GraphGrep for Inexact Matching using GLIDE query language

Outline

PART 1

- GLIDE syntax
- Parser
- Matching Process with wildcards
- Example
- Optimizations
- Conclusions

PART 2

- Results for Exact graph matching

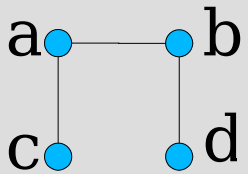
PART 1

Inexact Graph Matching with GraphGrep using GLIDE query language

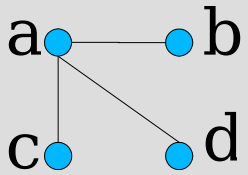
1/4. GLIDE syntax

- Main idea is to represent a graph and its branches in linear notation where each node is presented only once
- Nodes are represented using their labels and they are separated using slashes
- Branches are grouped using nested parentheses '(' and ')' anc cycles are broken by cutting an edge and labeling it with an integer anteceded by a %.

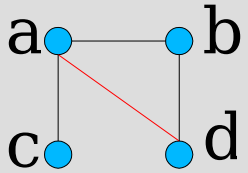
2/4. GLIDE syntax (example)



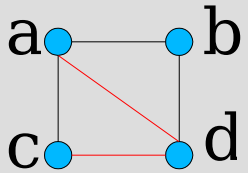
c/a/b/d/



a/(b/)(c/)(d/)



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



a%1/(c%2/)(b/d%1%2/)

3/4. GLIDE syntax - wildcards

Wilcards : “.”, “*”, “+”, “?”
Used to deal with inexact matching. Retrieved paths satisfies wildcards

“.” Matches any single node

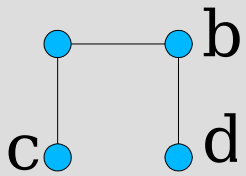
“*” Matches zero or more nodes

“+” Matches one or more nodes

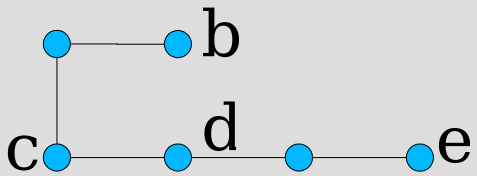
“?” Matches zero or one node

4/4. GLIDE syntax – wildcards (example)

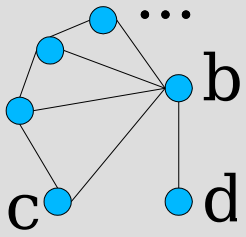
c/./b/d/



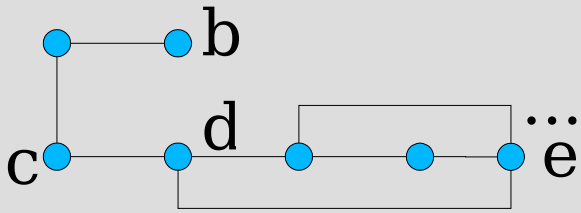
c/(./b/)(d/./e/)



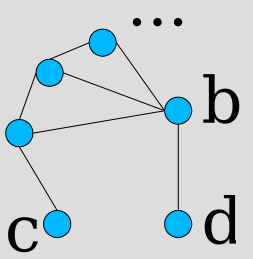
c/*/b/d/



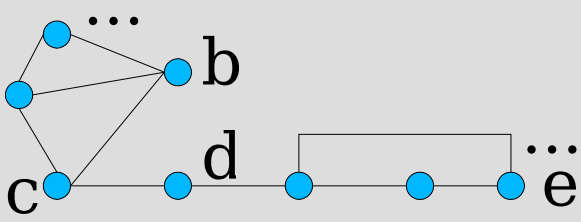
c/(./b/)(d/*/e/)



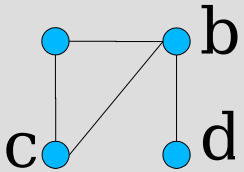
c/+/b/d/



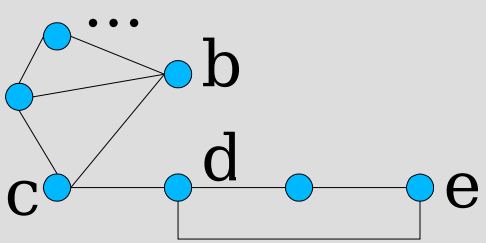
c/(*/b/)(d/+/e/)



c/?/b/d/



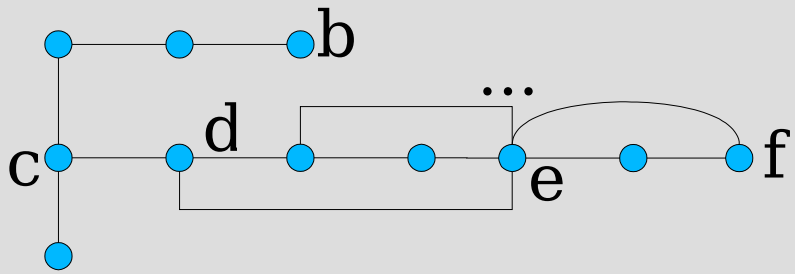
c/(*/b/)(d/?/e/)



Parser

- When a query in Glide is given, a parser computes all the wildcards and prepares a list of them for the next phase, the matching.

c/(././b/)(d/*/e/?/f)(./)



List of 4-uples (“id_node1”, “id_node2”, “value”, “operation”)

w[0] : 0 1 2 e
w[1] : 2 3 0 g
w[2] : 3 4 1 L
w[3] : 0-1 1 e

a/./././+/?/?/*/*/b/	[0 1 4 g]
a/*/*/*/?/?/b/	[0 1 0 g]
a/./*/*/?/b/	[0 1 2 g]
a/./+/+/?/?/?/b/	[0 1 3 g]
a/././?/b/	[0 1 3 L]

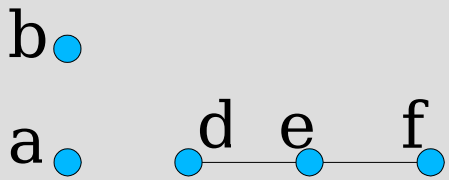
1/5. Matching Process

We perform Graph Matching of the query without wildcards.

Example

query: a/(./b/)(*/d/e/f/)(./)

then graph matching of the query below is performed



So we look for each graph in Database with the property upon.
(nodes and edges)

```
w[0] : 0 1 1 e
w[1] : 0 2 0 g
w[2] : 0-1 1 e
```

2/5. Matching Process

Output file of the simple matching process without caring of wildcards.

```
#qout1|GraphID|L
g0|2 4 5 0 1
g0|3 4 5 0 1
g0|6 4 5 0 1

#qout1|GraphID|L
g1|0 2 3 4 1
g1|5 2 3 4 1
```

3/5. Matching Process

Query : a/(./b/)(*/d/e/f/)(./)

w[0] : 0 1 1 e	a/./b/
w[1] : 0 2 0 g	a/*/d/e/f/
w[2] : 0-1 1 e	a/./

#qout1|GraphID|L
g0|2 4 5 0 1
- A node for wildcards cannot be among nodes solution, so 2,4,5,0,1 cannot be nodes that satisfies wildcards.
- Same node cannot be present more than once in each path

Local Sol:
w[0] : 2-3-4; 2-6-4; 2-7-4
w[1] : 2-5; 2-6-5
w[2] : 2-6; 2-7; 2-3

4/5. Matching Process

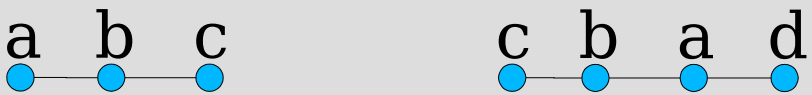
We look for a global solution for each possible combination of local sol in w[0],w[1],w[2]

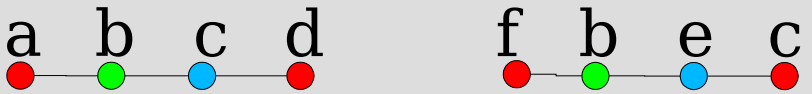
w[0] : 2-3-4; 2-6-4; 2-7-4
w[1] : 2-5; 2-6-5
w[2] : 2-6; 2-7; 2-3

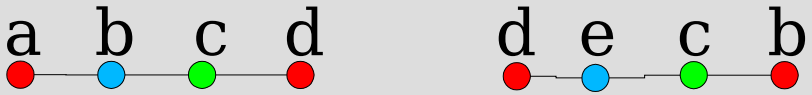
2-3-4; 2-5; 2-6; GLOBAL SOL
2-3-4; 2-5; 2-7; GLOBAL SOL
2-3-4; 2-5; 2-3; NO SOL
2-3-4; 2-6-5; 2-6; NO SOL
2-3-4; 2-6-5; 2-7; GLOBAL SOL
2-3-4; 2-6-5; 2-3; NO SOL
etc....

5/5. Matching Process

Rules for solution paths.
1) If one path is contained in another then NO SOLUTION
2) If two paths share the same initial node then NO SOL
3) If two paths share the same final node then NO SOL

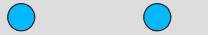
1) 

2) 

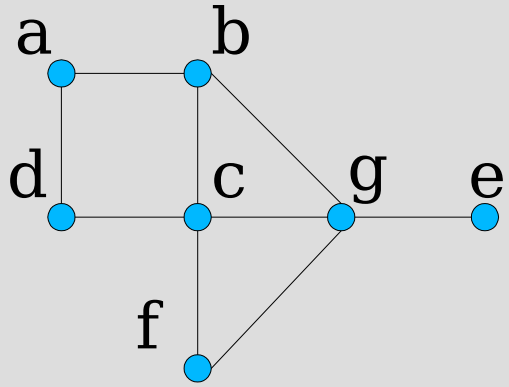
3) 

1/24. Example

Query: a(./)(+/./c/)



So we look in Database for each graph which contains nodes 'a' and 'c'



One graph of OUR DATABASE

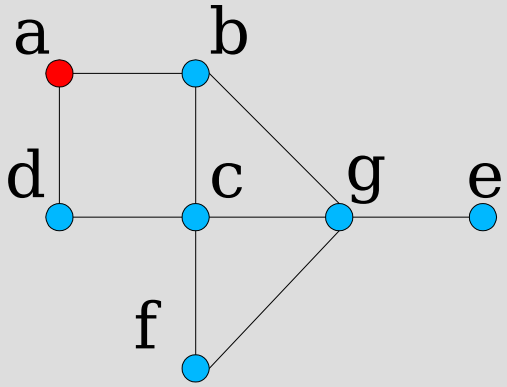
Parser table

w[0] = 0 -1 1 e
w[1] = 0 1 2 g

2/24. Example

Query: a(./)(+/./c/)

CHECKING FOR W[0]



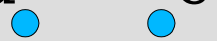
One graph of
OUR DATABASE

Parser table

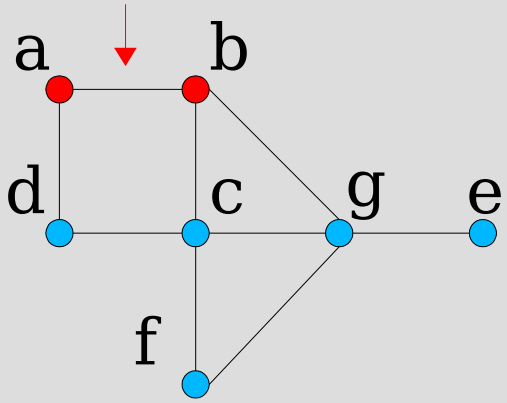
w[0] = 0 -1 1 e
w[1] = 0 1 2 g

3/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[0]



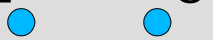
One graph of
OUR DATABASE

Parser table

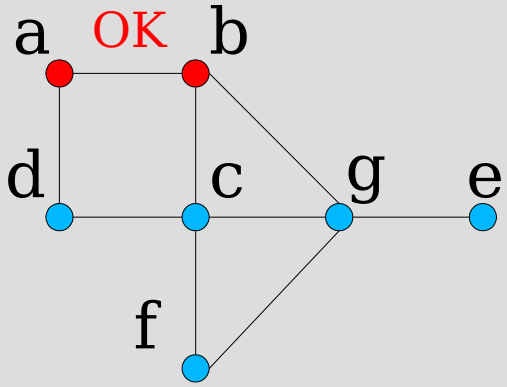
w[0] = 0 -1 1 e
w[1] = 0 1 2 g

4/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[0]



One graph of
OUR DATABASE

Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

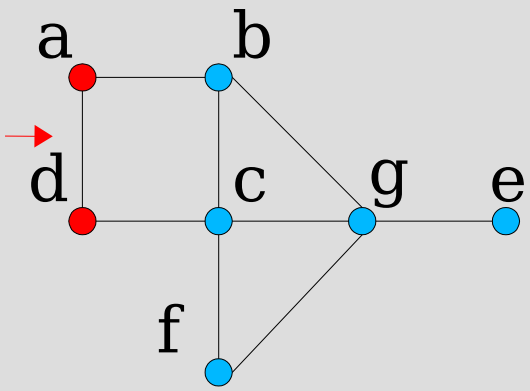
w[0]:{ab}

5/24. Example

Query: a(./)(+/./c/)

a c

CHECKING FOR W[0]



One graph of
OUR DATABASE

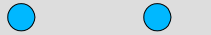
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

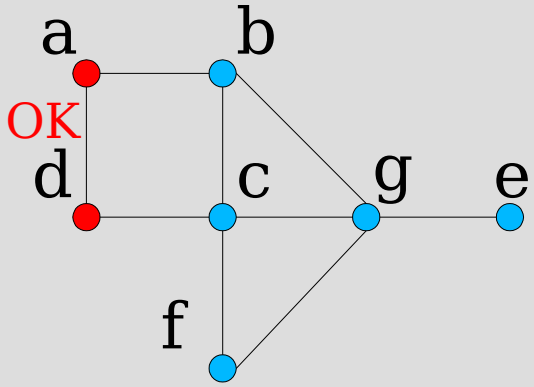
w[0]:{ab}

6/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[0]



One graph of
OUR DATABASE

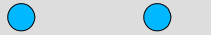
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

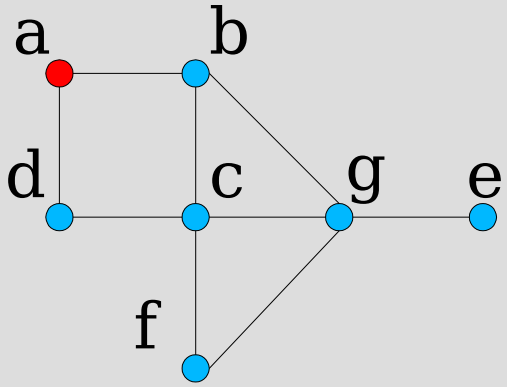
w[0]:{ab, ad}

7/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[1]



One graph of
OUR DATABASE

Parser table

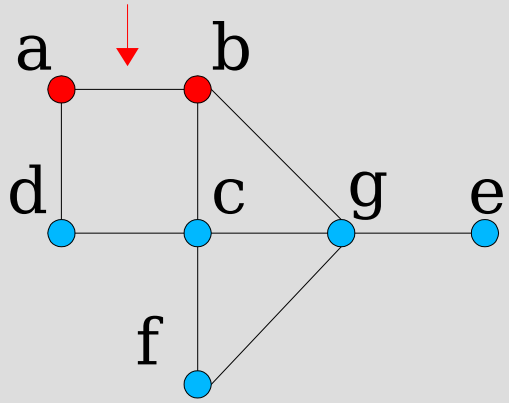
w[0] =	0	-1	1	e
w[1] =	0	1	2	g

w[0]:{ab, ad}

8/24. Example

Query: a(./)(+/./c/)

CHECKING FOR W[1]



One graph of
OUR DATABASE

Parser table

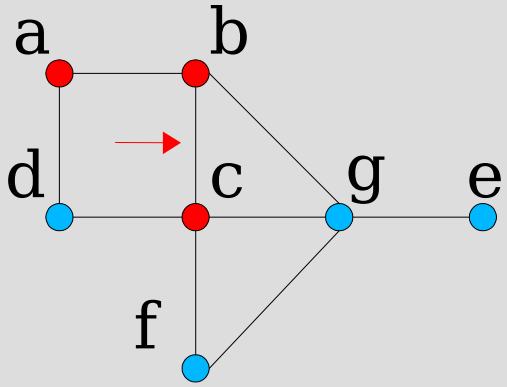
w[0] =	0	-1	1	e
w[1] =	0	1	2	g

w[0]:{ab, ad}

9/24. Example

Query: a(./)(+/./c/)

CHECKING FOR W[1]



One graph of
OUR DATABASE

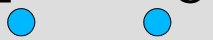
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

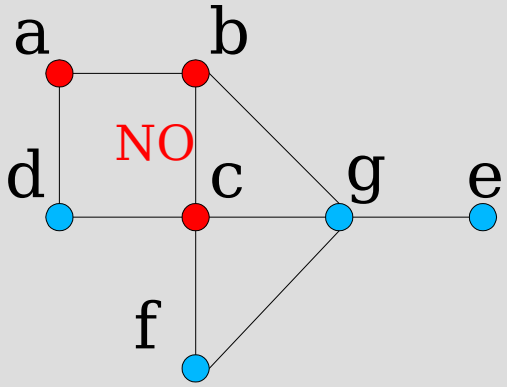
w[0]:{ab, ad}

10/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[1]



One graph of
OUR DATABASE

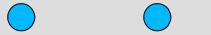
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

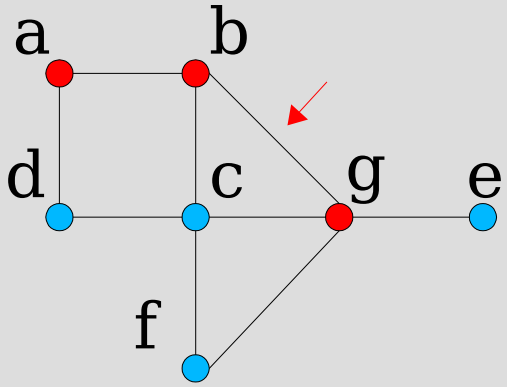
w[0]:{ab, ad}

11/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[1]



One graph of
OUR DATABASE

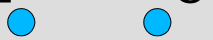
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

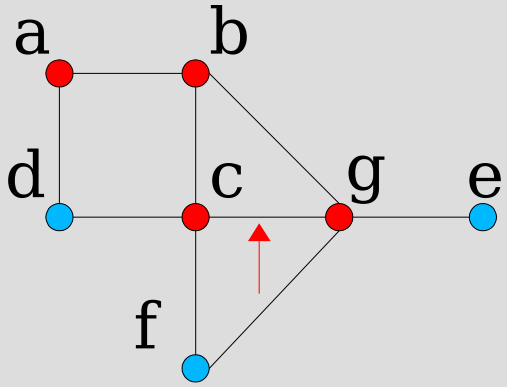
w[0]:{ab, ad}

12/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[1]



One graph of
OUR DATABASE

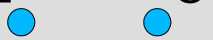
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

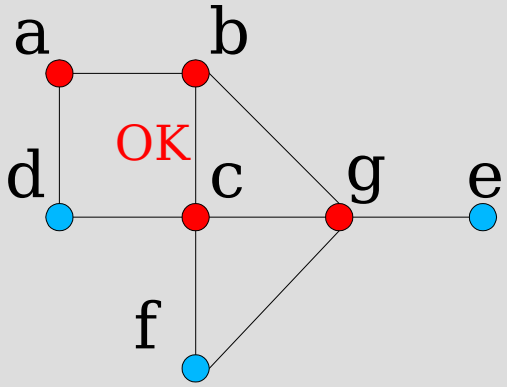
w[0]:{ab, ad}

13/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[1]



One graph of
OUR DATABASE

Parser table

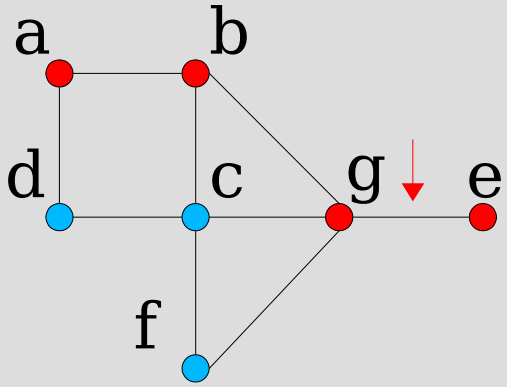
w[0] =	0	-1	1	e
w[1] =	0	1	2	g

w[0]:{ab, ad}
w[1]:{abgc}

14/24. Example

Query: a(./)(+/./c/)

CHECKING FOR W[1]



One graph of
OUR DATABASE

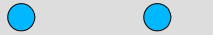
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

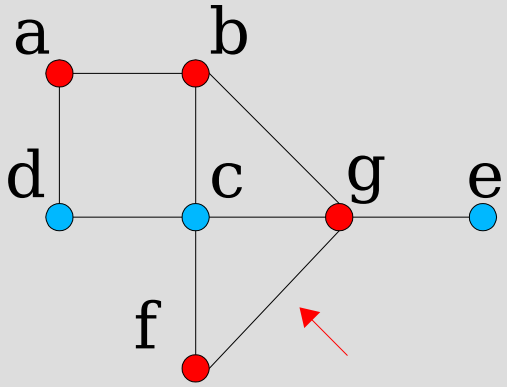
w[0]:{ab, ad}
w[1]:{abgc}

15/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[1]



One graph of
OUR DATABASE

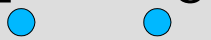
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

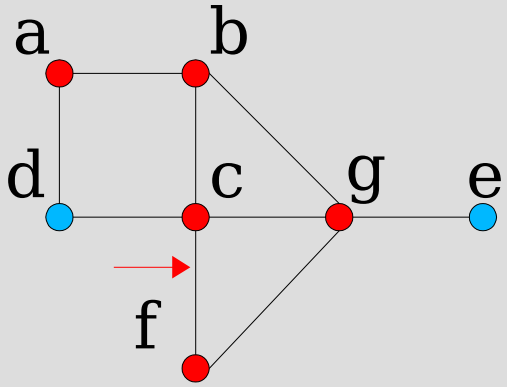
w[0]:{ab, ad}
w[1]:{abgc}

16/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[1]



One graph of
OUR DATABASE

Parser table

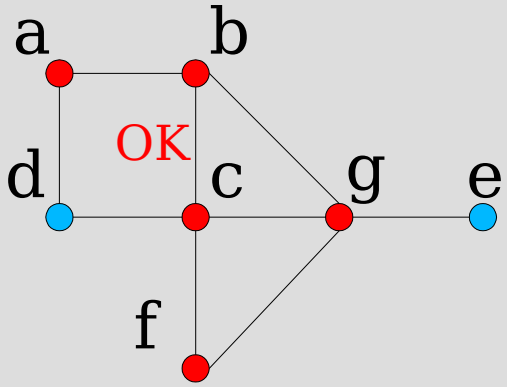
w[0] = 0 -1 1 e
w[1] = 0 1 2 g

w[0]:{ab, ad}
w[1]:{abgc}

17/24. Example

Query: a(./)(+/./c/)

CHECKING FOR W[1]



One graph of
OUR DATABASE

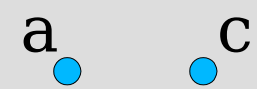
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

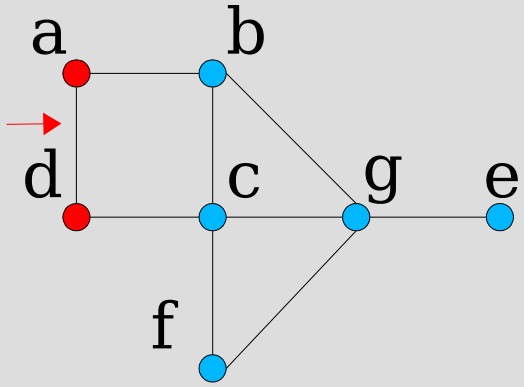
w[0]:{ab, ad}
w[1]:{abgc,abgfc}

18/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[1]



One graph of
OUR DATABASE

Parser table

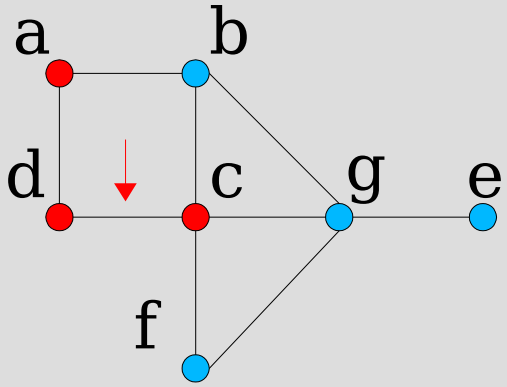
w[0] =	0	-1	1	e
w[1] =	0	1	2	g

w[0]:{ab, ad}
w[1]:{abgc,abgfc}

19/24. Example

Query: a(./)(+/./c/)

CHECKING FOR W[1]



One graph of
OUR DATABASE

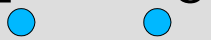
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

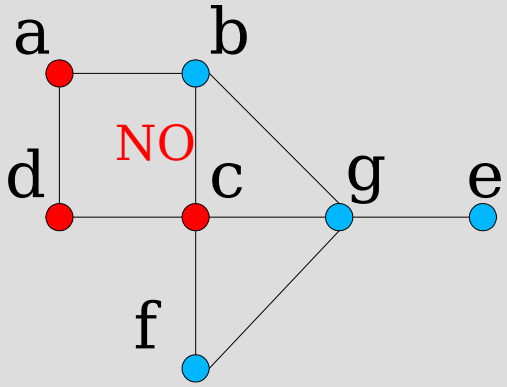
w[0]:{ab, ad}
w[1]:{abgc,abgfc}

20/24. Example

Query: a(./)(+/./c/)



CHECKING FOR W[1]



One graph of
OUR DATABASE

Parser table

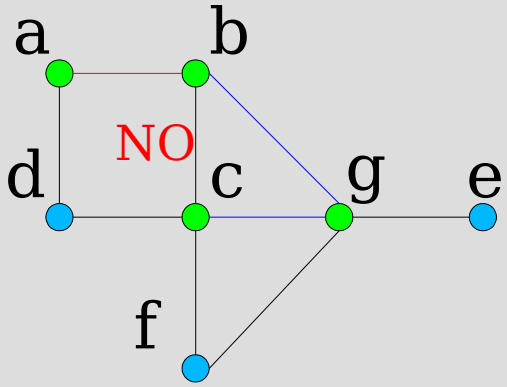
w[0] =	0	-1	1	e
w[1] =	0	1	2	g

w[0]:{ab, ad}
w[1]:{abgc,abgfc}

21/24. Example

Query: a(./)(+/./c/)

Cartesian Product
w[0][0] x w[1][0]



One graph of
OUR DATABASE

SOL : {∅}

Parser table

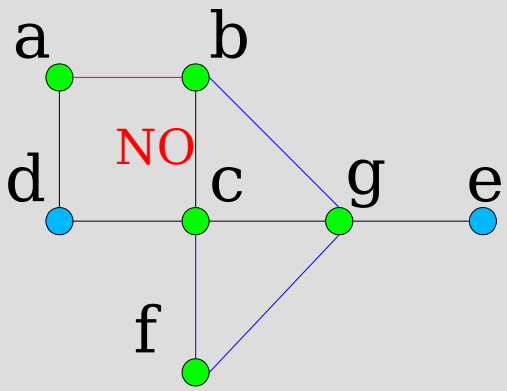
w[0] =	0	-1	1	e
w[1] =	0	1	2	g

w[0]:{ab, ad}
w[1]:{abgc,abgfc}

22/24. Example

Query: a(./)(+/./c/)

Cartesian Product
w[0][0] x w[1][1]



One graph of
OUR DATABASE

SOL : {∅}

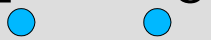
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

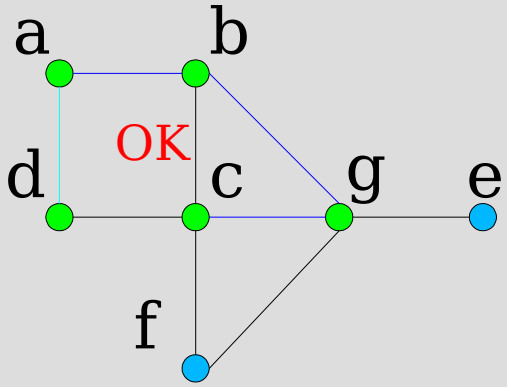
w[0]:{ab, ad}
w[1]:{abgc,abgfc}

23/24. Example

Query: a(./)(+/./c/)



Cartesian Product
w[0][1] x w[1][0]



One graph of
OUR DATABASE

SOL : {ad-abgc}

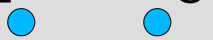
Parser table

w[0] =	0	-1	1	e
w[1] =	0	1	2	g

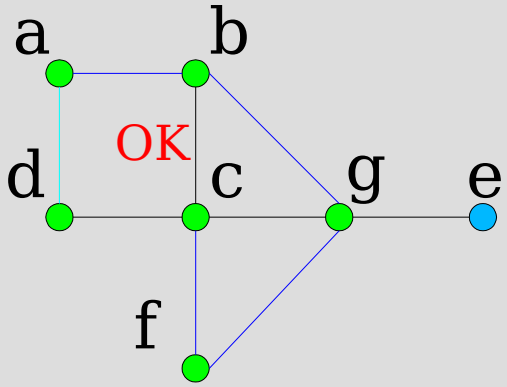
w[0]:{ab, ad}
w[1]:{abgc,abgfc}

24/24. Example

Query: a(./)(+/./c/)



Cartesian Product
w[0][1] x w[1][1]



One graph of
OUR DATABASE

SOL : {ad-abgc, ad-abgfc}

Parser table

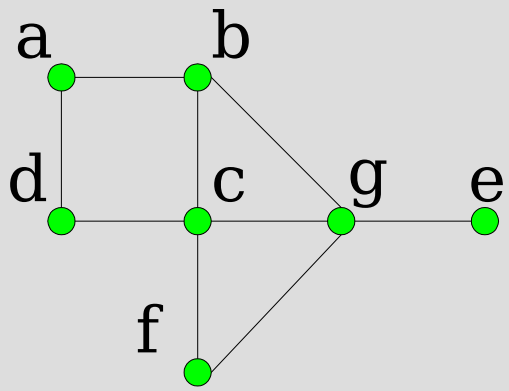
w[0] =	0	-1	1	e
w[1] =	0	1	2	g

w[0]:{ab, ad}
w[1]:{abgc,abgfc}

1/2. Optimization

(*, +) too expensive. NO OPTIMIZATION
(.) during visit if path is greater than val then “continue”
(?) during visit if path is greater than val then “continue”

Query: a(./c/)(?/g/)



w[0] = 0 1 1 e
w[1] = 0 2 1 L

w[0]:{abc, adc}
w[1]:{abg}

2/2. Optimization

Query: b%1/./b/(a%1/e%2%3/)(f%2/e/(f%3/)(b/b/./i/a/?/b/))

|DB| = 10000 graphs 30 nodes, 10 labels

~4000 results of Exact Matching
~1600 results of Inexact Matching

```
w[0] = 0   1  1 e
w[1] = 8   9  1 e
w[2] =10 11 1 L
```

TIME
1,7 secs without optimization
1 sec with optimization

1/2 Conclusion

- It could be very expensive to perform the matching (*,+)
- We give all possible matches user wants to find (maybe we can stop the process as soon as we find 1 result)
- We perform an exact inexact matching (we give the possibility to control the approximation, other systems don't)

2/2 Conclusion

With a |DB|=10000 graphs Meshes2D Regular,
30 nodes every graph 10 labels

query b%1/./b/(a%1/e%2%3/)(f%2/e/(f%3/)(b/b/./i/a/*/b/))

```
w[0] : 0 1  1 e
w[1] : 8 9  1 e
w[2] : 10 11 0 g
```

Nodes	12
Edges	11
Graphs after filtering	1312
# results1	~4000
# results2	~6700
Time	1,19 secs

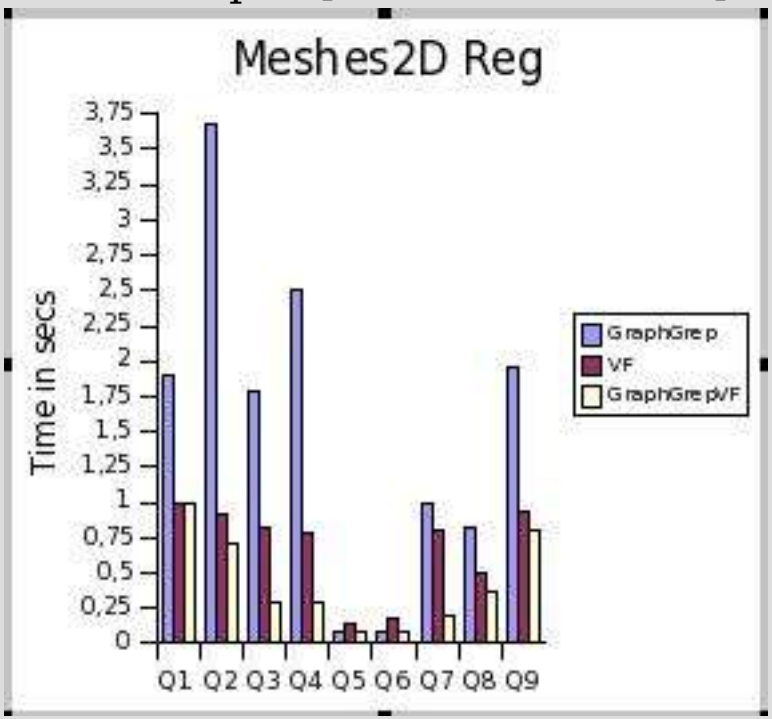
With a query 12 nodes and 16 edges with no wildcards time is ~1.12 seconds and ~2400 results

PART 2

Results and comparisons for EXACT Graph Matching of GraphGrepVF on our Benchmark

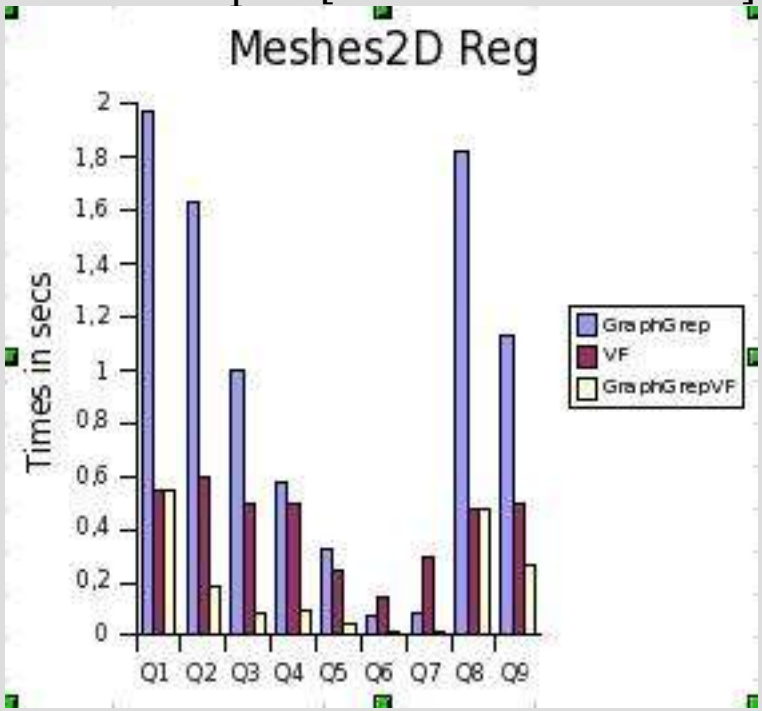
1/8. Results for exact matching Meshes2D Regular

10000 Graphs [50 nodes 8 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	40646	10000
Q2	8	10	7481	9136
Q3	12	16	2522	2522
Q4	16	24	1660	2522
Q5	4	4	0	0
Q6	50	84	0	0
Q7	50	84	424	424
Q8	2	1	7481	5825
Q9	8	7	14962	7516

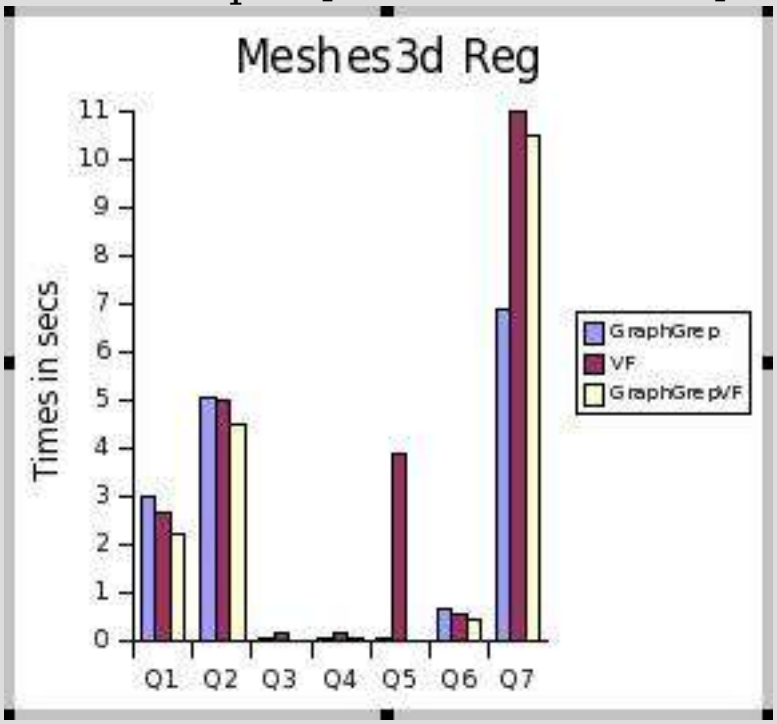
10000 Graphs [50 nodes 25 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	20317	10000
Q2	8	10	4159	3944
Q3	12	16	1250	1623
Q4	16	24	823	882
Q5	4	4	0	2774
Q6	50	84	0	0
Q7	50	84	1	1
Q8	2	1	19807	10000
Q9	8	7	3323	4027

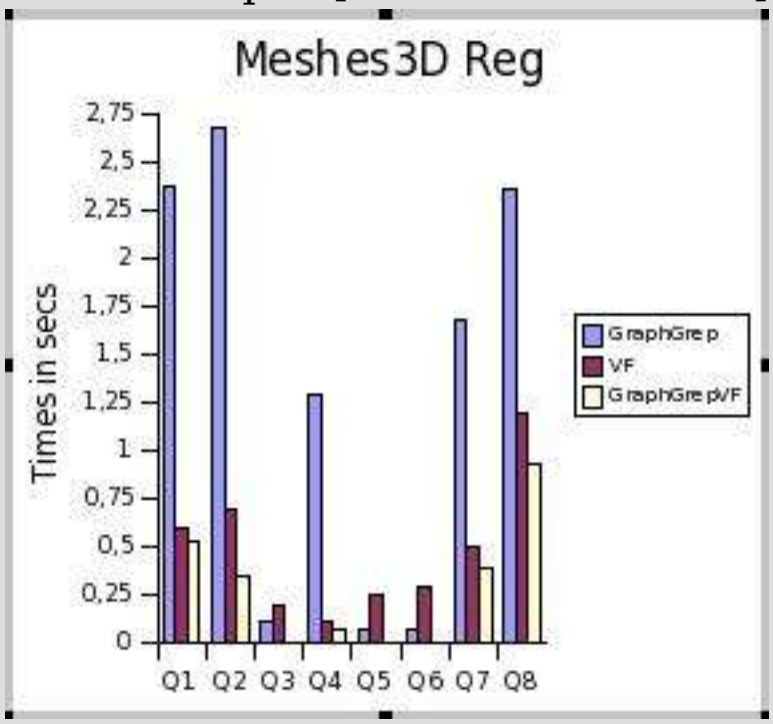
2/8. Results for exact matching Meshes3D Regular

10000 Graphs [51 nodes 5 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	300728	8768
Q2	8	10	118084	6270
Q3	4	4	0	0
Q4	51	103	0	0
Q5	51	103	2	3
Q6	2	1	58015	8361
Q7	8	7	987008	6885

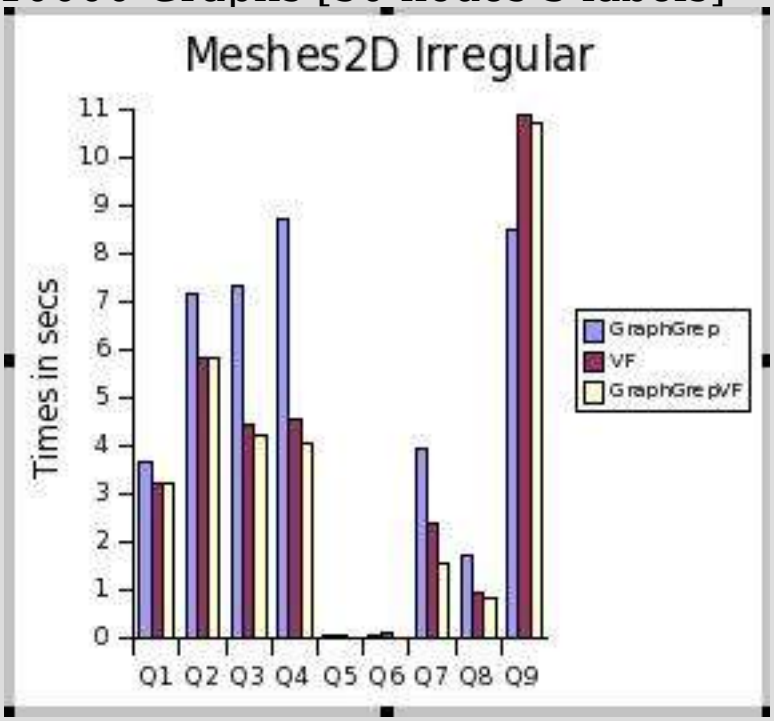
10000 Graphs [51 nodes 15 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	22570	9492
Q2	8	10	4940	5072
Q3	12	16	0	247
Q4	4	4	0	7119
Q5	51	103	0	0
Q6	51	103	1	1
Q7	2	1	22481	9098
Q8	8	7	39018	7290

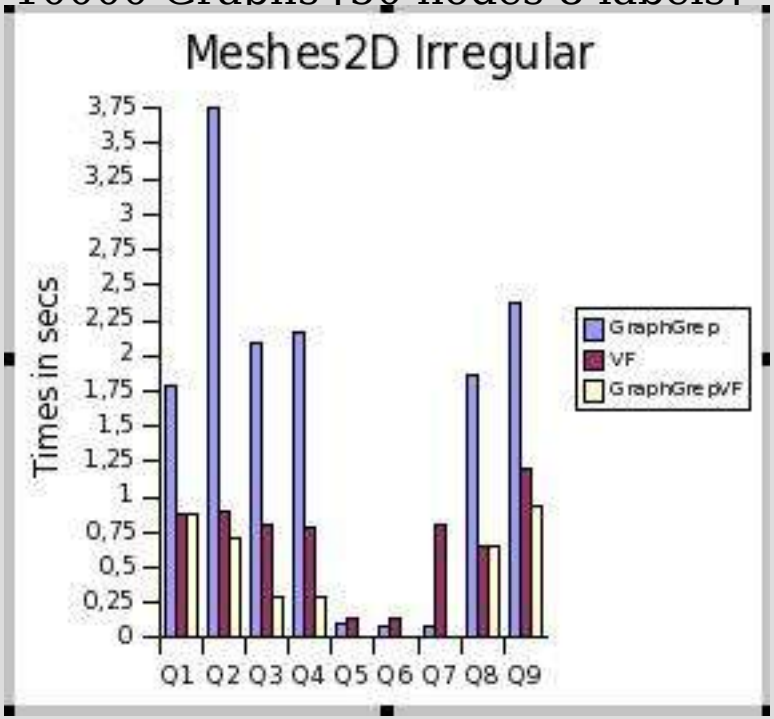
3/8. Results for exact matching Meshes2D Irregular

10000 Graphs [50 nodes 3 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	362096	10000
Q2	8	10	66272	10000
Q3	12	16	9411	9689
Q4	16	24	6900	9082
Q5	4	4	0	0
Q6	50	90	0	0
Q7	50	90	1	5402
Q8	2	1	126341	9991
Q9	8	7	977351	9972

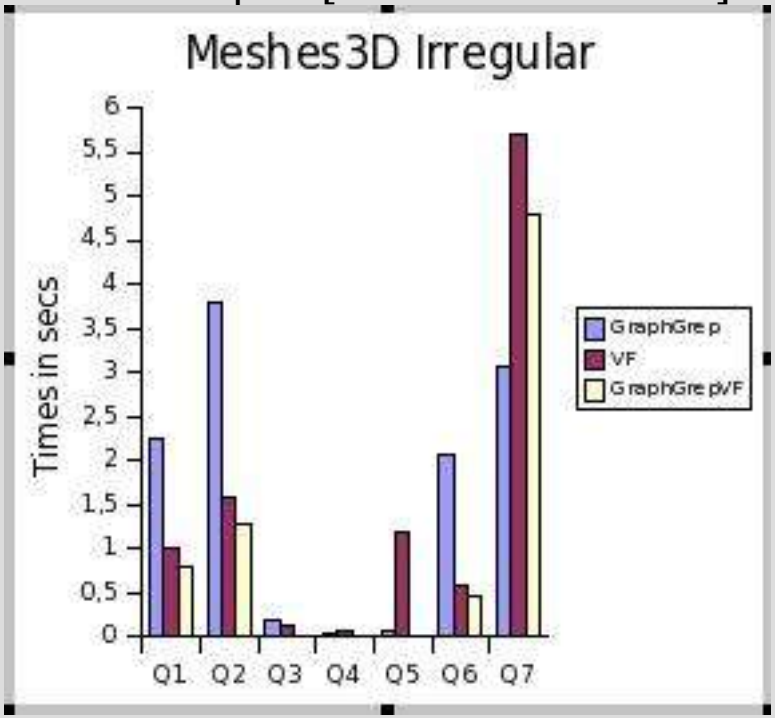
10000 Graphs [50 nodes 8 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	39780	10000
Q2	8	10	7396	9201
Q3	12	16	2520	2574
Q4	16	24	1660	2507
Q5	4	4	0	314
Q6	50	90	0	1
Q7	50	90	1	7
Q8	2	1	40376	10000
Q9	8	7	39018	7290

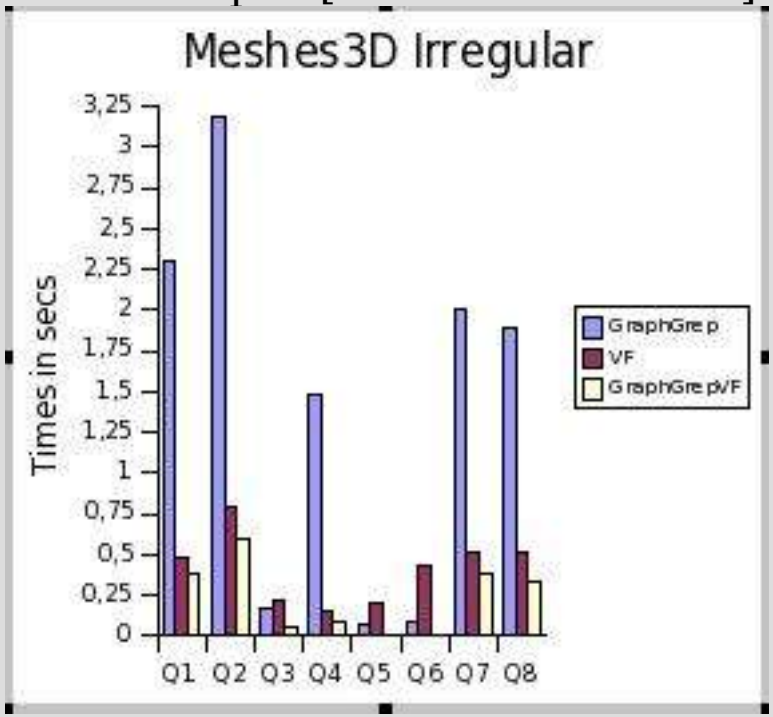
4/8. Results for exact matching Meshes3D Irregular

10000 Graphs [51 nodes 7 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	44864	9459
Q2	8	10	24267	5939
Q3	4	4	0	1592
Q4	51	109	0	0
Q5	51	109	1	2
Q6	2	1	45793	9416
Q7	8	7	181036	6394

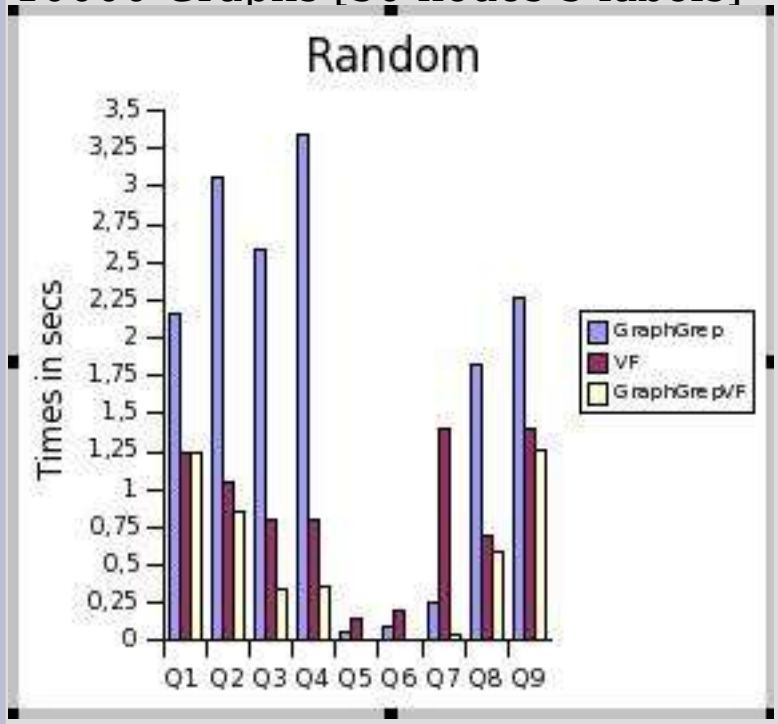
10000 Graphs [51 nodes 20 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	22443	9095
Q2	8	10	5175	6850
Q3	12	16	0	1002
Q4	4	4	0	7173
Q5	51	109	0	1
Q6	51	109	1	1
Q7	2	1	23055	9724
Q8	8	7	858	7311

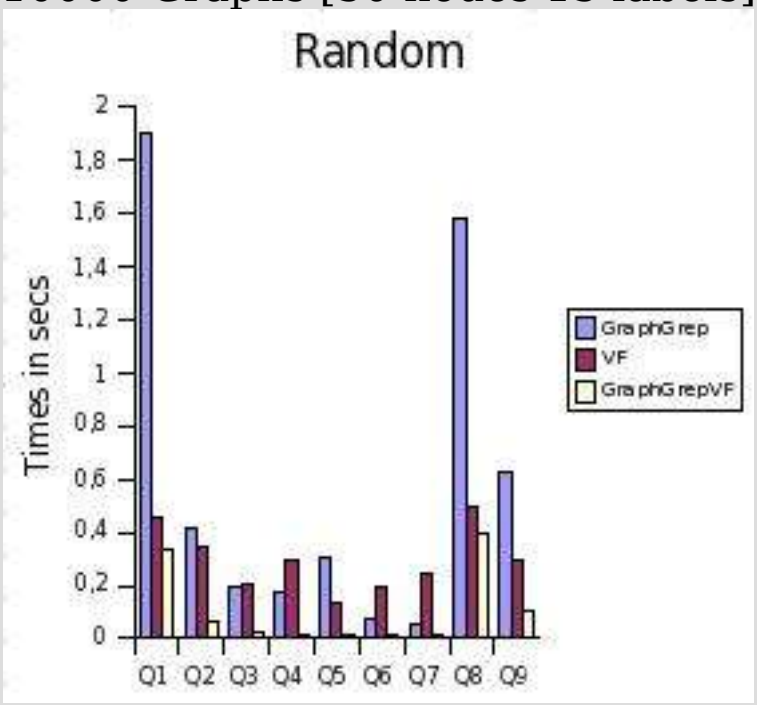
5/8. Results for exact matching Random

10000 Graphs [50 nodes 5 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	80000	10000
Q2	8	10	13298	6649
Q3	12	16	3265	3265
Q4	16	24	3384	3384
Q5	4	4	0	0
Q6	50	57	0	58
Q7	50	57	24	136
Q8	2	1	25196	8998
Q9	8	7	53880	6735

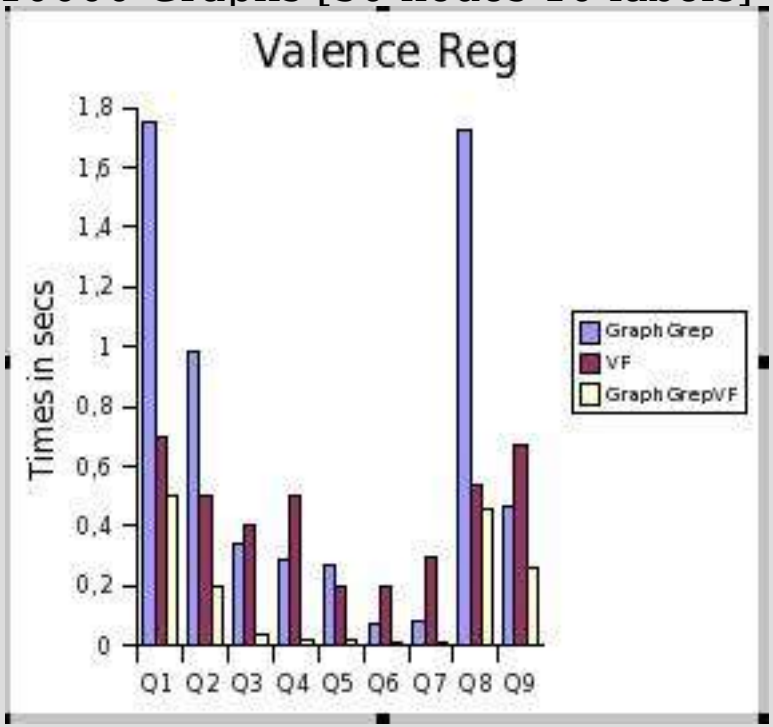
10000 Graphs [50 nodes 15 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	7716	8257
Q2	8	10	993	1162
Q3	12	16	195	545
Q4	16	24	148	153
Q5	4	4	0	854
Q6	50	57	0	0
Q7	50	57	12	1
Q8	2	1	7651	7830
Q9	8	7	684	2549

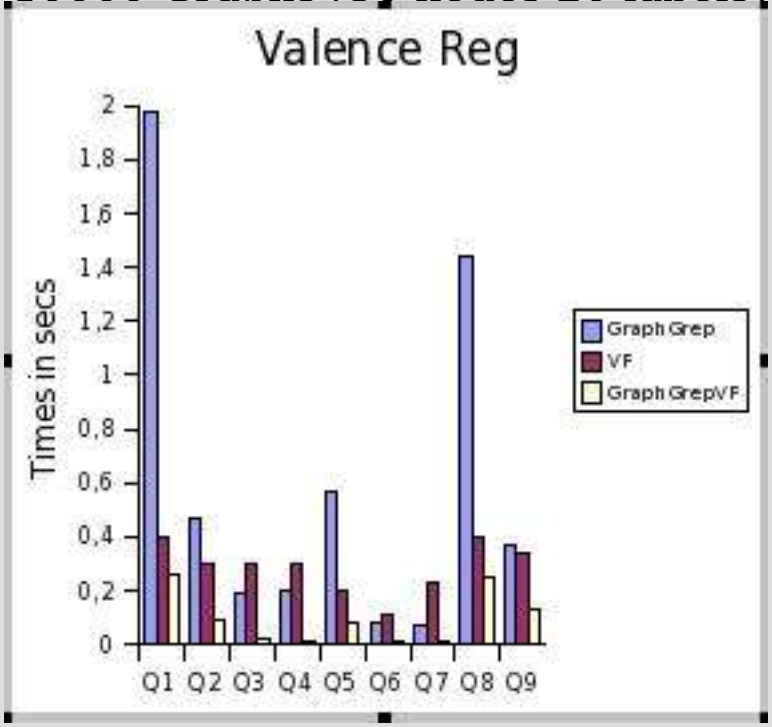
6/8. Results for exact matching Valence Regular

10000 Graphs [50 nodes 10 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	17644	8841
Q2	8	10	2439	3076
Q3	12	16	305	1209
Q4	16	24	271	433
Q5	4	4	0	2423
Q6	50	95	0	0
Q7	50	95	1	1
Q8	2	1	17596	9263
Q9	8	7	10476	2969

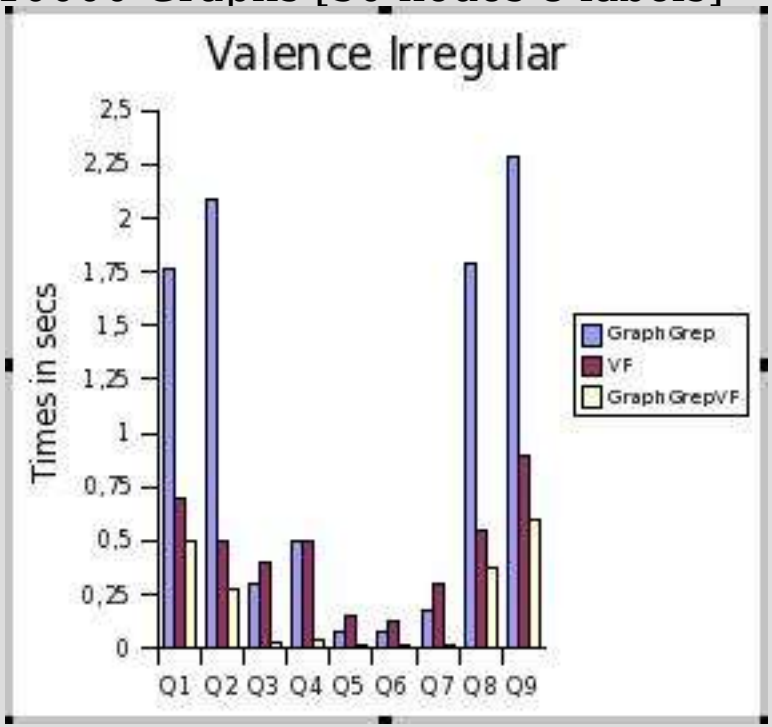
10000 Graphs [50 nodes 20 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	5163	8065
Q2	8	10	468	2839
Q3	12	16	42	900
Q4	16	24	31	322
Q5	4	4	0	5124
Q6	50	93	0	0
Q7	50	93	1	1
Q8	2	1	5163	7344
Q9	8	7	370	3367

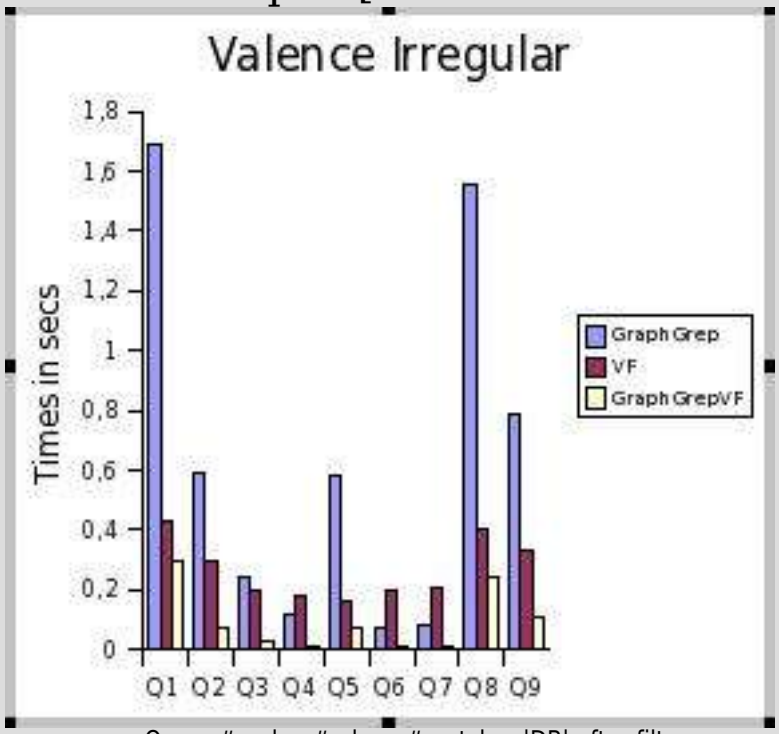
7/8. Results for exact matching Valence Irregular

10000 Graphs [50 nodes 8 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	17656	8828
Q2	8	10	2497	5293
Q3	12	16	626	626
Q4	16	24	693	858
Q5	4	4	0	0
Q6	50	71	0	0
Q7	50	71	1	58
Q8	2	1	17562	8781
Q9	8	7	10748	6306

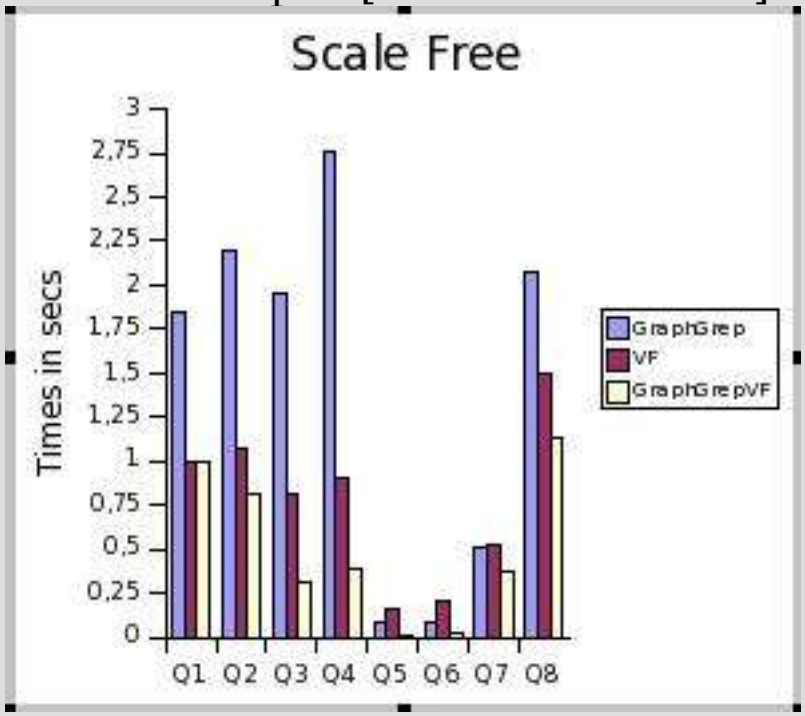
10000 Graphs [50 nodes 22 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	5263	7935
Q2	8	10	447	2953
Q3	12	16	60	1023
Q4	16	24	42	258
Q5	4	4	0	5294
Q6	50	91	0	1
Q7	50	91	1	1
Q8	2	1	5263	8094
Q9	8	7	380	3403

8/8. Results for exact matching Scale Free

10000 Graphs [50 nodes 5 labels]



Query	#nodes	#edges	#match	|DB| after filter
Q1	4	4	80000	10000
Q2	8	10	13162	6581
Q3	12	16	3333	3333
Q4	16	24	3248	3248
Q5	4	4	0	0
Q6	50	57	0	0
Q7	2	1	16647	7669
Q8	8	7	53336	6667

END