Conceptual Graphs are also Graphs

M. Chein and M.L. Mugnier

LIRMM (CNRS and Université Montpellier II),
161, rue Ada, 34392 Montpellier Cedex 5, France,
tel:(33)67148585, fax: (33)67148500, email: *name*@lirmm.fr

Research Report LIRMM, 95003, (January 1995)
— Revised version (September 95) —

Abstract. The main objective of this paper is to add one more brick in building the CG model as a knowledge representation model *autonomous* from logic. The CG model is not only a graphical representation of logic, it is much more: it is a declarative model encoding knowledge in a mathematical theory, namely *labelled graph theory*, which has efficient computable forms, with a fundamental graph operation on the encodings to do reasoning, *projection*, which is a labelled graph morphism. Main topics of this paper are: a generalized formalism for *simple* CGs; a strong equivalence between CSP (Constraint Satisfaction Problem) and labelled graph morphism. This correspondence allows the transportation of efficient algorithms from one domain to the other, and confirms that projection —or more generally labelled graph morphism— firmly moors CGs to combinatorial algorithmics, which is a cornerstone of computer science. The usual sound and complete first order logic semantics for CGs is still valid for our generalized model. This, plus the ease of doing important reasonings with CGs —for instance plausible reasonings by using some maximal join operations— without, at least for the moment, logical semantics, strengthens our belief that CGs must also be studied and developed independently from logic.

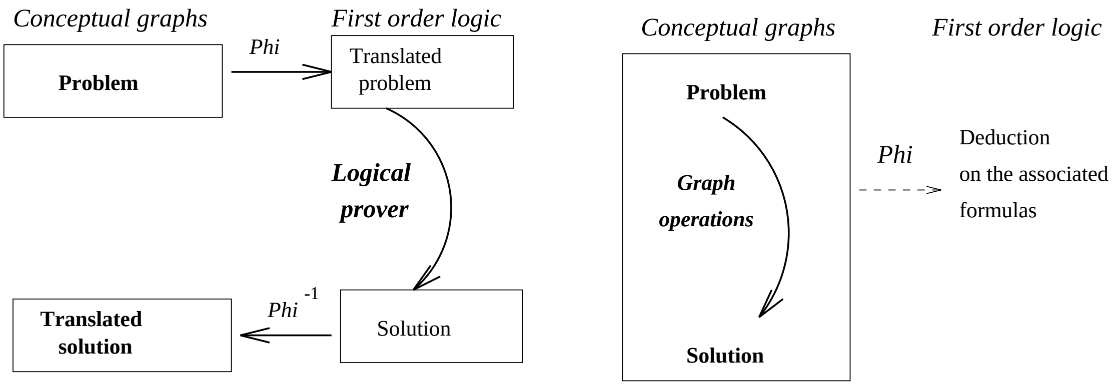
1 Introduction

"Conceptual graphs are significant as the basis for an operational knowledge science and technology encompassing natural language, formal language, visual language, and the wide range of reasoning processes that may be based on them", says Gaines [Gai93]. Works by Ellis, Levinson [Ell93] [Lev94] and others, have strengthen a fourth pillar —besides the three ones emphasized by Gaines: natural language, formal logic, and visual language— of CG theory : *graph theory and graph algorithms.* We share the Peirce group point of view that graph theory and algorithms are at the core of the CG model : this was present in [Sow84], but is not even mentioned by Gaines, and has become marginal in [Sow94a]. As a matter of fact graph theory is interesting in knowledge representation for many reasons. The principal reason is probably that natural language is the most important form of human knowledge representation, and graph models are very useful in linguistics, and specifically in semantics.

This is one of the bases of Sowa's work, and we will not develop this point here. In *human/machine interaction*, graphical representations of graphs allow the construction of useful interfaces, in particular convivial graph editors are easy to build (this is the visual language aspect). But, furthermore, a graph is a mathematical object which has "natural" implementations by cells and pointers (cf. [SW86]), which are analogous to the represented graphs. By combining this property with graphical representations, explanation modules may be built, allowing the user to follow computations on a faithful and "readable" image of the model. From a *computational point of view*, efficient algorithms are needed in order to construct usable systems based on CGs, and numerous efficient graph algorithms have been developed these last years.

One purpose of this paper is to add one more brick in building the CG model as a knowledge representation model *autonomous* from logic. The CG model is not (only) a graphical representation of logic, even if it may be used for that. That is to say (without repeating basic principles of knowledge representation as mentioned, for instance, by Sowa in [Sow94a], p.3) the CG model is: a *declarative* model encoding knowledge in a mathematical theory, namely *labelled graph theory*, which has efficient computable forms, with a fundamental graph operation on the encodings to do reasoning, *projection*, which is a labelled graph morphism.

The fact that the simple CG model has *consistent* and *complete* first order logic semantics, plus the ease of doing important reasonings —for instance plausible reasoning by using maximal joins— on CGs, without, at least for the moment, logical semantics, strengthens our belief that CGs must also be studied and developed "independently" from logic. This point of view is illustrated in Figure 1.



On the left: CGs as a graphical representation of logic.
On the right: CGs as a graph model, with a logical semantics.

Fig. 1. Conceptual graphs and logic

This paper is organized as follows. In section 2 a generalized formalism for the simple CG model is presented. By *simple* graphs we mean *non-nested* graphs, in particular there is no negation. The differences with Sowa's original model [Sow84], studied for instance in [CM92], are the following:

- a CG is not necessarily a connected graph;
- the concept type set is not necessarily a lattice. The concept-types and relation-types both form partial orders —which represent an *A Kind Of* relation;
- the conformity relation is replaced by an *Is A* relation between individual concepts and concept types (a typing of constants).

Generalized definitions for elementary specialization/generalization operations are proposed, and the main results concerning the specialization/generalization relation ($\leq$) remain valid :

- equivalence between $\leq$ and projection,
- existence of a unique irredundant CG in any equivalence class modulo $\leq$,
- furthermore, in the new model, the irredundant CG set has a *lattice* structure.

In our opinion, this slightly generalized model is simpler and more elegant than the previous one. The function Φ introduced by Sowa is studied. Sowa proved [Sow84] that this function may be considered as a *sound* logical semantics for CGs. We proved that this semantics is also *complete* [CM92]. We explain here why it is necessary to consider CGs in a *normal form* (any CG cannot contain more than one concept vertex with a given individual marker) in order to obtain completeness.

In Section 3 a strong equivalence between CSP (Constraint Satisfaction Problem) and graph morphism is established. This correspondence allows the transportation of efficient algorithms from one domain to the other, and confirms that projection —or more generally labelled graphs morphism— firmly moors CGs to combinatorial algorithmics, which is a cornerstone of computer science.

In a short conclusion it is indicated in what directions we are pursuing our research concerning theoretical and algorithmical aspects of the CG model.

2 The Formal Model

2.1 Descriptive Part

A structure, we call a *support*, provides background knowledge on a specific domain application. It represents an ontology of this domain, as the T-Box of terminological systems [Leh92]. It corresponds to the notion of a canon in [Sow84].

A *support* consists of a 5-tuple $S = (T_c,\ T_r,\ \sigma,\ I,\ \tau)$ where:

- T_c is an *ordered* set — not necessarily a lattice — of concept types, with a supremum $\top$ (the universal type) and an infimum $\bot$ (the absurd type),
- T_r is an *ordered* set of relation types, with a partition: $T_r = T_{r_{i_1}} + \ \dots\ + T_{r_{i_p}}$ where $T_{r_{i_j}}$ is the set of i_j-ary relations, $i_j \neq 0$; thus, two comparable relations types must have the same arity. The orders on T_c and T_r correspond to an *A Kind Of* relation between types,

- σ is a mapping which associates a signature with each relation type; the signature of a relation specifies the arity and the maximal concept types this relation can link. For any relation type $r \in T_{r_{i_j}}$, $\sigma(r) \in (T_c)^{i_j}$, and $\forall r_1, r_2 \in T_{r_{i_j}}, r_1 < r_2 \Rightarrow \sigma(r_1) \leq \sigma(r_2)$, where the order between signatures is the product order on $(T_c)^{i_j}$ (i.e. each i-ith argument of $\sigma(r_1)$ is less or equal to the i-ith argument of $\sigma(r_2)$),
- I is a set of individual *markers*; in addition there is the generic marker $*$; $I \cup \{*\}$ is provided with an order, such that $*$ is greater than all markers, and any two individual markers are non-comparable,
- τ is a mapping from I to $T_c \setminus \{\perp\}$; if the image of an individual marker m is a concept type t, t is called the type of m. τ corresponds to an *Is A* relation.

A *simple conceptual graph* related to a support S, $G = (R, C, U, label)$ is a bipartite multigraph[1], not necessarily connected. R and C denote the two classes of relation and concept vertices, $C \neq \emptyset$. U is the set of edges, and the edges incident to each relation vertex r are totally ordered —they are numbered from 1 to the arity of r. The i-ith neighbor of r in G is denoted by $G_i(r)$. Every vertex has a label defined by the mapping $label$. If $r \in R$, then $label(r) \in T_r$; if $c \in C$, then $label(c) \in (T_c \setminus \{\perp\}) \times (M \cup \{*\})$; the label of a concept type is a pair $(type(c), marker(c))$. $label$ satisfies the constraints given by σ and τ: $\forall r \in R$, $type(G_i(r)) \leq \sigma_i(type(r))$. $\forall c \in C$, if marker(c)$\in I$ then $type(c) = \tau(marker(c))$. Let us point out some aspects of our definitions, which slightly generalize the commonly used definitions of simple conceptual graphs.

- The concept type set is an order with a greatest and a least element, but not necessarily a lattice, as in [Sow84].
- The relation type set is ordered, and relations may be of any arity.
- A simple conceptual graph is not necessarily connected.

Another point is the τ relation, which plays the role of the conformity relation of [Sow84]. We impose that an individual marker always appears with the same type in concept labels[2]. But a more permissive solution could be as follows. Any concept label $l = (t, m)$, where $m \in I$, must satisfy $t \geq \tau(m)$, and not necessarily $t = \tau(m)$. Then, two types are associated with the label l: a "surface" type, t, and a "deep" type $\tau(m)$, this latter being the type used in operations.

[1] graph with possibly several edges between two vertices

[2] Let us explain why with an example. Let *Jumbo* be a circus elephant, i.e. the marker *Jumbo* conforms to the type *CircusElephant*. Now, suppose there are two graphs G_1 = [Elephant : *Jumbo*]-2-(AGNT)-1-[Eat:$*$] and G_2 = [CircusElephant : $*$]-2-(AGNT)-1-[Eat:$*$], with CircusElephant $\leq$ Elephant in T_c. Since one knows that [Elephant : *Jumbo*] refers to the same individual as [CircusElephant : *Jumbo*], G_2 should be considered as more general than G_1. However it is not possible to project G_2 onto G_1 (see section 2.2). When considering the logical semantics of CGs, one has the formulas: s_1 = *CircusElephant(Jumbo)* and $s_2 = \forall x\ CircusElephant(x) \rightarrow Elephant(x)$ coming from the logical interpretation of the support (see section 2.3), $f_1 = \exists y Elephant(Jumbo) \wedge Eat(y) \wedge Agent(y,\ Jumbo)$ and $f_2 = \exists x \exists y CircusElephant(x) \wedge Eat(y) \wedge Agent(y,\ x)$, respectively associated with G_1 and G_2. $f_1, s_1 \vdash f_2$, what cannot be inferred with the graph operations.

2.2 The Specialization/Generalization Relation between CGs

Elementary Operations. The fundamental notion for reasoning about CGs is the *specialization/generalization* relation on CGs — or *subsumption* relation. It is denoted by $\leq$. This relation can be defined in terms of a sequence of elementary relations, or by a global operation, called *projection*, which is a graph morphism.

The elementary specialization/generalization operations are internal operations on CGs, related to a given support. There are five specialization operations and five inverse operations, called generalization operations. In the list below, each specialization operation is followed by the inverse operation described in italics. All generalization operations are unary. One specialization operation, Disjoint Sum, is binary, the other four are unary. For all operations, the resulting graph is disjoint from the operand graph(s).

Let G be a CG, a CG H can be obtained from G by:

- **Simplify**: delete a duplicate relation vertex of G (i.e. a relation vertex r such that there is another relation vertex with the same type and exactly the same i-ith neighbor, for all i).
  Duplicate: duplicate a relation vertex of G.
- **Relation restrict**: decrease the label of a relation vertex r of G, providing that the new type (say t) conforms to σ: for all i ($1 \leq i \leq arity(r)$), $type(G_i(r)) \leq \sigma_i(t)$.
  Relation augment: increase the label of a relation vertex r of G (no constraint on this augmentation).
- **Concept restrict**: decrease the label of a concept vertex c of G —replace (t, m) by (t', m') with $t' \leq t$ and $m' \leq m$— providing that the new label conforms to τ: if m' is an individual marker, then $t' = \tau(m')$.
  Concept augment: increase the label of a concept vertex c of G —replace (t, m) by (t', m') with $t \leq t'$ and $m' = *$— providing that the new label conforms to σ: for all relation neighbor r of c, let $c = G_i(r)$, $type(c) \leq \sigma_i(r)$.
- **Join**: merge two concept vertices with same label.
  Split: split a concept vertex c into vertices c_1 and c_2, with the same label as c, such that the union of the sets of edges incident to c_1 and c_2 is the edge set of c —one of the two sets may be empty.
  And the unique binary operation:
- **Disjoint sum**: Let G_1 and G_2 be two disjoint CGs; H is obtained from G_1 and G_2 by juxtaposing G_1 and G_2.
  Substract: delete some (not all)[3] connected components of G.

G is a specialization of H ($G \leq H$) if G can be derived from H by specialization operations. Since there is a binary operation, the derivation of H from G may have other sources than H (see [MC93] for a formal definition of a specialization derivation). G is a generalization of H ($G \geq H$) if G can be derived from H by a sequence of generalization operations. Note that if $G \geq H$, then G can be obtained from H by considering H only (in terms of [MC93], the derivation from G to H is a path, and not a graph with several sources). It is straightforward to prove that $G \leq H$ *iff* $G \geq H$.

[3] Indeed we do not consider empty graphs

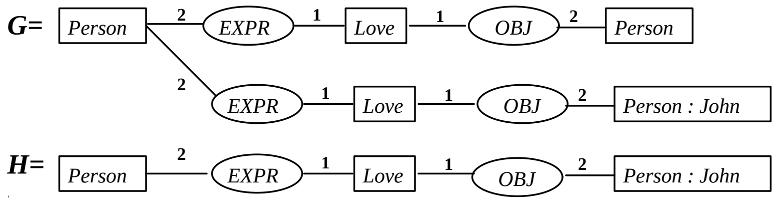
Projection. Projection is the fundamental operation on CGs, since it permits the *effective* computation of the $\leq$ relation. This operation is a graph morphism,(see section 3.1), keeping the order on the edges, and allowing restriction of vertex labels. More precisely: A *projection* Π from a conceptual graph $H = (R_H, C_H, U_H, lab_H)$ to a conceptual graph $G = (R_G, C_G, U_G, lab_G)$ is an ordered pair Π=(f, g) of mappings, f from R_H to R_G, and g from C_H to C_G, such that

1. $\forall\ rc \in U_H$, $f(r)g(c) \in U_G$, and if $c = H_i(r)$ then $g(c) = G_i(f(r))$ —i.e. it preserves edges and orderings on edges,
2. For each $r \in R_H$, $lab_H(r) \geq lab_G(f(r))$,
3. For each $c \in C_H$, $lab_H(c) \geq lab_G(g(c))$.

In the following, a projection is simply noted Π, with an implicit decomposition into f and g. Sowa [Sow84] proves that, given two conceptual graphs G and H, if $G \leq H$ then there is a projection from H to G. We prove in [CM92] that the reciprocal also holds (in [MC93] the proof is extended to canonical graphs). These proofs are still valid for the generalized model given in this paper.

Theorem 1 *Let G and H be two simple conceptual graphs; $G \leq H$* iff *there is a projection from H to G.*

Structure of the $\leq$ Relation $\leq$ is a reflexive and transitive relation, but, as several times remarked (e.g. [Jac88]), it is not a partial order because the antisymmetry property is not fulfilled (Figure 2). If $G \leq H$ and $H \leq G$, then G and H are said to be equivalent, denoted $G \equiv H$.



$G \leq H$ and $H \leq G$. G is redundant, while H is not.

Fig. 2. Equivalent graphs

A natural question is then to measure the difficulty of coming back to an order. The notion of an *irredundant* graph [CM92] provides a characterization of the quotient order of $\leq$. 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. Otherwise G is *redundant.* Note that an irredundant graph is not necessarily connected. It appeared later that this notion corresponds to the notion of a *core graph* for non-directed graphs with homorphism operation (e.g. [HN90]).

Property 1 [CM92] *Given two canonical graphs G and H, if $G \equiv H$ and H is irredundant, then G has a subgraph isomorphic to H.*

Corollary 1 [CM92] *Two irredundant conceptual graphs are equivalent* iff *they are isomorphic.*

Property 2 [CM92] *An equivalence class contains one and only one irredundant graph, which is the (unique) graph with the smallest vertex number.*

However, this characterization of an equivalence class is not polynomial (see part 3). It is shown in [Mug95] that, if the projection is polynomial for a class of graphs, then the computation of the irredundant form of a graph of this class is polynomial too.

The restriction of $\leq$ to the set of irredundant graphs is not only a partial order, it is a *lattice*.

Theorem 2 *The restriction of $\leq$ to the set of irredundant graphs is a lattice.*

Proof — Let G_1 and G_2 be two irredundant graphs and $G_1 + G_2$ be their disjoint sum. The following property is immediate: for every graph K, $K \leq G_1 + G_2$ *iff* $K \leq G_1$ and $K \leq G_2$. Indeed, a projection from $G_1 + G_2$ to K can be split into two projections, one from G_1 to K and one from G_2 to K. Reciprocally, the union of two projections, from G_1 to K and from G_2 to K yields a projection from $G_1 + G_2$ to K. Now there is a unique irredundant graph equivalent to $G_1 + G_2$, which is thus the greatest lower bound of G_1 and G_2. Thus, the irredundant graph set is an inf-semi-lattice. The irredundant graph set also has a maximum element, the graph $[\top : *]$. It is thus a lattice.
□

Note that this theorem is true even if the vertex labels (or the relation and concept types) are not ordered by a lattice. The only condition is that the concept type set forms a partial order with a greatest element. The theorem does not hold if one only consider connected graphs, even if the vertex labels are lattices.

This lattice structure is important when one wants to find minimal common generalization or maximal common specialization of two graphs. However, although it is easy to compute an equivalent graph to the infimum of two graphs (just compute their disjoint sum), this does not solve the problem of the effective computation of this infimum. The same problem arises with the computation of their supremum.

2.3 First Order Logic Semantics

Sowa [Sow84] proposes to associate with every conceptual graph G a well-formed formula $\Phi(G)$ of the first-order-predicate calculus. Given a support S, with each individual marker is associated a constant and with each relation or concept type is associated a predicate. With each concept vertex of a graph G related to S is associated a *term*: a distinct variable for each generic concept, and the constant corresponding to the marker for each individual concept. With each relation or concept vertex is associated an *atom*: for a concept c, the atom $type(c)(id(c))$, for a relation r, the atom $type(r)(id(c_1), ..., id(c_p))$, where c_i denotes the i-ith neighbor of r and $id(c_i)$ stands for the term associated with c_i. $\Phi(G)$ is the existential closure of the conjunction of the atoms associated with the vertices of G.

With the support is also associated a set of well-formed formulas, say $\Phi(S)$, which corresponds to a logical interpretation of the type orders.

1. for all $t_1, t_2 \in T_c$, if t_2 is covered by t_1[4], then:
   $\forall x t_1(x) \rightarrow t_2(x)$
2. for all $t_{r_1}, t_{r_2} \in T_r$, with same arity p, if t_{r_2} is covered by t_{r_1} then:
   $\forall x_1 \dots x_p t_{r_1}(x_1 \dots x_p) \rightarrow t_{r_2}(x_1 \dots x_p)$
3. $\forall x \top(x)$
4. $\forall x \neg\bot(x)$
5. For every $t_r \in T_r$, with arity p, and $\sigma(t_r) = (t_1 \dots t_p)$:
   $\forall x_1 \dots x_p t_r(x_1 \dots x_p) \rightarrow t_1(x_1) \wedge \dots \wedge t_p(x_p)$
6. For every individual marker m, with $\tau(m) = t$:
   $t(m)$

Axioms 1 and 2 are the logical interpretation of the *A Kind Of* links between types. Axioms 3 to 6 are not used in the proof of the completeness theorem, but they are needed for the model-theoretic semantics proposed in [MC94].

It can be directly proven that the generalization rules constitute a *sound* set of inference rules for the set of logical formulas associated with simple CGs (see [Sow84], [CM92]). More precisely:

Theorem 3 *Let $\Phi(S)$ be the set of formulas associated with a support S. Let G and H be simple CGs related to S. If $G \leq H$ then $\Phi(S), \Phi(G) \vdash \Phi(H)$.*

It is more difficult to prove that the generalization rules constitute a *complete* set of inference rules. As far as we know, [CM92] provides the first proof of this result. In order to obtain completeness property, CGs need to be in a specific form, we call a *normal form*: all individual vertices must have different markers. Any graph can be transformed to normal form by joining all vertices with same individual marker. When CGs are not in normal form, some logical deductions can not be simulated with the generalization rules. See for instance Figures 3 and 4: in both figures, formulas $\Phi(G)$ and $\Phi(H)$ are equivalent, but G and H are not equivalent, even non comparable (Figure 4).

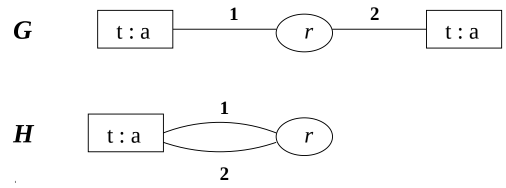
Let us mention that the specialization/generalization rules are not internal rules on the set of CGs in normal form. Even worse: there are graphs in normal form, say G and H, such that $G \leq H$ and there is no specialization derivation from H to G containing only CGs in normal form (Figure 5).

Theorem 4 [CM92], [MC94] *Let $\Phi(S)$ be the set of formulas associated with a support S. Let g and h be well formed formulas associated with G and H in normal form. If $\Phi(S), g \vdash h$ then $G \leq H$.*

3 Algorithmic Considerations

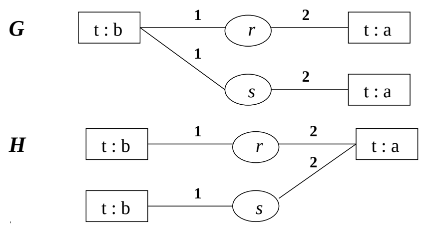
Determining whether there is a projection from a graph to another is an NP-complete problem —we call this problem *Projection*. [CM92] proves that problems *a priori* simpler are also NP-complete: *Equivalence* (given G and H, is G equivalent to H?),

[4] i.e. $t_2 \leq t_1$ and there does not exist a type t with $t_2 < t < t_1$. We consider here the minimal set of edges in T_C that is necessary to generate all formulas associated to the partial order.



$\Phi(G) = t(a) \wedge t(a) \wedge r(a,\ a)$. $\Phi(H) = t(a) \wedge r(a,\ a)$. $\Phi(G)$ and $\Phi(H)$ are equivalent, while G and H are not equivalent ($G > H$).

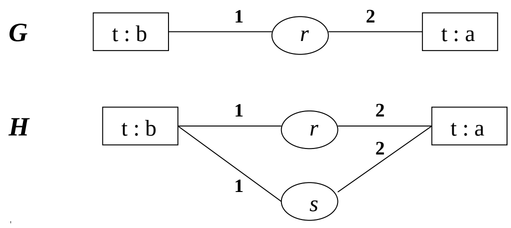
Fig. 3. The need for normal form (1)



$\Phi(G) = t(b) \wedge t(a) \wedge t(a) \wedge r(b,\ a) \wedge s(b,\ a)$.
$\Phi(H) = t(b) \wedge t(b) \wedge t(a) \wedge r(b,\ a) \wedge s(b,\ a)$.
$\Phi(G)$ and $\Phi(H)$ are equivalent, while G and H are non comparable.

Fig. 4. The need for normal form (2)

Redundancy (given G, is there a projection from G to one of its strict subgraphs?). This part is devoted to a further study of the projection from the viewpoint of its relations with two domains : graph theory and Constraint Satisfaction Problems (CSP).



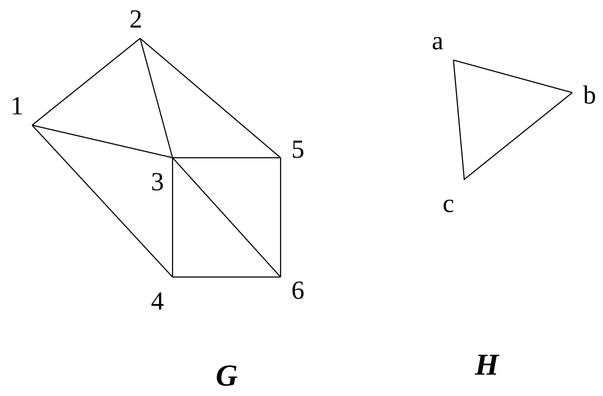
G and H are in normal form, but all derivations from G to H involve a graph which is not in normal form.

Fig. 5. Derivation of graphs in normal form

3.1 Conceptual Graphs and Graphs, Projection and Morphism

From a graph theory viewpoint, projection is a *morphism* —also called homomorphism. A *morphism* from a non-directed graph $G = (V_G, E_G)$ to a non-directed graph $H = (V_H, E_H)$, where V and E respectively denote the vertex set and the edge set, is a mapping $\Pi : V_G \rightarrow V_H$,which keeps the edges (i.e. for any edge xy of E_G, $\Pi(x)\Pi(y)$ is an edge of E_H). See Figure 6. When considering *directed* graphs, morphism must keep the edge direction (Figure 7). When there are labels on vertices and/or on edges, there are additional constraints; if the label sets have no specific structure, corresponding vertices (edges) must have the same label. The definition extends naturally to multigraphs.

We call *Morphism* the following generic problem: Given the two graphs G and H (both directed/non-directed, labelled or not, ...), is there a morphism from G to H?

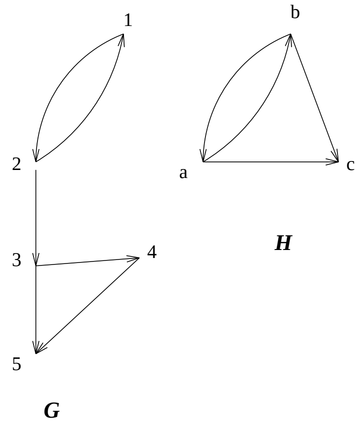


There is no morphism from G to H. But, there are several morphisms from G minus any edge, to H. For example, if the edge 12 is removed from G, $\{(1,\ a),\ (3,\ b),\ (4,\ c),\ (6,\ a),\ (5,\ c),\ (2,\ a)\}$

Fig. 6. Non directed graph morphism

Figure 6 shows an example of a non-directed morphism problem. It may also be interpreted as an instance of a graph vertex coloring problem (Given a graph $G = (V_G, E_G)$ and a positive integer k, is there a coloring of G with k colors, i.e. a mapping from V_G to $\{1, \ldots, k\}$, such that no adjacent vertices have the same image (color)?). Here it would be the 3-coloring problem ($k = 3$). More generally, there is a coloring of a graph G with k colors *iff* there is a morphism from G to the k-clique (graph with k vertices and all possible edges except for loops).

Simple conceptual graphs may be seen as particular non-directed graphs, labelled on vertices and edges (furthermore they are bipartite graphs). Conversely, non-directed graphs may be seen as particular conceptual graphs. Indeed, a non-directed graph may be seen, from the morphism operation viewpoint, as a directed graph, without loops, each edge being replaced by two symmetric directed edges



There are two morphisms from G to H: Π_1 and Π_2
$\Pi_1 = \{(1,\ a),\ (2,\ b),\ (3,\ a),\ (4,\ b),\ (5,\ c)\}$
$\Pi_2 = \{(1,\ b),\ (2,\ a),\ (3,\ b),\ (4,\ a),\ (5,\ c)\}$

Fig. 7. Directed graph morphism

(Figure 8). In other words, given two non-directed graphs G and H, there is a non-directed graph morphism from G to H *iff* there is a directed graph morphism from the directed graph associated with G to the directed graph associated with H. A directed graph is itself a specific case of directed labelled graph, with labels on vertices and edges, all vertices (*resp.* edges) having the same label. Finally, a directed graph labelled on vertices and edges is a specific case of conceptual graph: each edge $(x,\ y)$ is replaced by a binary relation vertex, with x as first neighbor, and y as second neighbor, and same label as the edge $(x,\ y)$.

Any polynomial case for Morphism, based upon the *structure* of G or H, provides a polynomial case for Projection. The complexity of Morphism has been particularly studied in the framework of the following problem, called *general coloring problem* or *H-coloring*, where H is a fixed graph, that may be directed or not: Given a graph G, is there a morphism from G to H? This problem is a generalization of the classical coloring problem, since H is not necessarily a clique. The complexity study of *H-coloring* focused on the structure of H, i.e. what are exactly the conditions on the structure of H for the problem to be polynomial? Concerning non-directed graphs, [HN90] proves that: *H-coloring* is polynomial if H is bipartite (or if H has a loop or is restricted to one vertex), and NP-complete in all other cases. Concerning directed graphs, [GWW92] and [HNZ93] exhibit some trees H, such that *H-coloring* remains NP-complete. Consequently, Morphism is NP-complete, even if H is a tree.

But Morphism is polynomial if G is a tree (this fact has been shown in several works independently). It has been shown in [HNZ94] that the problem remains polynomial when G has bounded treewidth (*i.e.* G is a partial k-tree[5], for a fixed

[5] A non-directed graph is a *k-tree* if it can be obtained from a k-clique by repeatedly adding a vertex connected to k existing vertices which form a k-clique. It is a *partial k-tree* if it is a spanning subgraph of a k-tree (some edges are removed). In particular, a tree is a

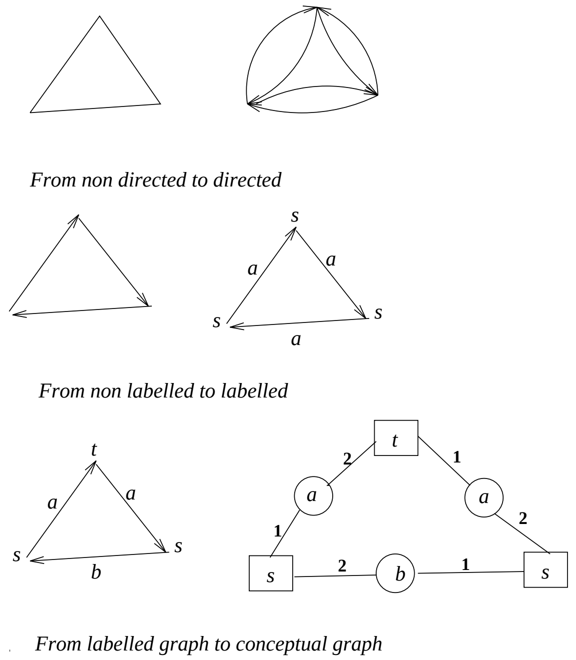


Fig. 8. Transformations

k). But note that determining for a given graph G and a given integer k, whether the treewidth of G is at most k is NP-complete [ACP87]. Concerning conceptual graphs, we give in [MC92]— also [Mug95]— a polynomial algorithm for computing a projection from a "conceptual tree" G to a conceptual graph H, and for counting the number of such projections. This projection algorithm from a tree to a graph can be exploited in a general projection algorithm (i.e. when G has any structure), as a preprocessing part [Mug95].

The above mentioned complexity studies do not consider the case of *labelled* graphs, i.e. the effects of some labelling function properties on the complexity. However such a work would be of great interest for conceptual graphs. If the condition on labels for two vertices (or edges) that match is the equality of their label, then the presence of labels may simplify the problem and provide some polynomial cases. For instance, when the labelling function is locally injective (i.e. injective on the neighborhood of each vertex (or if edges are labelled, on the set of edges incident to each vertex), then morphism is polynomial, whatever the structure of G and H may be (see [Kac84] for feature structures and [LB94] for conceptual graphs). But when labels are partially ordered then the condition for matching vertices (or edges) can be "for any vertex c, the label of the image must be *less or equal to* the label of c". Then, morphism becomes a projection, which can make the problem more complex. For example: the problem of determining whether there is an isomorphism between

1-tree. The *treewidth* of a graph is he minimum k for which the graph is a partial k-tree. For a directed graph, one considers the underlying non-directed graph.

two graphs is not classified but is supposed to be non NP-complete; for labelled graphs, the complexity remains the same if matching vertices necessarily have the same label; but the problem becomes NP-complete when there is a partial order on labels and the label of any vertex must be greater or equal to the label of its image (see the problem "isoprojection" in [CM92]).

3.2 Conceptual Graphs and Constraint Satisfaction Problems

There is a strong equivalence between the Projection problem and the well-known Constraint Satisfaction Problem (CSP) —e.g. [P.89]. The polynomial transformations between these problems, which are detailed below, keep the solution sets (actually an instance of one problem has the same set of solutions as its transformation into an instance of the other problem).

Let us consider a simple definition of a CSP, where constraints are defined in extension and are not typed. A *constraint network* $P = (X,\ C,\ D,\ R)$ involves:

- a set of n variables, $X = \{x_1,\ \dots,x_n\}$
- a set of constraints, $C = \{C_1,\ \dots,C_p\} \subset \mathcal{P}(X)$,
  When constraints are binary, i.e. involve two variables, the pair $(X,\ C)$ can be seen as a non directed graph, whose vertices are the variables, and edges are the constraints. In the general case, $(X,\ C)$ is an hypergraph.
- a set of variable domains, $D = \{D_1,\ \dots,D_n\}$
- a set of constraint definitions, $R = \{R_1,\ \dots,R_p\}$; for every constraint C_i, let $(x_{i_1}\ \dots\ x_{i_q})$ be any ordering of C_i variables, then $R_i \subset D_{i_1} \times\ \dots\ \times D_{i_q}$.

A *solution* of P is an assignment of values to the n variables such that all the constraints are satisfied, more formally:

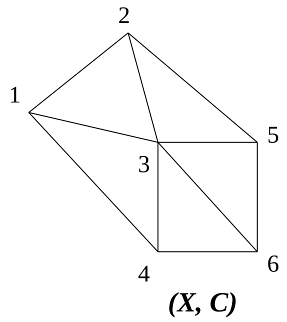
$$\begin{aligned} &\text{a mapping } \mathcal{I}\colon X \to \bigcup D_i \in D \\ &\qquad\qquad\quad x_i \mapsto a \in D_i, \\ &\text{and, for every constraint } C_j = (x_{j_1}\ \dots\ x_{j_q}),\ (\mathcal{I}(x_{j_1})\ \dots\ \mathcal{I}(x_{j_q})) \in R_i. \end{aligned}$$

A *constraint satisfaction problem* is the associated problem: Given a constraint network P, is there a solution to P?

In short, a solution of a CSP is a mapping from the variable set to the set of values of domains, which satisfies the constraints. In comparison, a morphism (or a projection)is a mapping from one vertex set to another vertex set, which keeps the edges. This formulation summarizes the strong equivalence of CSP and Morphism.

Theorem 5 *The problems* CSP *and* directed labelled graph Morphism *are polynomially equivalent, and both transformations from one problem to the other satisfy:*
(1) Every solution to one problem is a solution to the other one.
(2) The constraint graph and the graph to be mapped (G) have the same underlying non-directed graph.

This theorem basically comes from [FV93], where the equivalence between both problems is mentioned (but the transformations are not given in this paper). We give here a proof schema. It may be easily adapted for Projection instead of Morphism.



$D1=D2= \ldots = D6 = \{a, b, c\}$

For all constraint Ci={x, y},

Ri =

x	y
a	b
a	c
b	a
b	c
c	a
c	b

This constraint network represents an instance of the 3-coloring problem. The constraint graph and the graph to be colored are the same graph. There is no solution to the CSP: cf. Figure 6, which represents the same 3-coloring problem (as an instance of Morphism).

Fig. 9. A constraint network

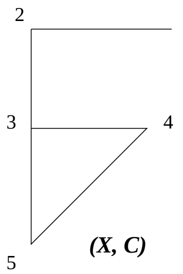
Transformation from Morphism to CSP. Let $G = (V_G,\ E_G)$ and $H = (V_H,\ E_H)$ be an instance of Morphism. We first consider G and H as directed antisymmetric graphs, without loops and without labels, we then extend the transformation.

A network $P = (X,\ C,\ D,\ R)$ is built from G and H as follows. (X, C) corresponds to the non-directed graph underlying G, that is: $X = V_G$, and $C = E_G$ with removal of the edge directions. With each variable x_i is associated the domain $D_i = V_H$. With each constraint C_i is associated the definition $R_i = E_H$.

When considering G and H as more complex graphs, variable domains and/or constraint definitions are restricted. E.g. if there are labels on edges, then with each constraint $C_i = (x_j\ x_k)$ is associated the definition $R_i = \{(y\ z) \in E_H \mid label(y,\ z) = label((x_j\ x_k))\}$. If there are loops in G, then for each vertex x_i with a loop, $D_i = \{y \in V_H | (y,\ y) \in E_H\}$, i.e. a loop may be seen as a kind of label.

This transformation is polynomial in the size of the instance of Morphism. It is straightforward to check that there is a morphism from G to H *iff* there is a solution to the CSP, and that points (1) and (2) of the theorem are fulfilled. Figure 10 shows the constraint network obtained from the instance of Morphism of Figure 7.

Transformation from CSP to Morphism. Let us first consider a binary CSP, with the constraint network $P = (X,\ C,\ D,\ R)$. From P, we build $G = (V_G,\ E_G,\ label_G)$ and $H = (V_H,\ E_H,\ label_H)$, an instance of directed labelled graph Morphism (only edges need to be labelled). G is built from the constraint graph: $V_G = X$, $E_G = C$, giving any direction to edges (we take here the order fixed by the constraint definitions). The labels are the "names" of the constraints. Each edge of E_G is labelled by the name of the constraint it comes from. H is built from D and R. V_H is the union of the variable domains, $V_H = \bigcup D_i \in D$. E_H is the disjoint union of the constraint definitions, that is, for each pair $(a\ b)$ of constraint definition



D1=D2= ... = D5 = {a, b, c}

1	2
a	*b*
b	*a*

2	3
a	*b*
b	*a*
a	*c*
b	*c*

3	4
a	*b*
b	*a*
a	*c*
b	*c*

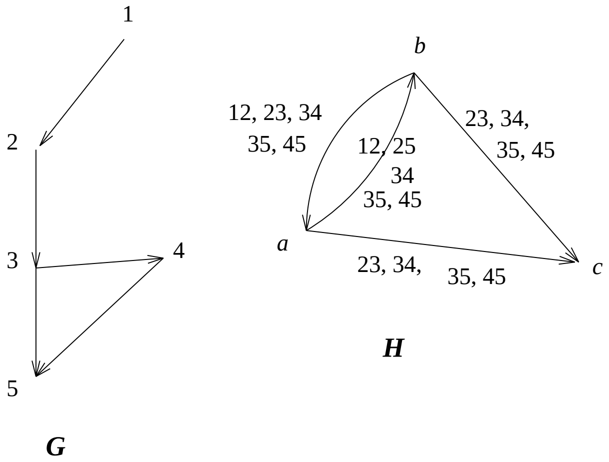
3	5
a	*b*
b	*a*
a	*c*
b	*c*

4	5
a	*b*
b	*a*
a	*c*
b	*c*

Fig. 10. Transformation from Morphism(Figure 7) to CSP

R_j, one builds an edge (a b) labelled by the name of C_j.

This transformation is polynomial in the size of the constraint network. It is straightforward to check that there is a solution to the CSP *iff* there is a morphism from G to H, and that points (1) and (2) of the theorem are satisfied. Figure 11 shows the graphs obtained from the constraint network of Figure 10.



Lists of labels represent multi-edges, e.g. there are 5 edges from a to b.

Fig. 11. Transformation from CSP(Figure 10) to Morphism

H actually is a multigraph. It is possible to remove the multi-edges, by subdividing each edge. One obtains a bipartite graph. Labels on edges disappear, and vertices are now labelled: new vertices are labelled by the name of the corresponding constraint, and original vertices are all labelled with the same new value.

When the CSP is not binary, it can be either first transformed into a binary CSP (the transformation is polynomial, see for instance [Ber70]), or the previous transformation can be extended. G and H become non directed bipartite graphs, one vertex class corresponds to constraints, these vertices are labelled by the name of the corresponding constraint, the other class corresponds to variables, and these vertices are labelled with the same value. The edges incident to any "constraint vertex" are totally ordered, what we represent by labelling the edges from 1 to the constraint arity. In short, one obtains a kind of conceptual graph.

These transformations show that Morphism (or Projection) and CSP are polynomially equivalent. It was already known since both are NP-complete problems. But, the interesting points are:
- The graph to be mapped (i.e. G) and the constraint graph have the same non-directed underlying graph. Thus, any polynomial case known for CSP and based on the structure of the constraint graph directly applies to the G of Morphism, and reciprocally. For instance, it is well-known in the CSP community that the problem CSP is polynomial when the constraint graph is acyclic (or is a tree) [Fre82].
- Every solution to the CSP *is* a morphism from G to H. Thus, the algorithmic techniques for determining whether there is a solution, but also enumerating the solutions, can be transferred from one domain to the other.

4 Conclusion

The simple CG model studied in this paper is a firm basis for building a knowledge representation language. We are indeed involved in the construction of a software platform (CORALI) for developing applications in which all kinds of knowledge are represented by CGs, and all reasonings are supported by projection.

But this model is too restrictive for many applications. Extensions are needed. Although it is easy to extend the descriptive features, extensions we are interested in have to respect essential characteristics of the kernel: objects must be sorts of graphs and inferences must be done by graph algorithms, not by graphical translations of logical operations. For instance, an extension to nested CGs —in the sense of [Sow94b]— is proposed in [MC94].

Acknowledgements

We thank Anne Preller for the example of Figure 4, which shows the problems with individual markers in the proof of logical completeness.

References

[ACP87] S. Arnborg, D.G. Corneil, and A. Proskurovski. Complexity of finding embeddings in a k-tree. *SIAM Journal Algebraic Discrete Methods*, (8):277–284, 1987.

[Ber70] C. Berge. *Graphes et hypergraphes.* Dunond, 1970.

[CM92] M. Chein and M.L. Mugnier. Conceptual Graphs: fundamental notions. *Revue d'Intelligence Artificielle*, 6(4), 1992. In english.

[Ell93] G. Ellis. Efficient Retrieval from Hierarchies of Objects using Lattice Operations. In J.F. Sowa G.W. Mineau, B. Moulin, editor, *Lecture Notes in Artificial Intelligence, 699, Proceedings of the International Conference on Conceptual Structures'93*, pages 274–293. Springer Verlag, 1993.

[Fre82] E.C. Freuder. A sufficient condition for backtrack-free search. *JACM*, pages 29–45, 1982.

[FV93] T. Feder and M.Y. Vardi. Monotone Monadic SNP and Constraint Satisfaction. In *Proceedings of the 25th ACM STOC*, pages 612–622, 1993.

[Gai93] B.R. Gaines. Representation, discourse, logic and truth : Situating knowledge technology. In G.W. Mineau, B. Moulin, and J.F. Sowa, editors, *Lecture Notes in Artificial Intelligence, 699, Proceedings of the International Conference on Conceptual Structures'93*, pages 36–63. Springer Verlag, 1993.

[GWW92] W. Gutjahr, E. Welzl, and G. Woeginger. Polynomial graph coloring. *Discrete Applied Math.*, (35):29–45, 1992.

[HN90] P. Hell and J. Nešetřil. On the complexity of H-coloring. *J. Combin. Th.*, (B)(48):33–42, 1990.

[HNZ93] P. Hell, J. Nešetřil, and X. Zhu. Complexity of tree homomorphisms. Manuscript, 1993.

[HNZ94] P. Hell, J. Nešetřil, and X. Zhu. Duality and Polynomial Testing of Tree Homomorphisms. Manuscript, 1994.

[Jac88] M.K. Jackman. Inference and the Conceptual Graph Knowledge Representation Language. In S. Moralee, editor, *Research and Development in Expert Systems IV*. Cambridge University Press, 1988.

[Kac84] H. Ait Kaci. A lattice-Theoretic Approach to Computation Based on A Calculus of Partially Ordered Type Structures. PhD Thesis, University of Pennsylvania, 1984.

[LB94] M. Liquiere and O. Brissac. A class of conceptual graphs with polynomial iso-projection. In *Supplemental proceedings of the International Conference on Conceptual Structures*, 1994. Revised version.

[Leh92] F. Lehmann, editor. *Semantics Networks in Artificial Intelligence*. Pergamon Press, 1992.

[Lev94] R. Levinson. UDS: A Universal Data Structure. In W.M. Tepfenhart, J.P. Dick, and J.F. Sowa, editors, *Lecture Notes in Artificial Intelligence, 835, Proceedings of the International Conference on Conceptual Structures'94*, pages 230–250. Springer Verlag, 1994.

[MC92] M.L. Mugnier and M. Chein. Polynomial algorithms for projection and matching. In *Proceedings of the 7th Workshop on Conceptual Graphs*, New Mexico State University, Las Cruces, New Mexico, 1992.

[MC93] M.L. Mugnier and M. Chein. Characterization and Algorithmic Recognition of Canonical Conceptual Graphs. In J.F. Sowa G.W. Mineau, B. Moulin, editor, *Lecture Notes in Artificial Intelligence, 699, Proceedings of the International Conference on Conceptual Structures'93*, pages 294–311. Springer Verlag, 1993.

[MC94] M.L. Mugnier and M. Chein. Représenter des connaissances et raisonner avec des graphes. 52 pages. To appear in Revue d'Intelligence Artificielle, 1994.

[Mug95] M.L. Mugnier. On generalization/specialization for conceptual graphs. *J. Expt. Theor. Artif. Intell.*, 7(3):325–344, 1995.

[P.89] Meseguer P. Constraint satisfaction problems: an overview. *AICOM*, 2(1):3–17, 1989.

[Sow84] J.F. Sowa. *Conceptual Structures - Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[Sow94a] J.F. Sowa. Knowledge representation : Logical, philosophical, and computational foundations. Preliminary Edition, August, 1994.

[Sow94b] J.F. Sowa. Logical Foundations for Representing Object-oriented Systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 5, 1994.

[SW86] J.F. Sowa and E.C. Way. Implementing a semantic interpreter using conceptual graphs. *IBM Journal of Research and Development*, 30(1):57–69, 1986.