May 10, 2023

This document is part of the paper "$\mathcal{ELKG}_{app}$: An Approach to Represent Multi-dimensional MK in the Web of Data". It presents the matching algorithm which is used in the paper.

1 Algorithm

Algorithm 1 Transform RDF Reification data into ELKG data

Input: Input file containing dataset in the RDF Reification format.
Output: Output file containing dataset in the form of pred:[sub, obj, uid1, uid2] and uid1:[C,t, t_s, t_e]

procedure CREATE A DICTIONARY CONTAINING A KEY AS PREDICATE AND OTHER RELATED INFORMATION AS VALUES
  Read RDF Reification dataset as $\mathcal{G}_{\mathcal{R}}$
  for triples $\in \mathcal{G}_{\mathcal{R}}$ **do** ▷ number varies according to MK. We consider that three dimensions of MK are present
    if predicate == rdf:subject **then**
      store object position as sub and subject position as uid1
    end if
    if predicate == rdf:object **then**
      store object position as obj
    end if
    if predicate == rdf:predicate **then**
      store object position as pred
    end if
    if the predicate != rdf:subject and predicate != rdf:object and predicate != rdf:predicate and predicate != source MK predicate **then**
      if the predicate == truth value MK predicate **then**
        store object position as C
      end if
      if the predicate != truth value MK predicate **then**
        if predicate == time MK predicate **then**
          store object position as t
        end if
        if predicate != time MK predicate **then**
          if predicate == time interval MK predicate **then**
            store the object position of triple as t_s and store object position of the next triple as t_e
          end if
          if predicate != time interval MK predicate **then**
            if predicate == source MK predicate **then**
              store object position as mobj and store predicate position as mpred
            end if
          end if
        end if
      end if
    end if
    Store triples in DATA ← *predicate_dictionary* and *statement_id dictionary*. *predicate_dictionary* as {pred:[sub, obj, uid1, uid2], mpred:[uid1, mobj, uid3, uid4]}, *statement_id dictionary* as uid1:[$\overline{C}$,t, t_s, t_e] ▷ uid2 and uid4 values should be null if there is no nested MK
  end for
  Call Creation of Compressed file with *predicate_dictionary* and *statement_id dictionary*
end procedure

Algorithm 2 Creation of the compression_file, of the decompression_file, and of the dictionaries

Input: Input file DATA containing pred:[sub, obj, uid1, uid2], uid1:[mkv1, mkv2, mkv31, mkv32]
Output: Compressed and Decompressed dictionary files and distinct predicate (pred) dictionary files.

1: **procedure** CREATE THE *COMPRESSION_FILE*, *THE DECOMPRESSION_FILE* AND THE *CREATE_DICTS* FILES
2: Initialization: a variable (counter) to 90000. Create two dictionaries; One for the compression file, and another for the decompression file
3: Read DATA
4: **for** each pred in DATA **do**
5: Check
6: **if** sub is new **then**
7: Replace sub with counter value and add sub:counter in compression_file dict.
8: Add counter:sub in decompression_file dict.
9: **end if**
10: **if** pred is new **then**
11: pred:pred in compression_file dict
12: Add pred:pred in decompression_file dict.
13: **end if**
14: **if** obj is new **then**
15: Replace obj with counter value and add obj:counter in compression_file dict.
16: Add counter:obj in decompression_file dict.
17: **end if**
18: **if** uid1 is new **then**
19: Replace uid1 with counter value and add uid1:counter in compression_file dict.
20: Add counter:uid1 in decompression_file dict.
21: **end if**
22: **if** uid2 is new **then**
23: Replace uid2 with counter value and add uid2:counter in compression_file dict.
24: Add counter:uid2 in decompression_file dict.
25: **end if**
26: Append to sub, obj, uid1, mkv1, mkv2, mkv31, mkv32, param4 and uid2 dictionaries.
27: **end for**
28: Write compression_file
29: Write decompression_file.
30: Write the pred dictionaries with the compression value to a unique file dictionary _<pred> along with uid1 and uid2 in the form of pred:[sub, obj, uid1, mkv1, mkv2, mkv31, mkv32, uid2].
31: **end procedure**

Algorithm 3 Query processing

Input: compression_file, decompression_file and pred-dictionaries files
Output: Result as per query, stored in comma-separated values format.

1: **procedure** PROCESS *QUERIES*
2: Display choice selection MENU:
3: **for** Insertion **do**
4: Read query pattern.
5: Create pred dictionaries for the given query pattern.
6: Open dictionary file corresponding to the entered pred.
7: **if** File exist **then**
8: Append query pattern into the file
9: **else** Create a new dictionary file for pred and append into it.
10: **end if**
11: **end for**
12: **for** Search **do**
13: Read query.
14: Compress query by loading compression_file.
15: Replace given sub, obj with their compression values.
16: Sort queries:
17: **for** each pred in query **do**
18: Read dictionary_ $< pred >$
19: Assign aweight to pred by the length of dictionary_$< pred >$
20: **end for**
21: Call bubble_sort() with weights.
22: After sorting Call Process_query()
23: **end for**
24: **for** ASK query **do**
25: All the steps shown in the search are the same.
26: Except here no need to decompress the result dictionary.
27: Check result dictionary
28: **if** non-empty - Display YES **then**
29: **else** empty - Display NO
30: **end if**
31: **end for**
32: **end procedure**

Algorithm 4 Process_query

```
Input: Dataset and query read by Algorithm 3
Output: The result of query.
procedure EXECUTE QUERIES
    Display choice selection MENU:
    for each query pattern in query do
        call matching_function(query pattern)
    end for
    Preparing display:
    for each variable associated with SELECT line do
        if variable is in result dictionary then
            Check yes or no
            if yes then
                copy result dictionary for the query pattern.
            end if
            if no then
                skip.
            end if
        end if
    end for
    Return the result to search() function.
    Read decompression_file for decompressing result dictionary.
    for each parameter in result dictionary do
        Replace parameters with their decompression value.
        Display result.
    end for
end procedure
```

Algorithm 5 matching_function

```
Input: line send by Algorithm 4 .
Output: Result send back to Algorithm 4 .
procedure EXECUTE QUERY PATTERN CONTAINING QUERY
    for each query pattern do
        if sub is variable, obj is variable then.
            if sub variable is new then
                Append the contents of sub_dict of result dictionary with that of sub_dict of current dictionary. And Add variable to list in vars['sub']
            end if
            if sub is variable and is previously encountered then
                Find the position of previous encounter and select the associated_dictionary.
                for each key in associated_dictionary do
                    if key exists in current dictionary's sub_dict then
                        Make union of current dictionary's sub_dict[key] and result dictionary's associated_dictionary[key].
                        Store it as value of associated_dictionary[key].
                    end if
                    if key not found in sub_dict of current dictionary then
                        Delete the key from associated_dictionary of result dictionary.
                    end if
                end for
            end if
            if obj variable is new then
                Append the contents of obj_dict of result dictionary with that of obj_dict of current dictionary. Add variable to list in vars['obj']
            end if
            if obj is variable and is previously encountered then
                Find the position of previous encounter and select the associated_dictionary.
                for each key in associated_dictionary do
                    if key exists in current dictionary's obj_dict then
                        Make union of current dictionary's obj_dict[key] and result dictionary's associated_dictionary[key].
                        Store it as value of associated_dictionary[key].
                    end if
                    if key not found in obj_dict of current dictionary then
                        Delete the key from associated_dictionary of result dictionary.
                    end if
                end for
            end if
        end if
```

```
35:         if sub is given, obj is variable then
36:             Keep the matching given sub in result dictionary's sub_dict.
37:             for each non_matching key in sub_dict do
38:                 Remove sub from obj_dict of result dictionary.
39:                 Remove obj from uid_dict of result dictionary.
40:             end for
41:             Remove the non_matching keys.
42:             if obj variable is new then
43:                 Append the contents of obj_dict of result dictionary with that of obj_dict of
    current dictionary and add variable to list in vars['obj']
44:             end if
45:             if obj is variable and is previously encountered then
46:                 Find the position of previous encounter and select the associated_dictionary.
47:                 for each key in associated_dictionary do
48:                     if key exists in current dictionary's obj_dict then
49:                         Make union of current dictionary's obj_dict[key] and result dictionary's
    associated_dictionary[key].
50:                         Store it as value of associated_dictionary[key].
51:                     end if
52:                     if key not found in obj_dict of current dictionary then
53:                         Delete the key from associated_dictionary of result dictionary.
54:                     end if
55:                 end for
56:             end if
57:         end if
58:         if sub is variable, obj is given then
59:             Keep the matching given obj in result dictionary's sub-dict.
60:             for each non-matching key in obj-dict do
61:                 Remove sub from sub-dict of result dictionary.
62:                 Remove obj from uid-dict of result dictionary.
63:             end for
64:             Remove the non-matching obj keys.
65:             if sub variable is new then
66:                 Append the contents of sub-dict of result dictionary with that of sub-dict of
    current dictionary.
67:                 Add variable to list in vars['sub']
68:             end if
69:             if sub is variable and is previously encountered then
70:                 Find the position of previous encounter.
71:                 Previously at sub, obj or uid position, select the associated_dictionary.
72:                 for each key in associated_dictionary do
73:                     if key exists in current table's sub-dict then
74:                         Make union of current dictionary's sub-dict[key] and result table's associ-
    ated_dictionary[key].
75:                         Store it as value of associated_dictionary[key].
76:                     end if
77:                     if key not found in sub-dict of current dictionary then
78:                         Delete the key from associated_dictionary of result dictionary.
79:                     end if
80:                 end for
81:             end if
82:         end if
83:         if sub is given, obj is given then
84:             Keep the matching given obj in result dictionary's sub-dict.
85:             for each non-matching key in obj-dict do
86:                 Remove sub from sub-dict of result dictionary.
87:                 Remove obj from uid-dict of result dictionary.
88:             end for
89:             Remove the non-matching keys.
90:             Keep the matching given sub in result dictionary's sub-dict.
91:             for each non-matching key in sub-dict do
92:                 Remove sub from obj-dict of result dictionary.
93:                 Remove obj from uid-dict of result dictionary.
94:             end for
95:             Remove the non-matching keys.
96:         end if
97:         Start UID1 processing.
98:         if UID is variable in user query then check:
99:             if UID variable is new then
100:                Append the contents of uid-dict of result dictionary with that of uid-dict of
    current dictionary.
101:                Add variable to list in vars['uid']
102:            end if
103:            if UID is variable and is previously encountered then
104:                Find the position of previous encounter.
105:                Previously at sub, obj or uid position, select the associated_dictionary.
106:                for each key in associated_dictionary do
107:                    if key exists in current table's uid-dict then
108:                        Make union of current dictionary's uid-dict[key] and result table's asso-
    ciated_dictionary[key].
109:                        Store it as value of associated_dictionary[key].
110:                    end if
111:                    if key not found in uid-dict of current dictionary then
112:                        Delete the key from associated_dictionary of result dictionary.
113:                    end if
114:                end for
115:            end if
116:        end if
```

```
117:         if UID1 is given then
118:             Keep the matching given UID in result dictionary's uid-dict.
119:             for each non-matching key in uid-dict do
120:                 Remove sub from sub-dict of result dictionary.
121:                 Remove obj from obj-dict of result dictionary.
122:             end for
123:             Remove the non-matching keys.
124:         end if
125:         Predicate parameters processing.
126:         for each parameter do
127:             if parameter if variable then
128:                 Check if parameter is new OR Check if parameter matches with previous parameter variables and match
129:             end if
130:             if parameter is given then
131:                 Match given parameter accordingly with result table.
132:             end if
133:             if parameter is not required then
134:                 No processing required.
135:             end if
136:         end for
137:         Start UID2 processing.
138:         if UID2 is variable then
139:             Check if UID2 is new OR
140:             Check if UID2 matches with previous variables and match accordingly
141:         end if
142:         if UID2 is given then
143:             Match given UID2 with result table
144:         end if
145:     end for
146: end procedure
```