Knowledge Acquisition with a Pure Graph-Based Knowledge Representation Model

Application to the Sisyphus-I Case Study

Jean-François Baget, David Genest and Marie-Laure Mugnier

LIRMM, 161, rue Ada, 34392 Montpellier Cedex 5, France
email: {baget,genest,mugnier}@lirmm.fr

Abstract

We present a purely *declarative* graph-based formal model, which could be the basis of a *generic* modeling framework. Indeed, all kinds of knowledge, be it facts, queries, rules or constraints are represented by *labelled graphs*. Furthermore, the originality of this model is that reasonings are performed by simple to understand *graph operations*. Thus, not only knowledge is represented in an easily readable format, but the problem solving can be followed step-by-step by the user in the same graphical format. We show this graph-based approach at work in modeling a problem solving process (the Sisyphus-I case study), and explain why in our opinion this model is suited to knowledge acquisition and specifically to knowledge engineering.

1 INTRODUCTION

Labelled graphs and conceptual graphs. Different kinds of labelled graphs have long been used to represent knowledge. In artificial intelligence they have been mainly investigated under the name of semantic networks. (Sowa, 1976)(Sowa, 1984) proposed a model of this family, named *conceptual graphs* (CGs). Actually the common vocable of CGs hides a great diversity of works difficult to compare. Besides a common base called "the simple graph model" in this paper, the original model has been developed along multiple viewpoints.

One of the more attractive features of labelled graphs in general for knowledge acquisition and representation is their *visual* aspect. Indeed people like drawings. Basically a CG is a bipartite labelled graph, where one class of nodes represents entities and the other represents relationships between these entities. It is thus easily editable and interpretable by a person, in particular if it has a small number of nodes and edges or if it has a special structure enhanced by the drawing.

Another fundamental point is that CGs are also a formal knowledge representation model. They are provided with reasoning operations which are sound and complete with respect to deduction in first order logics (see (Sowa, 1984)(Chein & Mugnier, 1992) for simple graphs, (Chein *et al.*, 1998) for nested graphs, and (Sowa, 1984)(Wermelinger, 1995)(Kerdiles & Salvat, 1997) for more general graphs equivalent to first order logic).

A graph-based knowledge representation model. Since 1991, our team has been studying CGs as a *graphical* knowledge representation model, where "graphical" is used in the sense of (Schubert, 1991), i.e. a model that "uses graph-theoretic notions in an essential and nontrivial way". Indeed, not only CGs are displayed as graphs but also reasonings are based on graph operations. Our aim is not to copy down first order logic nor to propose a "universal" modeling medium. We develop a KR model with, in some sense, limited expressiveness (we cannot, by example assert a disjunction as true) but graph-based reasonings. We see two main advantages in this kind of reasonings. From a computational viewpoint, reasonings benefit from combinatorial algorithms. From a modeling viewpoint, reasonings can be visualized in a natural way and

are simple to understand for an end-user. We believe this property is particularly significant for knowledge acquisition (this feature will be further discussed in the last section of this paper). For an in-depth discussion about graph-based reasonings, see (Chein, 1997).

The formal bases of our work are the following. Basic objects are simple CGs, i.e. labelled graphs. The fundamental reasoning operation, called projection, is a graph matching between simple CGs. Reasonings are logically founded, since projection is sound and complete w.r.t. deduction in first order logic. We built extensions of this kernel (for instance (Salvat & Mugnier, 1996)(Chein *et al.*, 1998)) keeping a pure declarative graph-based model with logically founded reasonings. We also developed the software CoGITo (Haemmerlé, 1995)(Chein, 1997), and its extension CoGITaNT (Genest & Salvat, 1998), a workbench for building knowledge-based applications, where every piece of knowledge is described by CGs. The theoretical framework and CoGITo have been used in several applications ((Bos *et al.*, 1997)(Genest & Chein, 1997) describe the applications we are currently involved in).

Modeling the Sisyphus-I case study. The aim of this paper is to show this graph-based approach at work on the Sisyphus-I case study, which has been largely discussed within the knowledge acquisition community (Sisyphus'94, 1994). The Sisyphus-I problem statement describes a resource allocation problem (specifically, the task is to allocate the members of a research group to different offices, given certain constraints) and a solution proposed by an expert, as an annotated protocol. The aim of this initiative for the KA community was "to compare different approaches to the modeling of problem solving processes in knowledge-based systems and the influences of the models on the knowledge acquisition activities".

This case study has been imported into the conceptual graph community as a common test-bed in order to ease comparison of the different approaches derived from Sowa's original theory (cf. the SCG-1 initiative associated to ICCS'99). The paper presenting our answer to this initiative (Baget *et al.*, 1999) details the graph-based formal model and focuses on the way the problem is solved within this framework. Here we will address the Sisyphus-I case study from a knowledge acquisition viewpoint and point out expertise modeling aspects. As the fuzziness of the annotated protocol has already been largely commented, and our interrogations about it are not essentially different to those that can be found in (Sisyphus'94, 1994) we will not develop these points.

The Sisyphus-I problem is a resource allocation task, therefore basically a constraint satisfaction problem. In our previous applications, reasonings were mainly based on deduction (given a knowledge base and a query, find whether the query can be deduced from the knowledge base). Therefore, at least superficially, i.e. in its formulation, the Sisyphus-I problem is not of same nature as the problems we used to solve in our framework.

Thus, when we studied the problem statement, an immediate question arose: how much could the problem be represented and solved within our framework and more generally keeping a pure graph-based approach? The first decision was to keep a completely *declarative* model and to build a *generic* solution. Another decision, architectural this time, was to design a system able to solve the problem in an entirely *automatic* way. The reason for this choice was not that we did not attach importance to human involvement in the problem solving process. Indeed it is unrealistic, in many "real problems" to hope to find a solution satisfying the user without offering him the possibility to add, remove and reformulate data during the solving process. But our first objective was to empirically test the theoretical framework and to offer a base for a modeling environment. In its present state the prototype is an extension of CoGITaNT and not yet a usable KA tool.

Paper organization. The paper is organized as follows. In section 2 the theoretical framework is presented. Section 3 proposes a modelization of the Sisyphus case study within this framework. Section 4 is devoted to the prototype architecture and the obtained experimental results. We end with a discussion of our framework from a knowledge acquisition viewpoint.

2 THE GRAPH-BASED MODEL

In this part the formal graph-based model underlying our knowledge acquisition approach is outlined. Definitions are deliberately simplified. For more details see (Baget *et al.*, 1999).

2.1 Presentation

All objects in this model, be it facts, queries, rules, or constraints, are labelled graphs. The elementary reasoning operation is a graph matching (or graph morphism in mathematical terms) known as projection. It maps a graph onto a more specific one. This operation is logically founded. Indeed basic objects (called simple graphs) correspond to positive, conjunctive and existential first-order logic. And projection between simple graphs is equivalent to deduction. The simplest form of reasoning based on projection is the following. Let F be an asserted fact and Q be a query. Every projection from Q to F is an answer to the query. Three kinds of graphs are defined: simple graphs, rules and constraints.

- A *simple graph* is interpreted as a fact or a query.
- A *rule* allows one to add new knowledge. It is composed of an hypothesis and a conclusion, and is used in the following way: given a simple graph, if the hypothesis of the rule matches the graph, then the information contained in the conclusion is added to the graph. Rules are split into inference rules (also called implicit knowledge rules) and transformation rules.
- A *constraint* defines conditions for a simple graph to be valid. It is composed of a condition part and a mandatory part. A graph satisfies a constraint if for every matching of its condition part, its mandatory part also matches the graph.

Though formal definitions will be provided later, we first give an intuitive presentation of the global reasoning scheme. The problem to be solved can be summarized as follows: given a *knowledge base* (composed of facts, implicit knowledge rules, transformation rules and constraints) and a *query*, is it possible for this knowledge base to *evolve* in such a way that the query is satisfied? In this global scheme facts describe an initial world. Implicit knowledge rules complete world descriptions. Transformation rules define possible transitions from a world to other ones. Constraints are used to verify the validity of worlds (the world description, enriched by implicit knowledge rules, must satisfy every constraint). A successor of a valid world is obtained by a single application of a transformation rule on this world. Solving the problem is to find a chain of valid worlds evolving from the initial one such that the last one matches the query.

2.2 Elementary Notions

We first present basic notions about the simple graph model. Ontological knowledge is encoded into a *support* describing available concept and relation types ordered by a specialization relation. Facts are represented by *simple graphs*, which nodes represent individual instances of concepts and relations between them. The fundamental reasoning operation is a graph morphism called *projection*. It induces a subsumption relation between simple graphs. These basic notions correspond to those introduced in (Sowa, 1984).

Support. Basic ontology is encoded in a structure called a *support*. We consider here a very simplified version $\mathcal{S} = \{\mathcal{T_C}, \mathcal{T_R}, \mathcal{I}\}$ of a support, where: T_C is a partially ordered set of concept types whose greatest element is $\top$; T_R is a partially ordered set of relation types. For the Sisyphus-I problem, we only use binary relations (but in general relations may be of any arity greater or equal to one). Each relation type has a signature, which gives the greatest possible concept type for each argument. I is a countably infinite set of individual markers.

Simple graphs. Facts are represented by *simple graphs*. A simple graph is a bipartite labelled multigraph (e.g. two nodes can be linked by more than one edge) $G = \{V_C, V_R, E, l\}$, where V_C is the set of concept nodes, V_R the set of relation nodes, E is the set of edges, and l labels nodes and edges. The label of a concept node is composed of a concept type from T_C and a marker, which may be either an element of I (then the node represents a specific individual) or the generic marker denoted by $*$. The label of a relation node is a relation type from T_R. Every edge is labeled by 1 or 2, for indicating which is the first and the second argument of each relation node. Note that in the case of binary relations, we can use oriented edges to simulate these numbers. Labels on concept nodes must respect the signature defined for their relation neighbors.

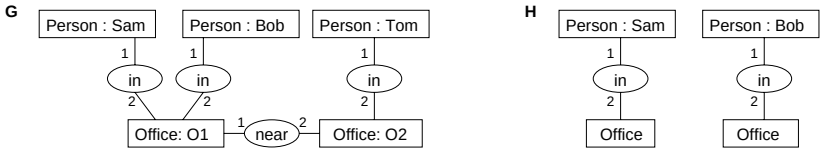

FIGURE 1: Two simple graphs, G and H

Fig. 1 shows two simple graphs. Concept nodes are represented by rectangles, containing the type and marker of the node (for readability purpose, generic markers are not represented). Relation nodes are represented by ovals containing the type of the node. The graph G represents three *persons*, *Sam*, *Bob* and *Tom*, and two *offices*, *O1* and *O2*. *Sam* and *Bob* are *in* the *office O1*, *Tom* is *in* the *office O2*, and *office O1* is *near office O2*.

Projection. Asking for the existence of a projection from a graph H to a graph G can be seen as querying whether all information contained in H is also contained in G. If it is the case, H is said to subsume G.

More precisely, a projection from a simple graph H into a simple graph G is a mapping Π from the nodes of H into the nodes of G that may decrease nodes labels (for every node of H, $l(H)$ is more generic than $l(\Pi(H))$), and preserves the adjacency relation (if there is an edge between two nodes x and y in H, then there must be an edge with the same label between $\Pi(x)$ and $\Pi(y)$).

By example, there is one projection from the graph H into the graph G in fig. 1. H expresses that *Sam* is in an *office* and *Bob* is also in an *office*. As a query, it could be read as "Are *Sam* and *Bob* each in an *office*?" The nodes labelled *Sam* and *Bob* in graph H can be projected into the nodes having the same label in graph G, the two *offices* in graph H are both projected into the *office O1* in graph G. The projection of relation nodes is determined by the projection of their neighbors.

We can see in this example that projection is generally not surjective, nor injective. In order to express that two concept nodes of a graph represent distinct entities (thus they can never be projected onto the same node) we use a symmetrical relation called *diff*. If every pair of concept nodes is connected by a *diff* relation node then projection behaves as an injective mapping over the concept nodes set.

2.3 Inference Rules

Rules are used to represent knowledge of form "if information A is contained in a graph, then information B can be added to this graph". They are represented by a bi-colored simple graph, the first color subgraph being the hypothesis part, the second color subgraph being the conclusion part.

Rules. We obtain a rule by assigning a color on $\{0, 1\}$ to each node of a graph, in such a way that the subgraph generated by 0-colored nodes is a simple graph. The subgraph generated by 0-colored nodes is called the *hypothesis* of the rule, and the subgraph generated by 1-colored nodes is called its *conclusion*. Concept nodes of the hypothesis part with at least one neighbor in the conclusion part are the *frontier* nodes.

In the drawing of a rule, we represent hypothesis nodes in a white rectangle or oval, and conclusion nodes in black. To differentiate rules from other modeling constructs, we indicate them by the symbol ➔. The rule *R1* in fig. 2 expresses that "every office is near to itself" and rule *R2* in the same fig. that *near* is a symmetrical relation.

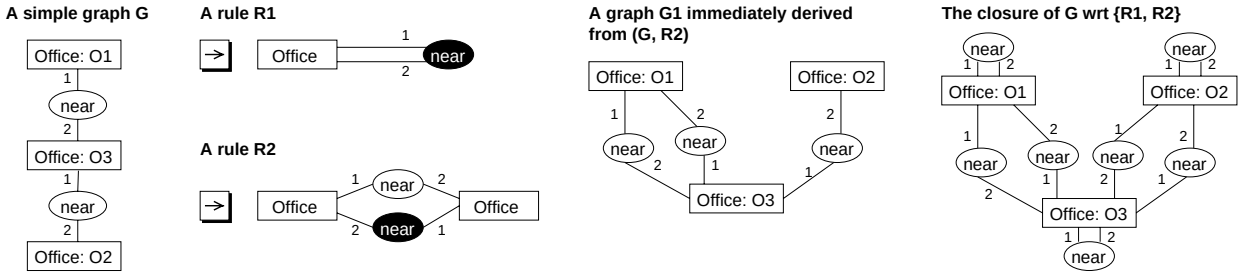

FIGURE 2: Rules, derivation and closure of a graph

Applying a rule. A rule R is said to be *applicable* to a simple graph G if there is a projection, say Π, from the hypothesis of R into G. In this case, the result of the application of R to G following Π is the simple graph G' obtained from G and the conclusion of R by restricting the label of each frontier node c in the conclusion to the label of its image $\Pi(c)$ in G, then joining c to $\Pi(c)$.

In fig. 2, the graph *G1* is obtained by applying *R2* to G once. Note that when a rule is applicable to a graph following a projection, it can be indefinitely applied to the resulting graph following the same projection, but all graphs obtained are equivalent to the graph built by the first application. In what follows a rule is applied *only once* to a graph following a given projection.

The derivation process. Let G be a simple graph and $\mathcal{R}$ be a set of rules. We say that a graph G' is immediately derived from $\{G, \mathcal{R}\}$ if there exists a rule $R \in \mathcal{R}$ and a projection Π from the hypothesis of R into G such that G' is the result of the application of R to G following Π. We say that a graph G' is derivable from $\{G, \mathcal{R}\}$ if there exists a sequence of immediate derivations from G to G'.

A decidable fragment. Let G and H be two simple graphs, and $\mathcal{R}$ be a set of rules. The problem of knowing whether there exists a graph G' derived from G such that there exists a projection from H into G' is semi-decidable (Coulondre & Salvat, 1998). It is obviously decidable if we restrain ourselves to rules with only relation nodes in their conclusion part (as will be done in the Sisyphus modeling). In this case, we can compute the *closure* of a graph with respect to a set of rules. A graph G is said to be *closed* w.r.t. such a set of rules $\mathcal{R}$ if all information that can be added by a rule is already present in G (i.e. more formally, for each rule $R \in \mathcal{R}$, each projection from the hypothesis of R into G can be extended to a projection from R as a whole into G). A *closure* of a simple graph G by such a set of rules $\mathcal{R}$ is a simple graph derived from $(G, \mathcal{R})$, closed w.r.t. $\mathcal{R}$. All closures are equivalent with respect to projection, and the minimum element of this equivalence class is called *the* closure of G w.r.t. $\mathcal{R}$ and is denoted by $G^*_{\mathcal{R}}$. Fig. 2 presents the closure of G w.r.t. $\{R1, R2\}$.

Our deduction problem is then equivalent to finding a projection of H into the closure of G w.r.t. $\mathcal{R}$.

2.4 Constraints

In order to express a judgment on the validity of a graph, we add the notion of constraints. A simple graph is said to be valid if it satisfies every constraint defined in the knowledge base. We present in this paper two kind of constraints, positive ones, and negative ones. For positive constraints, an intuitive semantic could be "if information A is present, then we must also find information B". For negative constraints, it could be "if information A is present, then we must not find information B". Let us add that a more general form of constraint can be handled, corresponding to the following semantics: "if information A is present and B is absent then we must find information C and not D". But it is not needed in the Sisyphus-I modelization.

Formal constructs. A constraint, be it positive or negative, is defined in the same way as a rule, as a simple graph provided with a coloration of its nodes with two colors 0 and 1. The subgraph generated by nodes whose color is 0 is called the *condition* of the constraint. The condition must be a valid simple graph. In particular, the condition can be an *empty graph*.

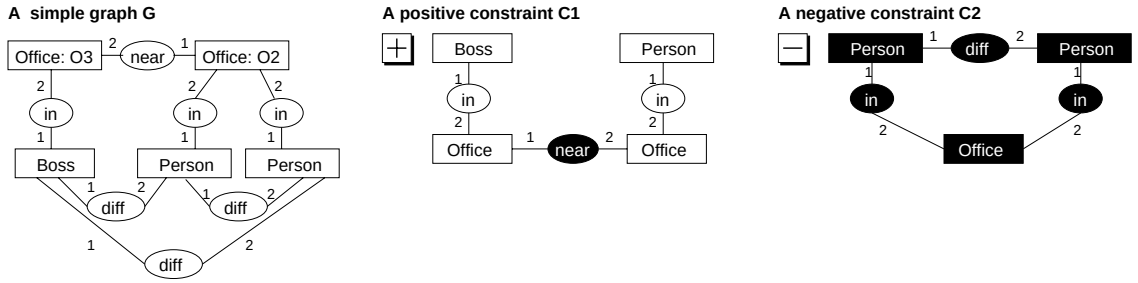

FIGURE 3: Positive and negative constraints

In the drawing of a constraint (indicated by ⊞ in the case of positive constraints and ⊟ in the case of negative ones), condition nodes are represented in white and the others in black. Fig. 3 presents two constraints: a positive one, *C1*, expressing that "if a *boss* is *in* an *office* and a *person* is in an *office*, then this latter office must be *near* the *boss office*", and a negative one, *C2*, expressing that "no two *different persons* can be in the same *office*".

Validity of a graph. Let G be a simple graph, and C be a positive constraint. We say that G *satisfies* C if *every* projection Π of the condition of C into G can be extended to a projection of C as a whole.

Let G be a simple graph, and C be a negative constraint. We say that G *satisfies* C if *no* projection Π of the condition of C into G can be extended to a projection of C as a whole.

Let G be a simple graph, $\mathcal{C}$ be a set of positive and negative constraints. The graph G is said to be *valid* with respect to $\mathcal{C}$ if G satisfies every constraint of $\mathcal{C}$.

The simple graph G defined in fig. 3 violates constraint *C1* as well as *C2*. By example, projecting both the *boss* and the *person* of the condition of *C1* into the *boss* of G, and both *offices* of *C1* into the *office O3* of G, we verify that *C1* is not satisfied, since *O3* is not *near* to itself. In the same way, *C2* is violated since there are two *different persons* in the office *O2* in G. Of course, violation of the constraint *C1* reveals an evident weakness in the modelization, which will be corrected in the next paragraph.

Validity of a graph given implicit knowledge. Let G be a simple graph, $\mathcal{R}$ be a set of rules (having only relation nodes in their conclusion) representing implicit knowledge, and $\mathcal{C}$ be a set of positive and negative constraints. We say that the knowledge base $(G, \mathcal{R})$ is valid with respect to $\mathcal{C}$ if and only if the closure $G^*_{\mathcal{R}}$ of G is valid with respect to $\mathcal{C}$.

Building the closure of the graph G in fig. 3 w.r.t. the rules defined in fig. 2, we obtain a graph satisfying *C1*, but still violating *C2*.

2.5 Transformation Rules

Consider that *asserted facts* (simple graphs) and *implicit knowledge* rules describe a *world*. Constraints determine whether this world is valid or not. Let us now introduce another set of rules, called *transformation rules*. These rules are used to *generate* new worlds which may be valid or not.

The simplest transformation rules have the same form as inference rules defined in part 2.2. but the intuitive semantic becomes "if information A is present in a world, then generate a new world by adding information B to this world". A more general form could be handled, with hypothesis split into a positive part ("if information A is present") and a negative part ("and if information B is absent"). Such a rule would be encoded by a graph with three colors. It is applicable to a graph if there is a projection from the positive part of the hypothesis onto the graph that cannot be extended to a projection of the whole hypothesis ("A is found but we cannot find B"). Note that it would not be possible to generalize in the same way implicit knowledge rules because the derivation mechanism would become non monotonic (for instance the "unique closure" property would not hold true anymore).

Given a valid *initial world* $(G, \mathcal{R})$, a set of constraints $\mathcal{C}$ and a set of transformation rules $\mathcal{T}$, answering a query H consists in building a sequence of successive valid worlds issued from the initial one, such that the last one is an answer to H. Below is a formal definition for such a derivation.

Valid immediate transformation Let G be a simple graph, $\mathcal{R}$ be a set of rules representing implicit knowledge, $\mathcal{C}$ be a set of positive and negative constraints, and $\mathcal{T}$ be a set of *transformation rules*. We say that a graph G' is a *valid immediate transformation* of G with respect to $(\mathcal{R}, \mathcal{C}, \mathcal{T})$ if and only if: $(G, \mathcal{R})$ is valid with respect to $\mathcal{C}$, G' is immediately derived from $(G^*_{\mathcal{R}}, \mathcal{T})$ and $(G', \mathcal{R})$ is valid with respect to $\mathcal{C}$.

Valid transformation. We say that a graph G' is a *valid transformation* of G with respect to $(\mathcal{R}, \mathcal{C}, \mathcal{T})$ if and only if there exists a sequence $G = G_0, G_1, \ldots, G_k = G'$ of graphs such that each G_{i+1} is a valid immediate transformation of G_i with respect to $(\mathcal{R}, \mathcal{C}, \mathcal{T})$. We also say that this sequence is a *constrained derivation* from G to G'.

The general deduction problem. The global problem to be solved can now be expressed as follows: given a *facts graph* G, a set $\mathcal{R}$ of implicit knowledge rules, a set $\mathcal{C}$ of constraints, a set $\mathcal{T}$ of transformation rules, and given a query H, is it possible to find a constrained derivation from G to a graph G' such that H can be projected into G'.

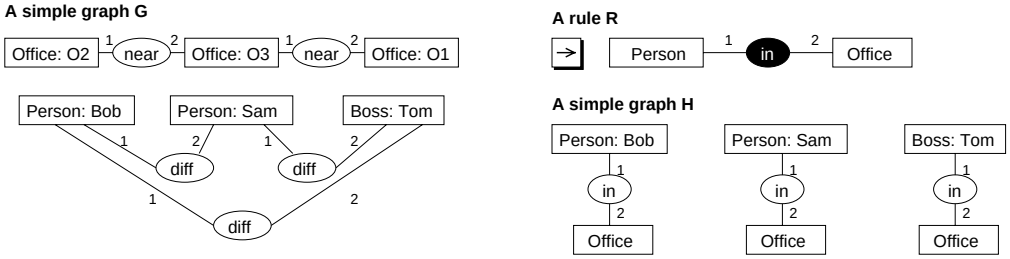

FIGURE 4: A graph G, a transformation rule R, and a query H

By example, let G be the graph defined in fig. 4, $\mathcal{R}$ be the set of rules defined in fig. 2, $\mathcal{C}$ be the constraints defined in fig. 3, $\mathcal{T}$ be the set of transformation rules containing only the rule R in fig. 4, expressing that we can assign an office to each person, and the query H be the graph defined in fig. 4. This query expresses that we want all three persons to be placed into an office. In order to solve the problem, we first compute the closure of G w.r.t. $\mathcal{R}$, which is a valid graph, then apply the transformation rule once, say by placing *Bob* into the third office. Now, we verify that this graph is valid, and try another assignment, until the query is satisfied. Fig. 5 traces the transformation rule applications leading to a correct answer.

Tom → 1
  Sam → 1 Violation: two in the same office
  Sam → 2 Violation: far from boss
  Sam → 3
    Bob → 1 Violation: two in the same office
    Bob → 2 Violation: far from boss
    Bob → 3 Violation: two in the same office
Tom → 2
  Sam → 1 Violation: far from boss
  Sam → 2 Violation: two in the same office
  Sam → 3
    Bob → 1 Violation: far from boss
    Bob → 2 Violation: two in the same office
    Bob → 3 Violation: two in the same office
Tom → 3
  Sam → 1
    Bob → 1 Violation: two in the same office
    Bob → 2 **Solution found**

FIGURE 5: A trace of the backtrack leading to the solution

2.6 Knowledge Acquisition Strategy

The knowledge acquisition strategy is directly grounded on the graph-based model. It consists of the following steps (their order is not significant, because of the strong interdependence between different kinds of knowledge)

- **Define the ontology.** The first step is to define all types for entities and relations we need to represent, and the specialization relation between them.
- **Represent facts.** Simple graphs offer a readable way to represent the positive information defining the initial world. *Sisyphus:* We represent as facts information on persons, offices and projects, according to the defined ontology.
- **Define the query.** What is a satisfying result ? What information should be contained in a world, to be a solution to the problem ? *Sisyphus:* It is an assignment of each person into an office.
- **Represent implicit knowledge.** Not only this set of rules is used to factorize initial information, but it can be of crucial importance to compute every consequence of a given transformation of the world. By example, suppose that there exists a transformation rule that adds the information "the object A is *smaller* than the object B". Then implicit knowledge (asserting that *smaller* is a transitive relation) would be used to update the resulting graph.
- **Represent constraints.** Constraints define what a valid world is. They have to be satisfied by every world derived from the initial one. *Sisyphus:* Constraints translate obligations and interdictions for acceptable assignments in a rather straightforward way.
- **Represent possible evolutions of the world.** Transformation rules define the way the world evolves towards a satisfying one. *Sisyphus:* A person is assigned to an office at each step of the reasoning cycle.

3 MODELIZATION OF THE SISYPHUS CASE STUDY

Our modelization of the Sisyphus-I problem is based on the annotated protocol. We detail in this part the components of our solution, and show how the framework enables explicitation of the expert strategy.

3.1 The Support

The support defined in fig. 6 and 7 represents all types used in our modelization. Each relation type in fig. 7 is provided with its signature.

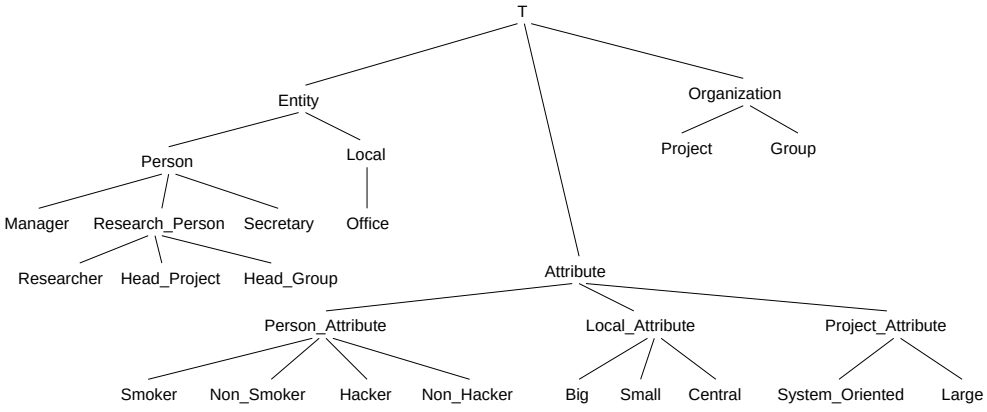

FIGURE 6: Hierarchy of concept types

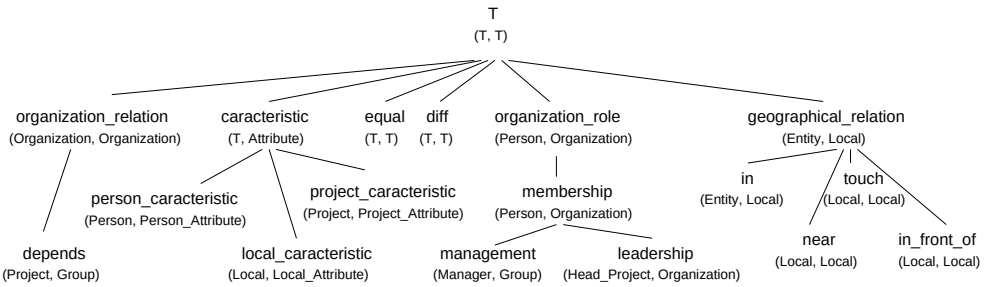

FIGURE 7: Hierarchy of relation types

3.2 Fact Graphs

The initial graph is obtained by making the disjoint union of the graphs represented in figs. 10, 11, 8 and 9. These graphs have been separated for better readability. To ensure completeness of the reasoning process, we compute the *normal form* of the resulting graph, by merging concept nodes having the same individual marker. For a readability purpose, we did not represent *diff* relations in these graphs, though in the graph of fig. 10, all pairs of concept nodes typed *Office* and having different individual markers are assumed to be linked by a relation node typed *diff*. The same assumption is done for the concept nodes typed *Project* in graph of fig. 11, and for concept nodes whose type is more specific than *Person* in graph of fig. 9.

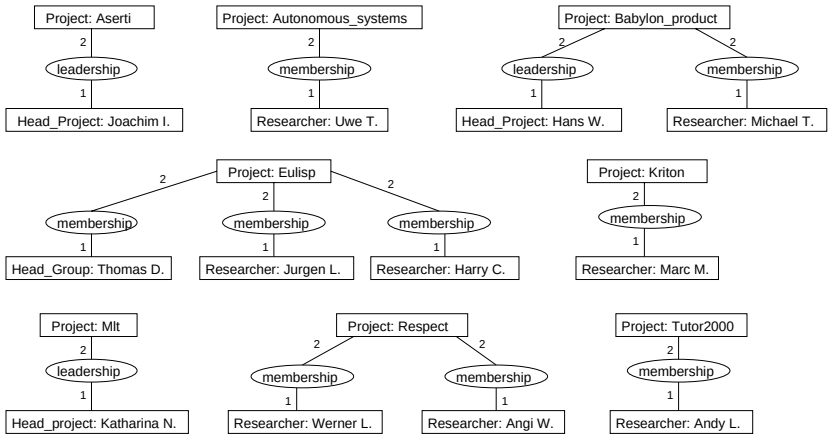

FIGURE 8: Team members of the YQT group

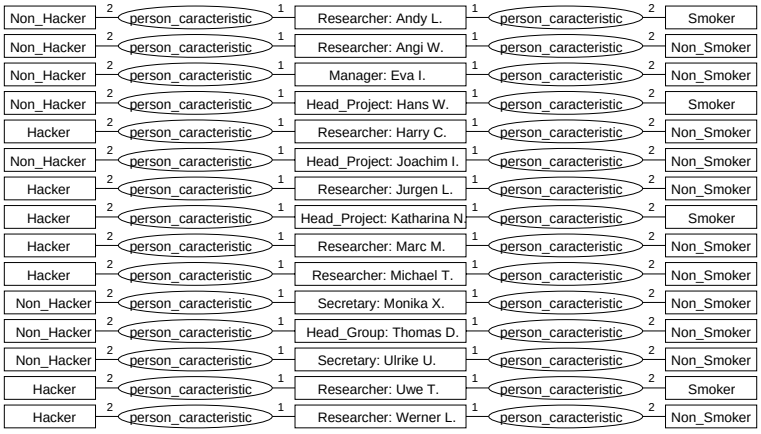

FIGURE 9: Personal data about members of the YQT group

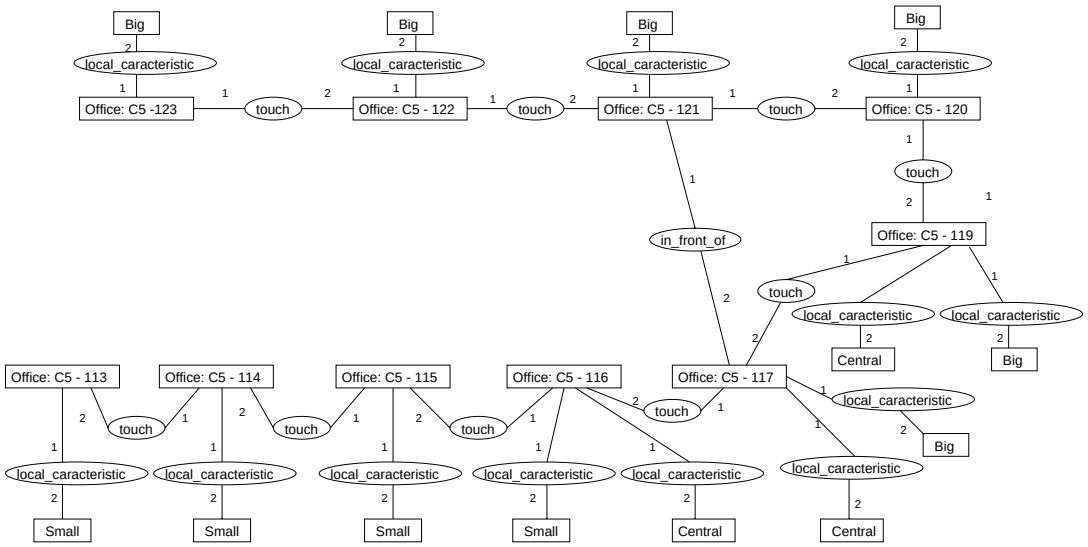

FIGURE 10: Geographical information for the first floor of the château of HNE

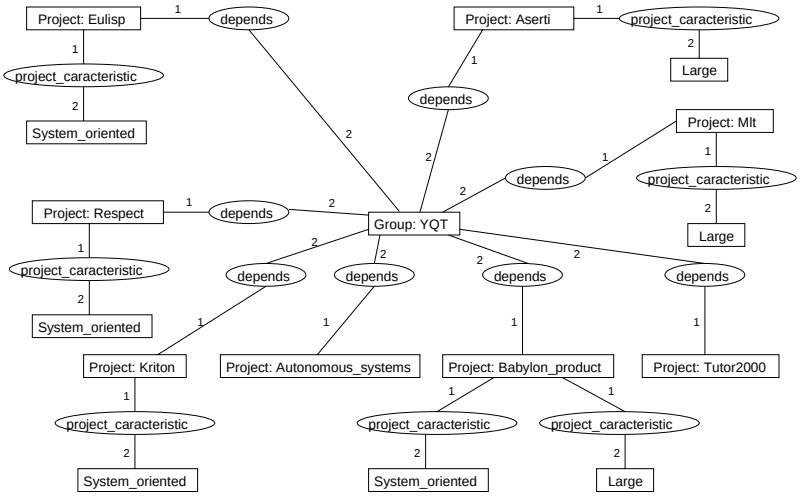

FIGURE 11: Organizational structure of the YQT group

3.3 The Query

The graph we want to deduce is the solution to the Sisyphus-I problem: we want every person being placed into an office. This graph is represented in fig. 12. Note that, due to individual markers, no *diff* relation is needed between the concept nodes representing persons.

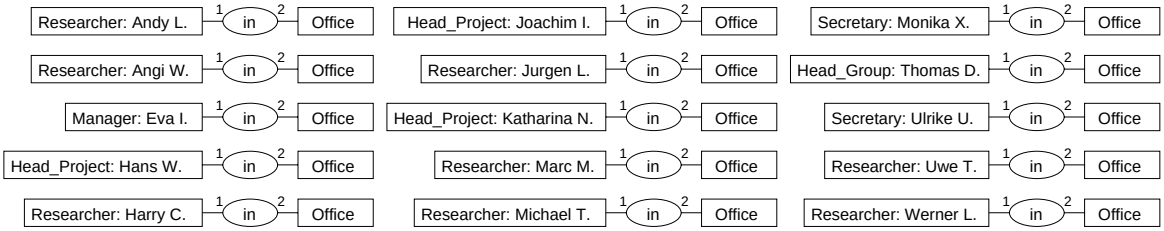

FIGURE 12: The query

3.4 Implicit Knowledge – the Set of Rules $\mathcal{R}$

The five rules represented in fig. 13 express that:

A. The relation *near* is symmetrical.
B. Two locals that *touch* the same one are considered *near*.
C. The relation *touch* is symmetrical.
D. Two locals that *touch* each other are considered *near*.
E. Two locals that are *in_front_of* each other are considered *near*.

Note that the two last rules could be replaced by expressing that *in_front_of* and *touch* are two relation types more specific than *near*. We also used rules expressing that *in_front_of* and *diff* are symmetrical.

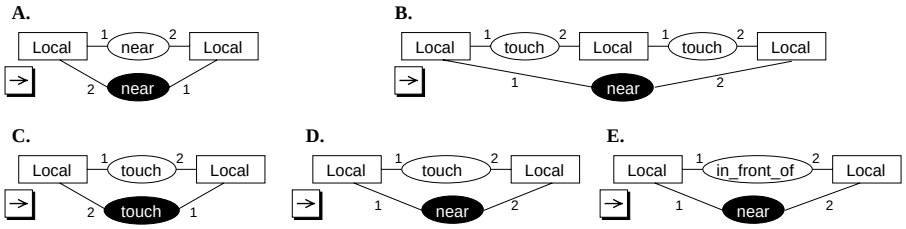

FIGURE 13: Rules representing implicit knowledge

3.5 Constraints – the Set $\mathcal{C}$

Constraints related to possible assignments can be either positive or negative. We sorted these constraints in three categories of decreasing priority: absolute, strong and weak. It is not possible to violate an *absolute* constraint. *Strong* constraints are to be satisfied by any solution, but, if the problem is over-constrained, the user may accept to reformulate it by modifying this set of constraints. *Weak* constraints represent preferences.

Every constraint used is described below, but whenever we have very similar constraints, only one of them is drawn. These constraints are represented in order of declining priorities.

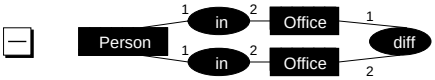

FIGURE 14: Ubiquity ?

The graph represented in fig. 14 is a negative constraint. It expresses that a person cannot be into two different places at the same time. This is an *absolute* constraint.

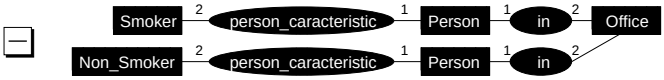

FIGURE 15: Smoker – Non-Smoker antagonism

The graph represented in fig. 15 is a negative constraint. It expresses that no office can host a *Smoker* and a *Non_Smoker*. This is a *strong* constraint.

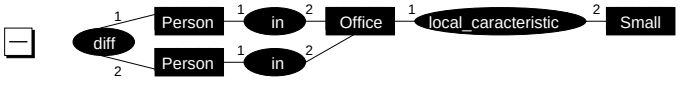

FIGURE 16: Number of persons in small offices

The graph represented in fig. 16 is a negative constraint. It expresses that a small office cannot host more than one person. This is a *strong* constraint.

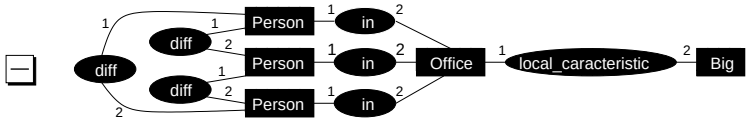

FIGURE 17: Number of persons in big offices

The graph represented in fig. 17 is a negative constraint. It expresses that a big office cannot host more than two persons. This is a *strong* constraint.

FIGURE 18: Head of group accessibility

The graph represented in fig. 18 is a positive constraint. It expresses that the head of group needs a central office (if the head of group is in an office, then this office has to be a central one). This is a *strong* constraint. In the same way, the manager would be pleased to have a central office, but we consider this as only a *weak* constraint.

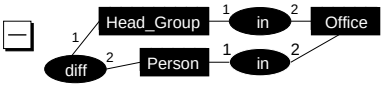

FIGURE 19: Privileges of the head of group (1)

The graph represented in fig. 19 is a negative constraint. It expresses that the head of group needs to be alone in his office. This is a *strong* constraint. In the same way, the manager as well as heads of large projects would be pleased to be alone in their office, but we consider this as only *weak* constraints.

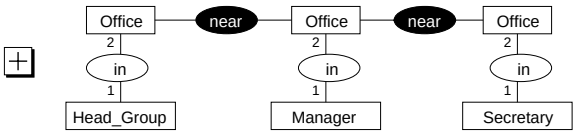

FIGURE 20: Manager's neighborhood

The graph represented in fig. 20 is a positive constraint. It expresses that the manager needs to be near the head of group as well as the secretariat. This is a *strong* constraint. In the same way, heads of large projects should be close to the head of group as well as the secretariat, but we only consider this as only a *weak* constraint.

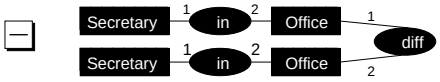

FIGURE 21: Secretariat holds secretaries

The graph represented in fig. 21 is a negative constraint. It expresses that no two secretaries can be in different offices. Note that this constraint can be satisfied since there are only two secretaries. Otherwise, should we want "secretaries-only" offices, we could express it by a positive constraint: "if a person is in the same office as a secretary, this person must also be a secretary". This is a *weak* constraint.

FIGURE 22: Privileges of the head of group (2)

The graph represented in fig. 22 is a positive constraint. It expresses that the head of group would like to have a big office. This is a *weak* constraint.

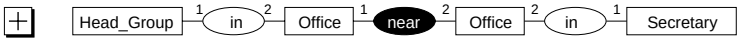

FIGURE 23: Secretariat's accessibility

The graph represented in fig. 23 is a positive constraint. It expresses that the secretaries' office should be located close to the office of the head of group. This is a *weak* constraint.

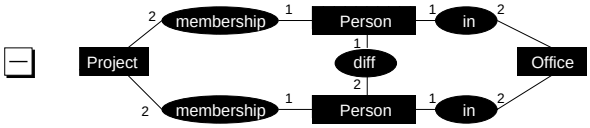

FIGURE 24: Synergy

The graph represented in fig. 24 is a negative constraint. It expresses that members of a same project should not share offices. This is a *weak* constraint.

3.6 Transformations – the Set of Rules $\mathcal{T}$

The simplest set of transformation rules $\mathcal{T}$ that would do the job is composed of a single rule: "if you can find a *person* and an *office*, try to place that *person* in that *office*" (fig. 25).

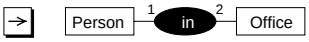

FIGURE 25: Assigning a person to an office

This rule is rather inefficient and is a poor representation of the expert strategy, since it leads the system trying to allocate an office to a person even if this person already has one (after which the constraint of "non ubiquity" (fig. 14) would reject this double allocation). A trivial improvement would be to use the rule represented in fig. 26 : "try to place a *person* into an *office* if this *person* does not already have one" (positive hypothesis is drawn in white, negative hypothesis in grey, and conclusion in black). Note that the "non-ubiquity" constraint becomes useless. This rule could be further improved by requiring that the office is not full. This would imply to add a concept type *"Full"*, subtype of *Local_Attribute*, and implicit knowledge rules saying that a *Small* (resp. *Big*) office with one person (resp. two persons) inside is full. But we have chosen to privilege simplicity and to distinguish as much as possible between the different kinds of knowledge (here between transformation rules – what should be tried – and constraints).

In the next section, we show how the expert strategy can be better modeled using an order on the transformation rule set.

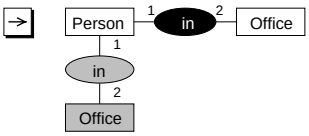

FIGURE 26: Placing an unassigned person into an office

3.7 Modeling the Expert Strategy

We detail in this section how ordering constraints and transformation rules can bring a better explicitation of the expert strategy.

Ordering constraints. First, what should be done in the case no solution is found satisfying every constraint? In this case, Siggi knows which constraints can be given up in order to find a solution. We simulate that knowledge by adding an external judgment on the relative strengths of these constraints as explained in section 3.5.

To take into account the different priorities assigned to constraints, we used the following algorithm. We sorted all constraints along a total order such that we have *weak* constraints first, then *strong* constraints, then *absolute* constraints. If we find a solution satisfying every constraint, then we return this solution. Otherwise, we remove the weakest constraint and try to find a solution to this modified problem. If we can find a solution by removing only weak constraints, we answer "yes, there is a solution if we remove the following constraints ...". If the only solutions are obtained by removing at least one strong constraint, then we answer: "no, there is no solution unless you accept to remove the following constraints ...". If there is still no solution after having removed all weak and strong constraints, then we answer "there is no solution to the problem".

Sorting constraints is thus a way to assert preferences on the set of acceptable solutions. Solutions satisfying the same set of constraints are considered equally acceptable. Note that, if we give up a constraint, our model does not try to minimize the number of violations of this constraint. We could use more sophisticated mechanisms to evaluate our solutions, considering, by example, the number of constraint violations. In this case, we should either minimize the number of constraints violated in *every* world between the initial one and the one satisfying the request, which would cause some loss in efficiency, or just minimize violations in the world satisfying the request.

Ordering transformation rules. The expert *knows* that it is *better* to assign some persons before others. One way to introduce this strategy in the reasoning model is to split the transformation rule into several ones, one for each category of person (according to their type), or even for each person (according to their marker). Note that other strategies based on categories of offices (e.g. allocate large offices before single ones) or dealing differently for specific people or offices (e.g. allocate *Thomas D.* to *office 117* first) are easily modeled in this framework. The expert priorities are simulated by a complete ordering of these rules, which is taken into account by the backtracking process. This strategy does not modify the completeness of our method, since a "bad" strategy will only lead to a slower research.

Computational efficiency can be enhanced by adding information in transformation rules (for instance the head of group allocation rule may be replaced by "if a head of group is without office and if there is a large and central office, try this allocation") but once again we prefer to clearly distinguish between allocation constraints and actions. Ultimately all constraints defined in the Sisyphus-I case study could be integrated into transformation rules, but with some loss of simplicity.

To sum up, the expert strategy is explicited in a totally declarative way:

- by an ordered set of constraints, clustered into three categories (absolute, strong and weak constraints). The order inside each category has an effect on efficiency but also on the constraint relaxation process, which occurs when the problem is over-constrained.
- by an ordered set of transformation rules

Finally let us add that a possible extension of our framework could be to allow dividing the query into an ordered set of smaller queries, and to explore the solution space in a goal-oriented manner following the query ordering.

4 IMPLEMENTATION

This section first outlines the system architecture. We then present some considerations about the computational complexity of the solving process and experimental results. The last part shows how the system can cope with changes in data.

4.1 System Architecture

We implemented the constrained derivation mechanism on top of the CoGITaNT platform (Genest & Salvat, 1998). Our work is yet a prototype and not a definitive tool. CoGITaNT can be seen as a tool for manipulating graphs, computing projections and applying rules, all graphs involved being read in a textual format called BCGCT format (Haemmerlé, 1995). As can be seen in fig. 27, the prototype is based upon a client–server architecture. A *Graphical Editor* has been designed, which communicates with the *Editor Server*. It allows the user to edit and modify simple graphs, rules and constraints which form the knowledge base located on a distant server.

We have added on top of CoGITaNT a constrained derivation engine (it is not yet provided with the necessary optimizations), which takes care of the whole backtracking process. This engine is intended to communicate with a user-friendly *Problem Manager* client, via an extended server, the *Constrained Derivation Server*. It should enable the user to select rules and constraints needed for a particular problem solving and visualize a step by step evolution of the KB by interacting with the graphical editor. This *Problem Manager* is not yet ready, and we had to hard-code the Sisyphus-I problem on top of the constrained derivation engine.

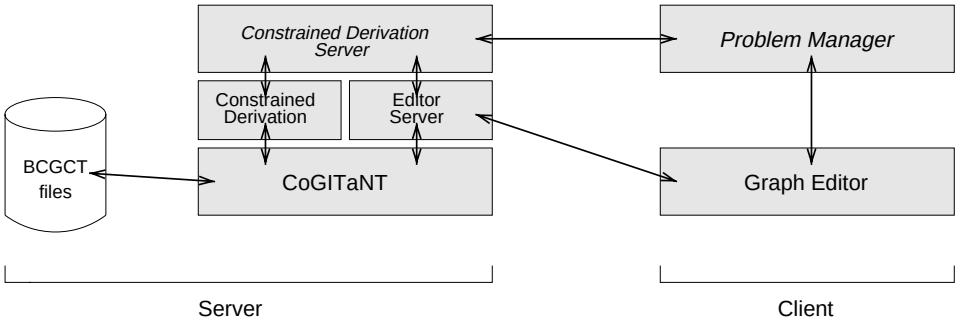

FIGURE 27: System architecture

4.2 Combinatorial Considerations

The existence of a projection is a NP-complete problem (Chein & Mugnier, 1992). The existence of a deduction from a simple graph and a set of inference rules is a semi-decidable problem (Coulondre & Salvat, 1998). For the specific case of rules adding only relation nodes, the problem is obviously decidable. Deciding whether a graph is valid with respect to a set of constraints is a co-NP-complete problem (as we need to exhibit two projections in order to prove that a constraint is violated). If the set of rules $\mathcal{R}$ representing implicit knowledge can only add relation nodes to a graph, the constrained derivation problem is semi-decidable. If the set of rules $\mathcal{T}$ representing the possible transformations of the world has the same property, the problem is decidable. This is the case in our modeling of the Sisyphus-I problem.

Let us add that, though the elementary operation (computing a projection from H to G) is intractable in the general case, it has been proven polynomial for specific structures of H (for instance, H is a tree). As all graphs to be projected (the query, hypothesis of rules, and conditions of constraints) are designed by a human being, we can expect they possess a "simple" structure, thus they can be efficiently processed.

We consider now the tree representing the possible applications of the transformation rule. For each step, we have to choose between *# of persons* $\times$ *# of offices = 150* possible assignments of a person into an office. There are *# of persons = 15* such steps, and this leads us to the study of the validity of 150^{15}

different worlds. Even if we consider that constraint violations prune many branches of this tree, this is not reasonable.

Since our processing engine still lacks necessary optimizations, we have chosen, for this problem only, to force a particular person to be assigned at each step, and this choice leads to *only* 10^{15} possible worlds. We also forced these assignments to follow the specific order suggested by the wizard Siggi. These restrictions, however incomplete in a general case, are well-founded in the case of our modelization of the Sisyphus-I problem. We are currently studying the possibility of using heuristics to compute automatically a best order with respect to rules and constraints.

4.3 Experimental Results

The research engine found *2880* solutions to the Sisyphus-I problem. These solutions satisfy all constraints excepted a weak one which expresses that heads of large projects should be next to the secretariat and the head of group. Obviously, this constraint cannot be satisfied. Given our modeling, the number of solutions can be explained as follows:

- The head of group can only be in the office 117 or 119. *(2 solutions)*
- The manager can only be placed in a small central office, i.e. the office 116. *(1 solution)*
- The positions of the head group and manager determine the position of the secretariat (the office which has not been assigned to the head of group). *(1 solution)*
- There are only three small offices left, and they must be assigned to the three heads of large projects, which should be alone in their office. *(6 solutions)*
- The two smokers must be together and have the choice between the 4 big offices left. *(4 solutions)*
- There are 15 possible sets of couples for the 6 researchers left, of them 5 are impossible (they are in the same project). There are 6 possible assignments for these couples in the 3 last big offices. *(60 solutions)*

Finally, we have the $2 \times 1 \times 1 \times 6 \times 4 \times 60 = 2880$ different solutions exhibited by our research engine. Two of the computed solutions are presented in fig. 28. The first solution we find is the solution A in fig. 28. The beginning of the research tree used to find this solution is presented in fig. 29. We can see there one backtrack, as the first secretary is assigned a small office (no explicit constraint forbids it), and problems arise only when we try to assign the same small office to the second secretary. We explored 85 different worlds before generating the first valid solution (so we had 70 backtracks), and we must remind here that each of these worlds required to solve the problem of its validity, which is a co-NP-complete problem. However, the length of the computation to find the first solution for this particular problem (i.e. prove that the problem is overconstrained, remove a constraint and find an acceptable solution) was around 30 seconds.

Office	**Solution A**	**Solution B**
113	Hans W. (Head of Large Project)	Katharina N. (Head of Large Project)
114	Katharina N. (Head of Large Project)	Hans W. (Head of Large Project)
115	Joachim I. (Head of Large Project)	Joachim I. (Head of Large Project)
116	Eva I. (Manager)	Eva I. (Manager)
117	Thomas D. (Head of group)	Monika X. Ulrike U. (Secretaries)
119	Monika X. Ulrike U. (Secretaries)	Thomas D. (Head of group)
120	Andy L. Uwe T. (Researchers, smokers)	Angi W. Mark M. (Researchers, non smokers)
121	Michael T. Mark M. (Researchers, non smokers)	Harry C. Werner L. (Researchers, non smokers)
122	Jurgen L. Werner L. (Researchers, non smokers)	Jurgen L. Michael T. (Researchers, non smokers)
123	Angi W. Harry C. (Researchers, non smokers)	Andy L. Uwe T. (Researchers, smokers)

FIGURE 28: Two from the 2880 solutions to the Sisyphus-I problem

Thomas D. → 113 Violation: Not Central
Thomas D. → 114 Violation: Not Central
Thomas D. → 115 Violation: Not Central
Thomas D. → 116 Violation: Small Office
Thomas D.→ 117
Eva I. → 113 Violation: Not Central
Eva I. → 114 Violation: Not Central
Eva I. → 115 Violation: Not Central
Eva I. → 116
Monika X. → 113 Violation: Far from Manager
Monika X. → 114 Violation: Far from Manager
Monika X. → 115
Ulrike U. → 113 Violation: Far from Manager
Ulrike U. → 114 Violation: Not with other Secretary
Ulrike U. → 115 Violation: Two persons in a small office
Ulrike U. → 116 Violation: Two persons in a small office
Ulrike U. → 117 Violation: Head of group must be alone
Ulrike U. → 119 Violation: Not with other Secretary
[. . .] (120, 121, 122, 123) Violation: Not with other Secretary
Monika X. → 116 Violation: Two persons in a small office
Monika X. → 117 Violation: Head of group must be alone
Monika X. → 119
Ulrike U. → 113 Violation: Far from Manager
Ulrike U. → 114 Violation: Not with other Secretary
Ulrike U. → 115 Violation: Not with other Secretary
Ulrike U. → 116 Violation: Two persons in a small office
Ulrike U. → 117 Violation: Head of group must be alone
Ulrike U. → 119
Hans W. → . . .

FIGURE 29: A trace of the backtrack leading to the solution A

4.4 Coping with Changes in Data

A second problem statement is given in order to test how the model reacts with borderline cases. This second problem is obtained from the first one by a change in data: the head of project *Katharina N.* has left and a researcher *Christian I.* has joined. He belongs to the same project, is a smoker and a hacker.

In order to represent this change, we have to modify the data and the query in an obvious way, by removing every information about *Katharina N.* and replacing it by information about *Christian I.* in the facts (fig. 8 "team membership" and fig. 9 "personal information"), and updating the query (fig. 12) in the same way.

According to our modelization there are less constraints about *Christian I.* than about *Katharina N.* There are now 8640 different solutions to the problem. As in the first problem, the last weak constraint (members of the same project should not share offices) cannot be satisfied.

An interesting issue when the data change is to enable the user to take advantage of a part or all existing allocations instead of running the problem completely again. In our framework, there are several ways of expressing that some assignments have to remain unchanged: as facts, as constraints or even as transformation rules (but it seems to be the less appropriate way to do it).

If it is absolutely mandatory that some assignments remain the same, put them in the facts (for instance: "*Thomas D.*, the head of group, has the *office 117*". This is a fact and the allocation of *Thomas D.* is no more part of the task to solve).

In a more flexible way, we can express that some persons would rather stay at the same place. We do it by transforming each or some of the previous assignments into a negative constraint such as the one represented in fig. 30. This constraint expresses that *Jurgen L.* should not be in an office different from the one he has already been assigned to. Should we want to disturb no member of the YQT team, such a constraint would be defined for each member of the group except for the new one. Then, it is not surprising that the system proposes only one solution : *Christian I.* occupies *Katharina N.*'s room.

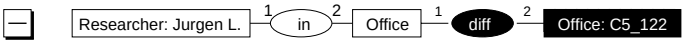

FIGURE 30: Transforming a previous assignment in a constraint

5 DISCUSSION

Conclusion. In this paper we described a graph-based formal model, which could be the basis of a generic modeling framework. We illustrated its use in modeling the Sisyphus-I case study. This model has the following features:

1. The knowledge and data structures are *labelled graphs*. Thus they are easily read and edited.
2. Reasonings are also based on *graph* operations. Thus graphs are not only used as a "graphical notation" for knowledge representation, but their structure offer an original viewpoint for designing efficient algorithms, as discussed in (Coulondre & Salvat, 1998). The key notion, that of a "projection", is a graph matching, simple to understand. It can be graphically represented, a feature which enables a user to follow reasonings. This aspect is significant for developing functionalities like explanations or model evaluation in KA.
3. It is totally *declarative*.
4. It is *generic* in the sense that it is not dedicated to any class of problem.

Applied to the Sisyphus-I case study, it notably allows to:

- describe the problem solving process with knowledge constructs that fit well with intuitive categories easily identifiable in the documentation. A clear distinction is made between the different kinds of knowledge involved in the modeling.
- describe all kinds of knowledge with a uniform language.
- describe the problem in a highly generic way. No domain knowledge has been needed. Mechanisms that solve the problem are general. No characteristic peculiar to Sisyphus-I nor to the class of resource allocation problems needs to be used.
- explicit the strategy of the expert. The knowledge engineer describes the task at an abstract level and does not have to write any line of code.
- easily modify the modeling, because everything is given in a declarative way. In addition, knowledge constructs may have any form (in particular they may comprise generic entities or not), which allows to solve variants of the original problem with exactly the same process, for instance: check whether a set of assignments is valid relative to constraints (for this problem assignments are encoded in facts and the query is empty) ; compute partial solutions (the query does not comprise every member of YQT) ; extend partial solutions (some assignments are encoded in facts) ; after a change in data find a solution that modifies as less as possible some or all current assignments (such requirements can be encoded with weak constraints as it has been done for the variant of the Sisyphus-I problem).

The approach that seems to be the closest to ours in the collection of papers devoted to Sisyphus-I (Sisyphus'94, 1994) is that of (Gaines, 1994) because of three important features: *declarativity*, *genericity* and the *visual* aspect. But one difference is that the visual language is used at an interface level only. Reasonings are performed by a KL-ONE like classifier. The user manipulates graphs but it would not be possible to exhibit reasonings made by the system in the same formalism (whereas the graph-based model naturally contains the possibility to *visualize reasonings*). Of course the system presented in (Gaines, 1994) is more completed. In particular it allows the user to intervene into the solving process, which is not possible with our prototype. Again we used the Sisyphus-I case study to test the graph-based model potentialities and we did not aim at building a directly usable KA tool.

Potentialities for Knowledge Engineering. In our opinion the graph-based model presents two benefits for knowledge engineering.

First, it is both a *formal* model (thus authorizing non ambiguous knowledge modeling and automatic treatment) and a *visual* language easily readable by an expert. As noticed in (Bos *et al.*, 1997), "Experts appreciate graphic notations. The very simple graphic concepts of Conceptual Graphs (boxes, ovals and arrows) and the use of labels chosen in the expert language make modeled knowledge understandable by the expert with no effort, whereas they were afraid by textual formal languages". In a knowledge engineering process, several representations are used at different stages of the knowledge based system construction (Gaines & Shaw, 1995) ; first, the elicitation phase develops a system specification expressed in a humanly understandable

mediating representation ; next this informal representation is encoded into a formal representation, which then leads to the operational knowledge-based system. (Gaines & Shaw, 1995) point out that one major problem in this process is to ensure the *fidelity* of the transformation from the mediating representation to the formal model. One of the solutions they propose is to create mediating representations that are formal (see also the discussion in (Lukose *et al.*, 1995)). The graph-based model presented here, as others in the conceptual graphs family, is a candidate to be the base of formal mediating representations.

An original feature of our graph-based model is that reasonings can be *graphically* represented and understood by experts. This property can be a precious help to verify the validity of an expertise modeling. Indeed it is very difficult to ensure that the expert reasoning is correctly modeled (see for instance (Aussenac-Gilles & Matta, 1994)). One solution would be to give the expert the possibility to run the system and to follow the reasonings step by step in order to check that what is done is correct from his/her viewpoint. This way of doing has been adopted by (Bos *et al.*, 1997) in a project, in which we partake, that aims at providing tools for modeling and simulating human organizations ; the key idea is to use simulation inside the design cycle as a mean of enriching and building the modeling and not only as a tool for final validation.

Further investigations and developments. One feature of conceptual graphs not really investigated here is that they allow to deal with different levels of knowledge. We modeled the expert strategy by way of declarative structures (constraints clustered in several categories and transformation rules) given in a total order that is explicitly part of the strategy. This ordering does not influence the existence of a solution nor the set of solutions, but it acts upon the way the solution space is explored. One could also imagine less crude ways of ordering declarative structures. Thus a more satisfying way of dealing with this ordering would be to reify it by representing it as a *meta-knowledge* in the graph-based formalism. More generally meta-knowledge should allow the description of ordered tasks. The major difficulty here is not to describe the succession of tasks. Several works have proposed modeling languages based on nested graphs (an extension of simple graphs, where a concept node may have a description which is a nested graph) that are suited to process descriptions. The difficulty is to define operations for managing this knowledge. As far as we know all works concerning the description and the execution of processes using conceptual graphs solve the execution problem by procedural extensions attached to specific structures or with an extended language including procedural capabilities (among others (Lukose, 1995)(Bos *et al.*, 1997)). It is an open question to know whether it can be solved in a generic and declarative way.

As mentioned in the introduction of this paper one important feature of conceptual graphs is that they possess *sound* and *complete logical semantics*. However another feature is that it is easy to extend the model with graph operations and to use it without having a logical interpretation, at least at the beginning. Simple graphs and inference rules are provided with sound and complete logical semantics, but finding a logical interpretation of transformation rules and constraints as well as the global constrained derivation is an open problem.

The *prototype* is not yet a real KA tool. Further developments are needed in order to enable communication with the user in a friendly way, give the user the possibility to intervene at several stages of the solving process, and improve computational complexity.

References

Aussenac-Gilles, N. & Matta, N. (1994). Making a method of problem solving explicit with MACAO. *International Journal of Human-Computer Studies, Special Issue: Sisyphus: Models of Problem Solving*, 40(2):193–219.

Baget, J.-F., Genest, D., & Mugnier, M.-L. (1999). A pure graph-based solution to the SCG-1 initiative. In *Conceptual Structures: Standards and Practices, proceedings of the seventh International Conference on Conceptual Structures (ICCS'99), Blacksburg, USA*, volume 1640 of *LNAI*, pages 355–376. Springer.

Bos, C., Botella, B., & Vanheeghe, P. (1997). Modelling and simulating human behaviours with conceptual graphs. In *Conceptual Structures: Fulfilling Peirce's Dream, proceedings of the fifth International Conference on Conceptual Structures (ICCS '97), Seattle, USA*, volume 1257 of *LNAI*, pages 275–289. Springer.

Chein, M. (1997). The CORALI project: From conceptual graphs to conceptual graphs via labelled graphs. In *Conceptual Structures: Fulfilling Peirce's Dream, proceedings of the fifth International Conference on Conceptual Structures (ICCS '97), Seattle, USA*, volume 1257 of *LNAI*, pages 65–79. Springer.

Chein, M. & Mugnier, M.-L. (1992). Conceptual graphs: Fundamental notions. *Revue d'intelligence artificielle*, 6(4):365–406.

Chein, M., Mugnier, M.-L., & Simonet, G. (1998). Nested graphs: A graph-based knowledge representation model with FOL semantics. In *Proceedings of the 6th International Conference "Principles of Knowledge Representation and Reasoning" (KR'98), Trento, Italy*, pages 524–534. Morgan Kaufmann Publishers.

Coulondre, S. & Salvat, E. (1998). Piece resolution: Towards larger perspectives. In *Conceptual Structures: Theory, Tools and Applications, proceedings of the sixth International Conference on Conceptual Structures (ICCS '98), Montpellier, France*, volume 1453 of *LNAI*, pages 179–193. Springer.

Gaines, B. R. (1994). A situated classification solution of a resource allocation task represented in a visual language. *International Journal of Human-Computer Studies, Special Issue: Sisyphus: Models of Problem Solving*, 40(2):243–271.

Gaines, B. R. & Shaw, M. L. (1995). Knowledge and requirements engineering. In *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems, Banff, Canada*.

Genest, D. & Chein, M. (1997). An experiment in document retrieval using conceptual graphs. In *Conceptual Structures: Fulfilling Peirce's Dream, proceedings of the fifth International Conference on Conceptual Structures (ICCS '97), Seattle, USA*, volume 1257 of *LNAI*, pages 489–504. Springer.

Genest, D. & Salvat, E. (1998). A platform allowing typed nested graphs: How CoGITo became CoGITaNT. In *Conceptual Structures: Theory, Tools and Applications, proceedings of the sixth International Conference on Conceptual Structures (ICCS '98), Montpellier, France*, volume 1453 of *LNAI*, pages 154–161. Springer.

Haemmerlé, O. (1995). *CoGITo : une plateforme de développement de logiciels sur les graphes conceptuels*. PhD thesis, Université Montpellier II, France.

Kerdiles, G. & Salvat, E. (1997). A sound and complete CG proof procedure combining projection with analytic tableaux. In *Conceptual Structures: Fulfilling Peirce's Dream, proceedings of the fifth International Conference on Conceptual Structures (ICCS '97), Seattle, USA*, volume 1257 of *LNAI*, pages 371–385. Springer.

Lukose, D. (1995). Using executable conceptual structures for modelling expertise. In *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems, Banff, Canada*.

Lukose, D., Mineau, G., Mugnier, M.-L., Moller, J.-U., Martin, P., Kremer, R., & Zarri, G. P. (1995). Conceptual structures for knowledge engineering and knowledge modelling. In *Supplementary Proceedings of ICCS'95, Santa Cruz, USA*.

Salvat, E. & Mugnier, M.-L. (1996). Sound and complete forward and backward chaining of graph rules. In *Conceptual Structures: Knowledge Representation as Interlingua, proceedings of the fourth International Conference on Conceptual Structures (ICCS '96), Sydney, Australia*, volume 1115 of *LNAI*, pages 248–262. Springer.

Schubert, L. (1991). Semantic Networks are in the Eye of the Beholder. In Sowa, J. F., (Ed.), *Principles of Semantic Networks*, pages 95–108. Morgan Kaufmann.

Sisyphus'94 (1994). *International Journal of Human-Computer Studies, Special Issue: Sisyphus: Models of Problem Solving*, volume 40(2). Academic Press.

Sowa, J. F. (1976). Conceptual graphs for a database interface. *IBM J. Research and development*, 20(4).

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

Wermelinger, M. (1995). Conceptual graphs and first-order logic. In *Conceptual Structures: Applications, Implementation and Theory, proceedings of the third International Conference on Conceptual Structures (ICCS'95), Santa Cruz, USA*, volume 964 of *LNAI*, pages 323–337. Springer-Verlag.