GraphMatching : An Unification of GraphGrep and VF2 Algorithms

Outline

1.1 / 1.2 - Graphs SubIsomorphism (NP-complete problem)

GRAPH ISOMORPHISM

Two graphs are isomorphic if there is a one-to-one correspondence between their vertexes and there is an edge betweeen two vertexes of one graph if and only if there is and edge between the two corresponding vertexes in the other graph.

SUBGRAPH ISOMORPHISM

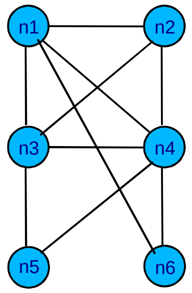
Like above but one graph is the subgraph of another graph.

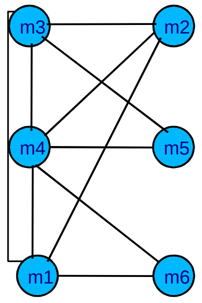
GRAPHS SUBISOMORPHISM or **MONOMORPHISM**

Finding occurrences of a graph in another graph.

1.2 / 1.2 - Graphs SubIsomorphism (NP-complete problem)

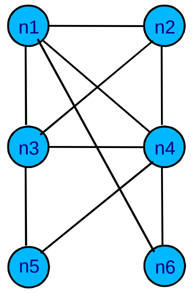
Graph isomorphism

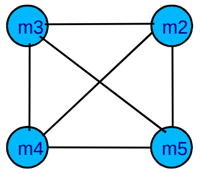




MAPPING
(n1,m1)
(n2,m2)
(n3,m3)
(n4,m4)
(n5,m5)
(n6,m6)

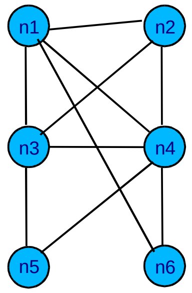
Subgraph isomorphism

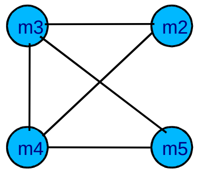




MAPPING
(n1,m3)
(n2,m2)
(n3,m4)
(n4,m5)

Monomorphism





MAPPING
(n1,m3)
(n2,m2)
(n3,m4)
(n4,m5)

n1,...,n6,m1,..,m6 not labels. No semantic attributes in general definition

2.1 / 2.9 - GraphGrep

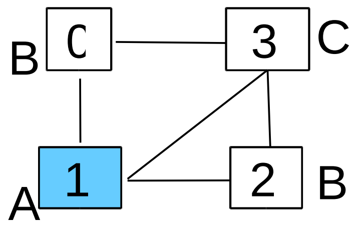
- Nodes with *label-node*
- Edges are undirected and unlabeled

GraphGrep Algorithm

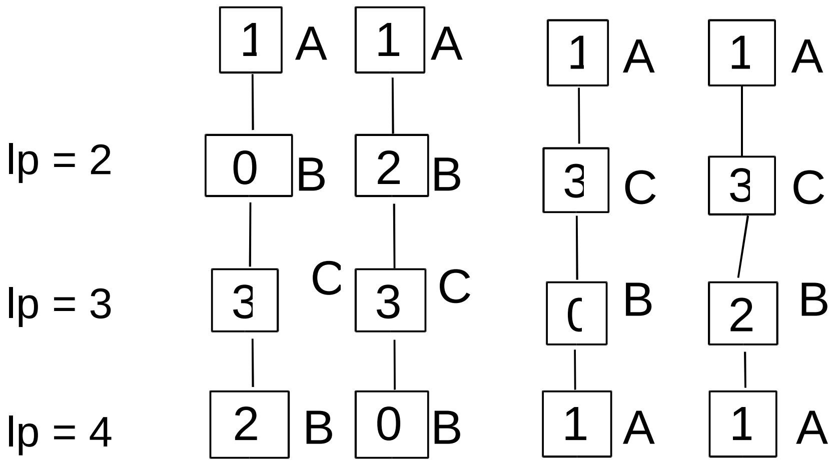
```
INPUT : Database of graphs, querygraph

➲ [PREPROCESSING] - Build the Database to
  represent the graphs as a sets of paths. (just
  once)

➲ 1 - Filter the Database based on the submitted
  query to reduce the search space

➲ 2 - Perform exact matching
```

2.2 / 2.9 - Sets of Paths

For each graph and for each node, find all paths that start at this node and have length one up to a constant value l_p



lp = 4



A={(1)}

AB={(1, 0), (1,2)}

AC ={(1, 3)}

ABC={(1,0,3), (1,2,3)}

ACB={(1, 3, 0), (1,3,2)}

ABCA={(1 ,0 ,3 ,1),(1, 2, 3, 1)}

ABCB ={(1 ,2,3 ,0),(1, 0, 3, 2)}

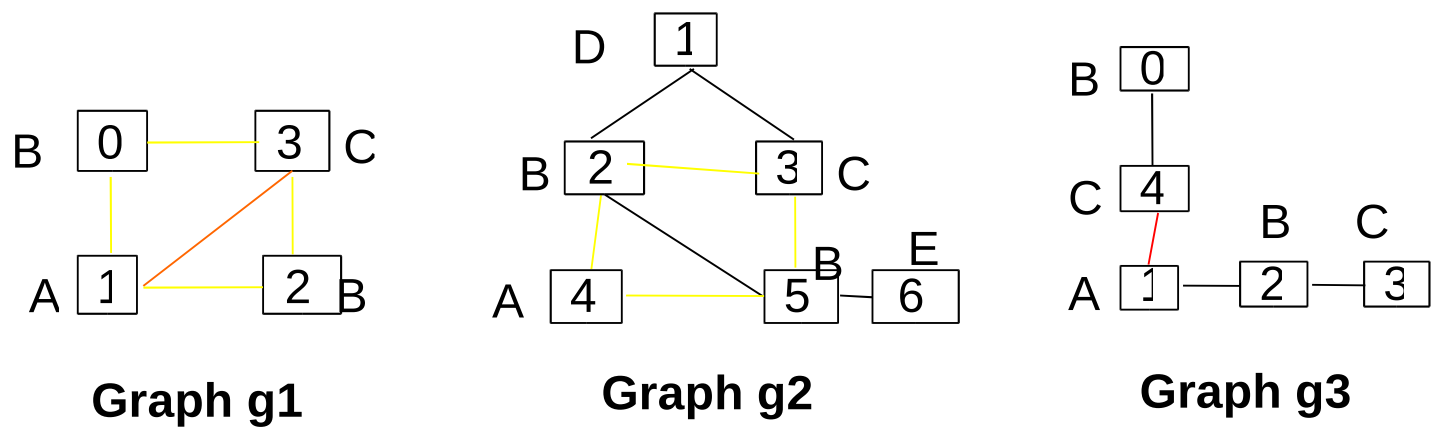
B={(0),(2)}

BA={(0,1),(2,1)}

BC={(0,3), (2, 3)}

………..

2.3 / 2.9 - Graphs Fingerprint

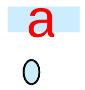


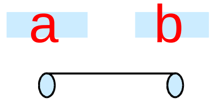
Key	g_1	g_2	g_3
h(CA)	1	0	1
……			
h(ABCB)	2	2	0

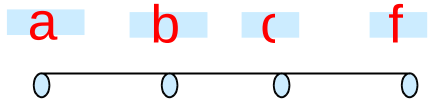
The keys of the hash table are the hash values of the label paths. Each row contains the number of id-paths associated with a key (hash value) in each graph.

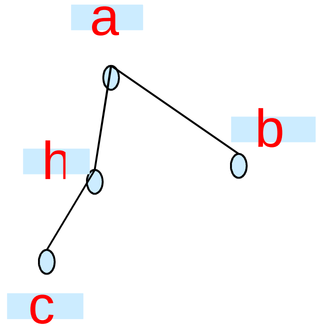
2.4 / 2.9 - Glide: graph query language

We need an interface to represent graphs.
Each node is presented only once.
It can be seen as a linear representation of a tree generated in a DFS



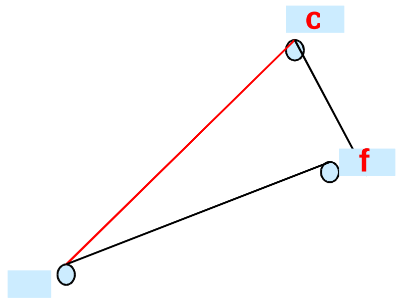






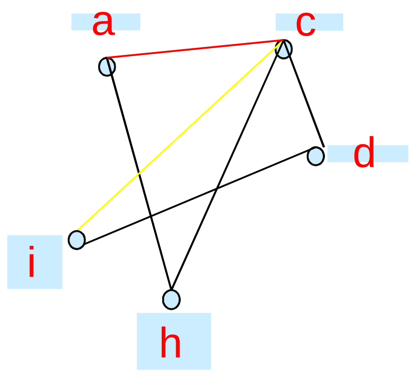
Node	a/
Edge	a/b/
Path	a/b/c/f/
Branches	a/(h/c/)b/

2.5 / 2.9 - Glide: graph query language



Cycle

c%1/f/i%1/



Cycles

a%1/h/c%1%2/d/i%2/

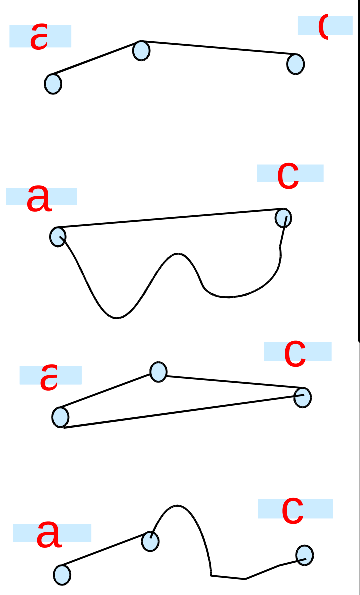
wildcards

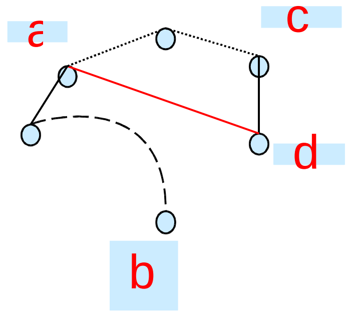
1) . a/./c

2) * a/*/c/

3) ? a/?/c

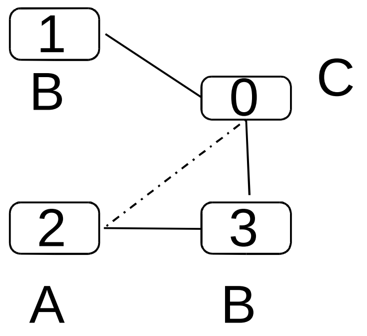
4) + a/+/c



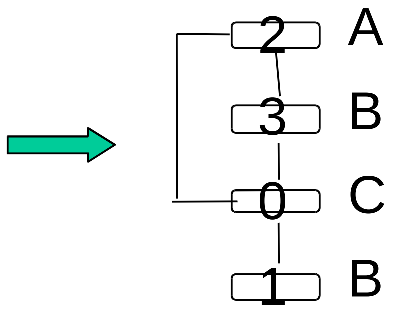


a%1/(./*/b/)./
c/d%1/

2.6 / 2.9 - Parsing a query graph



A%1/B/C%1/B/



lp = 4 →

$A^*B\underline{C}A^*$
$\underline{C}B$

lp = 3 →

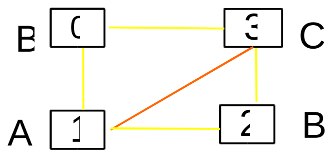
$A^*\ B\underline{C}$,
$\underline{C}B$
$\underline{C}A^*$

Use small components of the *query graph* and of the *database graphs* to filter the database and to do the matching

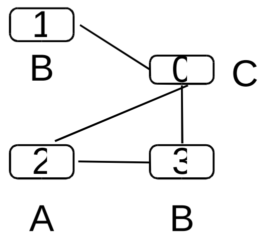
2.7 / 2.9 - Filtering

Key	Query
h(CA)	1
……	
h(ABCB)	1

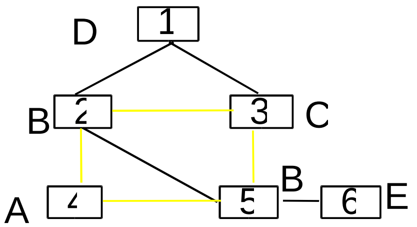
Key	g_1	g_2	g_3
h(CA)	1	0	1
……			
h(ABCB)	2	2	0



Graph g1

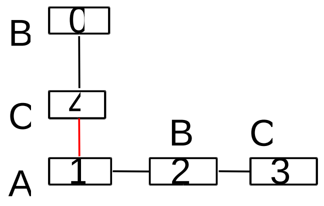


Query



Discarded

Graph g2

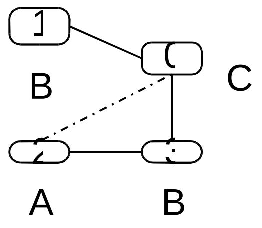


Discarded

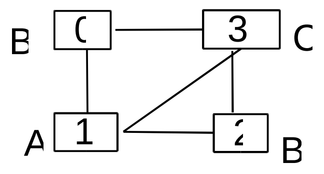
Graph g3

2.8 / 2.9 - Subgraph Matching

A*BCA*
CB



Query



Graph g1

Select the set of paths in g1 matching
the patterns of the query

ABCA = {(1, 0, 3, 1),(1, 2, 3, 1)}

CB = {(3,0),(3,2)}

Combine any list from ABCA with any list of CB accordingly '*' and '_'

ABCACB = {((1, 0, 3, 1),(3, 0)),

((1, 0, 3, 1),(3, 2)),((1, 2, 3, 1),(3, 0)),

((1, 2, 3, 1),(3, 2))}

Remove lists if they contains equal nodes in the positions not involved above

ABCACB ={removed,

((1, 0, 3, 1),(3, 2)),((1, 2, 3, 1),(3, 0)),

removed}

2.9 / 2.9 - Complexity

Building the Database (preprocessing)

- Linear in the size of the DB
- Linear in the number of the nodes in the graphs
- Polynomial in the valence of the nodes
- Exponential in the value of lp (small constant!)

$O(\Sigma_i^{|D|} (n_i \, m_i^{lp}))$ with m the maximum valence (degree)

Subgraph Matching

- Linear in the size of the database
- Exponential in p x lp with p the number of query graph patterns
- No exponential dependency on the data graph size

$O(\Sigma_i^{|Df|} ((\underline{n}_i \, m_i^{lp})^p))$ with Df the size of DB after filtering and

$\underline{n}$ the maximum number of nodes having tha same label

MEMORY cost is $O(\Sigma_i^{|D|} (l_p n_i \, m_i^{lp}))$

3.1 / 3.8 - VF2 Graph Matching Algorithm

- Matching process is carried out by using a State Space Representation (SSR). A State represents a partial solution of the matching between 2 graphs and a transition between states corresponds to the addition of a new pair of matched nodes.
- A set of feasibility rules is introduced for pruning states corresponding to partial matching solutions not satisfying the required graph isomorphism

3.2 / 3.8 – SSR Approach

- Solutions to the matching problem could be obtained computing all the possible partial solutions and selecting the ones satisfying the wanted mapped type (Brute Force approach).
- In order to reduce the number of paths to be explored during the search, for each state on the path from s_0 to a goal state, we impose that the corresponding partial solution verifies some coherence conditions, depending on the desired mapping type. States which don't satisfy a feasibility rule can be discarded from further expansions.

3.3 / 3.8 - The Matching Algorithm

```
INPUT:
OUTPUT:
BEGIN

  REPEAT

    FOREACH        IN




       FOREACH      IN

       ENDFOR
    ENDFOR

    UNTIL            OR
END.
```

C(s) is the local valid mapping.
S(k) is the set of states computed at the k-th iteration, that is the states whose partial mapping invoves k nodes. At each iteration the algorithm determines all the coherent partial solutions that map k+1 nodes

3.4 / 3.8 - The feasibility rules

Let us call feasibility function the function **F** that express the f.r.
Note that **F** is a function of s and the pair (n,m).

$$Q(s) = \{(n,m) \in P(s) \mid F \text{ holds}\}$$

$F = F_{syn} \wedge F_{sem}$, F_{syn} guarantees the syntactic coherence
F_{sem} guarantees the semantic coherence

The feasibility rules must be simultaneously verified to allow the insertion of the considered pair.

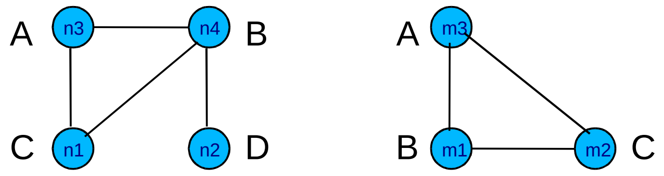
$$\mathbf{F_{syn}} = R_{coherence} \wedge R_{prun1} \wedge R_{prun2}$$

The semantic feasibility function F_{sem} is satisfied if the attributes of nodes and branches, corresponding in the found mapping, are equal

1 - iff for each node m' connected to m in the partial mapping, the corresponding node n' is connected to n
2 - iff the num of node connected to n that are in $T_1(s)$ is $\geq$ to the num of node connected to m that are in $T_2(s)$
3 - iff the num of node connected to n that are neither in $C_1(s)$ nor in $T_1(s)$ is $\geq$ to the num of node connected to m that are neither in $C_2(s)$ nor in $T_2(s)$

3.5 / 3.8 – Example coherence

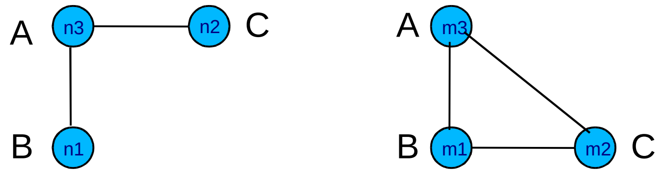
$C(s) = \{(n,m) \in N_1 x N_2$ |n is mapped onto m in the current partial solution}
$C_1(s) = \{n \in N_1 \mid \exists m \in N_2 \mid (n,m) \in C(s)\}$
$C_2(s) = \{n \in N_2 \mid \exists n \in N_1 \mid (n,m) \in C(s)\}$
$T_1(s) = \{n \in N_1 - C_1(s) \mid \exists n' \in C_1(s) \mid (n,n') \in B_1(s)\}$
$P(s) = T_1(s) \times T_2(s)$



State s:

$C(s) = \{(n_1,m_2),(n_3,m_3)\}$
$C_1(s) = \{n_1,n_3\}$ $C_2(s) = \{m_2,m_3\}$

$T_1(s) = \{n_4\}$ $T_2(s) = \{m_1\}$
$P(s) = \{(n_4,m_1)\}$
$Q(s) = \{(n_4,m_1)\}$

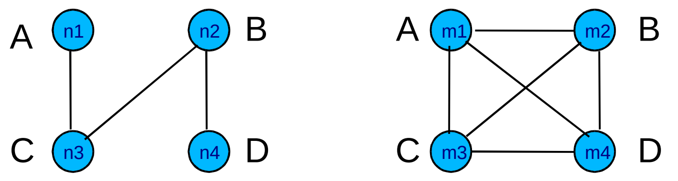


State s:

$C(s) = \{(n_1,m_1),(n_3,m_3)\}$
$C_1(s) = \{n_1,n_3\}$ $C_2(s) = \{m_1,m_3\}$

$T_1(s) = \{n_2\}$ $T_2(s) = \{m_2\}$
$P(s) = \{(n_2,m_2)\}$
$Q(s) = \emptyset$

3.6 / 3.8 – Example pruning



$C(s) = \{n_1,m_1\}$

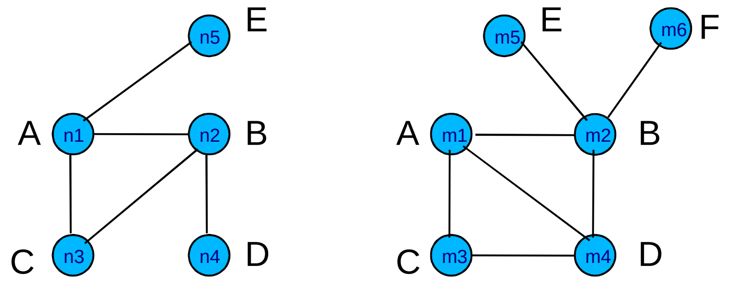
$C_1(s) = \{n_1\}$ $C_2(s) = \{m_1\}$

$T_1(s) = \{n_3\}$ $T_2(s) = \{m_4,m_2,m_3\}$

$P(s) = \{(n_3,m_4),(n_3,m_2),(n_3,m_3)\}$

RULE 2

(n_3,m_3) $0 \geq 2$ PRUNING.



$C(s) = \{n_1,m_1\}$

$C_1(s) = \{n_1\}$ $C_2(s) = \{m_1\}$

$T_1(s) = \{n_2,n_3,n_5\}$ $T_2(s) = \{m_4,m_2,m_3\}$

RULE 3

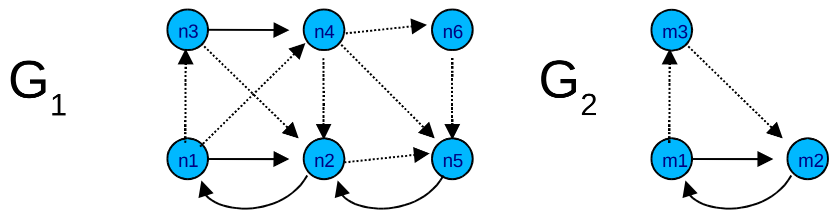
(n_2,m_2) $1 \geq 2$ PRUNING.

1 - iff for each node m' connected to m in the partial mapping, the corresponding node n' is connected to n

2 - iff the num of node connected to n that are in $T_1(s)$ is ≥to the num of node connected to m that are in $T_2(s)$

3 - iff the num of node connected to n that are neither in $C_1(s)$ nor in $T_1(s)$ is ≥to the num of node connected to m that are neither in $C_2(s)$ nor in $T_2(s)$

3.7 / 3.8 - Discussion



- Without the use of the feasibility rules : 228 states
- With R_{coh} : 40 states
- Using all the rules : 21 states

3.8 / 3.8 - Complexity

- Cost to verify if the new state satisfies the feasibilty rules.
- Cost to calculate sets T_1,T_2,etc
- Cost to generate P(s)

It is proven that cost for the exploration of a single state is $\Theta(N)$

Best case
In each state only one of the potential successors satisfies the feasibility rules(in the hypothesis that an isomorphism exists). So number of states is N and complexity is $\Theta(N^2)$. Spatial is $\Theta(N^2)$.

Worst case
Each state must be explored. It is proven that complexity is $\Theta(N!N)$. Spatial is $\Theta(N^2)$

4 - The need of a Benchmark

- We need to know the behavior or every algorithms on every kind of graphs, every combinations of nodes, labels, query, number of matches etc.

The following kinds of graphs have been considered:

1) *Randomly Graphs* with different values of the edge density μ
(where μ is the probability that an edge is present between two distinct nodes)
2) Regular Meshes with different dimensionality: 2D, 3D
3) Irregular Meshes with different dimensionality: 2D, 3D
(like regular with the addition of ρN random edges uniformly distributed)
4) Bounded Valence Graphs with different values of valence
(every node has a number of edges lower than valence)
5) Irregular Bounded Valence Graphs
(like regular but 10% of all edges are moved)
6) Scale Graphs with $\alpha,\beta,\gamma,\delta$ p,q

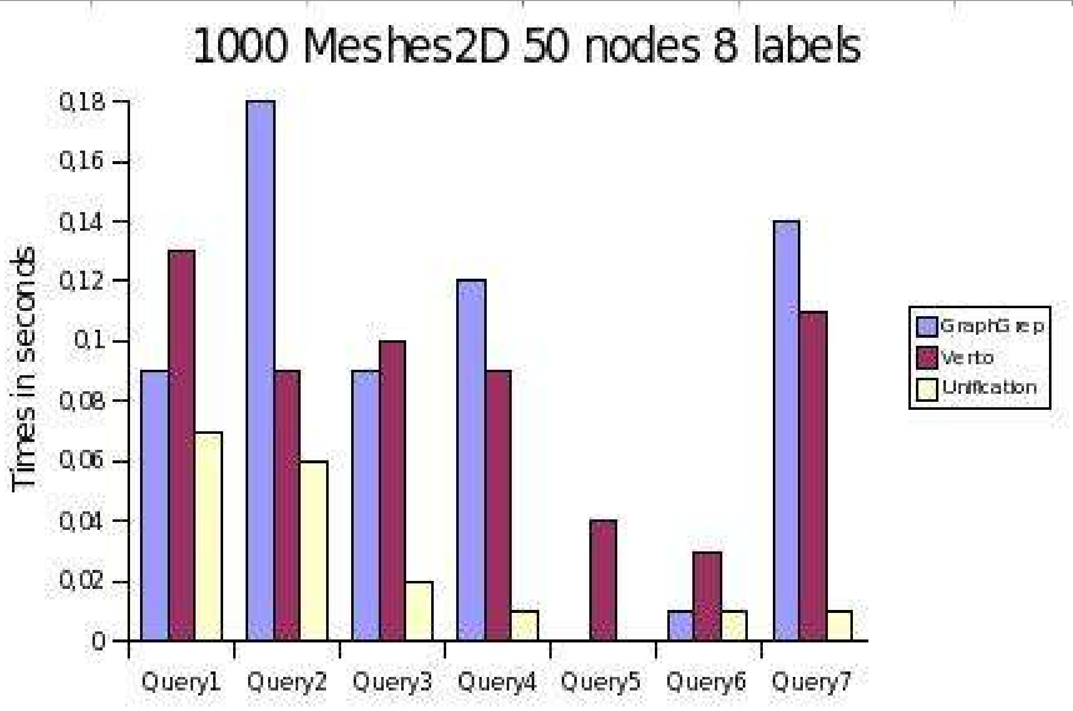
5.1 / 5.10 – Unification Method and Results

- When we have a lot of matches GraphGrep performs better than VF2 because worst case of VF2
- When we have a few of matches GraphGrep performs better because the pruning

***Graphgrep* + *VF2* =>** **Unification** (new algorithm):

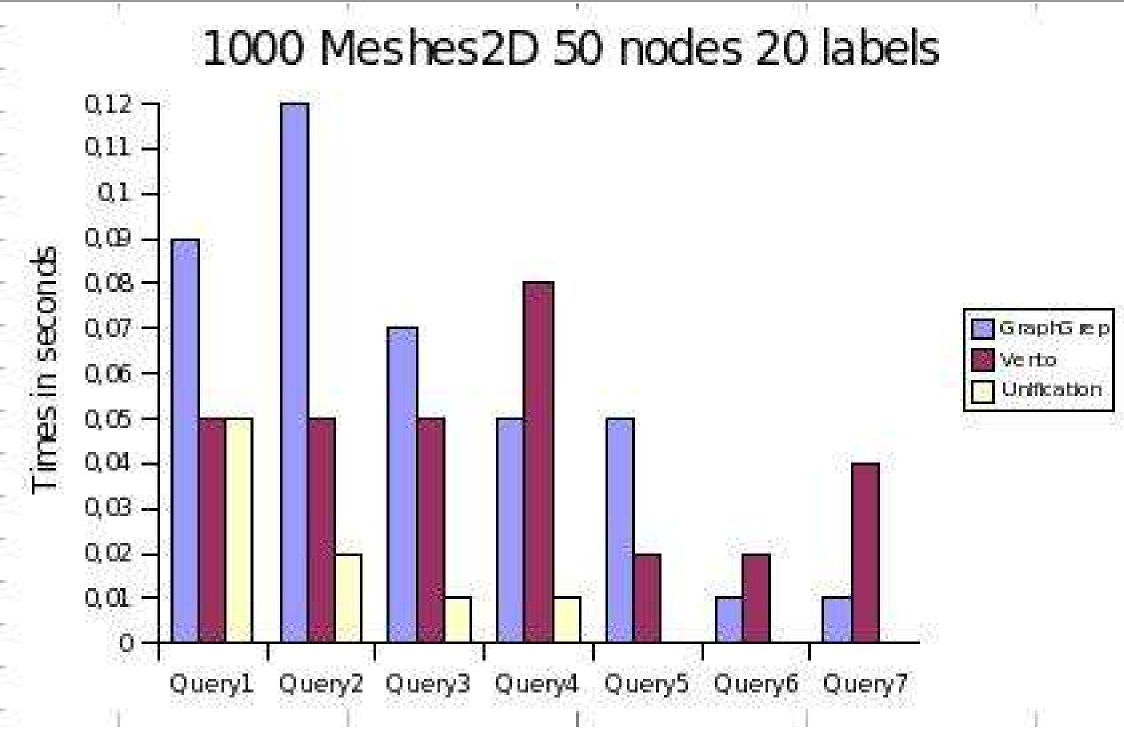
1) Use GraphGrep for pruning and apply VF2 to the pruned DataBase of graphs

5.2 / 5.10 - Unification Method and Results



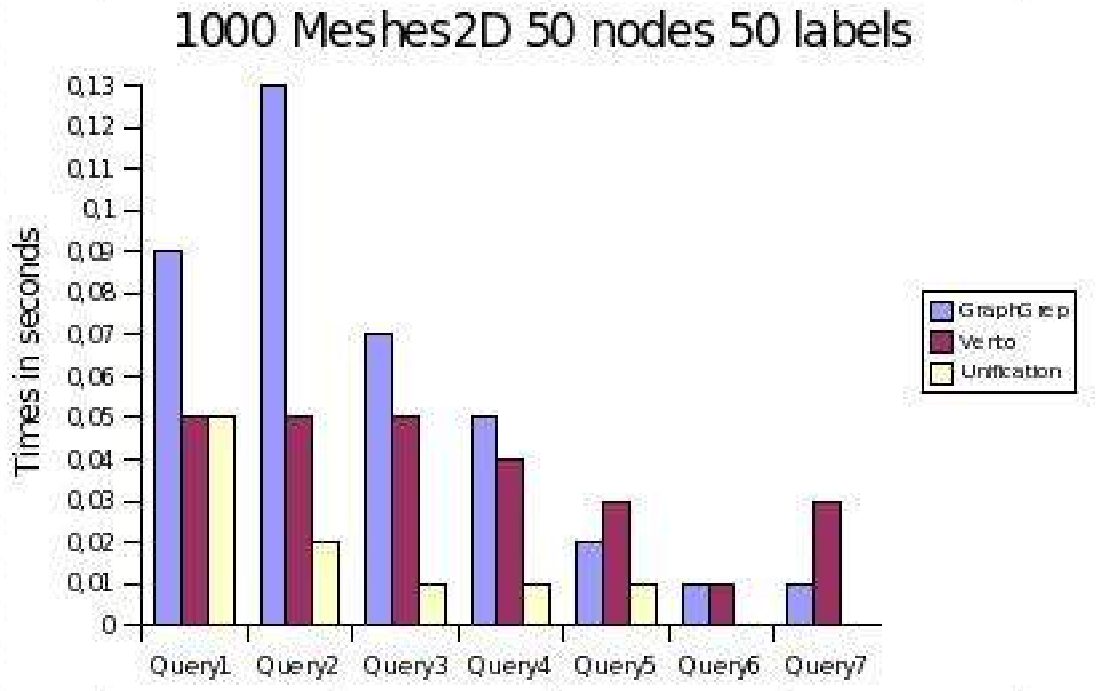
QueryName	Num Nodes	Num Edges	|DB| after filtering	#Matches
Query1	4	4	1000	4148
Query2	8	10	908	758
Query3	12	16	243	243
Query4	16	24	243	158
Query5	4	4	0	0
Query6	50	84	0	0
Query7	50	84	40	40

5.3 / 5.10 - Unification Method and Results



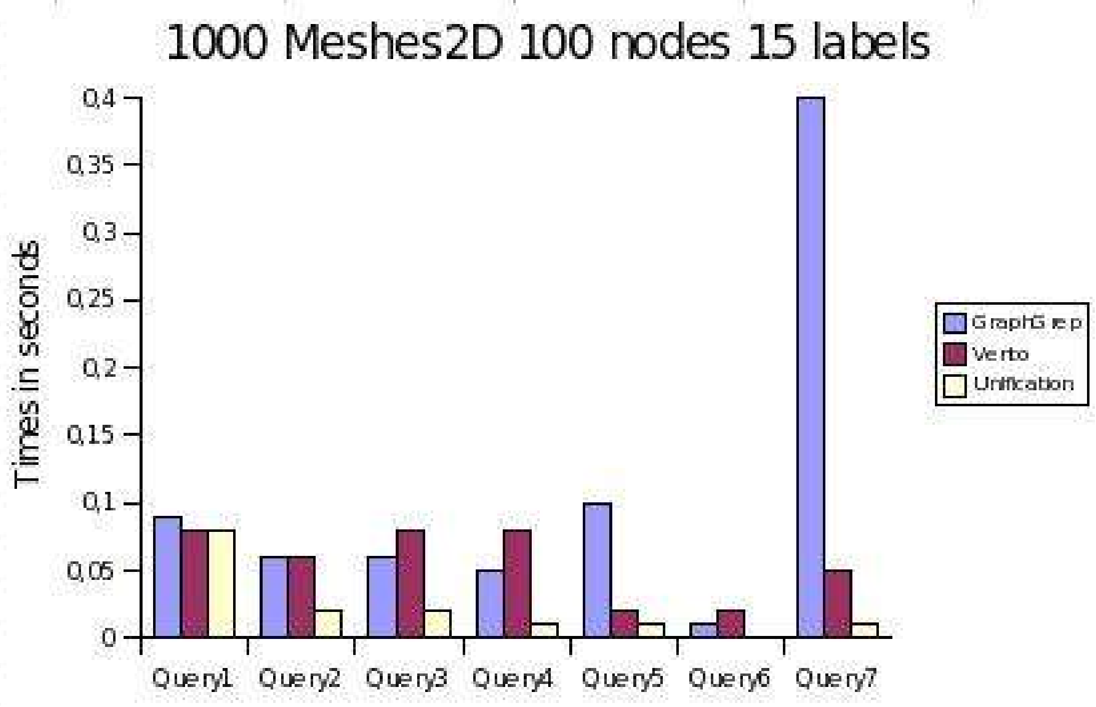
QueryName	Num Nodes	Num Edges	|DB| after filtering	#Matches
Query1	4	4	1000	2067
Query2	8	10	461	419
Query3	12	16	281	115
Query4	16	24	75	75
Query5	4	4	57	0
Query6	50	84	0	0
Query7	50	84	4	1

5.4 / 5.10 - Unification Method and Results



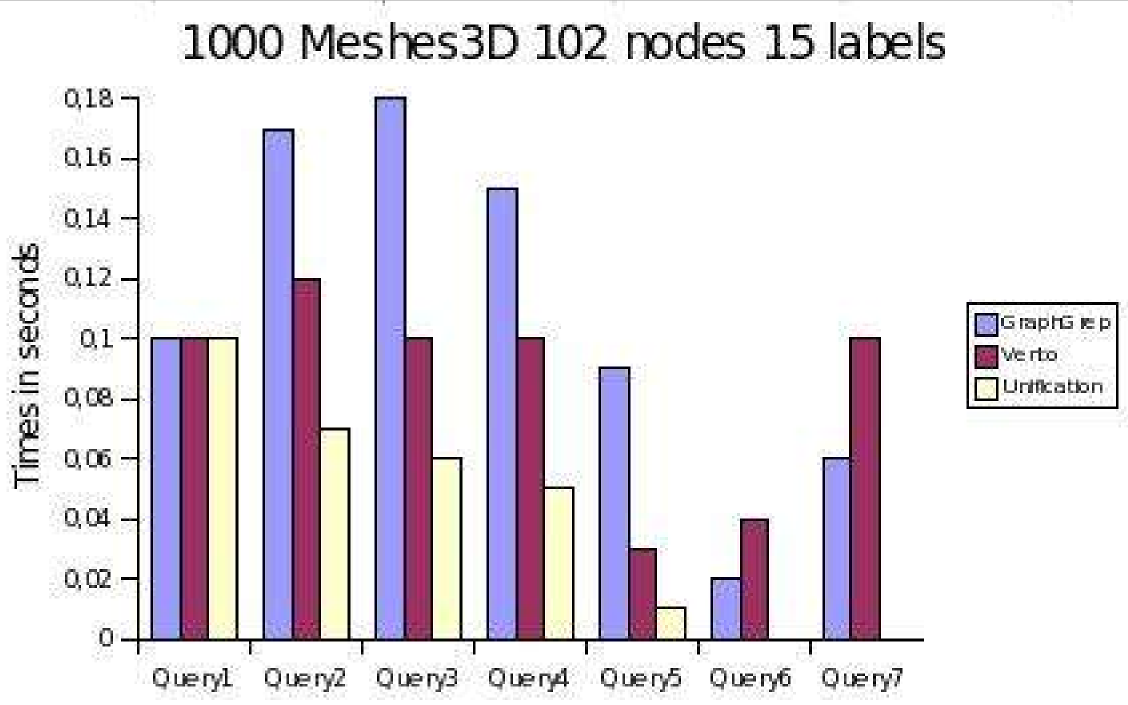
QueryName	Num Nodes	Num Edges	|DB| after filtering	#Matches
Query1	4	4	1000	2069
Query2	8	10	665	418
Query3	12	16	188	116
Query4	16	24	75	75
Query5	4	4	310	0
Query6	50	84	0	0
Query7	50	84	1	1

5.5 / 5.10 - Unification Method and Results



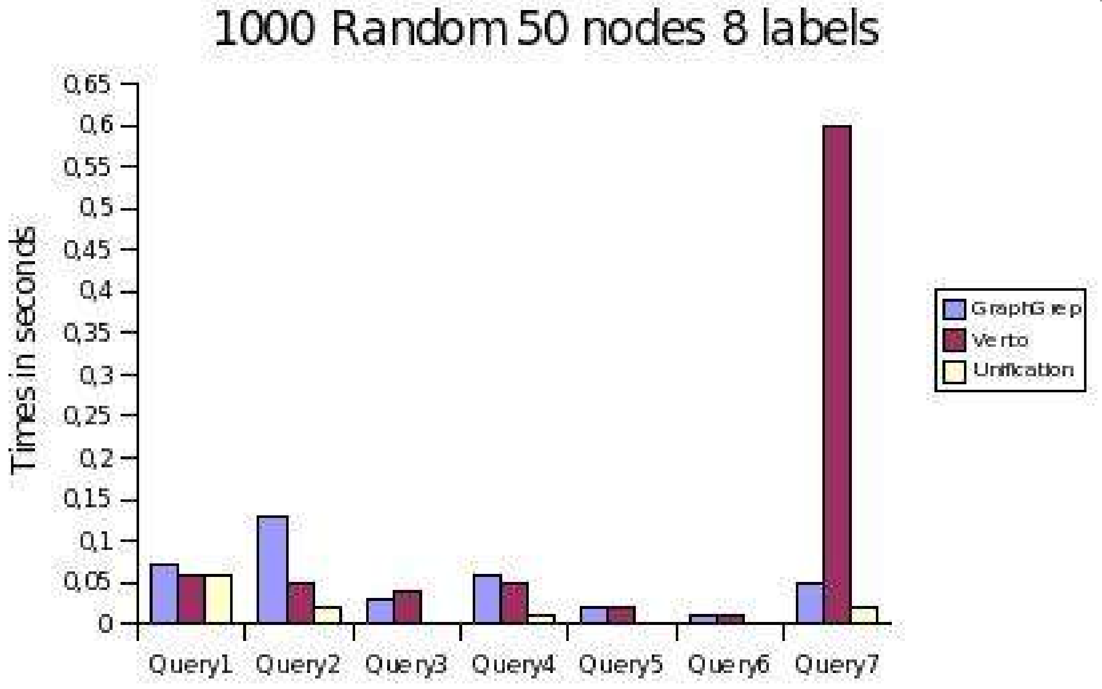
QueryName	Num Nodes	Num Edges	|DB| after filtering	#Matches
Query1	4	4	1000	3365
Query2	8	10	280	482
Query3	12	16	233	178
Query4	16	24	78	33
Query5	4	4	1000	0
Query6	100	178	0	0
Query7	100	178	47	47

5.6 / 5.10 - Unification Method and Results



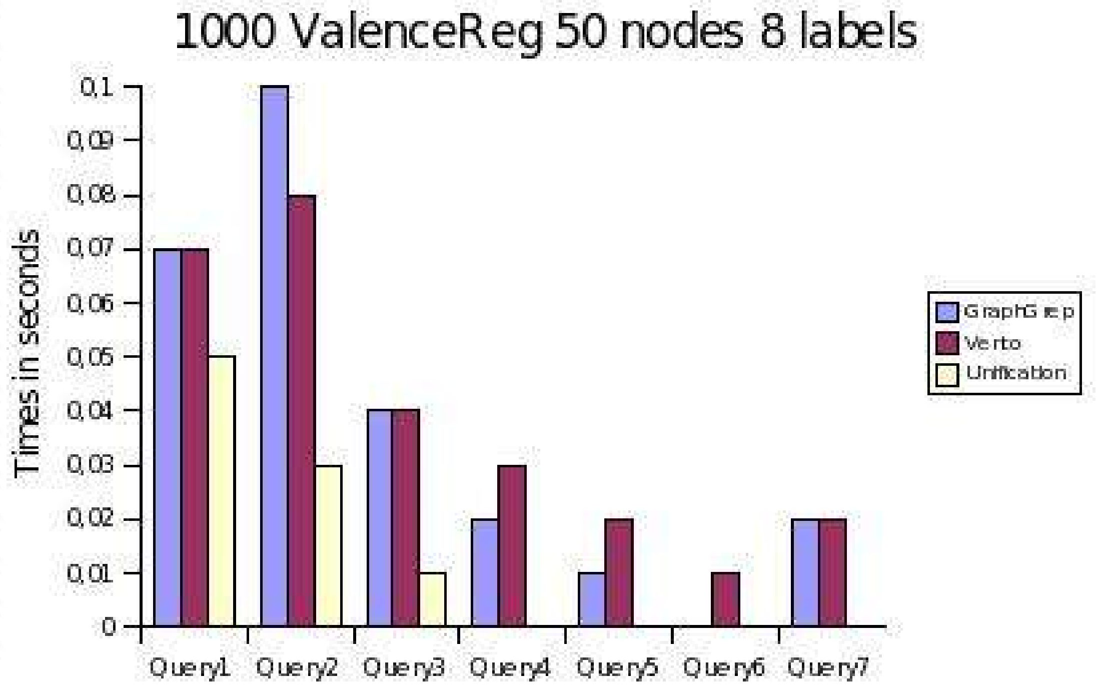
QueryName	Num Nodes	Num Edges	|DB| after filtering	#Matches
Query1	4	4	1000	3728
Query2	8	10	763	760
Query3	12	16	806	248
Query4	16	24	574	108
Query5	4	4	903	0
Query6	102	230	4	0
Query7	102	230	4	1

5.7 / 5.10 - Unification Method and Results



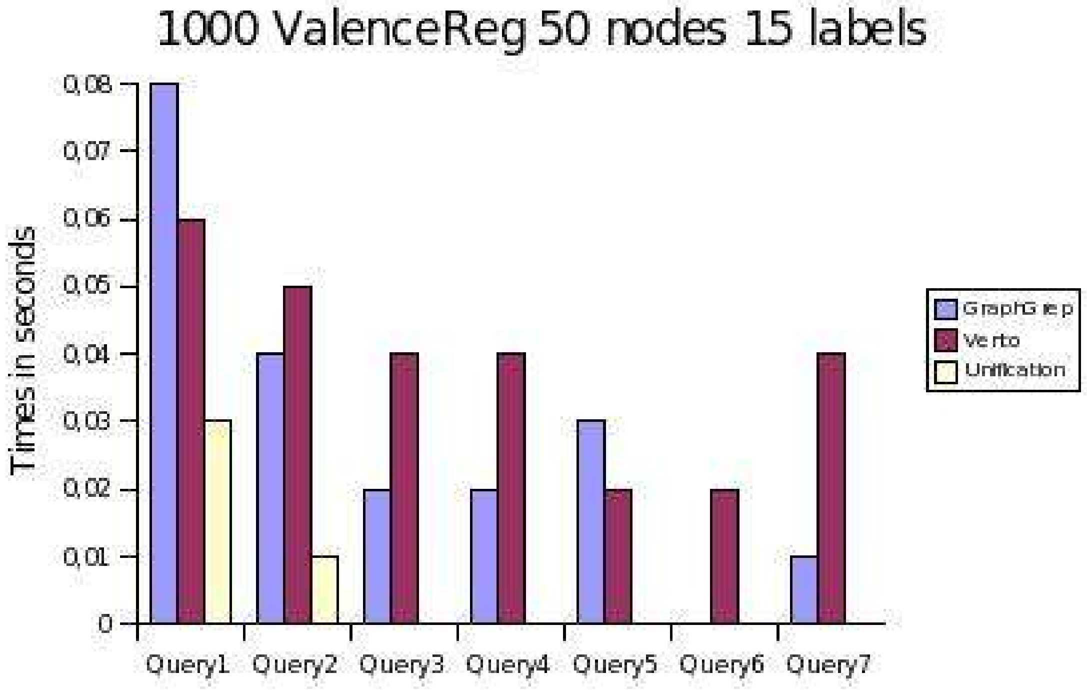
QueryName	Num Nodes	Num Edges	|DB| after filtering	#Matches
Query1	4	4	880	1760
Query2	8	10	664	245
Query3	12	16	175	66
Query4	16	24	94	66
Query5	4	4	310	0
Query6	50	122	0	0
Query7	50	122	2	1

5.8 / 5.10 - Unification Method and Results



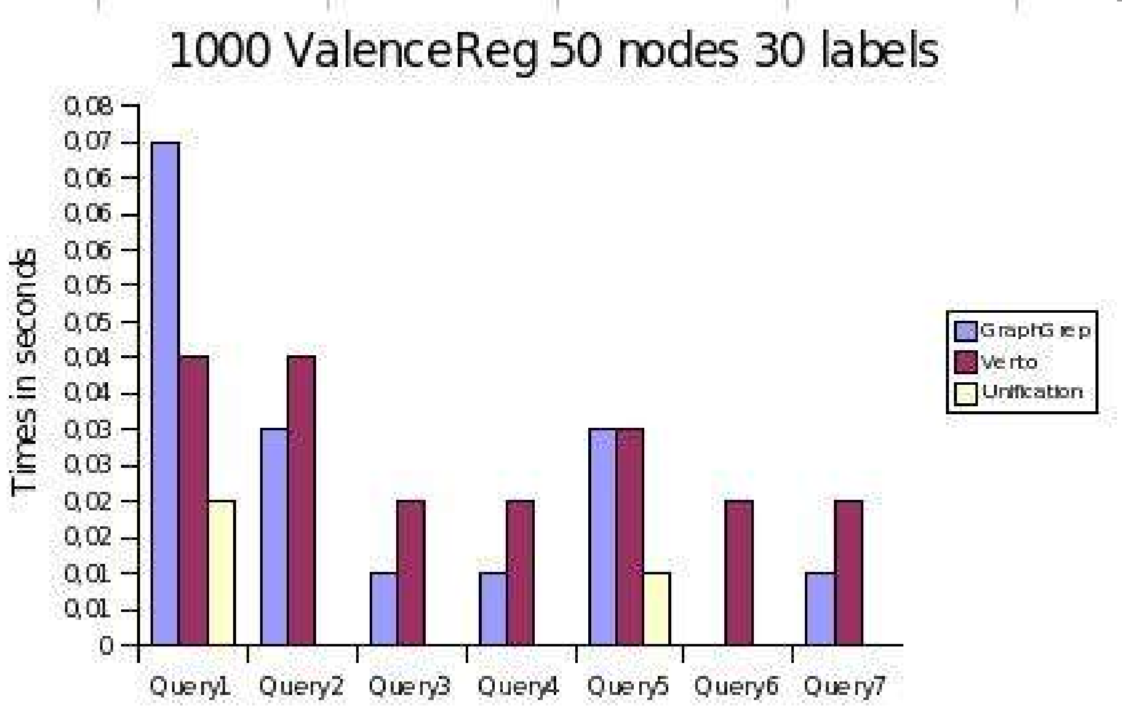
QueryName	Num Nodes	Num Edges	|DB| after filtering	#Matches
Query1	4	4	886	1772
Query2	8	10	548	270
Query3	12	16	127	90
Query4	16	24	73	55
Query5	4	4	51	0
Query6	50	70	0	0
Query7	50	70	1	1

5.9 / 5.10 - Unification Method and Results



QueryName	Num Nodes	Num Edges	|DB| after filtering	#Matches
Query1	4	4	841	762
Query2	8	10	135	88
Query3	12	16	71	21
Query4	16	24	75	21
Query5	4	4	230	0
Query6	50	74	0	0
Query7	50	74	1	1

5.10 / 5.10 - Unification Method and Results



QueryName	Num Nodes	Num Edges	|DB| after filtering	#Matches
Query1	4	4	622	476
Query2	8	10	129	24
Query3	12	16	13	3
Query4	16	24	2	2
Query5	4	4	239	0
Query6	50	73	0	0
Query7	50	73	1	1

6 - Conclusions and Future Work

- Extending GraphGrep for the inexact subgraph matching

- Extending GraphGrep for the others kind of mapping

- ...Implementing VF2 for undirected graphs

References

- D. Shasha, J.T-L Wang, R. Giugno, “*Algorithmics and Applications of Tree and Graph Searching*”,Proceeding of the ACM Symposium on Principles of Database Systems (PODS), Madison, Wisconsin, June 2002
- R. Giugno, D. Shasha, “*GraphGrep: A Fast and Universal Method for Querying Graphs*”, Proceeding of the IEEE International Conference in Pattern recognition (ICPR), Quebec, Canada, August 2002.
- L.P. Cordella, P. Foggia, C. Sansone,M. Vento,“*An efficient Algorithm for the inexact Matching of ARG Graphs Using a Contextual Transformation Model*”, Proc. of the 13th International Conference on Pattern Recognition, Wien, Austria, vol. III, IEEE Computer Society Press, pp. 180-184, 1996.
- L.P. Cordella, P. Foggia, C. Sansone, F. Tortorella, M. Vento, “*Graph Matching: A fast Algorithm and its Evaluation*”, Proc. of the 14th International Conference on Pattern Recognition, Brisbane, Australia, August, 16-20, pp. 1582-1584, 1998.
- L.P. Cordella, P. Foggia, C. Sansone,M. Vento,“*Performance Evaluation of the VF Graph Matching Algorithm*”, Proc. of the 10th International Conference on Image Analysis and Processing, IEEE Computer Soc. Press, Los Alamitos (California), pp. 1172-1177,1999.