ATLANTIS PRESS
Atlantis Studies in Uncertainty Modelling, volume 3
Joint Proceedings of the 19th World Congress of the International Fuzzy Systems Association (IFSA), the 12th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT), and the 11th International Summer School on Aggregation Operators (AGOP)

CG2A: Conceptual Graphs Generation Algorithm

Adam Faci[a,b] and **Marie-Jeanne Lesot**[a] and **Claire Laudy**[b]
[a]Sorbonne Université, CNRS, LIP6, F-75005 Paris, France {adam.faci,marie-jeanne.lesot}@lip6.fr
[b]LRASC, Thales, 91477 Palaiseau, France claire.laudy@thalesgroup.com

Abstract

Conceptual Graphs (CGs) are a formalism to represent knowledge. The production of CG benchmarks is currently a crucial need in the community to validate algorithms. This paper proposes CG2A, an algorithm to build synthetic CGs exploiting most of their expressivity. CG2A takes as input constraints that constitute ontological knowledge including a vocabulary and a set of CGs with some label variables, called γ-CGs, as components of the generated CGs. Extensions also enable the automatic generation of the set of γ-CGs and vocabulary to ease the database generation and increase variability.

Keywords: Conceptual Graphs, Data generation, Predictability, Variability

1 Introduction

Conceptual graphs (CGs) [5] refer to a family of formalisms of graph-based knowledge representation, close to existing semantic web languages such as RDF(S) [14, 3] and OWL [15]. Their advantages include their data modeling capacities, grounded on first-order logic (FOL) semantics, as well as the possibility to manage knowledge through graph-based operations. They differ from other graph-based semantic knowledge representations by the clear distinction between ontological knowledge and factual knowledge which ensures conformity of reasoning with FOL formulas. They have been extended to include fuzzy sets based information[17]. CGs have many applications in research and industry, e.g. in security [11], semi-structure data modeling [19], software development [20], clustering [16], music [10] or decision-making [18] to name a few. One drawback is the major difficulty in designing a CG database without prior expertise. It may be one of the reasons why there has been for a long time the need of CG datasets of quality [1, 7], in particular for benchmarking. Indeed, existing CGs are either private properties, small examples to illustrate the formalism or specific use cases that only represent a reduced part of the formalism, for instance with no ontological part. As discussed in Section 2, T_{nat} [2] is a translation algorithm from RDF(S)/OWL [14, 3, 15] to CGs. The quality of the resulting CG base depends on the quality of the RDF(S)/OWL input base and only a part of the CG formalism is taken into account. The notion of quality is here mainly understood as the combination of two criteria: variability, i.e. the fact that many datasets varying on several characteristics can be generated from the same input, and immediate predictability, i.e. the fact that the characteristics of the resulting database can be derived from the input without mining of the base. Two additional criteria are used: expressiveness, i.e. how much of the CG formalism is represented, and computational efficiency.

This paper proposes CG2A (Conceptual Graphs Generation Algorithm), an algorithm generating a CG database from a set of constraints corresponding to ontological knowledge. Factual knowledge is generated from the input ontological knowledge defined as a vocabulary and a set of CGs with some label variables, called γ-CGs. The ontological knowledge thus constitutes an underlying model of the generated dataset. A benefit is that the user has explicit knowledge on datasets generated from this model, without analysis of the generated datasets. It is inspired by the benchmark generation process in the clustering community, where synthetic datasets are generated from given data distributions that determine expected results for a clustering algorithm running on these datasets. CG2A has been used to validate cgSpan [9], an algorithm mining frequent patterns in CGs.

In order to generate realistic datasets, without a total randomization of labels and structure, ontological knowledge is required as input. This corresponds to

constraints on the generated CGs domain. Still this input can be generated automatically from a reduced set of numerical parameters based on three proposed extensions to the algorithm, respectively automating the generation of the vocabulary, the γ-CGs and the γ-CGs variables. The generated CGs domain is therefore extended to all CGs that can be defined over the ontological knowledge generated from the given set of numerical parameters. The use of these extensions reduces immediate predictability as the generated ontological knowledge is not known without further analysis. Consequently the CG2A version to use depends on the use case: on one hand it is possible to define all input ontological knowledge or reuse an existing one to represent a specific situation; on the other hand the use of automatically generated input enables an easier and faster CG generation and leads to more variability.

Section 2 presents a short reminder about the Conceptual Graphs formalism, including the proposed γ-CGs, and a state of the art on CG dataset generation. Section 3 presents the proposed Conceptual Graphs Generation Algorithm as well as its randomization modules. Section 4 describes the conducted experimental study, detailing the proposed criteria, to measure variability and efficiency, and to assess qualitatively immediate predictability and representativity of CGs formalism. Section 5 concludes the paper and discusses some directions for future works.

2 State of the art

2.1 Conceptual Graphs

Conceptual graphs [5] are a family of formalisms for knowledge representation, made of ontological and factual knowledge. A CG is a bipartite graph representing factual knowledge referring to a vocabulary that represents the ontological knowledge.

A vocabulary is a 5-tuple $\mathscr{V} = (T_C, T_R, \sigma, I, \tau)$. T_C and T_R, that respectively correspond to concept and relation types, are two partially ordered disjoint finite sets, where ordering corresponds to generalisation. Each relation type is associated to a signature that specifies its arity and the maximal concept type of each of its arguments. σ maps each relation type t_r to its signature $\sigma(t_r)$ represented as a tuple $(t_1, \ldots, t_n)$ where $\forall\, i \in\ T_C$ and n is the relation arity. I is a set of individual markers used to instantiate concept nodes. τ is a mapping from I to T_C that defines the type instantiated by each individual marker.

A CG is a bipartite labeled multigraph represented as a 4-tuple G = *(C,R,E,label)* defined over such a vocabulary $\mathscr{V}$. C and R correspond to concept and relation nodes, E denotes the set of the edges connecting elements of C and R. *label* is a labelling function from C to $T_C \times\ I$ and from R to T_R. For any $r \in\ R$, $label(r) = t_r \in\ T_R$ is the type of r and for any $c \in\ C$, $label(c) = (t_c, i_c) \in\ T_C \times\ I$ where t_c is the type of c and i_c is the optional individual marker of c.

We extend this formalism to represent CGs where some labels are replaced with variables, named γ-Conceptual Graphs and inspired by λ-BGs from the CG formalism [5]. A γ-CG $\Gamma = ((v_1, D_1)\ldots(v_n, D_n))G,\ n \geq\ 1$ is a conceptual graph G with n variables v_i and their respective domains D_i. Each variable v_i is assigned to a label of G, either a relation type label, a concept type label or a marker label. It is illustrated in Fig. 6 where v_1, v_2 and v_3 are respectively assigned to a concept type, marker and relation type. For a variable v_i associated with a relation type t_r, its domain D_i is a subset of T_R reduced to relation types of same arity, i.e. $D_i = \{t \in\ T_R, arity(t) = arity(t_r)\}$. For a variable v_i associated with a concept type t_c, its domain D_i is a subset of T_C reduced to the concept types respecting all constraints imposed by the signatures of connected relation nodes. For a variable v_i associated with a marker m_i, the domain is a subset of I reduced to markers of same or more specific concept types, i.e. $D_i = \{m \in\ I, type(m) \leq\ type(m_i)\}$.

Finally we define a neighborhood as a node and its connected nodes, for instance a relation node and its connected concept nodes.

2.2 Conceptual graphs data generation

To the best of our knowledge, there is no CG dataset of quality available. Available CG datasets are for instance based on flat hierarchy, i.e. with no order defined between types, as the conceptual graph dataset for NLP/NLU [8][1]. Industrial CG datasets are not publicly available.

CG datasets can be obtained as the result of translation from datasets respecting other formalisms. The main differences with a proper generation algorithm are that the goal is different and that the resulting dataset depends on the chosen input dataset and its formalism. T_3 and T_{nat} [2] are algorithms translating datasets expressing knowledge in the RDF(S)/OWL formalism [14, 3, 15] to the CG formalism. They are implemented in CoGUi[2], a tool to visualize and manipulate CGs. Their main validation criterion is the equivalence between reasoning in RDF(S)/OWL before translation

[1] `https://github.com/alexge233/conceptual_graph_set`

[2] `http://www.lirmm.fr/cogui/`

and reasoning in the CGs formalism after translation: they aim at ensuring that the same conclusions are deduced from the same premises in both datasets, and that reasoning remains identical when translating back to RDF(S)/OWL. In this regard, T_3 is a sound and complete translation w.r.t. RDF(S) but not intuitive visually. Indeed, it represents the RDF(S)/OWL triplets constituted of subject, object and predicate by a blank relation node linking these three elements as concept nodes. As a consequence the fact that relations in CGs correspond to relations between entities is not represented. It is more intuitive to represent relations nodes connecting concepts nodes as predicates linking subject and object, as is the case of T_{nat}.

In addition T_{nat} exploits the separation between background knowledge and factual knowledge by translating the predicates as binary relation nodes linking the subject and object, both translated as concept nodes. This translation ensures two properties that enable a better representation of CGs but hinders the reasoning equivalence. First, a *separability condition* has to be satisfied by the input RDF(S)/OWL dataset: it states that any entity in the knowledge base appears either as a class, a property or an instance (in the RDF(S) sense). Otherwise the entity is considered ambiguous and different choices are made depending on the situation: if a violation of this separation requirement between classes and properties occurs, the ambiguous predicates are ignored; if a violation occurs between classes and instances, or properties and instances, the triples involving the ambiguous entity as an instance are ignored. Second, a distinction between ontological and factual triples is performed to populate either the vocabulary or the conceptual graphs when a new triple is processed. This distinction stems from the flexibility of RDF(S) that does not impose a clear distinction between factual and ontological knowledge. A particularity of CG databases constituted with T_{nat} is that only relations of arity 2 are built because of RDF(S) restrictions. This drawback is minimised by the fact that relation of arity greater than 2 can always be brought back to a set of relations of arity 2, and conversely. This is immediate considering that CGs are graph-based representations of first-order logic formulas and that relations correspond to atomic formulas, which are 2-decomposable [12].

3 CG2A: generation from a set of constraints

CG2A is a three step algorithm that generates a CG dataset from ontological knowledge. It ensures representativity of the CG formalism as well as variability and immediate predictability of the generated base characteristics. First CG2A generates a CG by randomly combining input γ-CGs until reaching a specified minimum size. Then variables are assigned random values from their respective domains. Finally the nodes in the generated CGs with the same individual marker are merged to increase the connectivity of the resulting CG. CG2A iterates until a specified number of CGs is reached. This section first describes CG2A input and details its three steps. It then presents its extension modules automating the generation of input.

3.1 Input

CG2A takes five parameters. They include *maxCGs*, the number of generated CGs, *minSize*, the minimum size of each generated CG in number of nodes, and *maxSpe*, the maximum number of specializations operated on each type label variable. More precisely, each generated CG is increased in size until reaching *minSize*, then the type label variables are specialized a number of times between 0 and *maxSpe*, and the iteration resumes until reaching *maxCGs* CGs in output base. These three parameters thus determine the stopping conditions of the algorithm. The two other parameters are a vocabulary $\mathscr{V}$ and a set of γ-CGs $\mathscr{G}$, as detailed hereafter.

The vocabulary, as formally presented in Section 2.1, contains a hierarchy on concept types, a hierarchy on relation types and the set of relation signatures. The individual markers set is populated during generation when a concept node is instantiated. The set $\mathscr{G}$ of γ-CGs are the components of the generated CGs. Compared to a classic CG, as described in Section 2.1, some labels are replaced with a variable referring to a list of values from the vocabulary. Thus γ-CGs are configurable constraints.

3.2 Proposed algorithm

Fig. 1 gives the pseudo-code of CG2A, commented below: CG2A generates sets of CGs by randomly combining elements from the set $\mathscr{G}$ of input γ-CGs into bigger CGs. Let $G_c = (C_c, R_c, E_c, label_c)$ be the currently generated CG and $\Gamma = ((v_1, D_1) \dots (v_n, D_n))G = (C, R, E, label)$ be a γ-CG from the input set $\mathscr{G}$. First G variables are instantiated with values from their domains, and the ones assigned to type labels are specialized from 0 to *maxSpe* times using hierarchies from $\mathscr{V}$. Then $G_o = (C_c \uplus C, R_c \cup R, E_c \uplus E, label_c \uplus label)$ is formed from the join of G_c and G where $\uplus$, based on coreferent nodes merge [4], is a specific union whose differences follow: if there are elements of C_c with an individual marker similar to one of C, only the most specialised is kept. Then neighborhoods are merged so that both neighborhoods are connected to the resulting

Input: $\mathscr{V} = (T_C, T_R, \sigma, I, \tau)$, $\mathscr{G}$,
maxCGs, *minSize*, *maxSpe*.

- Initialize $\mathscr{G}_o$ to an empty set
- Iterate until size($\mathscr{G}_o$)$\geq$*maxCGs*
  - Initialize $G_c = (C_c, R_c, E_c, label_c)$ to an empty CG
  - Iterate until $size(G_c) \geq minSize$
    1. Get $(v_1, \ldots, v_n)G = (C, R, E, label)$ in $\mathscr{G}$
    2. Assign value to each variable v_i
    3. Specialize each type label var from 0 to *maxSpe* times
    4. $G_c = Join(G_c, G)$
  - Add G_c to $\mathscr{G}_o$
- Return $\mathscr{G}_o$

Figure 1: Pseudo-code of the proposed CG2A.

node, i.e. elements of E_c and E corresponding to the two merged nodes are reassigned to the resulting node.

This join operator is illustrated in Fig. 2, where the node colour indicates their associated markers: the two green nodes, resp. at the right end of the current CG and at the top of the added CG, are merged. They are not necessarily of same type; the most specific type is retained, indeed as illustrated in the example, the connected signatures enforce a specialisation of this type.

Without operator $\uplus$, the algorithm would obtain for each generated CG a set of unconnected instantiated elements from $\mathscr{G}$. The connectivity of the resulting CGs thus depends on the number of common nodes. There are other techniques available for graph fusion based on the join operator [13, 6], but this simple fusion operator based on coreferent nodes merge operator is sufficient in this case.

CG2A stops CGs combinations upon reaching the desired minimum size, *minSize* and stops generation upon reaching the desired number of generated CGs, *maxCGs*. Since CGs of potentially several nodes are added at the same time, the resulting CGs are typically greater than *minSize*.

The advantages of using γ-CGs instead of directly defining many variants of a CG is that the process is automatic and that from one designed γ-CG, many can be generated while keeping its structure and its semantic. As a consequence CG2A guarantees variability from one input as well as predictability thanks to the knowledge of the input γ-CGs and their characteristics.

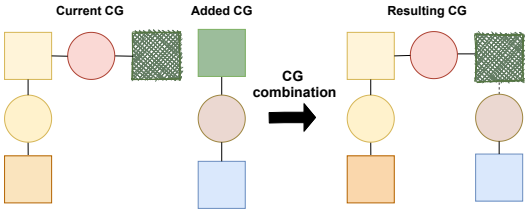

Figure 2: CG join step in CG2A. In this representation, concept nodes are squares and relation nodes are circles.

3.3 Input generation to increase variability

This section presents three modules to generate automatically the input so as to increase variability and ease the generation. All mentioned numerical parameters can be replaced by mean and standard deviation, and drawn from the associated normal distribution.

3.3.1 Automatic generation of vocabulary

This module generates automatically the vocabulary $\mathscr{V}$ from four parameters: the desired depth of hierarchies both for concept and relation types, the maximum number of children of each node of the hierarchy and the number of individual markers for each concept type. This generation is random, however the four parameters ensure a number of fixed characteristics in the resulting vocabulary.

As illustrated on Fig. 3 and 4, for concepts and relations respectively, a hierarchical structure is generated until the desired depth is reached and random unique labels are assigned to each node of the hierarchy. The hierarchy of concept types is a rooted tree with the most general type, denoted "Top" in Fig. 3. For each concept type, a list of individual markers is generated. For relation types, a top type is respectively defined for each arity, e.g. denoted T_3 for the case of arity 3 illustrated in Fig. 4. Then signatures are defined using the previously generated hierarchy of concepts for each relation type, with each relation top type having a default signature with only Top as concept type restriction. A more specific relation type has a more restrictive signature, meaning that the specified restrictions require an identical or more specific concept type. It is illustrated in Fig. 4 where at each step the hierarchy is deepened and signatures are defined as identical or more restrictive than signatures of more general relation types.

3.3.2 Automatic generation of input γ-CGs

This module generates automatically a set of input γ-CGs so as to define $\mathscr{G}$, as illustrated in Fig. 5. The generated γ-CGs actually have no defined variable, but

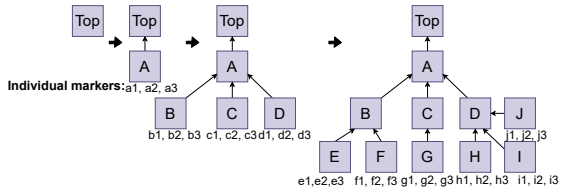

Figure 3: Automatic generation of a hierarchy of concept types, here performed in three steps. Parameters are: Depth = 4; Maximum number of children = 3; Number of individual markers per type = 3.

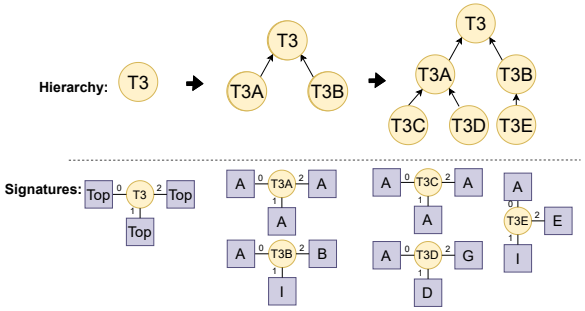

Figure 4: Automatic generation of a hierarchy of relation types with their signatures, here performed in two steps. Parameters are: Depth = 3; Maximum number of children = 3.

as this module can be used independently, one can subsequently define variables manually or use automatic generation. This module takes as input a vocabulary $\mathscr{V}$ (possibly generated automatically using the module described in the previous subsection 3.3.1) and two numbers: the number of γ-CGs to be generated and their minimum size. While designing CGs requires proficiency in the formalism and also requires to respect of the vocabulary constraints, only two numbers restrict the domain of γ-CGs generation.

The γ-CGs module is similar to applying CG2A to the set of signatures taken from the input vocabulary $\mathscr{V}$. The only difference is that each label is treated as a variable with no constraint, so that they all have a randomly attributed label and are randomly specialized.

3.3.3 Automatic generation of variables

This module generates variables in input γ-CGs. Instead of having to choose which labels are variables and their respective domains, this module takes a set of γ-CGs and a matching vocabulary as input, which can be generated with the previous modules, as well as five numbers: the numbers of concept types, relation types and individual marker variables per CG, the number of values per variable and the number of specialisations.

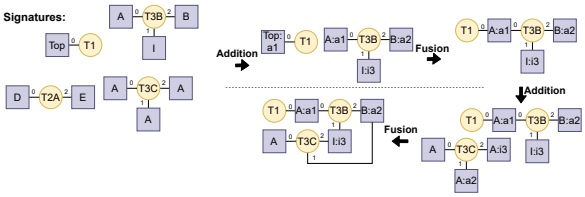

Figure 5: Automatic generation of input γ-CGs.

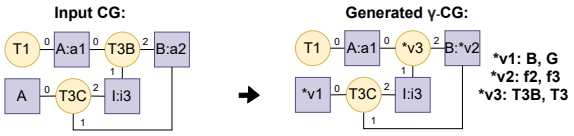

Figure 6: Automatic generation of γ-CGs variables.

It may be run even if variables have already been defined in the γ-CGs to increase their number.

First, for each CG, variables are attributed to a relation type, a concept type or an individual marker. Then a list of values is associated with each variable. Fig. 6 illustrates this operation with the variables v_1, v_2 and v_3. For a relation type as v_3 in Fig. 6, the module chooses from relations with the same arity and identical or less restrictive concept types. For a concept type as v_1 in Fig. 6, the module chooses from concept types equal to or more specific than the ones compatible with the signatures of the neighborhood. For an individual marker as v_2 in Fig. 6, it chooses from individual markers with an assigned concept type equal to or more specific than the concept node type. Because of these restrictions, first relation type variables are defined, then concept type variables and finally individual marker variables. Then, all assigned variables corresponding to type labels are specialized a number of times up to the number of specialisations parameter *maxSpe*.

4 Experimental study

This section presents the experimental study conducted to evaluate CG2A both per se and compared to existing translation techniques. The input used in these experiments as well as the desired properties of variability, predictability, representativity and efficiency are presented in the following subsection. Then results are subsequently examined in view of each criterion.

4.1 Experimental protocol

We consider four criteria to optimize for a data generation algorithm. First the algorithm has to enable *variability* in the generated data from one input. Indeed from a unique ensemble of ontological knowl-

edge the possibility to produce datasets with various sizes and characteristics may be required to assess the breadth of corresponding possible facts. Second it has to provide a certain level of *immediate predictability*. This means that from a given data generation algorithm, denoted dGenA, the expected results of a data mining algorithm, denoted dMinA, run on a database generated by dGenA can be defined. Obviously the expected results will differ depending on dMinA goal and whether deductions from the dGenA input are relevant regarding this goal. This criterion is essential to enable dMinA validation but is difficult to quantify. Third the generation algorithm has to achieve *representativity*, i.e. exploit as much of the CG formalism as possible. Any fragment that cannot be represented limits the use of the generation algorithm to the subset of situations where this fragment is unnecessary. Fourth the *computational time* has to be minimised. Indeed it may be crucial to obtain quickly a set of examples datasets when given ontological knowledge. Besides many generations may be required while testing parameters variations to satisfy expectations.

These criteria are used to compare CG2A with T_{nat} using the following configuration. The input of T_{nat} is an RDF(S)/OWL dataset[3], modified to solve some issues when parsing for translation. The modified dataset includes an ontology constituted of 39 concept types and 35 relation types organized in a hierarchy with a depth of 5 and between 1 and 3 children for each non-leaf type. An option proposed by T_{nat} to split CGs in several connected components has been used, so that each resulting CG is a connected graph. Otherwise it results in one big unconnected CG. The input for CG2A matches the characteristics of the database generated with T_{nat} to ensure that we mostly evaluate the influence of the algorithmic part rather than the variability induced by parameters. Running T_3 on the same dataset results in a unique CG of about 6000 vertices and no ontology (other than RDF/RDF(S) knowledge). All relation nodes are the "triple" relation node to connect elements of a triple. As it does not lead to many CGs nor a proper ontology, T_3 is irrelevant for our concerns and in consequence its results are not used in what follows.

The first row of Tab. 1 displays the results of T_{nat} and the following ones display the average results across 100 runs on CG2A and its variants with each extension module individually (Auto Voc, Auto γCG and Auto Var respectively) and CG2A with all extension modules (Full Auto).

[3] `http://www.semanticbible.com/ntn/ntn-view.html`

4.2 Variability results

In order to assess numerically the notion of variability, we consider the following criteria: the average size of generated CGs in number of nodes, denoted NbN, and the average number of unique labels, denoted NbL, in one CG, both with their standard deviation, and the distribution of relation arity from 1 to 3, denoted Ar1, Ar2 and Ar3. These criteria are defined for the sake of comparison with existing techniques providing CG datasets. They are not optimized or even considered by the translation algorithm T_{nat} that aims at maximizing conformity of reasoning between the input and output databases. CG2A and its variants can represent relations of arities greater than 3, but the use Ar1, Ar2 and Ar3 seems to suffice for the presented experiments.

It can first be observed that the translation of T_{nat} results in a huge CG of hundreds of nodes and many CGs of a few nodes. This is why in Tab. 1 the standard deviation of NbN is significantly more important than the average NbN, and the average NbL is relatively small. Moreover Ar1 and Ar3 are zero, which is due to the fact that the RDF(S)/OWL languages do not represent relations of arities different than 2.

CG2A leads to significantly smaller standard deviations, and the resulting CGs characteristics are close to the input parameters.

It is expected that the use of automatically generated input leads to more variability in the expected results. It can be observed that, indeed, the use of random vocabulary increases NbL standard deviation and relation arities while the use of random CGs increases the standard deviation of NbN and arities. The results of the fully automated CG2A combine these consequences.

4.3 Predictability results

Predictability refers to the possibility to define apriori the results a data mining algorithm is expected to obtain when run on a generated data set. It can be put in balance with the cost of the specific resources to deploy and efforts to undertake to define the input, in particular the ontological knowledge.

T_{nat} possesses the advantage that it generates a CG database from a RDF(S)/OWL dataset, without requiring any prior knowledge on this dataset. However this implies that without mining the input dataset first, T_{nat} cannot be considered as predictable. As such, it does not meet the immediate predictability aim.

CG2A can be considered as predictable as the generated CG are defined as combination of the input γ-CG that are defined over the input vocabulary: $\mathscr{G}$ together with $\mathscr{V}$ determine the expected results whose charac-

Test	NbN (avg. ± sd.)	NbL (avg. ± sd.)	Ar1	Ar2	Ar3
T_{nat}	15.2 ± 321	3 ± 1	0	3	0
CG2A	36.3 ± 4	22.5 ± 4	0.5	44	3
Auto Voc	33 ± 3.5	55 ± 14	4	34	9
Auto γCG	39.9 ± 2	22 ± 4.1	6	22	31
Auto Var	35.3 ± 4	32 ± 7	0.4	42	7
Full Auto	35 ± 4	67 ± 17	8	33	26

Table 1: Results for one run of T_{nat} and different versions of CG2A averaged across 100 runs (see experimental protocol in Sec. 4.1).

teristics are known.

CG2A used with automatic generation of vocabulary or γ-CGs changes the nature of the expected results, that are defined in terms of their general characteristics rather than specific information. As compared to CG2A, in the case of Auto Voc, only the general characteristics of the vocabulary are known, its specificities are not. In the case of Auto γ-CG, only the general characteristics of the components used to build the generated CG are known (number and size of the γ-CGs, as well as the signatures they are built on): Auto γ-CG adds to the CG2A variation from γ-CGs to generated CGs another variation, from the signatures to the input γ-CGs.

CG2A used with automatic variable generation, Auto Var, does not modify predictability significantly. We consider that automatic variables slightly reduce the immediate predictability by increasing variability.

The fully automated CG2A variant, that includes the three random modules, combines their respective properties and defines expected results in terms of their general characteristics. Overall, the results are significant when compared with T_{nat}: there is much more variability with CG2A, and while CG2A has to cope with a balance between variability and immediate predictability, T_{nat} does not enable immediate expected results.

4.4 Representativity results

As discussed earlier, T_{nat} outputs CG databases only including relations of arity 2, that are unbalanced with respect to ontological or factual knowledge, i.e. that mostly comprehend one of the two types. Yet, it can be argued that the advantage of disposing of relations with varying arity only is a question of perspective or reformulation and that the lack of balance is due to the considered input RDF(S)/OWL knowledge bases. Similarly, the fact that T_{nat} often results in bases constituted of one or two huge conceptual graphs and the rest containing only a few nodes is mostly due to the available RDF(S)/OWL bases, rather than the algorithm itself.

CG2A and its variants more naturally avoid these drawbacks. They enable the representation of most of the CG formalism as reminded in Sec. 2. CG2A retains most of T_{nat} advantages by using the CoGui formalism and adds the possibility to generate a large proportion of relation nodes with various arities, and to have both a wide vocabulary and a considerable quantity of CGs, i.e. both ontological and factual knowledge. Besides when defining the input, e.g. the characteristics of the vocabulary, the user can determine the extent of the CGs formalisms that is exploited, which is one main advantage of CG2A. Generally speaking, CG2A ensures that the user can choose more precisely the characteristics of the resulting base.

4.5 Efficiency results

In the conducted experiments, depending on the stopping conditions parameters, most CG2A runs last less than one second and never exceed 5 seconds. The computer used has 16GB of memory and a i5-2500k processor. The use of the automatic generation modules increases the computational time, with a factor 2, however the time spent to design input without these modules is not accounted for. T_{nat} generation lasts much longer, with factor of three up to ten depending on the size of the input base. At some point if the input is too massive, T_{nat} aborts so the tests could not be pursued.

5 Conclusion

This papers proposes CG2A, an algorithm to produce Conceptual Graphs. CG2A enables more variability in the generated dataset than any other known method as it offers a lot of variance in the size and labels of CGs as well as a reasonable proportion of relation nodes with an arity different than 2 in the generated CGs. As such, numerous different situations can be tested through the use of CG2A, either a strongly constrained domain to test a specific case or a more relaxed generation to test a broad variety of situations. In addition the CG formalism is well represented with deep hierar-

chies, variation in the signatures and various arities of relation nodes. Finally, when using a different method to obtain a CG dataset, it is not possible to define expected results without first mining the dataset or having prior knowledge on this dataset.

Ongoing works aim at extending CG2A to generate more complex CGs, e.g. nested or fuzzy CGs. Another direction considers a different possibility in terms of predictability, expressed as a desired distribution over CGs parameters so as to ensure that the resulting dataset respects such a distribution.

References

[1] J.-F. Baget, O. Carloni, M. Chein, D. Genest, A. Gutierrez, M. Leclere, M.-L. Mugnier, E. Salvat, R. Thomopoulos, Towards benchmarks for conceptual graphs tools, in: CS-TIW'2006, Aalborg University Press, 2006, pp. 72–86.

[2] J.-F. Baget, M. Croitoru, A. Gutierrez, M. Leclère, M.-L. Mugnier, Translations between rdf(s) and conceptual graphs, in: Springer (Ed.), Conceptual Structures: From Information to Intelligence, Springer, Berlin, Heidelberg, 2010, pp. 28–41.

[3] D. Brickley, R. V. Guha, B. McBride, Rdf schema 1.1, W3C recommendation 25 (2014) 2004–2014.

[4] M. Chein, M.-L. Mugnier, Concept types and coreference in simple conceptual graphs, in: Int. Conf. on Conceptual Structures, Springer, 2004, pp. 303–318.

[5] M. Chein, M.-L. Mugnier, A Graph-Based Approach to Knowledge Representation: Computational Foundations of Conceptual Graphs (Part. I), Springer, 2008.

[6] M. Chein, M.-L. Mugnier, Conceptual graphs are also graphs, in: Int. Conf. on Conceptual Structures, Springer, 2014, pp. 1–18.

[7] M. Croitoru, B. Hu, S. Dashmapatra, P. Lewis, D. Dupplaw, L. Xiao, A conceptual graph based approach to ontology similarity measure, in: Int. Conf. on Conceptual Structures, Springer, 2007, pp. 154–164.

[8] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, P. Kalnis, Grami: Frequent subgraph and pattern mining in a single large graph, Proc. of the VLDB Endowment 7 (7) (2014) 517–528.

[9] A. Faci, M.-J. Lesot, C. Laudy, cgSpan: Pattern Mining in Conceptual Graphs, in: Int. Conf. on Artificial Intelligence and Soft Computing, 2021.

[10] M. D. Fowler, John cage's silent piece and the japanese gardening technique of shakkei: Formalizing Whittington's conjecture through conceptual graphs, Journal of Mathematics and Music 13 (1) (2019) 4–26.

[11] Z. Fu, F. Huang, K. Ren, J. Weng, C. Wang, Privacy-preserving smart semantic search based on conceptual graphs over encrypted outsourced data, IEEE Transactions on Information Forensics and Security 12 (8) (2017) 1874–1884.

[12] P. Jeavons, D. Cohen, M. C. Cooper, Constraints, consistency and closure, Artificial Intelligence 101 (1-2) (1998) 251–265.

[13] C. Laudy, J.-G. Ganascia, C. Sedogbo, High-level fusion based on conceptual graphs, in: Int. Conf. on Information Fusion, IEEE, 2007, pp. 1–8.

[14] F. Manola, E. Miller, B. McBride, Rdf primer, W3C recommendation 10 (1-107).

[15] D. L. McGuinness, F. Van Harmelen, Owl web ontology language overview, W3C recommendation 10 (10) (2004) 2004.

[16] A. Pérez-Suárez, J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa, A review of conceptual clustering algorithms, Artificial Intelligence Review 52 (2) (2019) 1267–1296.

[17] R. Thomopoulos, P. Buche, O. Haemmerlé, Representation of weakly structured imprecise data for fuzzy querying, Fuzzy Sets and Systems 140 (1) (2003) 111–128.

[18] S. Tremblay, J.-F. Gagnon, D. Lafond, H. M. Hodgetts, M. Doiron, P. P. Jeuniaux, A cognitive prosthesis for complex decision-making, Applied ergonomics 58 (2017) 349–360.

[19] V. Varga, C. Săcărea, A. E. Molnar, Conceptual graphs based modeling of semi-structured data, in: Int. Conf. on Conceptual Structures, Springer, 2018, pp. 167–175.

[20] S. V. Vlasenko, G. A. Efimenko, D. V. Gnezdilov, O. I. Brikova, Approaches to conceptual graphs notations using in digital manufacturing software environments, in: 2019 IEEE Conf. of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2019, pp. 731–735.