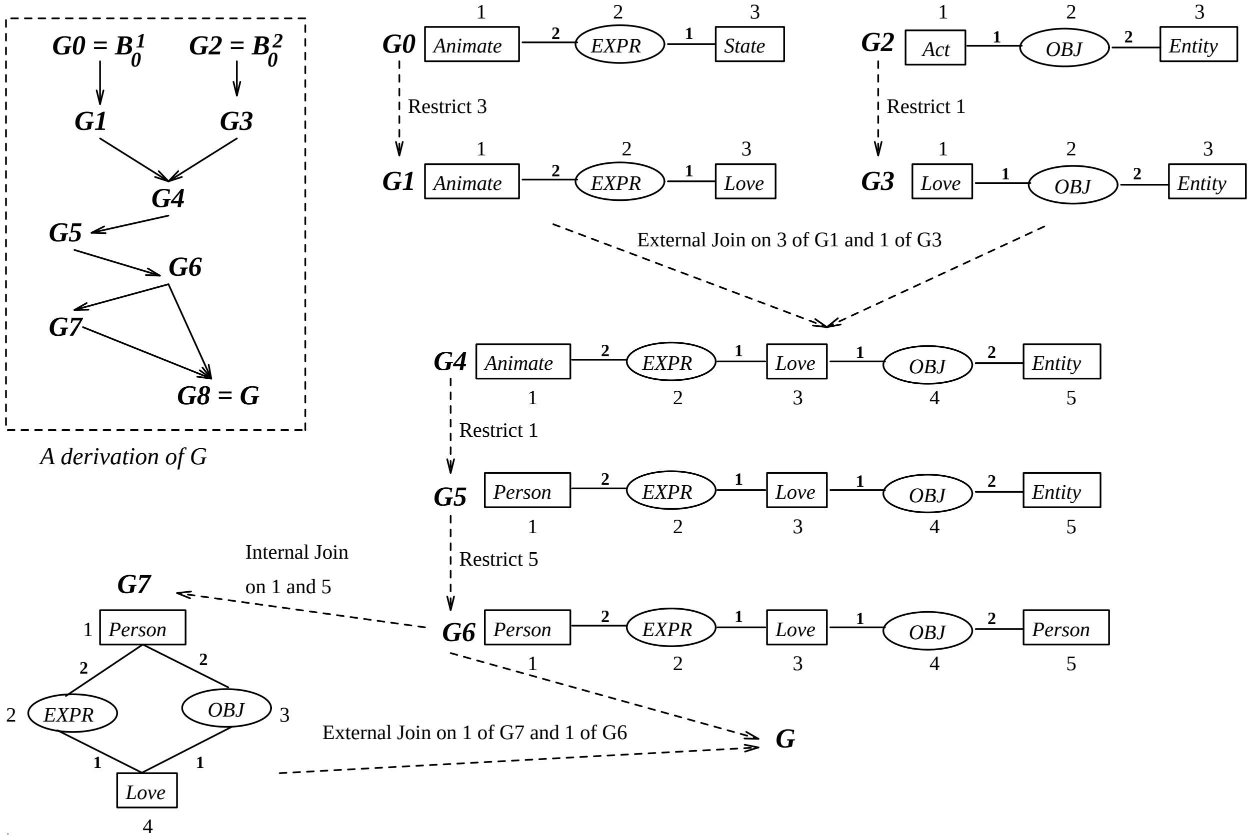
A new conceptual graph (say G) can be derived from others by a composition of formation rules. G is said to be a *specialization* of every conceptual graph H from which it is derived, denoted $G \leq H$. More precisely (Figure 3):

A *derivation graph* of a conceptual graph G is a finite directed graph, without circuits, whose vertices are labelled by conceptual graphs, such that:

1. there is a unique sink labelled by G,
2. let x be any vertex with label H; then,
   (a) x is a source, and H is a well-formed graph, *or*
   (b) x has one predecessor labelled K, and H is obtained from K by a simplify, a restrict or an internal join, *or*
   (c) x has two predecessors, respectively labelled K_1 and K_2, and H is obtained from K_1 and K_2 by an external join.

G is a *specialization* of every conceptual graph H appearing in the derivation graph.

A *derivation sequence*, as used by Sowa (Sowa 1984), is then defined in terms of a derivation graph. Given a derivation graph $\mathcal{D}$ of G, consider any *topological sorting* x_0 ... x_n of $\mathcal{D}$ (*i.e.* any total ordering x_0 ... x_n of $\mathcal{D}$ vertex set, such that, for every edge ab of $\mathcal{D}$, $a = x_i$ and $b = x_j$ implies $i < j$). The label sequence ($G_0 = label(x_0)$ ... $G_n = label(x_n)$) is a derivation sequence of G (Figure 3).



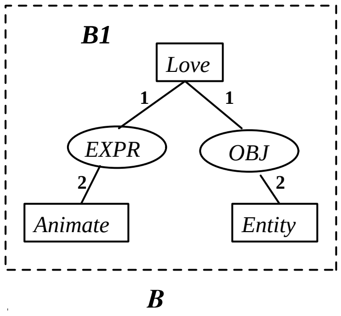
G is the graph of Figure 2.
B_0^1 and B_0^2 are relative to the support of Figure 1.
(G_0 G_1 ... G_8) is a derivation sequence of G

Figure 3: Derivation graph of a conceptual graph

2.2.2 Extension to canonical graphs

Sowa introduced in (Sowa 1984) the notion of a *canonical basis*, which is a set of distinguished conceptual graphs. The constraints these graphs implicitly define are preserved under the formation rules. The canonical basis is then used as a generator set, to derive conceptual graphs, which are said to be *canonical*.

A *canonical basis* $\mathcal{B} = \{B_1, \ ... \ , \ B_n\}$ is a finite set of well-formed conceptual graphs. The set of canonical graphs is defined as the *closure* of the canonical basis under the formation rules.



$\mathcal{B}$ is a canonical basis relative to the support of Figure 1.
Figure 3 shows that B_1 is a well-formed graph, since $B_1 = G_4$.

Figure 4: A canonical basis with one element

Note that the notion of a canonical basis is a generalization of $\mathcal{B}_0 \cup \{[\top : *]\}$, where $\mathcal{B}_0$ is the star graph set of a support S and $[\top : *]$ is the universal type generic c-vertex. Indeed, given any well-formed graph G, if G is restricted to a single c-vertex, it is derived by restriction from $[\top : *]$, otherwise it can be derived from $\mathcal{B}_0$. Reciprocally, if a conceptual graph G is derived from $\mathcal{B}_0 \cup \{[\top : *]\}$, then G is well-formed since $[\top : *]$ and all elements of $\mathcal{B}_0$ are well-formed graphs, and the formation rules retain the property of being well-formed.

Property 1 *Given any conceptual graph G defined on a support S, G is well-formed relative to S* iff *it can be derived from the set $\mathcal{B}_0 \cup \{[\top : *]\}$.*

A *canonical derivation graph* is a derivation graph — as defined in 2.2.1 — with all sources belonging to $\mathcal{B}$. A canonical graph can also be defined as a graph with canonical derivation graph. E.g., in Figure 3, the subgraph induced by $\{G_4, \ G_5, \ G_6, \ G_7, \ G_8\}$ is a canonical derivation graph of G (graph of Figure 2) relative to $\mathcal{B}$ (base of Figure 4).

2.3 Projection and specialization ($\leq$)

Let us first recall the definition of a graph morphism in a classical graph theory sense. A *morphism* from a directed graph $G_1 = (X_1, U_1)$ to a directed graph $G_2 = (X_2, U_2)$, where X and U denote the vertex set and the edge set respectively, is a mapping Π from X_1 to X_2 such that for every edge xy of U_1, $\Pi(x)\Pi(y)$ is an edge of U_2. $\Pi(G_1)$ denotes the subgraph of G_2, which is the image of G_1 for Π.

Projection is a graph morphism, keeping the order on the edges, and provided with additional rules on labels.

A *projection* Π from a conceptual graph $G_1 = (R_1, C_1, U_1, lab_1)$ to a conceptual graph $G_2 = (R_2, C_2, U_2, lab_2)$ is an ordered pair $\Pi=(f,g)$ of mappings, f from R_1 to R_2, and g from C_1 to C_2, such that

1. For every edge rc of U_1 labelled i, $f(r)g(c)$ is an edge of U_2 labelled i,
2. For all $r \in R_1$, $lab_1(r) = lab_2(f(r))$,
3. For all $c \in C_1$, $lab_1(c) \geq lab_2(g(c))$.

In the following, a projection is simply noted Π, with an implicit decomposition into f and g. Sowa proved that, given two conceptual graphs G and H, if $G \leq H$ then there is projection from H to G (Sowa 1984). The proof holds true for canonical graphs. Here the reciprocal property for canonical graphs is provided.

Theorem 1 *Given two canonical graphs G and H, $G \leq H$* iff *there is projection from H to G.*

Proof — With following lemmas 1 and 2.

Lemma 1 *Given two canonical graphs G and H, if $G \leq H$ then there is projection from H to G.*

Proof — (see also (Sowa 1984), theorem 3.5.4). A specific projection from G_1 to G_2 can be associated with each elementary specialization rule producing G_2 from G_1. (For the following proofs, note that the projections associated with simplify, restrict and internal join are surjective; in the case of an external join leading from G_1 and G_2 to G_3, let $f_1 : G_1 \rightarrow G_3$ and $f_2 : G_2 \rightarrow G_3$ be the associated projections, then $f_1(G_1) \cup f_2(G_2)$ is exactly G_3).

Consider a derivation graph of G where H occurs; any path from H to G defines a projection, obtained by composition of the specific projections associated with each edge of the derivation graph.
□

Lemma 2 *Given two canonical graphs G and H, if there is projection from H to G then $G \leq H$.*

Let us denote by α the set of "unary operations" on conceptual graphs, *i.e.* simplify, restrict, and internal join. Given any set $\mathcal{G}$ of conceptual graphs, $\mathcal{G}^\alpha$ denotes the closure of $\mathcal{G}$ under α.

Lemma 2.1 *Given two conceptual graphs H and K, $K \in \{H\}^\alpha$ (i.e. K can be derived from H with a sequence of unary operations)* iff *there is a surjective projection from H to K.*

Proof — If $K \in \{H\}^\alpha$, let us consider a derivation graph from H to K (it is a path). A projection is associated with this path, and it is surjective, since all involved "elementary" projections are surjective (see proof of Lemma 1).
Reciprocally, let Π be a surjective projection from H to K. The following specialization sequence can be associated with Π.
(1) For every c-vertex c of H, restrict the label of c to the label of $\Pi(c)$. One obtains H_1.

(2) Successively join all c-vertices of H_1 having the same image in K. More precisely, for every c-vertex c of K, if $\Pi^{-1}(c) = \{c_1, \ldots, c_p\}$, then merge the c-vertices of H_1 corresponding to the c_i, by doing $p-1$ internal joins (these H_1 c-vertices have the same label). One obtains H_2.
(3) Successively simplify all r-vertices of H_2 having the same image in K. More precisely, for every r-vertex r of K, if $\Pi^{-1}(r) = \{r_1, \ldots, r_q\}$, then merge the r-vertices of H_2 corresponding to the r_i, by doing $q-1$ simplifications (these H_2 r-vertices are duplicate vertices). One obtains K.
□

A *cover* of a graph G is a set of subgraphs of G such that every vertex and every edge of G appears in at least one of these subgraphs. In the case of conceptual graphs, only connected subgraphs are considered.

Lemma 2.2 *A well-formed graph G is canonical, relative to a basis $\mathcal{B}$,* iff *there is a cover of G with elements of $\mathcal{B}^\alpha$.*

Proof —

$\Rightarrow$ Consider a canonical derivation graph of G. Each maximal path of this derivation graph, leading from a source $B_i \in \mathcal{B}$ to G, defines a projection from B_i to G, say Π (Lemma 1). Then:
(a) $\Pi(B_i)$, the image of B_i in G, is an element of $\{B_i\}^\alpha$, since there is a surjective projection from B_i to $\Pi(B_i)$ (Lemma 2.1).
(b) The set of the image graphs associated with all maximal paths of the derivation graph is a cover of G. It is proven by induction on the vertex number of the derivation graph, say n. (b) is true for $n = 1$. Suppose (b) is true for all derivation graphs with $n < p$. Consider $n = p$. If G has one predecessor H, let f be the elementary projection from H to G; since the derivation subgraph with sink H has a vertex number strictly smaller than p, then a cover of H, say $\{H_1, \ldots, H_q\}$, is built from the maximal paths of the derivation subgraph. $\{f(H_1), \ldots, f(H_q)\}$ is a cover of G, because f is surjective. If G has two predecessors H and K, let f_1 and f_2 be the elementary projections from H to G, and from K to G respectively; let $\{H_1, \ldots, H_q\}$ and $\{K_1, \ldots, K_{q'}\}$ be the covers of H and K respectively. Then $\{f_1(H_1), \ldots, f_1(H_q)\} \cup \{f_2(K_1), \ldots, f_2(K_{q'})\}$ is a cover of G, because $f_1(H) \cup f_2(K) = G$.

$\Leftarrow$ Let us consider a cover of G with $(n+1)$ elements of $\mathcal{B}^\alpha$. It can be supposed, without loss of generality, that there is an ordering $(C_0, \ldots, C_n)$ of this cover, such that, for all $1 \leq i \leq n$, C_i shares at least one c-vertex with $\bigcup_{j=0}^{i-1} C_j$. Indeed, such ordering must exist since G is a connected graph. Given this ordering, a graph sequence $(G_0 \ldots G_n)$ can be built with $G_0 = C_0$ and, for all $1 \leq j \leq n$, G_j is obtained from G_{j-1} and C_j by merging the vertices of G_{j-1} and C_j having the same image in G. In other words, G_j is the subgraph of G induced by the union of G_{j-1} and C_j vertex sets. Then $G_n = G$. A canonical derivation graph of G can be constructed from the sequence $(G_0 \ldots G_n)$, by decomposing the subgraph unions in elementary formation operations (each union can be replaced by an external join, an internal join sequence, then a simplify sequence), and substituting each C_j with one of its canonical derivation graphs.
□

Proof of Lemma 2 — Consider a projection Π from H to G. Since G is canonical, there is a cover of G with elements of $\mathcal{B}^\alpha$, say $\mathcal{C}$ (Lemma 2.2). Let us consider $\mathcal{C}' \subseteq \mathcal{C}$, the subset of graphs sharing at least one c-vertex with $G \setminus \Pi(H)$. The subgraph G' of G covered by $\mathcal{C}'$ may have several connected components, say G'_1 ... G'_p. Each of these G'_i is a canonical graph.
A canonical derivation graph of G can be built as follows.
(1) Consider a graph sequence (G_0 ... G_p), such that each G_j, $0 \leq j \leq p$, is a subgraph of G, $G_0 = \Pi(H)$, $G_p = G$, and for $1 \leq j \leq p$, G_j is obtained from G_{j-1} and G'_j by merging the common subgraphs of G_{j-1} and G'_j; with this sequence is associated a derivation graph of G, say $\mathcal{D}_1$, having sources G'_1, ..., G'_p and $\Pi(H)$,
(2) consider a canonical derivation graph of H, then, complete it by performing a sequence of unary operations leading from H to $\Pi(H)$ (Lemma 2.1); a canonical derivation graph of $\Pi(H)$ is obtained;
(3) build $\mathcal{D}_2$ by substituting, in $\mathcal{D}_1$, $\Pi(H)$ with its canonical derivation graph built in (2), that is identifying the node labelled $\Pi(H)$ in $\mathcal{D}_1$ and the sink of $\Pi(H)$ derivation graph;
(4) build $\mathcal{D}_3$ by substituting, in $\mathcal{D}_2$, each G'_i with one of its canonical derivation graphs.
$\mathcal{D}_3$ is a canonical derivation graph of G and it contains H. Thus, $G \leq H$.
□

Let us introduce some specific projections.

A projection Π from G_1 to G_2 is an *isoprojection* if it is bijective. G_2 can be obtained from G_1 with a sequence of restrict.

A projection Π from G_1 to G_2 is a *conceptual graph morphism* if it fulfills "for all $c \in C_1$, $lab_1(c) = lab_2(\Pi(c))$". $\Pi(G_1)$ can be obtained from G_1 with a sequence of internal join and simplify.

A conceptual graph morphism is an *isomorphism* if it is bijective. For the formation rules, G_1 is considered to be identical to G_2. Note that, due to the fixed arity for a given relation type, a conceptual graph isomorphism can also be defined as a conceptual graph morphism whose reciprocal is a conceptual graph morphism. This does not work for the isoprojection, since c-vertex labels can be restricted.

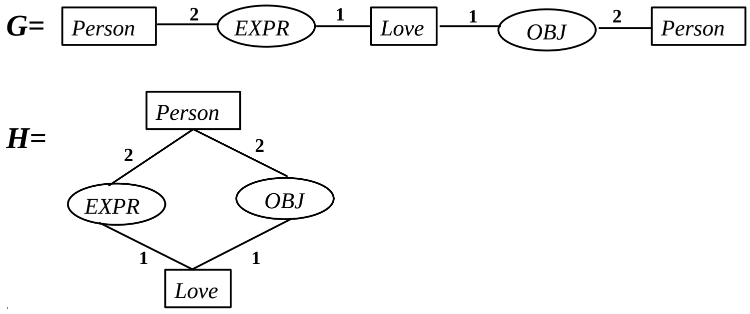
E.g. in Figure 3, there is an isoprojection from G_4 to G_6, an injective morphism from G_1 to G_4, and a non-injective morphism from G_6 to G_7.

The decision problem associated with the projection is NP-complete. The same holds for the isoprojection and morphism. The conceptual graph isomorphism problem is equivalent to the isomorphism problem on classical graphs (Chein and Mugnier 1992). Recall that this latter problem is still unclassified and that its class is called "isomorphism complete".

2.4 Injective projection and specialization ($\leq_i$)

Many applications use projection in a restricted form, which is *injective projection* (see for instance (Levinson and Ellis 1991) and (Fargues *et al.* 1986)). Then, given two conceptual graphs G and H, G is said to be a specialization of H *iff* there is an injective projection from H to G (in other words, there is an isoprojection from H to a subgraph of G). Let us denote this new specialization relation by $\leq_i$. The change between $\leq$ and $\leq_i$ is not slight, from several viewpoints.

From a semantic viewpoint, $\leq_i$ drastically restricts the comparability between two graphs. Indeed, two distinct c-vertices necessarily represent distinct entities (Figure 5). Note the use of $\leq_i$ supposes that, for any conceptual graph, labelling mapping is injective on the individual marker set (*i.e.* there is at most one c-vertex with a given individual marker).



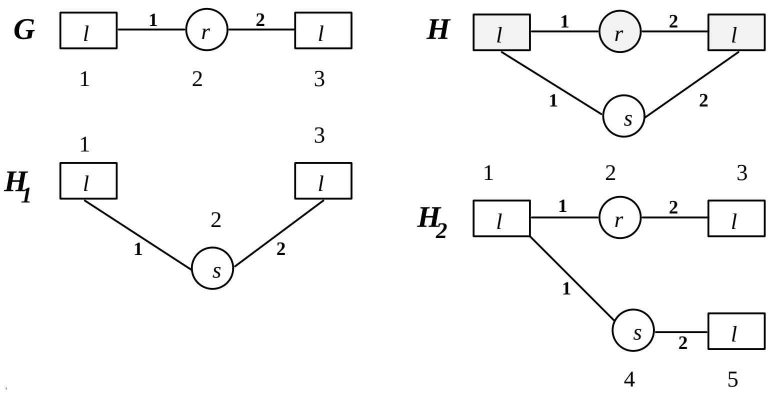
G could represent the sentence "A person x loves a person y". For $\leq_i$, x and y are distinct individuals. H could represent the sentence "A person x loves itself". $H \leq G$, but G and H are not comparable for $\leq_i$.

Figure 5: $\leq$ and $\leq_i$

Correspondence with the formation rules is lost. Indeed, consider two conceptual graphs G and H, such that $H \leq_i G$, and an injective projection, say Π, from G to H. $\Pi(G)$ can be obtained from G by a sequence of restrict; no external join is needed, and it is forbidden to perform an internal join or a simplify. But, to obtain H from $\Pi(G)$, at least external and internal joins are needed (see Figure 6).

Furthermore $\leq$ and $\leq_i$ do not have the same structure, as detailed in the next section: $\leq_i$ provides an order on conceptual graphs, while $\leq$ does not, because it is not antisymmetric.

Let us add that the decision problem associated with the injective projection is also NP-complete. But $\leq$ and $\leq_i$ do not behave the same relative to some restricted forms of graphs (see Figure 8).



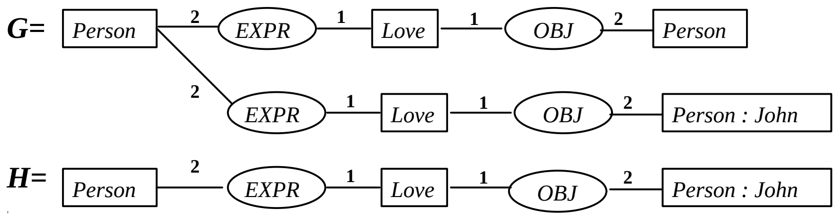
Consider Π, the unique projection from G to H. $\Pi(G)$ (represented in gray) is here isomorphic to G. Obtaining H from G (or from $\Pi(G)$) needs at least an external join (for instance, join the c-vertices 1 of G and 1 of H_1, leading to H_2), then an internal join (join the c-vertices 3 and 5 of H_2, leading to H).

Figure 6: $\leq_i$ and derivation rules

2.5 Structure of specialization relations

2.5.1 The structure of $\leq$

$\leq$ is a transitive relation, since the serial composition of two derivations is a derivation. It is also reflexive since a derivation can be restricted to a single element. Thus, $\leq$ is a *preorder*. As several times remarked (Ellis 1991) (Jackman 1988), it is not an order because the antisymmetry property is not fulfilled (Figure 7). An natural question is then to measure the difficulty of coming back to an order. This aspect has been studied in (Chein and Mugnier 1992). We recall here the main results.



$G \leq H$ and $H \leq G$

Figure 7: $\leq$ is not an order

If $G \leq H$ and $H \leq G$, then G and H are said to be equivalent, denoted $G \equiv H$. If $G \leq H$ and $G \not\equiv H$, then G is said to be strictly more specific than H. A graph G is an *irredundant* graph if it has no strict subgraph equivalent to it. Thus G is irredundant *iff* there is no projection from G to one of its strict subgraphs. (Chein and Mugnier 1992) contains other characterizations of irredundant conceptual graphs.

Property 2 *Given two canonical graphs G and H, if $G \equiv H$ and H is irredundant, then G*

has a subgraph isomorphic to H.

Proof (Chein and Mugnier 1992) — If $G \equiv H$, then there is a projection $\Pi : H \to G$ and there is a projection $\Pi' : \Pi(H) \to H$. $\Pi'(\Pi(H))) = H$, otherwise there would be a projection from H to one of its strict subgraphs, and H would not be irredundant. Thus $\Pi(H)$ and H are isomorphic.

□

Corollary *Two irredundant conceptual graphs are equivalent* iff *they are isomorphic.*

Property 3 *An equivalence class contains one and only one irredundant graph, which is the (unique) graph with the smallest vertex number.*

Proof (Chein and Mugnier 1992) — Previous corollary implies that an equivalence class contains at most one irredundant graph. Given an equivalence class, let us consider some graph G, with a minimal vertex number. G is irredundant, otherwise G would possess some strict subgraph equivalent to it, and this would contradict the hypothesis concerning G.

□

However, this characterization of an equivalence class is not polynomial. The problem of deciding whether two conceptual graphs are equivalent is NP-complete. And deciding whether a given conceptual graph is irredundant is co-NP-complete (*i.e.* is the complement of an NP-complete problem, here "is there a projection from a graph to one of its strict subgraphs ?") (Chein and Mugnier 1992) .

Suppose we are given an algorithm, say PROJECTION(G, G'), where G and G' are conceptual graphs; PROJECTION returns a projection from G to G' if any exists, otherwise it returns $\emptyset$. Then, using PROJECTION as an oracle, an algorithm can be constructed for computing the irredundant form of any conceptual graph, whose complexity is polynomially related to the complexity of PROJECTION (the number of calls to PROJECTION is actually linear in the number of G r-vertices).

The following reasoning is iteratively applied. If G is not irredundant, then there is a projection from G to one of its strict subgraphs, therefore there is a projection from G to one of the subgraphs obtained by removing one r-vertex of G. Such a subgraph, say $G \setminus \{r\}$, may have several connected components. Since G is connected, and the image by projection of a connected graph is necessarily connected, one only has to check whether there is a projection from G to a connected component of $G \setminus \{r\}$. If such a projection exists, say Π, then G is equivalent to $\Pi(G)$. And the irredundant form of G is the irredundant form of $\Pi(G)$. The algorithm is more precisely described below.

```
IRREDUNDANT(G)
// returns the irredundant form of G
   (1) R ← R_G          // where R_G is the r-vertex set of G
       ToCheck ← R_G // the set of eventually redundant r-vertices
   (2) While ToCheck ≠ ∅
         (2.1) Let r ∈ ToCheck; remove r from ToCheck
               The subgraph of G induced by R \ {r} may have several connected components
         (2.2) For all connected component C
                 (2.2.1) Π ←PROJECTION(G, C)
                 (2.2.2) If Π ≠ ∅ then
                           R ← {r-vertices of Π(G)}
                           ToCheck ← {r-vertices of Π(G)}∩ ToCheck
                           // Some r-vertices of Π(G) may have already been checked
                           Exit(For Loop)
   (3) Return the conceptual graph induced by R
```

Let $proj$ be the complexity of computing a projection from G to a graph of smaller or equal size. Since the total size of the connected components obtained by removing an r-vertex of G is less than the size of G, step (2.2) is in $O(proj)$. And step (2) is in $O(|R_G| \times proj)$. Thus the complexity of IRREDUNDANT is linearly related to the complexity of PROJECTION. It is shown in part 3, that PROJECTION(G_1, G_2) is polynomial when G_1 is a tree (see Figure 8). As connected subgraphs of a tree are trees, IRREDUNDANT(G) is polynomial when G is a tree.

2.5.2 The structure of $\leq_i$

$\leq_i$ is a transitive relation since the composition of two injective projections is an injective projection. It is reflexive since the identity mapping (or an automorphism) is an injective projection. And the antisymmetry property (modulo the isomorphism) is ensured by the following fact: if $G \leq_i H$, then the G vertex number is greater or equal to the H vertex number; if $H \leq_i G$ also holds true, then, given the constraints on the vertex labels in a projection, G and H are isomorphic. $\leq_i$ is thus an order. In other words, two conceptual graphs are equivalent for $\leq_i$ *iff* they are isomorphic.

3 Algorithmic considerations

Since the problems associated with the projection or the injective projection are NP-difficult, a common approach to solving them involves using backtracking techniques(McGregor 1982). Efficiency in the practice of these algorithms is linked to the relevance of their heuristics (exploiting the fact that conceptual graphs are labelled graphs and some characteristics of the application domain). Another perspective is proposed which consists of restricting the *form* of the graphs in order to come to polynomial cases. Algorithms presented here are based on the *tree* notion, roughly said a conceptual graph without cycles.

A *generic* version of each algorithm is first presented. Instead of conceptual graphs, directed graphs, labelled on vertices and edges, are considered. There is a trivial bijection between these graphs and conceptual graphs with only binary r-vertices. Instead of projection, a generic morphism operation is considered. It is called ρ_1-ρ_2-*morphism*, where ρ_1 and ρ_2 are predicates defining compatibility relations on vertex and edge labels. A ρ_1-ρ_2-morphism Π from a directed labelled graph $G_1 = (X_1,\ U_1,\ lab_1)$ to a directed labelled graph $G_2 = (X_2,\ U_2,\ lab_2)$ is a directed graph morphism from $(X_1,\ U_1)$ to $(X_2,\ U_2)$ such that
(1) For all $x \in X_1$, $\rho_1(lab_1(x), lab_2(\Pi(x)))$,
(2) For all $xy \in U_1$, $\rho_2(lab_1(xy), lab_2(\Pi(x)\Pi(y)))$.

A projection can be seen as a specific instance of a ρ_1-ρ_2-morphism, with ρ_1 is $\geq$ and ρ_2 is $=$. From an algorithmic point of view, the generic case contains all basic principles. Let us add that a significant number of conceptual graph systems only consider binary r-vertices.

Figure 8 summarizes the complexity of problems we are interested in, around projection and injective projection. Algorithms are provided for all mentioned polynomial cases.

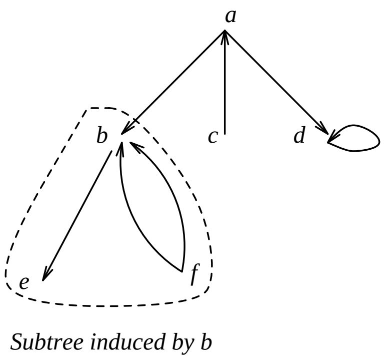
Question	G and H are graphs	G is a tree	G and H are trees
Is there a projection $\Pi : G \to H$?	NP-Complete	Polynomial	Polynomial
Given an integer k, is the number of $\Pi : G \to H$ smaller than k ?	#P-Complete	Polynomial	Polynomial
Is there an injective projection $\Pi : G \to H$?	NP-Complete	NP-Complete	Polynomial
Given an integer k, is the number of injective $\Pi : G \to H$ smaller than k ?	#P-Complete	#P-Complete	#P-Complete

Figure 8: Complexity of Projecti83 mugnier@ludwig.lirmm.fr:mugnier on and Injective projection

3.1 Polynomial algorithms for the projection

Concerning "projection", algorithms are developed for the following problems: given a graph G and a tree T, decide whether there is a projection from T to G, exhibit one or all projections from T to G, count the projections from T to G. As previously noted, algorithms in the generic binary case are first described (3.1.1), and then the general case with the projection is considered (3.1.2).

3.1.1 Generic morphism from a tree to a graph



The labels are not represented

Figure 9: Tree with root a

Given two vertices, say a and b, $links(a, b)$ denotes the set of edge labels between a and b, taking the edge direction into account —for instance r_+ (resp. r_-) if there is an edge ab (resp. ba) labelled r. Then, given any vertices a, b, c and d, $links(a, b)$ is said to be compatible with $links(c, d)$, if for all label $l \in links(a, b)$, there is a label $l' \in links(c, d)$, such that $\rho_2(l,\ l')$.

A directed labelled graph is said to be a *tree*, if its underlying undirected graph has no cycles, except for loops and multi-edges between two vertices. Given any tree, say T, any vertex, say a, will be chosen at a starting point for the operations. Then, a will be considered as the root of T, thus providing a father/son relationship between vertices. However, let us point out that this direction from the root to the leaves is independent of the direction of the edges. Given a rooted tree, and one of its vertices, say c, the *subtree induced by* c is the rooted subtree with root c (Figure 9).

Let $T = (X_T, U_T, lab_T)$ be a tree, and $G = (X_G, U_G, lab_G)$ be any graph. First notice that there is a generic morphism from T to G *iff*, given any vertex a of T chosen as the root, there is a vertex c of G and a generic morphism from T to G which maps a onto c. Such an operation is easily recursively defined as follows.

Π is a ρ_1-ρ_2-MORPHISM from T to G, with $\Pi(a) = c$, if it satisfies:
- *(i)* $\rho_1(label(a), label(c))$,
- *(ii)* $links(a, a)$ is compatible with $links(c, c)$
- *(iii)* for all a_i son of a, $c_j = \Pi(a_i)$ is a neighbour of c or is c itself, such that
  - *(iii-a)* $links(a, a_i)$ is compatible with $links(c, c_j)$,
  - *(iii-b)* Π_{T_i}, the restriction of Π to the subtree T_i induced by a_i, is a ρ_1-ρ_2-MORPHISM from T_i to G (and by construction, $\Pi_{T_i}(a_i) = c_j$).

Notes — If there is no loop on a, then (ii) is always true; if there is a loop on a and no loop on c, then (ii) is always false; in (iii), $c_j \neq c$ if there is no loop on c.

This recursive definition immediately leads to an algorithm for computing a morphism from T to G. A function called MORPHISM(a, E) is first provided, where a is the root of T, and E is a vertex set of G; the function returns $E' \subseteq E$ such that $\forall c \in E'$, $\exists \Pi : T \rightarrow G \mid \Pi(a) = c$. There is a morphism from T to G *iff* MORPHISM(a, X_G) $\neq \emptyset$ (See Figure 10 for a running example).

MORPHISM(a, E)
//$a \in X_T$; $E \subseteq X_G$; returns $\{c \in E \mid \exists \Pi : T \rightarrow G,\ \Pi(a) = c\}$
(1) $E \leftarrow \{c \in E \mid \rho_1(label(a), label(c))$ and $links(a,a)$ is compatible with $links(c,c)\}$
(2) If $E = \emptyset$ or a is a leaf, return E
(3) For all a_i sons of a,
  (3.1) For all $c \in E$
    $W_{c,a_i} \leftarrow \{c_j$ neighbour of c or $c_j = c \mid links(a, a_i)$ and $links(c, c_j)$ are compatible$\}$
  (3.2) $E_i \leftarrow \bigcup \{W_{c,a_i}\}_{c \in E}$
  (3.3) $E_i \leftarrow$ MORPHISM(a_i, E_i)
  (3.4) For all $c \in E$,
    $V_{c,a_i} \leftarrow W_{c,a_i} \cap E_i$
  (3.5) $E \leftarrow \{c \in E \mid V_{c,a_i} \neq \emptyset\}$
(4) Return E

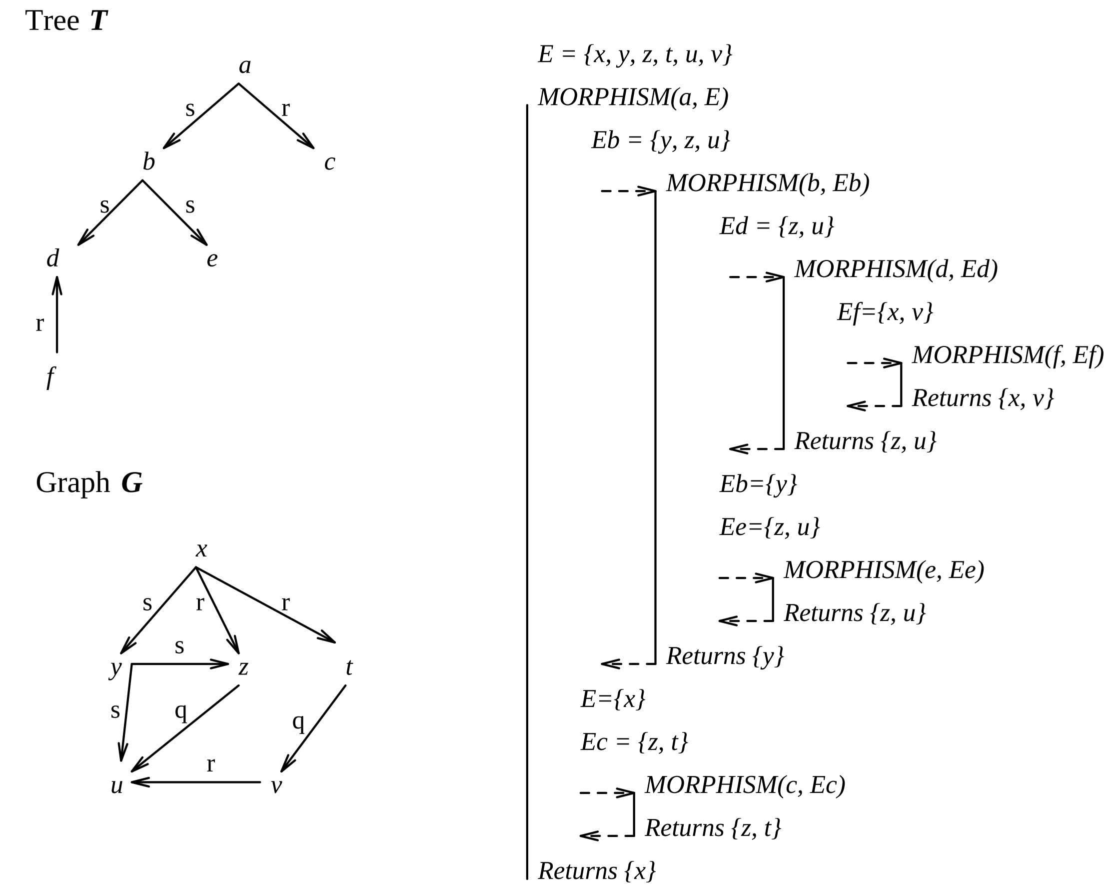
The algorithm stops since the depth of the tree strictly decreases at each recursive call.

Complexity of MORPHISM

One assumes in the following that the labels of two vertices (respectively edges) can be compared by ρ_1 (respectively ρ_2) in a constant time. If not, let $comp_1$ and $comp_2$ be these complexities. Then, the supplementary cost for all algorithms described in this paper is at most $max\ (|X_T| \times |X_G| \times comp_1, |U_T| \times |U_G| \times comp_2)$.

Property 4 *The complexity of MORPHISM(a, X_G) is $O(m_T \times m_G)$, where m denotes the edge number.*

Proof — By induction on $|X_T|$. For $|X_T| = 1$, only (1) and (2) are executed. (1) is in $O(m_T \times m_G)$, (2) is in constant time, so the property is trivially true. Let T be a tree with root a, and $|X_T| = n$, $n > 1$. Let us analyse step (3) by summing at each substep the operations done for all sons of a. (3.1) is in $O(degree(a) \times \sum_{c \in X_G} degree(c)) = O(degree(a) \times m_G)$. Suppose there is a vector with size $|X_G|$, that is initialized before the first recursive call, and used to mark the vertices of X_G, with different marks for different uses. Then (3.2) can be done in $O(|\{\text{sons of } a\}| \times m_G)$. For all a_i son of a, $|E_i| \leq |X_G|$, and $|X_{T_i}| \leq n$, thus, using the induction hypothesis, MORPHISM(a_i, E_i) is in $O(m_{T_i} \times m_G)$. Since $\sum_{a_i\ son\ of\ a} m_{T_i} \leq m_T$, (3.3) is in $O(m_T \times m_G)$. (3.4) can be done in $O(|\{sons\ of\ a\}| \times m_G)$, for instance: for all a_i, mark the vertices of E_i ; then restrict all V_{c,a_i} to the marked elements. (3.5) is in $O(|\{sons\ of\ a\}| \times |X_G|)$. Summing the complexities of the substeps, one obtains $O(m_T \times m_G)$. $\square$

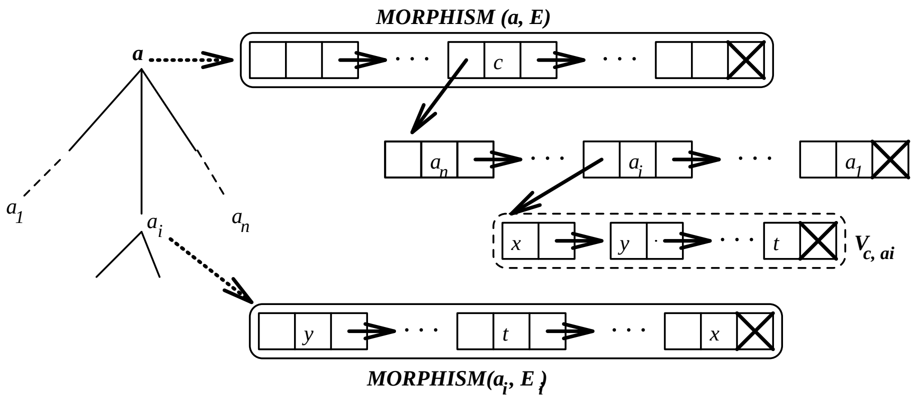


T and G vertex labels are not represented. They are supposed to be pairwise compatible relatively to ρ_1. ρ_2 is the equality.

Figure 10: Illustration of MORPHISM

Let us suppose that the V_{c,a_i} sets computed by MORPHISM(a, E) are kept outside the function. Consider for instance, the structure built in Figure 11. This does not imply increasing complexity. In order to exhibit a morphism Π, one only has to traverse this structure from the root of T to its leaves. $\Pi(a)$ in MORPHISM(a, E) is chosen, then recursively: if $\Pi(b) = c$, and b_i is a son of b, then choose $\Pi(b_i)$ in V_{c,b_i}. If all solutions are to be enumerated, the structure is traversed in the same way, combining all possible choices for all vertices. Rather than (or before) enumerating all morphisms from T to G, it may be interesting to count the number of such morphisms. The number of morphisms mapping the root a of T onto a vertex of $E \subseteq X_G$ is defined as follows.

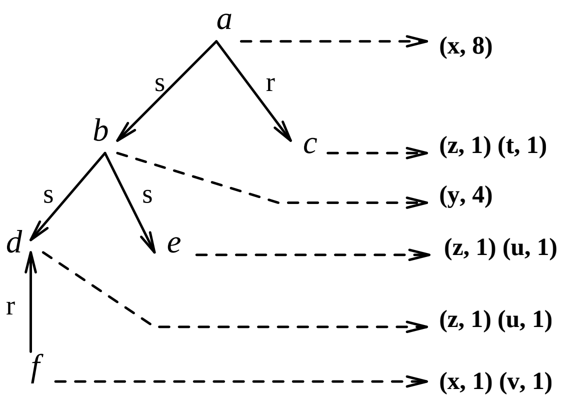
$NB(a, E) = \sum\{NB(a, c),\ c \in E\}$
$NB(a, c) =$ 0 if the condition "$\rho_1(label(a), label(c))$ and $links(a, a)$ is compatible with $links(c, c)$" is not fulfilled
1 if a is a leaf
Otherwise, $\prod\{NB(a_i, V_{c,a_i}),\ a_i$ son of $a\}$
—See (3.4) of MORPHISM for definition of V_{c,a_i}—



With each $a \in T$ is associated the set E' =MORPHISM(a, E). With each $c \in E'$ is associated the set V_{c,a_i}, for all a_i son of a.

Figure 11: The structure built by MORPHISM(a, E)

$NB(a, E)$ is computable by simply adding counters to MORPHISM. For instance, instead of returning a vertex set, the function returns a set of couples $(c,\ NB(a,c))$, where $NB(a,c) \neq 0$ (Figure 12). This is done without increasing the complexity.

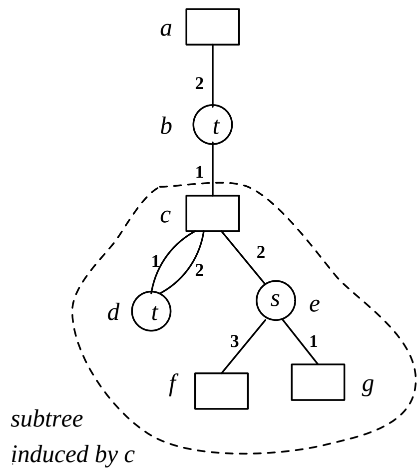


The tree refers to Figure 10. With each vertex $a \in T$ is associated a list of couples $(c, NB(a,c))$, where $NB(a,c)$ is the number of morphisms Π from the subtree induced by a to G, with $\Pi(a) = c$. There are eight morphisms from T to G.

Figure 12: Counting the morphisms from T to G

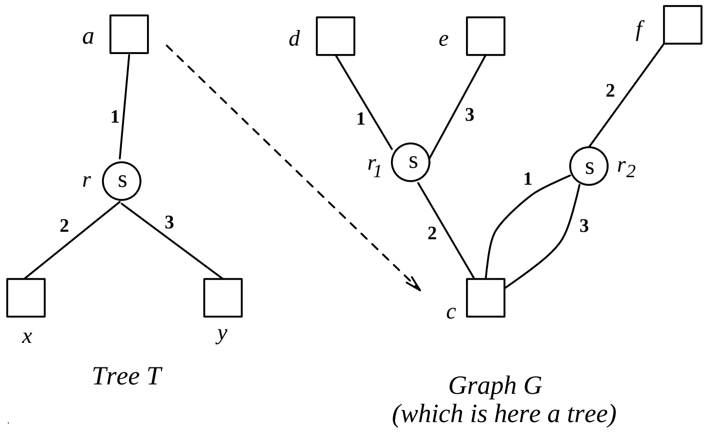
3.1.2 Projection from a conceptual tree to a conceptual graph

Let us now consider conceptual graphs. A *conceptual tree* is defined as a conceptual graph without cycles, except for cycles created by multi-edges between an r-vertex and one of its neighbours. The root is here a c-vertex, and the *father/son* relationship is reserved for c-vertices. The term *successor* is used for both kinds of vertices, according to the direction from the root to the leaves. The successors of a c-vertex c are the r-vertices which link c and its sons, or accept c as a unique neighbour. The successors of an r-vertex are its neighbours, except for the one which is the father of the others (Figure 13). Note that r-vertices play a preponderent role. Indeed, assuming that Π is a projection from one conceptual graph, say H, to another, say G, given an r-vertex r of H and $\Pi(r)$, there is a unique way for mapping each neighbour of r onto the neighbours of $\Pi(r)$ (Figure 14). The converse is not true: when there are duplicate r-vertices in G, knowing the images of all c-vertices does not determine the images of the r-vertices (although it is usually not important whether one duplicate r-vertex or the other is chosen). Also note that, given a conceptual tree T, an r-vertex of T can link a father to several of its sons. When trying to construct a projection from T to a conceptual graph G, these c-vertices are now interdependent for the choice of an image in G.



d and e are *successors* of c, while f and g are *sons* of c.

Figure 13: Conceptual tree



All c-vertices have the same label. There is only one projection from T to G mapping a on c. It is defined by $\Pi(r) = r_2$ (this implies that $\Pi(y) = \Pi(a) = c$ and $\Pi(x) = f$).

Figure 14: Projection with n-ary r-vertices

Given any r-vertex r, P_r notes the partition on $\{1, \ldots, degree(r)\}$, induced by the equality of r neighbours, that is i and j are in the same class of P_r if the i-ith and j-ith neighbours of r are the same vertex. For c a neighbour of r, $P_r[c]$ denotes the class of P_r which corresponds to c —*i.e.* the set of the numbers on edges between r and c. Given r and r' r-vertices, P_r is said to be *thinner* than $P_{r'}$ if each class of P_r is included in (or equal to) a class of $P_{r'}$. In other words, $P_{r'}$ is obtained from P_r by identifying (restriction + internal join) some neighbours of r. For instance, in Figure 14, $P_{r_1} = P_r = \{\{1\},\{2\},\{3\}\}$; $P_{r_2} = \{\{1,3\},\{2\}\}$; so P_r is thinner than P_{r_2}.

A projection from a tree T to a graph G can now be defined as follows. Let a and c be any c-vertices, respectively from T and G.

Π is a PROJECTION from T to G, with $\Pi(a) = c$, if it satisfies:

I *(i)* $label(a) \geq label(c)$,

(ii) for all r successor of a, $r' = \Pi(r)$ is a neighbour of c such that

(ii-1) $type(r) = type(r')$

(ii-2) $P_r[a] \subseteq P_{r'}[c]$, and,

II *(ii-3)* for all a_i successor of r, $c_j = \Pi(a_i)$ is the neighbour of r' (unique if it exists) such that

(ii-3-1) $P_r[a_i] \subseteq P_{r'}[c_j]$,

(ii-3-2) Π_{T_i}, the restriction of Π to the subtree T_i induced by a_i, is a PROJECTION from T_i to G (and by construction, $\Pi_{T_i}(a_i) = c_j$).

(ii-2) and *(ii-3-1)* ensures that P_r is thinner than $P_{r'}$. Parts **I** and **II** correspond to decomposition of the algorithm in two crossed recursive functions. **I** keeps the name PROJECTION and **II** is called PROJ-r.

```
PROJECTION(a, E)
//a ∈C_T; E ⊆ C_G; returns {c ∈ E | ∃Π : T → G, Π(a) = c}.
    (1) E ← {c ∈ E | label(a) ≥ label(c)}
    (2) If E = ∅ or a is a leaf, return E
    (3) For all r successors of a,
        (3.1) For all c ∈ E
              W_{c,r} ← {r' neighbour of c | type(r)=type(r') and P_r[a] ⊆ P_{r'}[c]}
        (3.2) E_r ← ⋃{W_{c,r}}_{c∈E}
        (3.3) E_r ← PROJ-r(r, E_r)
        (3.4) For all c ∈ E,
              V_{c,r} ← W_{c,r} ∩ E_r
        (3.5) E ← {c ∈ E | V_{c,r} ≠ ∅}
    (4) Return E

PROJ-r(r, E)
//r ∈R_T; E ⊆ R_G and fulfills conditions defined by PROJECTION;
//returns {r' ∈ E | ∃Π : T → G, Π(r) = r'}.
    (1) E ← {r' ∈ E | P_r is thinner than P_{r'}}
    (2) If E = ∅ or |P_r| = 1, return E
    (3) For all a_i successors of r,
        (3.1) E_i ← ⋃{c_{r'} | P_r[a_i] ⊆ P_{r'}[c_{r'}]}_{r'∈E}
        (3.2) E_i ← PROJECTION(a_i, E_i)
        (3.3) E ← {r' ∈ E | c_{r'} ∈ E_i}
    (4) Return E
```

Complexity of PROJECTION

Property 5 *The complexity of PROJECTION(a, C_G) is $O(m_T \times m_G)$, where m denotes the edge number.*

Proof — By induction on $|C_T|$ (similar to the proof of property 4).

The other algorithms are only modified in their formulation. Finding one or enumerating the projections from T to G is done as previously, with introduction of structures associated to r-vertices. The number of projections from T to G mapping a onto a vertex of $E \subseteq C_G$ becomes:

$NB(a,E) = \sum\{NB(a,c),\ c \in E\}$

$NB(a,c) =$ 0 if the condition "$label(a) \geq label(c)$" is not fulfilled
1 if a is a leaf
Otherwise, $\prod\{n_r,\ r$ successor of $a\}$, with $n_r = \sum\{NB(r,r') \mid r' \in V_{c,r}\}$

$NB(r,r') =$ 0 if the condition "P_r thinner than $P_{r'}$" is not fulfilled
—For all a_i successor of r, let $c_{r'}$ be the neighbour of r' such that $P_r[a_i] \subseteq P_{r'}[c_{r'}]$ —
1 if there is an a_i such that $NB(a_i, c_{r'}) = 0$
Otherwise, *max* $\{NB(a_i, c_{r'}),\ a_i$ successor of $r\}$

3.2 Injective projection

Given two conceptual graphs G and G', deciding whether there is an injective projection from G to G' remains an NP-Complete problem, even if G is a tree, and the graphs only have binary r-vertices without multi-edges (by reduction from "subgraph isomorphism" (Garey and Johnson 1979)).

An injective projection from a tree to another tree is polynomially computable. The skeleton of an algorithm is described below. It is a trivial adaptation of the algorithm analyzed in (Reyner 1977). It involves the notion of *bipartite graph matching*: a matching of a bipartite graph is a set of its edges, such that no two edges share a common endpoint.

Let T and T' be directed labelled trees. There is an injective generic morphism from T to T' *iff* given any vertex of T chosen as the root, one can root T', such that there is an injective generic morphism from T to T' which maps the root of T onto the root of T'.

Let a and b be any vertices chosen as roots of T and T' respectively.
There is an INJECTIVE ρ_1-ρ_2-MORPHISM from T to T', mapping a onto b if
(i) $\rho_1(label(a), label(b))$ and $links(a,a)$ is compatible with $links(c,c)$
(ii) there is a matching of cardinality $|A|$ in the bipartite graph on A and B where:
$A = \{\text{sons of } a\}$, $B = \{\text{sons of } b\}$
there is an edge a_ib_j, $a_i \in A$, $b_j \in B$, if
(ii-1) $links(a,a_i)$ is compatible with $links(b,b_j)$
(ii-2) there is an INJECTIVE ρ_1-ρ_2-MORPHISM from the subtree induced by a_i to the subtree induced by b_j mapping a_i onto b_j.

Complexity of INJECTIVE MORPHISM

Property 6 *Previous algorithm can be implemented in* $O(m_T \times m_{T'} + |X_T|^{1,5} \times |X_{T'}|)$ *(or less depending on the complexity of the matching algorithm).*

Proof — The injectivity constraint adds to the complexity of MORPHISM a supplementary cost of $\sum_{a\in T,\ b\in T'} match_{a,b}$, where $match_{a,b}$ is the cost of the matching between the sons of a and b. A good matching algorithm is, for instance, the algorithm of (Hopcroft and Karp 1973) (also analyzed in (Papadimitriou and Steiglitz 1982)) which has complexity $O(|A|^{1,5} \times |B|)$, with $|A| \leq |B|$. Since $\sum_{a\in T,\ b\in T'} |sons(a)|^{1,5} \times |sons(b)| \leq |X_T|^{1,5} \times |X_{T'}|$, the property holds true.
□

Deciding whether there is an injective morphism from T to T' can be done without increasing the complexity (Moon 1987).

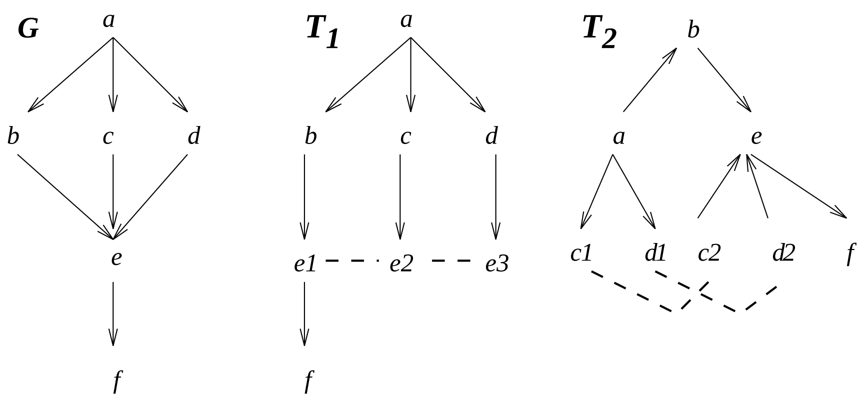
When coming to the general conceptual trees, the bipartite matching is computed on sets of r-vertices, and two corresponding r-vertices must have the same partition P_r. Using the same reasoning as previously described, the complexity can be roughly bound with $O(m_T \times m_{T'} + |R_T|^{1,5} \times |R_{T'}|)$.

Note that counting the injective projections between two rooted trees is an NP-hard problem (by reduction from the problem of counting the perfect matchings in a bipartite graph (Garey and Johnson 1979, theorem 7.9)).

3.3 Conclusion

Analyzing the polynomial cases for projection and injective projection has a theoretical and practical interest. First point, trees are frequent in conceptual graph applications, especially as specific sorts of conceptual graphs, for instance as elements of a canonical basis. Moreover some sophisticated operations are based on computation of a projection from a conceptual graph of a specific type to a general conceptual graph. For instance, checking if any conceptual graph G is canonical can be done by computing projections of elements of the canonical basis, say $\mathcal{B}$, onto G; the number of calls to projection is bounded by $|\mathcal{B}| \times \text{size}(G)$, thus, if the canonical basis consists of trees, the problem is polynomial (G and $\mathcal{B}$ being the data) (Mugnier and Chein 1993).

Second point, the projection algorithm from a tree to a graph can be exploited in a general projection algorithm, as a preprocessing part. Indeed, any graph, say G, can be represented by one of its "edge-covering trees" (*i.e.* trees containing all edges of G), say T_G, some c-vertices of G being split into several vertices (see Figure 15). G can be reconstructed by keeping the sets of vertices of T_G corresponding to the same vertex of G.



Both edge-covering trees of G are obtained by a breadth-first traversing of G, starting from a and b respectively.

Figure 15: Edge-covering trees

Then, the principle of a general algorithm for computing a projection from G to any graph H, could be as follows.

1. Compute an edge-covering tree of G, say T_G,
2. Apply PROJECTION(T_G, H). If it returns the empty set, then there is no projection from G to H; otherwise, with each vertex of T_G, say c, is associated a set of potential

images in H (see Figure 11), say I_c.

3. For each set $\{c_1, \ldots, c_p\}$ of T_G vertices corresponding to a same vertex c of G, replace each I_{c_i} with $\bigcap_{1 \leq i \leq p} I_{c_i}$.
4. Apply a backtrack algorithm, to satisfy the constraint "the vertices of T_G corresponding to a same vertex of G must have the same image in H".

This is a very sketchily described and generic algorithm. Note, for instance, that (1) can be done "virtually"; in (2), G only has to be traversed as it is a tree, using a vertex marking technique. Then, (3) is automatically done during (2), by slightly modifying the projection algorithm used in (2).

Besides precise designing of such an algorithm, a practical study is required to evaluate the gain of its introduction in pre-existing backtracking algorithms. The decisive factor for the complexity of the algorithm is the complexity of step (4). This complexity is exponential in the number of split vertices of G. Thus an important point is to find an edge-covering tree, which has good properties relative to the minimization of the split vertex number.

References

Berard-Dugourd, A., Fargues, J., Landau, M.C. and Rogala, J.P. (1988) Natural Language Information Retrieval from French Texts. In *Proceedings of the 3rd Workshop on Conceptual Graphs.* 4 p.

Chein, M. and Mugnier, M.L. (1992) Conceptual Graphs: fundamental notions. *Revue d'Intelligence Artificielle*, 6(4): 365–406. In english.

Ellis, G. (1991) Compiled Hierarchical Retrieval. In *Proceedings of the 6th Annual Workshop on Conceptual Graphs*, 187–207.

Fargues, J., Landau, M.C., Dugourd, A. and Catach L. (1986) Conceptual Graphs for Semantics and Information Processing. *IBM Journal of Research and Development*, 30(1):70–79.

Garey, M.R. and Johnson, D.S. (1979) *Computer and Intractibility - A Guide to the Theory of NP-Completeness* (W.H. Freeman and co).

Haussler, D. (1989) Learning Conjunctive Concepts in Structural Domains. *Machine Learning*, 4:7–40.

Hopcroft, J.E. and Karp, R.M. (1973) A $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. *SIAM Journal on Computing*, 2:225–231.

Jackman, M.K. (1988) Inference and the Conceptual Graph Knowledge Representation Language. In S. Moralee (ed.) *Research and Development in Expert Systems IV* (Cambridge University Press).

Kocura, P. and Kwong Ho, K. (1991) Aspect of Conceptual Graph Processor Design. In *Proceedings of the 6th Annual Workshop on Conceptual Graphs*, 317–329.

Lehman, F. (ed.) (1992) *Semantics Networks in Artificial Intelligence.* Pergamon Press.

Levinson, R., Pattern Associativity and the Retrieval of Semantics Networks. In Lehman, F. (ed.), *Semantics networks in Artificial Intelligence*, 573–600, (Pergamon Press).

Levinson, R. and Ellis, G. (1991) Multi-Level Hierarchical Retrieval. In *Proceedings of the 6th Annual Workshop on Conceptual Graphs*, 67–81.

McGregor, J.J. (1982) Backtrack Search Algorithms and the Maximal Common Subgraph Problem. *Software-Practice and Experience*, 12:23–34.

Mitchell, T.M. (1982) Generalization as search. *Artificial Intelligence*, 18:203–226.

Moon, J. C. (1987) $O(n^{2.5})$ Time Algorithms for the Subgraph Homeomorphism Problem on Trees. *Journal of Algorithms*, 8:106–112.

Mugnier, M.L. and Chein, M. (1993) Characterization and Algorithmic Recognition of Canonical Conceptual Graphs. In Sowa, J.F., Mineau, G.W. and Moulin, B. (eds) *Lecture Notes in Artificial Intelligence, 699, Proceedings of the International Conference on Conceptual Structures'93*, 294–311, (Springer Verlag).

Papadimitriou, C.H. and Steiglitz, K. (1982) *Combinatorial Optimization: Algorithms and Complexity*, (Prentice Hall).

Reyner, S.W. (1977) An Analysis of a good Algorithm for the Subtree Problem. *SIAM Journal on Computing*, 6(4):130–132.

Sabah, G. and Vilnat, A. (1991) Flexible Case Structure Implemented into a Deterministic Parser. In *Proceedings of the 6th Annual Workshop on Conceptual Graphs*, 343–356.

Sowa, J.F. (1984) *Conceptual Structures - Information Processing in Mind and Machine*, (Reading, MA: Addison-Wesley).