A Comparison of Different Conceptual Structures Projection Algorithms

Heather D. Pfeiffer and Roger T. Hartley

New Mexico State University
{hdp,rth}@cs.nmsu.edu

Abstract. Knowledge representation (KR) is used to store and retrieve meaningful data. This data is saved using dynamic data structures that are suitable for the style of KR being implemented. The KR allows the system to manipulate the knowledge in the data by using reasoning operations. The data structure, together with the contents of the transformed knowledge, is known as the knowledge base (KB). An algorithm and the associated data structures make up the reasoning operation, and the performance of this operation is dependent on the KB.

In this paper, the basic reasoning operation for a query-answer system, projection, is explored using different theoretical algorithms. Within this discussion, the associated algorithms will be using different KBs for their Conceptual Graph (CG) knowledge representation. The basic projection algorithm defined using the CG representation is looking for a graph morphism of a query graph onto a graph from the KB.

The overall running time for the projection operation is known to be a NP class problem; however, by modifying the algorithm, taking into account the associated KB, the actual time needed for discovering and creating the projection/s can be improved. In fact, a new projection algorithm will be defined that, given a typical query onto a carefully defined KB, presents a running time for the actual projection that only grows with the number of projections present.

1 Introduction

Query-Answer systems are very important in business and industry today. However, these systems need to be able to represent *knowledge* in the computer in order to use reasoning techniques when attempting to answer a query for a problem domain. In the computer, the description of the problem to be solved has become known as *knowledge representation*, *KR*. This representation must be able to store and retrieve meaningful data so that reasoning operations can be performed. The most common reasoning technique used in query-answer systems is *projection* of the query onto the stored knowledge. Later this work will discuss more about this technique, but first more about storing meaningful data will be presented.

One type of KR is semantic networks. These networks are displayed as a discrete graphical structure of vertices and arcs [1]. Within the graphical structure,

U. Priss, S. Polovina, and R. Hill (Eds.): ICCS 2007, LNAI 4604, pp. 165–178, 2007.

the vertices are called nodes and may be displayed as circles or boxes. The arcs are called links and are displayed as lines with arrows between the nodes. The nodes are related to each other through their links where the links are assigned a one-to-one correspondence to a conceptual meaning [2]. The nodes are sometimes called conceptual units and may be seen as objects within the network. These objects are of many different types including entities, attributes, events or even states. On top of the semantic network, abstract hierarchies are organized according to levels of generalization for the conceptual units. The links of the network form relational connections between the conceptual units, such that the valence (or parity) of the connection is the number of units that are associated with a particular unit. In a semantic network links are usually dyadic (binary) connecting two conceptual units together.

Even though there are multiple semantic network representations available, the representation that shows much flexibility is *conceptual structures*. Conceptual Structures (CS) are a logic based representation of C.S. Peirce's existential graphs [3] developed by John Sowa [4]. Conceptual structures are like a set of logic building blocks; the definitions for some of the blocks are presented beginning with the *type* block:

Definition 1. *A type is a labeling for an abstract idea which is either a conceptual unit or a relationship. These types are members to a set, T, that may form several structures including hierarchy trees, lattices, and other related structures. When the structure is a type hierarchy lattice, the set is labeled* T_C*, and the function* ctype *maps a conceptual unit to the type label in the structure. When the structure is a relation hierarchy tree, the set is labeled* T_R*, and the function* rtype *maps a relationship to the type label in the structure.*

A *referent* block would have the following definition:

Definition 2. *A referent is an abstract conceptual unit that has been instantiated with a factual value.*

Graph diagrams that are built out of the blocks of conceptual structures are *conceptual graphs (CG)* [4,5]. For this work, a conceptual graph has the following definition:

Definition 3. *A conceptual graph is a bipartite, connected, directed graph* $G = (V, E)$*, such that the set of all vertices (nodes)* V *is partitioned into two disjoint sets* V_C *and* V_R*. The vertices are labeled, and the set* V_C *is called the* concept *nodes and the set* V_R *is called the* conceptual relations *nodes.* $e \in E$ *is an ordered pair that connects an element of* V_C *to an element of* V_R *using a directed arc.*

The label of a concept node is a pair, $c =< type, referent >$*. The type is an element of the set* T_C*. The referent (if present) contains the individual instantiation for the type field.*

The label of a conceptual relation node is a pair, $cr =< type, signature >$*, where type is an element of the set* T_R*, and the signature is a pair,* $s =< I, O >$ *where* I *is the arcs that are directed into the conceptual relation and* O *is*

the arcs that are directed out from the conceptual relation. The signature is further defined by its subset category of either relation *or* actor*. The relation is a tuple,* $r =< type, c_1, c_2, ..., c_n >$*where type is defined above and in the signature* $I \subseteq V_C$ *and* $O \in V_C$*. The actor is a slightly different tuple,* $a =< type, c_1, c_2, ..., \{..., c_{n-1}, c_n\} >$ *where type is defined above and in the signature* $I \subseteq V_C$ *and* $O \subseteq V_C$*.*

Researchers M. Chein and M.-L. Mugnier [6] from the LIRMM group at the Universite Montpellier and other researchers [7,8] have done research on a subset of conceptual graphs known as *simple conceptual graphs (SCGs)* (see Sowa 3.1.2 [4]). As explained in Baget and Mugnier [7], these SCGs are connected, bipartite graphs where the arcs are labeled and finite but not directed, $SG = ((V_c, V_r), U, \lambda)$.

2 Foundational Projection

In general, the matching part of the projection algorithm is unification [9], and there are known linear unification algorithms for acyclic (tree) graphs [10]. Also, SCGs have been evaluated as both graph homomorphism and graph isomorphism. In their original paper from 1992 [11], Mugnier and Chein looked at general projection running times and injective projection. However, CGs and SCGs are not necessarily trees and only part of the algorithms presented next apply to injective projection, so these linear algorithms give guidance, but do not always directly apply.

As discussed in the Messmer and Bunke paper [12], a naive strategy with forward-checking for establishing a subgraph isomorphism is Ullman's backtracking in search tree algorithm [13]. Since Messmer and Bunke feel that it is a common technique with a good baseline subgraph isomorphism algorithm, the Ullman algorithm and its known complexity (from [13,12]) will be reiterated here for defining a basis for investigating projection algorithms. The basic idea of Ullman's algorithm is to take one vertex of the input vertices (query graph) at a time and map it onto a model (a graph from the KB) such that the resulting mapping represents a subgraph isomorphism for a subgraph of the model (KB graph) projected from the input graph (query graph) (see page 307 and 322 of Messmer and Bunke [12]). If at some point, the mapping being built does not represent a subgraph isomorphism then the algorithm backtracks and tries a different mapping. This process is continued until all vertices, $v_1, \ldots, v_M$ in V_I of the input graph are successfully mapped onto V of the model. This either produces a subgraph isomorphism from G to G_I or stops when a vertex in V_I can not be mapped to at least one vertex in V. In the second case, the algorithm backtracks to a new v_1 in V or v_{n-1}in V and tries to remap the subgraph isomorphism.

Even though this basic algorithm works well for small model and input graphs, it performs poorly as the graphs become larger. This is because all checks are being done locally. Ullman added a forward-checking procedure to know when it is not possible for v_n to be mapped onto an available vertex in V_I (see page 322

in Messmer and Bunke [12]), so that the algorithm can backtrack immediately and save computational steps. In the best case Ullman's algorithm is bounded by: $O(NIM)$ where $N = \#model$ graphs, $I = \#labeled\,vertices$ in input graph which come from the M set of labels, $M = \#labeled\,vertices$ in model graph that are unique. In the worst case the algorithm is bounded by: $O(NI^M M^2)$ where $N = \#model$ graphs, $I = \#vertices\ in\ the\ input\ graph$ and are unlabeled, $M = \#vertices\ in\ the\ model\ graph$ and are unlabeled.

As can be seen, even with this general algorithm, labeling of vertices greatly improves the efficiency of the algorithm. However, it should be noted, that this algorithm does not take into account any support or hierarchy knowledge information.

2.1 Operator

The *project* operator is defined through a mapping $\pi : u \rightarrow v$, where πu is a sub-element of v. When u and v are defined to be conceptual graphs, for graph u to be a subgraph of graph v then all of the nodes and arcs of u are in v [14], and the project operator π holds to the following rules [4,15]:

- Type preserving: For each concept c in u, πc is a concept in πu where $type(\pi c) \leq type(\ c\ )$, and $\leq$ is the subtype relation. If c is an individual, that is an actual instance of an object, then $referent(\ c\ ) = referent(\ \pi c)$.
- Structure preserving: For each conceptual relation r in u, πr is a conceptual relation in πu where $type(\pi r) = type(\ r\ )$. If the ith edge of r is linked to a concept c in u, the ith edge of πr must be linked to πc in πu.

Fig. 1. Query Graph

2.2 Operation

A projection operation uses the project operator, which is a matching on a graph morphism, graph data structures with either the support information for SCGs or hierarchies when full CGs, and the actual projection algorithm. Stated in Baget and Mugnier, "the elementary reasoning operation, projection, is a kind of graph homomorphism that preserves the partial order defined on labels" [7]. Not only does projection use a project operator (see its definition in the subsection above), but the support S of the graph be it a SCG or the defined type hierarchy if a CG produces a generalization subgraph during the projection operation.

For the rest of this work, the projection operation evaluation and comparison will be restricted to injective projection. This projection mapping is not necessarily one-to-one; that is, a concept or relation in u may have more than one concept or relation in v that πu is a valid mapping. In this respect, there is more than one valid projection from u to v . When the projection operation is performed using

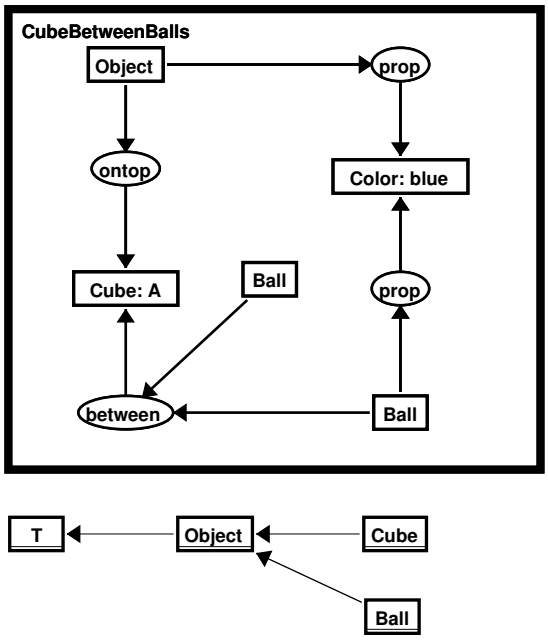

Fig. 2. KB Graph with Type Hierarchy

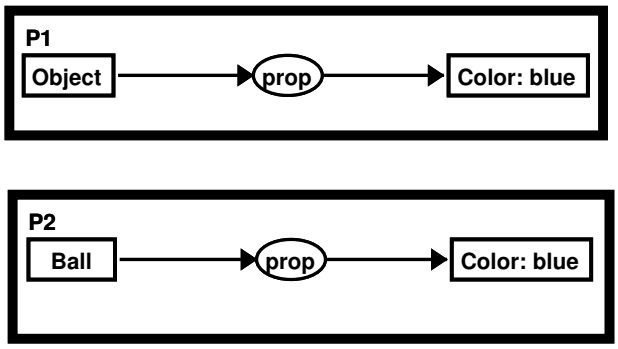

Fig. 3. Projection Results

the query graph from Figure 1 onto the KB graph and hierarchy of Figure 2, the two projections, $P1$ and $P2$, discovered are displayed in Figure 3. [1] Using the type hierarchy, both *object* and *ball* are matches; note, if no hierarchy were present, then there would be only one projection. This is a simple injective projection because of the small graphs, however, it can become complex very quickly.

[1] The figures in this section were generated by *CharGer* [16].

3 Previous Algorithms

3.1 Mugnier and Chein Projection

Given here is a discussion of the algorithms found in Marie-Laure Mugnier and Michel Chein's 1992 paper [11]. Before discussing the actual injective projection algorithm, given are some added basic definitions which will help the reader understand the algorithm. 1) Using the projection operation provided in the section above, the following additional rules on labels will be added to the graph morphism (from [11] page 240):

Definition 4. *Given two simple conceptual graphs G and G', a projection Π from G to G' is an ordered pair of mappings from (R_G, C_G) to $(R_{G'}, C_{G'})$, such that:*

(i) For all edges rc of G with label i, $\Pi(r)$ $\Pi(c)$ is an edge of G' with label i.
(ii) $\forall r \in R_G$, $type(\Pi(r)) = type(r)$; $\forall c \in C_G$, $type(\Pi(c)) = type(c)$.

There is a projection from G to G' if and only if G' can be derived from G by the elementary specialization rules [4,6].

2) Injective projection definition:

Definition 5. *Injective projection is a restricted form of projection where the image of G in G' is a subgraph of G' isomorphic to G.*

For the algorithm to compute the injective projection from T to G, it is broken into two parts. The first function is used to determine the **PROJ-ROOT** part of the definition. As seen in line 3.1 and 4, the function looks for the actual projection from T to G by comparing the relation vertices connected to concept vertex a in T to the relation vertices connected to concept c in G. The second function is used to determine the **PROJ-r** part of the definition. This function looks for possible mappings at each concept vertex by examining sub-trees. The complexity of this example, as proved on pages 248-249 of Mugnier and Chein92 [11], is such that if T is a tree and G is a cyclic conceptual graph then an injective projection algorithm is a NP-complete problem.

3.2 Croitoru Projection

Madalina Croitoru's projection algorithm is based on SCGs as described in her two 2004 papers [8,17]. This algorithm begins by starting from the foundational injective algorithm given by Mugnier and Chein [11], using SCGs with support which as stated in the Mugnier and Chein 1992 paper [11] is NP-complete. The change applied to this algorithm is to preprocess each graph pair looking for a *matching graph* as defined by the Definition 4.1 on page 8 of Croitoru2004a [8]. The actual matching graph definition is:

Definition 6. *Let $SG = (G, \lambda_G)$ and $SH = (H, \lambda_H)$ be two SCG's without isolated concept vertices defined on the same support S.*

The matching graph of SG and SH is the graph $M_{G \to H} = (V, E)$where:

- $V \subseteq V_R(G) \times V_R(H)$ *is the set of all pairs* (r, s) *such that* $r \in V_R(G)$, $s \in V_R(H)$, $\lambda_G(r) \geq \lambda_H(s)$ *and for each* $i \in \{1, \ldots, d_G(r)\}$ $\lambda_G(N^i_G(r)) \geq \lambda_H(N^i_H(s))$.

- E *is the set of all 2-sets* $\{(r, s), (r', s')\}$, *where* $r \neq r', (r, s), (r', s') \in V$ *and for each* $i \in \{1, \ldots, d_G(r)\}$ *and* $j \in \{1, \ldots, d_G(r')\}$ *such that* $N^i_G(r) = N^j_G(r')$ *we have* $N^i_H(s) = N^j_H(s')$.

These matching graphs indicate which relation vertices should be used as potential candidates for projection; therefore, reducing the search space. By this addition of preprocessing and then using the matching graphs, this makes the projection of $G \rightarrow H$ in its reduced form belong to a class of graphs on which finding the maximum clique can be solved in polynomial time [8].

3.3 Notio Projection

The Notio project is a general conceptual graph implementation with a well defined API [18]. It is currently being used by several projects [19,20,21] for working with basic reasoning operations with a CG KB. This is the author's derived theoretical algorithm (see Algorithm 1) from the Notio implementation code [18,22] for the injective projection algorithm (note: Southey never wrote any papers or documentation on the actual implemented algorithm). It should be noted for Algorithm 1, all the vertices are all labeled, but the edges are directed.

Also for the analysis of the execution times given above, the following definition of variables hold:

$| M_c |=$ #of concepts in the KB graph
$| M_r |=$ #of relations in the KB graph
$| Q_c |=$ #of concepts in the query graph
$| Q_r |=$ #of relations in the query graph
$| Q_e |=$ #of edges in the query graph
$| N |=$ #of graphs in the KB
$| KB_c |=$ #of concepts in the whole KB

As can be seen in the stated algorithm, in step 1: Notio collects all the concept and relation vertices from both the KB graph and query graph. This takes $O(| M_c | + | M_r | + | Q_c | + | Q_r |)$. In step 2: Notio attempts to see if any of the concept vertices from the KB graph maps to a concept vertex in the query graph. In this way attempting to see if there is any possible subgraph isomorphism of the KB graph to the query graph. In the worst case this step is bounded by: $O(| M_c || Q_c || KB_c |)$.

In step 13: Notio (if a possible mapping was indicated from step 2) will attempt to match all the relation vertices from the KB graph (along with their neighboring concepts along their edges) to query graph vertices with the same edge relationships. As a match is found for relation vertices in the query graph; those relation vertices are now only examined. At end of this step, it is checked that all relation vertices for the query graph were mapped. In the worst case this step is bounded by: $O(| M_r || Q_r || M_c || Q_c || KB_c | + | Q_e |^{|Q_r|})$ $O(| M_r || Q_r ||$

Algorithm 1. Notio Projection

```
 1: Get all concept and relation vertices from the KB and Query graphs
 2: for i ← 0, numfirstconcepts do                          ▷ all concepts in KB graph
 3:     for j ← 0, numsecondconcepts do                     ▷ all concepts in Query graph
 4:         foundmatch ← false
 5:         if (type(c_i) == type(c_j)) || (supertype(c_i) == type(c_j)) then
 6:             if (individ(c_i) == individ(c_j) || (individ(c_j) == ∅) then
 7:                 foundmatch ← true                       ▷ match all concepts in query graph
 8:             end if
 9:         end if
10:
11:     end for
12: end for
13: if foundmatch == true then
14:     for i ← 0, numfirstrelations do                     ▷ all relations in KB graph
15:         for j ← 0, numsecondrelations do                ▷ all relations in Query graph
16:             if (!relation[j].mapped) && (type(r_i) == type(r_j)) then
17:                 if match from r_j to match to each of its concepts then
18:                     relation[j].mapped = true           ▷ repeat line 2 for all
19:                 end if
20:             end if
21:
22:         end for
23:     end for
24:     foundmatch ← true
25:     for j ← 0, numsecondrelations do
26:         if !relation[j].mapped then
27:             foundmatch ← false
28:         end if
29:     end for
30: end if
31: if foundmatch == true then
32:     P ← build new subgraph projection
33:     return P                                            ▷ return new projection
34: else
35:     return ∅                                            ▷ no projection returned
36: end if
```

$M_c \mid\mid Q_c \mid + \mid Q_e \mid$) . In step 31: if a projection is found, it is returned. Therefore the leading step is line 13 for the over all running time, so the worst case bound for the whole KB is very close to the worst case bound given for Ullman's algorithm above: $O((\mid M_r \mid\mid Q_r \mid\mid M_c \mid\mid Q_c \mid\mid KB_c \mid + \mid Q_e \mid^{|Q_r|}))(\mid N \mid)$.

4 New Algorithm

After examining the above algorithms it was discovered that even though the running times were acceptable, the actual projection algorithms were not general.

That is, the user was confined by what parts of a valid conceptual graph could be present in the data or could only have one projection even if more than one was present. The desire to allow the user to use a directed, connected, bipartite conceptual graph (see Definition 3) that was cyclic for both the query and KB graphs prompted a new projection algorithm to be designed.

4.1 Supporting Information

In order to produce a new algorithm, new data structures and supporting routines were needed. Because in the KB the connection between the algorithm and data structures is critical, the new data structures and variables need to be designed around the actual algorithms.

Variables and Given values. Evaluating all the past projection algorithms, and looking at the data structures used for each knowledge base, the authors discovered that handling conceptual graphs as triples as oppose to vectors or linked lists makes the operation of projection much easier and cleaner to process. These authors are not the first researcher to think about using triples. Kabbaj and Moulin in 2001 [23] looked at CG operations using a bootstrapping step. It was at this time that they also looked at defining the join operation using triples as part of the matching data structure. However, they did not look at exploiting the triples in the actual algorithm of the operation.

All conceptual graphs in the KB and the query graph are stored not only with the general conceptual graph information, but also with a *C-R-C* list in a cs-triple format. Their definitions are given below:

- ***cs-triple*** is a 3-tuple, $T =< c_i, b, c_j >$, where c_i, c_j are concept nodes, and i and j are not equal. b is a conceptual relation (either a relation or actor node), and $(c_i, b) \in E$ and $(b, c_j) \in E$, and c_i and c_j are members in the signature of b.
- ***c-r-c list*** is a concept-relation-concept list that holds cs-triple information in which the 'b' in the 3-tuple is a relation node
- ***c-a-c list*** is a concept-actor-concept list that holds cs-triple information in which the 'b' in the 3-tuple is an actor node
- ***defining labels*** are all elements in a data structure hold a unique label; that includes concepts, relations, actors, and triples

During the performance of the projection operation two added data structures are used. One data structure holds the matching possibilities of the query concepts with the KB graph concepts, called the *match list*, and the second structure holds the matching triples from the KB graph for each concept in the query graph, called the *anchor list*. These data structures improve performance by making available preprocessed information at the time of the actual projection during which the projection graph or graphs are created.

Actual Supporting Routines. Because the conceptual information is the structural foundation of a conceptual graph and because the relationships between the concepts define the meaning of the graph, the new supporting routines

Algorithm 2. Projection

```
1: function NewProjection(Q, KB)                          ▷ Query and KB graphs
2:     P = ∅
3:     for each G ∈ KB do                                  ▷ All graphs in KB
4:         W ← A list from Q                               ▷ Preprocessing
5:         for each q_i ∈ W, where i = 1 to c(W) do
6:             if ((M ← MatchConcepts(q_i, G)) > ∅) then
7:                 for each n_j ∈ M, where j = 1 to M do
8:                     match = false
9:                     for each t_a ∈ Q do
10:                                ▷ where a = 1 to the # of cs-triples in crc list for q_i
11:                        for each s_b ∈ G do
12:                                ▷ where b = 1 to the # of cs-triples in crc list for n_j
13:                            if MatchTriple(t_a, s_b, true) == true then
14:                                add (n_j, (s_b, t_a)) to q_i ∈ W
15:                                match = true
16:                            end if
17:                        end for
18:                    end for
19:                    if match == false then
20:                        break out of loop and start next graph in KB
21:                    end if
22:                end for
23:            else
24:                break out of loop and start next graph in KB
25:            end if
26:        end for
27:        Pset = ∅                                        ▷ Projection processing
28:        for each q_i ∈ W, where i = 1 to c(W) do
29:            Pset = Projection(i, W, G, Pset)
30:        end for
31:        P ← P ∪ Pset
32:    end for
33:    return P                                  ▷ Set of projections from query onto KB
34: end function
```

have been defined around the *triple* relationship of the *C-R-C*. The routines for *MatchConcept, MatchHierarchy, MatchTriple, and MatchConcepts* are examples of the most important matching support routines.

4.2 Actual Algorithm

The overall algorithm (see Algorithm 2) for the projection of the query graph onto the KB is based on looking at all triples that are in the query graph and checking for a complete subgraph match of the query graph onto the KB graph during preprocessing. Because each triple in the query graph is unique, even if the node *type* is not, all projections can be found in the KB graph. Then after all matches of conceptual units and triples are found, the actual projection graphs

are built. However, because the temporary data structures are saved from the preprocessing, matching does not have to happen again at build time. The actual projection just uses the match list and anchor list already created to build up or create new the projection graphs. Because the anchor list contains all available projections, both injective and non-injective or homomorphism projections are found.

4.3 Execution Time

Now that the algorithm is split into two sections, there is a running time for answering the decision question of whether or not there is a projection, it will be called the *matching algorithm*, and a running time for the *actual projection*. For the new algorithms, three modifications have been made that affect the execution time of the projection operation: 1) all nodes and triples are uniquely labeled, 2) the edges are not labeled, but do have implied labeling through their directionality within the triples, and 3) the triples are not only part of the data structure of the KB, but also directly effect the actual projection algorithm. The *labeling* drives the execution time of the matching algorithm when doing an injective projection toward the running time for a subgraph 'labeled' isomorphism problem which can be solved in polynomial time as opposed to a straight subgraph isomorphism problem which is known to be NP-complete. The triples allow the matching algorithm to stop sooner when no projection is possible.

For the actual projection creation, the number of triples in the query graph drives the amount of time needed for the actual projection. The size of the graphs in the KB affects the base of the execution time, but the number of times the **Projection** function is executed is based on the number of triples in the query graph. In a typical query-answer scenario where the query graph would potentially contain normally two to four triples compared to possibly a thousand in the KB graph, this algorithm takes into account that the query graph is small. Because of that, the time to do thousands of graphs in a KB is only multiplied by a constant based on the maximum number of triples in a KB graph that the small query graph is projected onto. Therefore the execution time is only multiplicative in the number of projection available with this query graph. Since in the most common case there is only one projection, the actual projection creation algorithm becomes polynomial. Through this shift in problem class, the running time for the projection operation for a typical scenario within a query-answer application shows improvement.[2]

5 Comparison and Conclusion

Four different, yet related, projection algorithms have been described. Examining Table 1 comparisons between basic units, type of graphs, number of possible projections found, overall operation algorithm execution time and just actual projection creation execution time will be evaluated.

[2] The complete algorithms and running times were presented as part of the author's PhD dissertation [24].

Table 1. Comparison of four algorithms

	M&C	Croitoru	Notio	New Algorithm
basic unit	relations	relations	relations	concepts
works over	SCGs	SCGs	CGs	CGs
projections found	all	# relations	1	all
overall operation	NP-Complete	NP-Complete	NP-Hard	NP-Hard
actual projection	NP-Complete	NP-Complete	NP-Complete	NP

The Mugnier and Chein and Croitoru algorithms use SCGs, and Notio and the new algorithm work over full CGs. Looking back at the example shown when discussing the projection operation, Notio would only find one projection because it was only designed to look for a single projection graph. Croitoru's algorithm assumes that the total number of relations in the query graph equals the number of possible projections; therefore, with this example it will only find a single projection. However, if the KB graph has multiple projections to a single relation in the query graph; part of the projections would be missed.

It is not clear from the Mugnier and Chein 1992 paper if they can handle two concept pairs with the same relationship between them in a projection operation. However, from later work [25], it is indicated that the same relationship between different concepts can be found and multiple projections are possible between two CGs, but the execution time is at best NP-complete and only works on SCGs (no actors or directed graphs). Mugnier and Chein algorithm is also based on the *relations* found within the graph and must traverse all of their signatures to discover if there is a subgraph morphism. The new algorithm is based on the conceptual units, or *concepts*, within the graph and can stop searching as soon as there is no match for a concept or concept triple in the KB graph for one of query graph's conceptual unit.

Mugnier and Chein's algorithm does the whole projection operation as a single injective projection algorithm, where Croitoru, Notio and the new algorithm all use some form of preprocessing. Notio and the new algorithm have a complete separation between the preprocessing algorithm and projection; where, Croitoru uses the preprocessing algorithm inside of the actual projection, therefore, giving the same running time for both the overall algorithm and the actual projection. Notio does preprocessing at storage time that helps in constructing the projection. However, the actual projection algorithm after the preprocessing is still NP-Complete.

The new algorithm splits the overall projection algorithm into two parts, matching and projection construction. Then data structures are used between these two algorithms to use the structure of the graphs to help in the projection process. In the most common case the matching algorithm is the longest running part of the overall algorithm because the actual projection execution is polynomial. Therefore, in a typical scenario where the query graph is small, the new algorithm is not only able to find all projections for full conceptual graphs, but can use the data structures of the KB to do it faster. Future work is to determine if the actual projection algorithm for all injective projections can be performed in polynomial time by experimental results [24].

Acknowledgment

The authors would like to thank Dr. Desh Ranjan for reviewing the definitions and mathematical results that appear within this paper.

References

1. Lehmann, F. (ed.): Semantics Networks. Pergamon Press, Oxford, England (1992)
2. Schubert, L.: Extending the expressive power of semantic networks. Artifical Intelligence 7, 163–198 (1976)
3. Peirce, C.: Manuscripts on existential graphs. Peirce 4, 320–410 (1960)
4. Sowa, J.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Reading, MA (1984)
5. Sowa, J.: Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks/Cole (2000)
6. Chein, M., Mugnier, M.L.: Conceptual graphs: Fundamental notions. Revue d'Intelligence Artificielle 6(4), 365–406 (1992)
7. Baget, J.F., Mugnier, M.L.: Extensions of simple conceptual graphs: the complexity of rules and constraints. Journal of Artificial Intelligence Research (JAIR) 16, 425–465 (2002)
8. Croitoru, M., Compatangelo, E.: On conceptual graph projection. Technical Report AUCS/TR0403, University of Aberdeen, UK, Department of Computing Science (2004)
9. Corbett, D.: Reasoning and Unification over Conceptual Graphs. Kluwer Academic/Plenum Plublishers, New York (2003)
10. Paterson, M., Wegman, M.: Linear unification. J. Comput. Syst. Sci. 16, 158–167 (1978)
11. Mugnier, M.L., Chein, M.: Polynomial algorithms for projection and matching. In: Pfeiffer, H.D., Nagle, T.E. (eds.) Conceptual Structures: Theory and Implementation. LNCS, vol. 754, pp. 239–251. Springer, Heidelberg (1993)
12. Messmer, B., Bunke, H.: Efficient subgraph isomorphism detection: A decomposition approach. IEEE Transactions on Knowledge and Data Engineering 12, 307–323 (2000)
13. Ullman, J.: An algorithm for subgraph isomorphism. J. of the Assoc. for Computing Machinery 23(1), 31–42 (1976)
14. Harary, F.: Graph Theory. Addison-Wesley, Reading, MA (1969)
15. Willems, M.: Projection and unification for conceptual graphs. In: Ellis, G., Rich, W., Levinson, R., Sowa, J.F. (eds.) ICCS 1995. LNCS, vol. 954, pp. 278–282. Springer, Heidelberg (1995)
16. Delugach, H.: CharGer 3.3 - A Conceptual Graph Editor, University of Alabama in Huntsville, Alabama, USA (2004), `http://www.cs.uah.edu/delugach/CharGer`
17. Croitoru, M., Compatangelo, E.: A combinatorial approach to conceptual graph projection checking. In: Webb, G.I., Yu, X. (eds.) AI 2004. LNCS (LNAI), vol. 3339, Springer, Heidelberg (2004)
18. Southey, F., Linders, J.: NOTIO - a Java API for developing CG tools. In: Tepfenhart, W.M. (ed.) ICCS 1999. LNCS, vol. 1640, pp. 262–271. Springer, Heidelberg (1999)

19. Delugach, H.: CharGer: A graphical Conceptual Graph editor. In: Delugach, H.S., Stumme, G. (eds.) ICCS 2001. LNCS (LNAI), vol. 2120, Springer, Heidelberg (2001), [Online Access: July 2001] http://www.cs.nmsu.edu/hdp/CGTOOLS/proceedings/index.html
20. Benn, D., Corbett, D.: pCG: An implementation of the process mechanism and an extensible CG programming language. In: Delugach, H.S., Stumme, G. (eds.) ICCS 2001. LNCS (LNAI), vol. 2120, Springer, Heidelberg (2001), [Online Access: July 2001] http://www.cs.nmsu.edu/hdp/CGTOOLS/proceedings/index.html
21. Polovina, S., Hill, R.: Enhancing the initial requirements capture of multi-agent systems through conceptual graphs. In: Dau, F., Mugnier, M.-L., Stumme, G. (eds.) ICCS 2005. LNCS (LNAI), vol. 3596, pp. 439–452. Springer, Heidelberg (2005)
22. Southey, F.: Notio and Ossa. In: Delugach, H.S., Stumme, G. (eds.) ICCS 2001. LNCS (LNAI), vol. 2120, Springer, Heidelberg (2001), [Online Access: July 2001] http://www.cs.nmsu.edu/hdp/CGTOOLS/proceedings/index.html
23. Kabbaj, A., Moulin, B.: An algorithmic definition of cg operations based on a bootstrap step. In: Delugach, H.S., Stumme, G. (eds.) ICCS 2001. LNCS (LNAI), vol. 2120, Springer, Heidelberg (2001)
24. Pfeiffer, H.D.: The Effect of Data Structures Modifications On Algorithms for Reasoning Operations. PhD thesis, New Mexico State University, Las Cruces, NM (2007)
25. Mugnier, M.L., Leclere, M.: On querying simple conceptual graphs with negation. In: Data and Knowledge Engineering, DKE, Elsevier, Revised version of R.R. LIRMM 05-051 (2006)