A Cost-Bounded Algorithm to Control Events Generalization

Gaël De Chalendar, Brigitte Grau, Olivier Ferret

A Cost-bounded Algorithm to Control Events Generalization

Gaël de Chalendar, Brigitte Grau, Olivier Ferret

LIMSI-CNRS
BP 133, 91403 Orsay, France
email: (gael, grau, ferret)@limsi.fr

Abstract. In order to build descriptions of prototypical situations, we first developed a system, MLK (Memorization for Learning Knowledge), allowing us to gather events related to similar situations starting from descriptions found in texts, these events being represented by conceptual graphs. One of the stages to build these prototypes consists of generalizing some similar graphs in order to produce a description. In this paper, we present a cost bounded algorithm of conceptual graph generalization, proceeding by ascending clustering. The use of costs on the operations of generalization allows us to control the growth of the search space.

1. Introduction

Text understanding requires knowledge about concrete situations that are developed in order to recognize links between the different events. Such knowledge is often represented by schemas whose content describes characters and events involved in a situation. However handcoding schemas is a very difficult task, even on a limited domain. Automatic acquisition of situation descriptions has been [1], [2] and [3], but essentially to learn new specialized situations of predefined ones. Inside a framework that does not make any hypothesis on the existence of general knowledge, we have first conceived the system MLK (Memorization for Learning Knowledge) [4], [5], able to learn from the accumulation of its own experience. MLK memorizes each specific situation found in texts by aggregating it with an already memorized description if a similarity between their events is recognized. This process leads to incrementally build aggregated Thematic Units (TU) that are precursors of general schemas. Each event inside a TU is represented by a weighted conceptual graph (CG) [6]. In order to build a general description of a situation from a TU, we have studied how to generalize these events, i.e. conceptual graphs, to find a description level accounting for different formulations of a same event. This kind of problem is dependant on ascending clustering methods from positive examples. Generalization algorithms of conceptual graphs have already been proposed [7], [8], but they only generalize concepts, and not the graph itself. Generalizing both concepts and graphs entail a combinatory explosion when searching for all the possibilities. Therefore, we have developed a cost-bounded generalization algorithm that limits the search space and leads us to propose informative generalizations. A cost is associated with each generalization operator, and we take advantage of semantic knowledge to produce meaningful descriptions of events.

2. System Overview

The system input is made of a set of conceptual graphs describing events related to a situation of the same kind, for example *to come in a house, to attack a woman, to stab a man with a knife, to arrest the murderer*, etc. In order to elaborate a description of the situation from these events, a main stage is to find a right level of description leading to generalize those specific graphs describing a same kind of event. In the preceding example, it would consist of proposing a description where *the woman attack* and *the man stabbing* are generalized, whereas the two other events are left as it. Then the problem can be formulated as follows: given a set of conceptual graphs, find among them those that can be generalized while keeping an informative description level. Our purpose is not to find the common generalization of all the events, but a description level keeping the specificity of the kinds of events represented by the graphs. Coming back to the example, our goal is not to generalize the four events in a single one that would be *to carry out an action.*

The method we propose consists of developing a sub-set of the possible generalizations of each graph, by generalizing concepts and removing relations. Generalization is controlled by the use of semantic knowledge: a lattice of concept types and constraints on the arguments related to some concepts, given by canonical graphs. It avoids overgeneralizations that would not have any meaning. Developing all the generalizations of a graph is unconceivable, even if limiting them by comparison with domain knowledge. Therefore, in order to limit the size of the generalization space, a cost is associated to the generalization operators, allowing the system to associate a cost to each graph produced and to limit their formation by fixing a threshold. Costs are defined according to the task and encode the generalization level that is searched. All the generalizations are organized in configurations allowing the system to know which initial graphs are generalized. These configurations represent the different possibilities to describe the initial situation. The choice of one of them is done according to the number of initial graphs that are generalized and to the cost of the different configurations. Thus, the description level that is built is constrained both by the semantic knowledge and by the costs associated to the generalized graphs. This method allows us to bypass the difficulty coming from the impossibility to define negative examples in our task and then to use classical learning algorithms.

3. Conceptual Graphs

Semantic knowledge is represented in a lattice of types of concept and by canonical graphs associated to some types in order to precise their thematic roles (i.e. relations), as *agent*, *object*, etc., and semantic constraints on concepts that might fill them (see Fig. 1). We do not follow the Conceptual Graphs standard draft NCITS.T2/98-003. The "syntactic" knowledge held by star graphs is coded by our canonical graphs. Conceptual graphs that represent events (see Fig. 2) are obtained by application of formation rules to one or several canonical graphs, as described in Sowa [6]. So, these graphs are built with respect to the constraints defined in the knowledge base. A graph resulting from the application of these rules is a specialization of one (or several) initial graph(s). In Fig. 2, the graph results from a maximal joint between two graphs

produced after restrictions on types of concept in the canonical graphs associated to *stab* and *body*. See [9] for a detailed presentation of the formation rules and their relation with subsumption. The class of CGs we consider is the class of simple CGs derived from canonical graphs, with n-ary relations and with a distinguished concept node explained below.

[Stab]-
(agent) → [Human]
(dest.) → [Body]
(instrument) → [CuttingEdge]

Fig. 1. A canonical graph

[Stab]-
(agent) → [Man]
(dest.) → [Body] -
(partOf) → [Man]
(instrument) → [Knife]

Fig. 2. An event

As our graphs represent events, we distinguish a concept that is significant for the event type, named predicate (*Stab* in Fig. 2), this concept playing a particular role inside our application. Other kinds of concept result from abstractions done during the aggregation process in MLK. They already generalize instances linked to a same predicate that are found in texts. See [4] and [5] to find details about the process that build these graphs.

4. Graphs Generalization

4.1 Method

Some graphs belonging to the initial set may contain different predicates while being very similar, as in Fig. 3. If graphs are identical except for their predicate, and if these predicates are semantically close, these graphs have to be replaced by a graph that generalize them. The same principle applies if the graphs only differ by some details in the predicate arguments. The problem is then to find the least common generalization of several graphs, given one does not *a priori* know which graphs in the initial set have to be generalized.

Fig. 3. Similar conceptual graphs with different structures and concepts

Such a problem is similar to conceptual clustering, with conceptual graphs as the description language of the concepts to be learned [10]. Initial conceptual graphs are equivalent to first order logic formulas, made of a conjunction of positive predicates, the concepts and the relations of the graphs. This problem is close to the work of Mineau and Bournaud. Even if we do not search for a classification, which is the problem they have to solve, the generalization space has also to be generated in order to choose the generalization level we want to reach. In their work, a graph is represented by the set of its relations to avoid comparison of graphs, that is a NP-complete problem. However, this approach cannot be applied in our application given that the structure of the graphs may be modified during the generalization process when removing relations. We have to reason on graphs themselves in order to keep

specific links between a predicate and its arguments, i.e. to maintain their connectedness. It is then impossible to generate meaningful generalizations in a limited time without controlling the growth of the generalization space. The process we propose is a cost-bounded algorithm that associates a cost to each generalization operator and uses domain knowledge to avoid overgeneralizations. This algorithm has been conceived without taking into account the specificity of our application other than the presence of a central concept in conceptual graphs. So we consider that types of relations are organized in a lattice, even if our knowledge representation does not use this possibility.

We have defined three primitive generalization operators:

1. concept generalization: replacing a type of concept by its supertype in the lattice,
2. relation generalization: replacing a type of relation by its supertype in the lattice,
3. relation removing: removing a relation in the graph, and some associated concepts.

We have not defined an operator for removing concepts because removing a concept entails the removal of all the relations it is linked to. If we come back to the example in Fig. 3, removing the concept *Shop* leads to remove the relation *Loc*, and conversely removing the relation *Loc* leads to remove the concept *Shop* in order to keep a connected graph (in such a case, we keep the connected component that contains the predicate). Removing relations is then sufficient since removing concepts is strictly included in the results of the removal of relations, which may entail the removal of several concepts and relations.

These operators create a partial order between the resulting graphs, as defined in [6] and [9]. Precisely, the subsumption relation defined for conceptual graphs is the relation induced by the existence of a projection between two graphs [11].

We associate a cost to each operator since we dispose neither a domain theory (general knowledge about situations for example), nor a characterization of the pieces of knowledge we want to learn that would give us a formal proof of the quality of the generalization. We use an empirical evaluation based on semantics and the estimation of the loss of information when generalizing.

In order to find this cost, we have reasoned on the effect of each operation in terms of loss of information it entails. In our context, the cost of each operator is related to the concept it applies to: predicate or not, concept derived from the canonical graph of the predicate or not. When generalizing a concept that is not derived from the canonical graph of the predicate, named common concept, the generalization is made on characteristics that are peripheral to the event. Removing relations, that, as already said, causes the removal of concepts, entails the suppression of some of these characteristics. As a matter of fact other concepts cannot be removed, as one cannot remove relations belonging to the canonical graph of the predicate without losing the meaning of the graph (the canonicity of the resulting graph is no more verified). Lastly, the operation we consider as very costly is the generalization of the predicate, since, as we have seen in part 2, we want to avoid many generalizations of the predicates in order to keep kinds of events specific to the described situation. So the least common generalization is a graph achieved by preferentially generalizing concepts other than the predicates.

We have defined the following order of the costs of the operators, this ordering characterizing the loss of information:

generalization of a common concept ≤ generalization of a concept derived from the canonical graph ≤ removal of a relation ≤ generalization of the predicate

The relative order is more significant than absolute values that defines costs. The value given to each operator is connected to the threshold that is fixed to compute generalizations for a given application, the number of operations it is likely to realize, and the importance of each operation in relation to the others. These values have to be fixed experimentally according to the application.

Let us now present some specificity due to the conceptual graph formalism and our application. Firstly, it is not possible to suppress the predicate, since the resulting graph(s) would no longer refer to the described event. Secondly, if two non-connected graphs result from a suppression, we only keep the graph containing the predicate. Thirdly, a concept is never generalized in a concept more general than the one in the canonical graph. And last, when generalizing a predicate, a verification is done using the new canonical graph to ensure the canonicity of the resulting graph. A consequence of having this distinguished concept is that the complexity of the operations on CGs is reduced due to less possible matching between two graphs. But that does not reduce a lot the size of the effectively searched space.

4.2 The Generalization Algorithm

One stage of the algorithm consists in building the generalization space which contains, at the beginning, the conceptual graphs to be generalized, i. e. the root nodes. During the processing, each node resulting from a generalization is inserted in this space only if it does not already exist. A node of depth *n* is the result of *n* generalizations of a root node. For example, a conceptual graph that contains 3 concepts and 2 relations may be processed by applying 5 generalizations of concept or relation and 2 removal of relation. It would generate, at most, 7 new nodes. The cost associated to each node, related to the path towards the root node, is equal to the cost of its father plus the cost of the operation which has given it birth. The cost of a root node is zero. Applying such an algorithm until a common generalization is found, if it exists, is exponential in the general case. Bounding the set of the possible generalizations by fixing a maximal cost on the nodes entails a quite reasonable practical complexity (this latter point will be detailed in the following section) and all the generalizations having a cost less than this threshold are computed.

Fig. 4 shows an example where three conceptual graphs (the root nodes x1, x2 and x3) have been generalized in the graphs (a, b, c, d), (c, b, e) and (e, f, g, d) respectively. Values on the branches are the costs of the applied operators.

Costs which stop the generalization process, more than maintaining the semantics of the graphs, yield the possibility to control the growth of the generalization space. In order to improve more effectively the performance of the algorithm, we have implemented some principles. For example, if the cost to find a supertype common to the predicate of a graph and the predicates of each other graph of the initial set oversteps the given threshold, this graph is removed from the initial set. By using this principle, the algorithm applies on graphs that may be a generalization of two root graphs, and not on the very initial graphs, given they all contain a different predicate (see *Initialization* part of the algorithm in Fig. 6). On another hand, graphs are

indexed by their predicate and their number of relations and concepts, such as comparison of graphs is only done if these characteristics are identical.

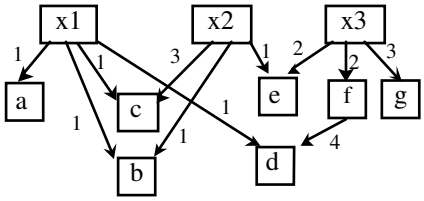

Fig. 4. A generalization space built from 3 root graphs

The aim of the algorithm is to generate all the possible descriptions of the initial set, the set of configurations. A configuration is a set of conceptual graphs, generalized or not, that represents a partition of the initial examples, the root graphs. After application of the algorithm, only the graphs that generalize several root graphs with the root graphs themselves are kept. Other nodes, that just generalize a unique initial conceptual graph, are useless for our purpose (for the example in Fig. 4, retained nodes appear in the first column in Fig. 5). The generalization cost of a node is the average cost of the generalization costs of this node, these latter costs correspond to the different path issued from the different root nodes. Finding the set of configurations consists of building all the subsets of nodes such as each initial node is generalized once and only once in the subset. Each configuration is evaluated by the average cost of its elements.

Nodes	Average costs	Origins	
c	(1+3)/2 = 2	x1	x2
b	(1+1)/2 = 1	x1	x2
d	(1+6)/2 = 3.5	x1	x3
e	(1+2)/2 = 1.5	x2	x3
x1	0	x1	
x2	0	x2	
x3	0	x3	

Configurations = {[c, x3] (2), [b, x3] (1), [d, x2] (3.5), [e, x1] (1.5), [x1, x2, x3] (0)}

Fig. 5. Results of the generalization algorithm

Configurations are computed during the generalization process (cf. *Updating configurations* in the algorithm). At each step, partial configurations, corresponding to the retained generalizations, are updated if the new graph generalizes at least two root graphs.

If more than one configuration has been built at the end of the processing, we choose the one that verifies the following criteria, applied in this order: a) the configuration that corresponds to the generalization of the maximum number of root graphs; b) the configuration containing less numerous elements; and c) the configuration having the minimal cost, other than zero, considering it is the most likely to be the most common specific description of the initial events. By means of these criteria, we encode that the best description of the initial situation corresponds

to the maximal regrouping of graphs, whose cost is less than the limit. In the example of Fig. 4, these criteria lead to choose the configuration [b, x3] having an average cost equal to 1. These criteria fit our application, even if, for other kinds of applications, another order or other criteria may have to be found.

E, set of initial graphs, E = {gi} and *n* = |E|.
Cgi, sets of generalized graphs from each root graph *gi*.
G, generalization space, the set of all computed generalized graphs. Each generalized graph is related to a list of root graphs it generalizes.
Cp, set of partial configurations, initialized to an empty set
cgp, cost to generalize a predicate
la, threshold that stops generalization

Notation: a -> b : a associated to b

Initialization
For all *g* ∈ *E*, and **For all** *g'* ∈ *E* **do**
  Create a set *Cg*
  If the predicate of *g* has a common supertype, *sc,* with length *d* of the path between the predicate of *g'* and *sc*, such as *d* x *cgp* < *la*
  then,
    create a copy g_C of *g*,
    replace the predicate of g_C by *sc*
    add g_C ot *Cg*
    add *g* to *G*
  endIf
endFor

Abstraction
While it exists a node to develop **do**
  For all *Cgi* ∈ *Cg*, **do**
    Selection of a node *nd* from *Cgi*
    Updating configurations with *nd* ***comingFrom*** *Cgi*
    generalize *nd* -> *son(nd)*
    add *son(nd)* to *Cgi*
  endFor
endWhile

Compute the final result

Selection
Depends on the way the generalization space is searched: in our case, the first found.
Suppress the chosen graph from *Cgi*.

Updating configurations with *nd* ***comingFrom*** *Cgi*
/ updating of the generalization space/
If *nd* ∉ *G*,
then add *nd* -> {root graph of *nd*} in *G*
else
  If *nd* had already been generated from the same root graph
  then
    If the cost of *nd* is lesser,
    then replace the old graph by *nd*
    endIf
  else
/ updating of the partial configurations /
  add, in *G,* the new root graph of *nd* to the list of its root graphs.
  remove all the configurations containing *nd*
  For all *cp* = {a, b, c ...}->{x1, x2, x3, ...}∈ *Cp* **do**
    If the list of root graphs of *cp* does not contain any *x'i*, root graphs of *nd*
    then
    create a copy of *cp*, add *nd* to the left member and the *x'i* to the right member:
    new configuration = {a, b, c,..., nd}->{x1, x2, x3, ..., x'1,x'2, x'3}
    add the new configuration to *Cp*
    endIf
  endFor
  create a new configuration with *nd*
  endIf
endIf

Fig. 6. The generalization algorithm

The algorithm we propose is general enough to be used in other applications. We will see in the latter sections that heuristics coming from the application domain and expressed with simple numeric values, entail the considerable pruning of the search space, allowing the algorithm to find only the more informative generalizations. As for all algorithms depending on heuristics, it is not easy to formally demonstrate the complexity of our algorithm, thus we have worked on a demonstration of the adequacy and the efficiency of this method by extensive tests.

5. Results

The process has been applied with success to the results of MLK, but this does not entail a validation on a quantity of data sufficiently large, because text representations that compose the input of MLK are handcoded at this moment. That is why, in order to test our algorithm in a better way, we conceived a test generator. Its knowledge (lattice of types and canonical graphs) is compatible with our application and the generator is parameterizable so that it can simulate different kinds of application.

5.1 Domain Knowledge

Our application exploits a lattice of types of concepts and a set of canonical graphs. The lattice (see Fig. 8), results from the work of Chibout [12] who created an ontology of approximately 3,000 concepts for verbs and entities. We extracted a sub-part from this ontology so that to obtain a lattice regular enough in depth and in breadth. Thus, 368 concepts were extracted and distributed in a lattice with an average depth of 5.59 for a branching factor of 2.75. We have selected 8 verbs from the 128 verbs retained to assign canonical graphs to them. These canonical graphs are defined relative to the types retained in the lattice and to produce homogeneous bench tests, so that global results will have a signification.

Fig. 7. Canonical graph of the concept denoted by the verb "*to drink*"

The 8 types are *ToCauseDeath*, *ToCauseToFeel*, *ToDivide*, *ToBreak*, *ToFly*, *ToAbsorb*, *ToEat* and *ToDrink*. Subtypes inherit of the canonical graph of their father. Figure 7 shows the canonical graph of *ToDrink*. The *Function* concept is the supertype of 55 concepts. Some of them are *Artisan*, *Killer* or *Deputy*. *Liquid* is the supertype of, among others, *Water*, *Alcohol* and *Syrup*

The lattice of relation types is limited in our application to a flat lattice: all the relations have the same supertype (T) and the same subtype (⊥). For our tests, we have used four relations: *Relation*, *Agent*, *Object* and *Place*.

5.2 Bench Test Generation

We have generated sets of graphs with controllable properties for the generalization. By specializing canonical graphs in directions randomly chosen and in different depths, our task is then to retrieve the original common graphs, or, better, some specialization of them that might not have appeared during the generation phase. So, our generator builds sets of graphs such as each set possesses a partition of its graphs where each part (named family) is made of graphs that are specialized from a graph more specific than their common canonical graph. This definition of the searched generalizations comes from our application for which it is not suitable to generalize events by their canonical graph: it does not fit an informative description.

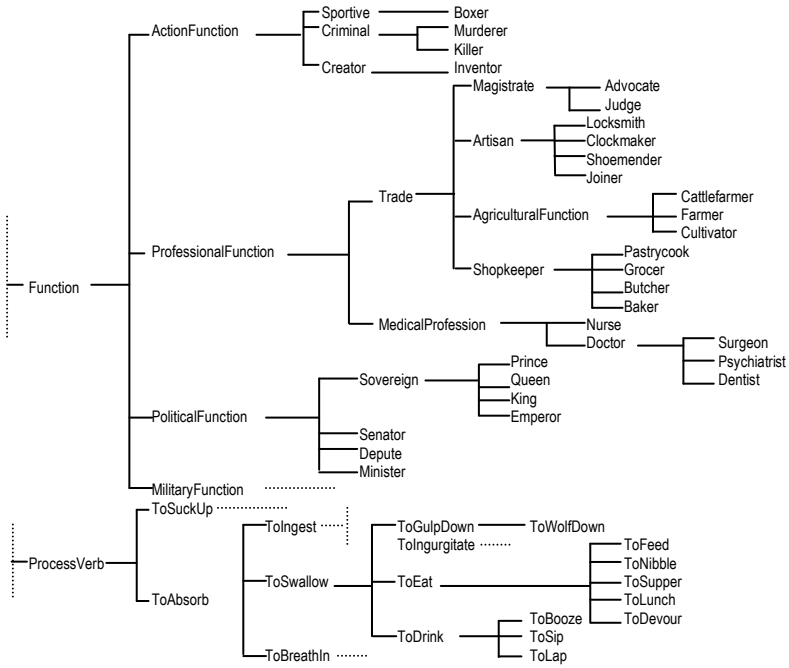

Fig. 8. Extracts from the concept ontology

To produce a bench test, the generator randomly chooses *n* predicates (P1 and P2 in Fig. 9), each one being the origin of one family. The canonical graph of each predicate is then specialized several times (s1 times in family 1 and s2 times in family 2) to generate *n* graphs, named ancestor graphs. Each ancestor graph is differently specialized (k1, k2, …, k5 times in family 1 and k1, k2 and k3 times in family 2), creating different specializations named cousin graphs.

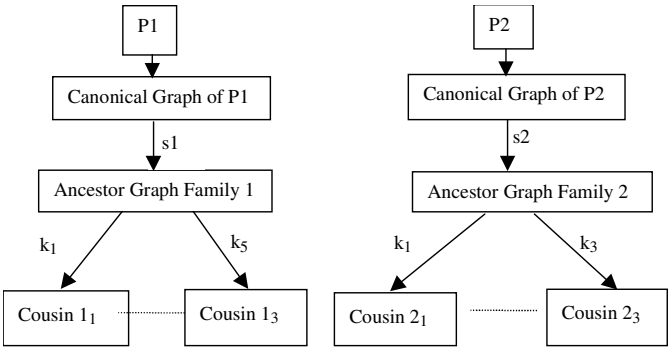

Fig. 9. Bench test generation framework

Input data of the generalization process are the set of all the cousin graphs of different families (the 8 cousin graphs), the goal being to retrieve the set made of the ancestor graphs. The numbers of specialization operations applied (restriction of type

belonging to the canonical graph or not, addition of a relation) are set at the time of the bench test specification.

We can see below an example of test containing four graphs with two families of two graphs each. The graphs have been obtained following the parameters presented in Fig. 10.

Graph 1:

```
[ToNibble]{
– (Agent) -> [Nurse],
– (Object) -> [Cake],
– (Rel) -> [Apple]}
```

Graph 2:

```
[ToDevour]{
– (Agent) -> [Nurse],
– (Object) -> [Meat],
– (Rel) -> [Rice]}
```

Graph 3:

```
[ToKill]{
– (Object) -> [Reptile],
– (Agent) -> [Assassin],
– (Rel) -> [Car]}
```

Graph 4:

```
[ToEmbed]{
– (Agent) -> [Killer],
– (Objet) -> [Insect],
– (Rel) -> [Water]
```

Predicates	**Canonical Graphs**	**Specializations for ancestors**		**Specializations for cousins**		
		predicate	***canonical graph concepts***	***predicate***	***canonical graph concepts***	***concept adding***
ToEat	**[ToEat]** -> (Agent) ->[Function] (Object) -> [Food]	0	3	1	1	1
				1	1	1
ToCause Death	**[ToCauseDeath]**-> (Agent) -> [Function] (Object)-> [Alive]	0	3	1	2	1
				1	2	1

Fig. 10. Parameters for generating a set of graphs

This example entails the construction of 6 configurations in a total time of 115s. We can see below the optimal solution obtained in 15 seconds and a non-optimal solution. Costs of operations are specified in section 5.4.

Optimal Solution:

```
[ToEat]{
– (Agent) -> [Nurse],
– (Object) -> [Food]}
Cost: 8.0
[ToCauseDeath]{
– (Agent) -> [Criminal],
– (Objet) -> [Animal]}
Cost: 10.0
```

Non-optimal Solution:

```
[ToEat]{
– (Agent) -> [MedicalProfession],
– (Object) -> [Aliment]}
Cost: 10.0
[ToCauseDeath]{
– (Agent) -> [Criminal],
– (Objet) -> [Animal]}
Cost: 10.0
```

If we look again at the definition of a good solution given section 4.2, the optimal solution in this example respects the established criteria: maximum of root graphs generalized, minimum number of graphs in the configuration and minimal cost in the case of configuration equality. In the non-optimal configuration, the concept *Nurse* in the two graphs of the first family has been overgeneralized in *MedicalProfession*. So, we really have obtained the best solution for a cost inferior to those of the other solution. The four other solutions, not presented here, have inferior costs but do not generalize all the graphs.

We will now describe two bench tests. The first one tests a difficult case with very specialized graphs and little generalization costs. It entails the generation of a great number of configurations and high computation times. The second one shows the efficiency of the system with higher costs.

5.3 A Borderline Case

We produced 33 sets of 4 graphs containing 3 to 5 concepts and belonging to 2 families. These graphs were produced by applying 3 to 8 specialization operations on canonical graphs. The threshold for the generalization was set to 10 and a limit of processing time was fixed to 1 hour by case. For this data set, we set low values to costs of operations in order to test the borderline case: the costs allow the system to compute all the ancestor graphs. These costs are the following:

1. any concept generalization: 1,
2. generalization of one concept of the canonical graph: 1,
3. predicate generalization: 2,
4. relation suppression: 1.

The generalization process finds 32.15 configurations on average by bench test, in an average time of 28mn, 30% of the generalization processes reaching the time limit. The 70% solved cases are solved in an average time of 15mn.

5.4 Costs Utilization

To show the efficiency of mindfully chosen costs, we carried out the same kind of tests, but with costs multiplied by 2. We generated 60 bench test with 4 graphs each. With a time limit set to 1 hour, we obtained 12,2 configurations on average by case. With 20mn., we obtained 11.9. So, 97.5% of the configurations obtainable in 1 hour were obtained in less than 20mn. The following results were obtained in this time limit: 96.7% of the bench tests lead to at least one solution and 73,3% of the cases did not reach the time limit.

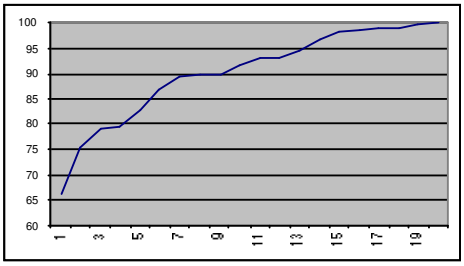

Fig. 11. Percentage of computed configurations relatively to the processing time

Figure 11 shows the percentage of configurations obtained for all bench tests in function of the time. It shows that in a reasonable time of 7mn., 90% of the solutions obtainable in 20mn (92.3% of those obtainable in 1 hour) are already obtained. Thus, according to task requirements, if it is possible to miss some solutions, then one can parameterize the generalization system with processing times really acceptable.

For these tests, the relevant solutions (i.e. informative) are thus those that contain generalized graph(s) equal to the ancestor graphs from which the graphs of the tests are issued. This relevant solution can only be found if, for each graph, the sum of the costs of the specialization operations applied to them is lower or equal to the generalization limit, here 10. The number of bench tests in this case is 22 (37%). With a processing time set to one hour, the relevant solution is found for all these cases and

in 20mn., 20 solutions are found (90.9%). In the other 63%, for which the theoretical cost of the optimum solution is higher than the limit, we find nevertheless 15.8% of configurations in which the number of generalized graphs is the same as the number of families. This can be explained by the fact that cousin graphs are randomly generated. If by chance the specialization process goes the same way for some graphs, then the effective relevant solution is more specific than the ancestor graph.

5.5 Analysis

It has to be noted that in the first series of tests, the obtainable relevant solutions are obtained in 70% of the cases (23/33). In the second series, 100% of the obtainable solutions are found in one hour, however it contains a least number or test cases. So, with low costs, a great number of solutions is found and we also miss some good solutions. On the contrary, with high costs, the number of possible solutions is lower and the system finds all of them. When we will apply the system on real data, we will have to choose costs according to these two extremes.

These tests show that a cost-bounded algorithm allows us to generalize conceptual graphs in a reasonable time, in tasks where costs on the generalization operators are definable. Differences between the two series of tests show the benefits of modulating the costs and the limit threshold in order to produce interesting generalizations and to avoid the production of graphs corresponding to overgeneralizations in the application, i.e. valid graphs with a too general description level.

It is not an easy task to show the validity of solutions obtained with unsupervised learning. Thus, we have conceived our test protocol in such a way that the searched solution can be characterized. In our application framework, the validation could be done externally by an expert who would answer the following question: "Does the produced solution seem coherent to you: are events described at a correct description level compared to the described situation?". In case of non-satisfaction, it is possible to execute the process again to search for new generalizations. Another possibility would be to validate a system using the learned knowledge.

6. Previous Works

Mineau [7] and Bournaud [8] have conceived algorithms to build a classification from an initial set of conceptual graphs. Even if we do not want to classify data, finding a generalization level can be seen as an equivalent problem as it requires the algorithm to compute possible generalizations. The major difference comes from the generalization operators used. Suppression of relations entails the necessity of a complete description of each generalized graph and not only a partial description by their relations. To limit the effective complexity, we have introduced costs associated with operators. These costs have to be fixed relative to the type of generalizations expected in the application. As in [8], we also use domain knowledge to constrain generalization.

Our work comes under Inductive Logic Programming (ILP) domain [13] and our problem is particularly close to the work of Esra Erdem and Pierre Flener [14], who redefine the minimal generalization in order to find a minimal set of generalized

clauses in function of an over-generalization criterion. In the field of ILP and conceptual graphs, our kind of problem is studied in the work of Marc Champesne on the reduction of the search space by the use of the notion of "empirical subsumption" [11]. However his results can only be applied to non-connected graphs whereas our algorithm handles connected ones.

About our application itself, i.e. learning descriptions of prototypical situations, our work is different of [1], [2] and [3] essentially because we do not hypothesize the existence of previous knowledge about situations and learning is completely unsupervised. The system MLK, which implements this application, gathers and selects events and their relevant characteristics when recurrent situations are found in texts, and the system proposed in this paper assumes the construction of a general description with respect to semantic knowledge.

7. Conclusion

Learning structures in order to describe concrete situations has lead us to construct the system MLK, able to identify events linked to a same situation. In the purpose of generating general descriptions for these situations, we have studied the generalization of events represented by conceptual graphs. However the generalization algorithm we propose is independent enough of our particular context and could be used with other applications.

This algorithm is completely implemented in Smalltalk. The tests made show the benefit of fixing costs on generalization operators in applications having only a weak domain theory.

The protocol of bench tests generation we have developed allows us to control different parameters such as the definition of an optimal solution, the number of operations to be applied in order to find this solution and the homogeneity of the bench tests. It presents two major advantages: testing the efficiency of our approach and allowing us to find values for the diverse parameters of the generalization algorithm by successive tests.

At this time, we are working on the automatic acquisition of ontologies from partial syntactic analysis of phrases. This work will allow us, in a middle-term prospect, to apply the described generalization process to a great amount of data very similar to Thematic Units. In a more long-term prospect, this should entail the application of the complete MLK system on data automatically extracted from large volumes of texts.

References

1. Raymond Mooney and Gerald DeJong. Learning schemata for natural language processing. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, 1985.
2. Michael J. Pazzani. Integrating explanation-based and empirical learning methods in OCCAM. In Proceedings of the Third European Working Session on Learning, Glasgow, 1988.

3. Ashwin Ram. Indexing, elaboration and refinement: incremental learning of explanatory cases. Machine Learning, 10(3):7-54, 1993.
4. Olivier Ferret and Brigitte Grau. An Aggregation Procedure for Building Episodic Memory, 15th IJCAI, Nagoya, 1997.
5. Olivier Ferret et Brigitte Grau, Construire une mémoire épisodique à partir de textes : pourquoi et comment ?, RIA, vol.12, n°3, pp. 317-409, 1998.
6. John F. Sowa. Conceptual Structures: Information Processing in Mind and Machine. Addison Wesley, 1984.
7. Guy W. Mineau. Induction on Conceptual Graphs: Finding Common Generalizations and Compatible Projections. Conceptual Structures - current research and practice. Eds. T. E. Nagle, J. A. Nagle, L. L. Gerholz and P. W. Eklund, New York, Ellis Horwood, 1992.
8. Isabelle Bournaud. Construction de hiérarchies conceptuelles pour l'organisation de connaissances. LMO'96, 1996.
9. Gerard Ellis. Compiled Hierarchical Retrieval. Conceptual Structures - current research and practice. Eds. T. E. Nagle, J. A. Nagle, L. L. Gerholz and P. W. Eklund, New York, Ellis Horwood, 1992.
10. K. B. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In Machine Learning, An Artificial Approach, Vol. 1, pp. 331-363, Morgan Kaufmann, 1983.
11. Marc Champesme. Using Empirical Subsumption to Reduce Search Space in Learning. International Conference on Conceptual Structures, ICCS'95. Santa Cruz, California, USA, 1995.
12. Karim Chibout and Anne Vilnat, Computational Processing of Verbal Polysemy with Conceptual Structures, Acts Conceptual Structures: Theory, Tools and Applications (6th ICCS), Montpellier, LNCS volume 1453, p. 367-374, 1998.
13. Stephen Muggleton et Luc De Raedt, Inductive Logic Programming: Theory and Methods. Journal of Logic Programming 1994 :19,20 :629-679.14. Esra Erdem et Pierre Flener. A Re-Definition of Least Generalizations, and Construction Modes as a New Declarative Bias for ILP, Unpublished, 1998, http://www.cs.utexas.edu/users/esra/papers/ErdFle98.ps.