Detecting Logic Bugs in Graph Database Management Systems via Injective and Surjective Graph Query Transformation

Yuancheng Jiang, Jiahao Liu, Jinsheng Ba
Roland Yap, Zhenkai Liang, Manuel Rigger

NUS National University of Singapore

From Relational to Graph

Relational Data Model

Students

ID	Name
001	Sam
002	Mary
003	Tine
004	Jay

StudentCourses

StudentId	CourseId
001	1
001	2
001	3
002	3
002	4
003	3
003	5
004	5

Courses

ID	Name
1	SQL Server
2	ASP.NET
3	MongoDB
4	Java
5	PHP

From Relational to Graph

Relational Data Model

Students

ID	Name
001	Sam
002	Mary
003	Tine
004	Jay

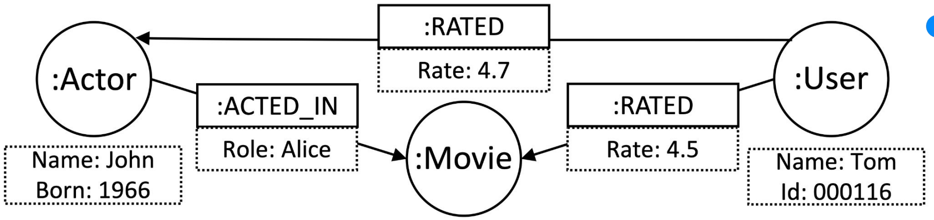
StudentCourses

StudentId	CourseId
001	1
001	2
001	3
002	3
002	4
003	3
003	5
004	5

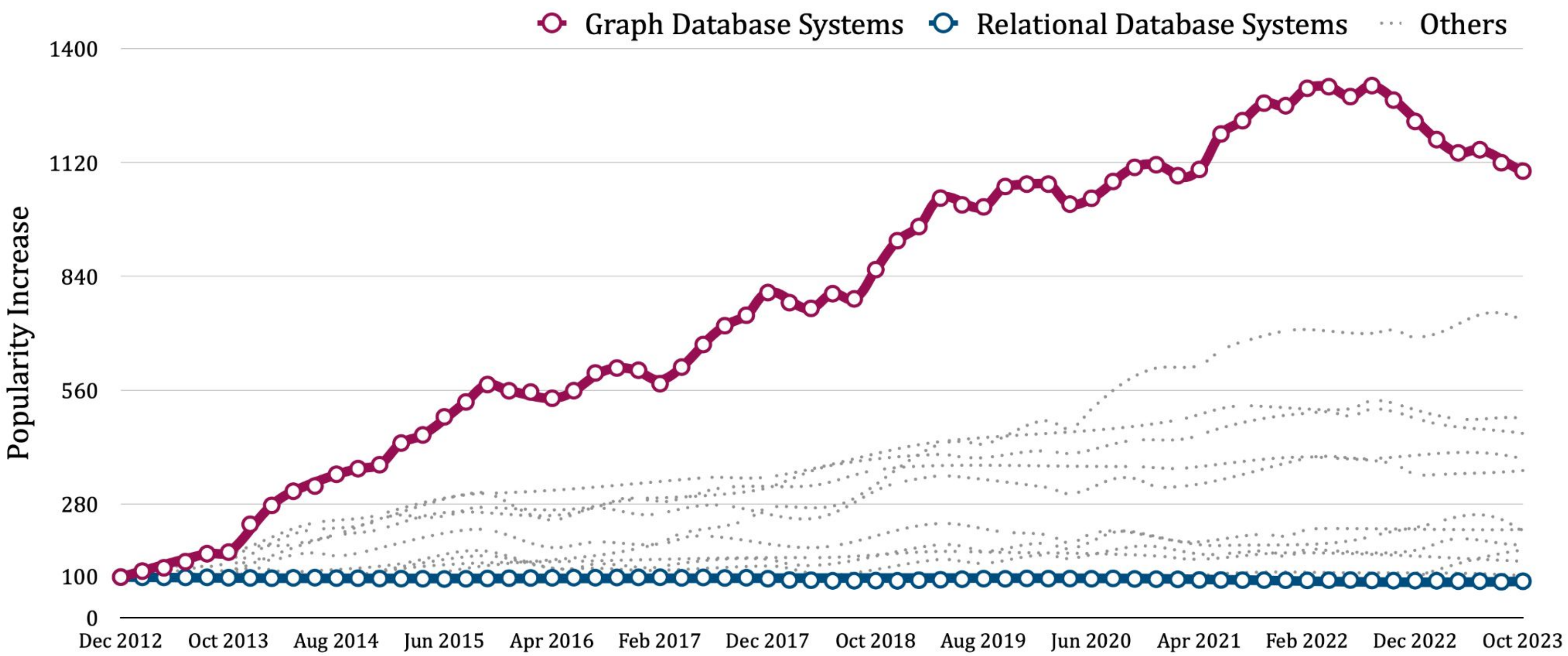
Courses

ID	Name
1	SQL Server
2	ASP.NET
3	MongoDB
4	Java
5	PHP

Graph Data Model

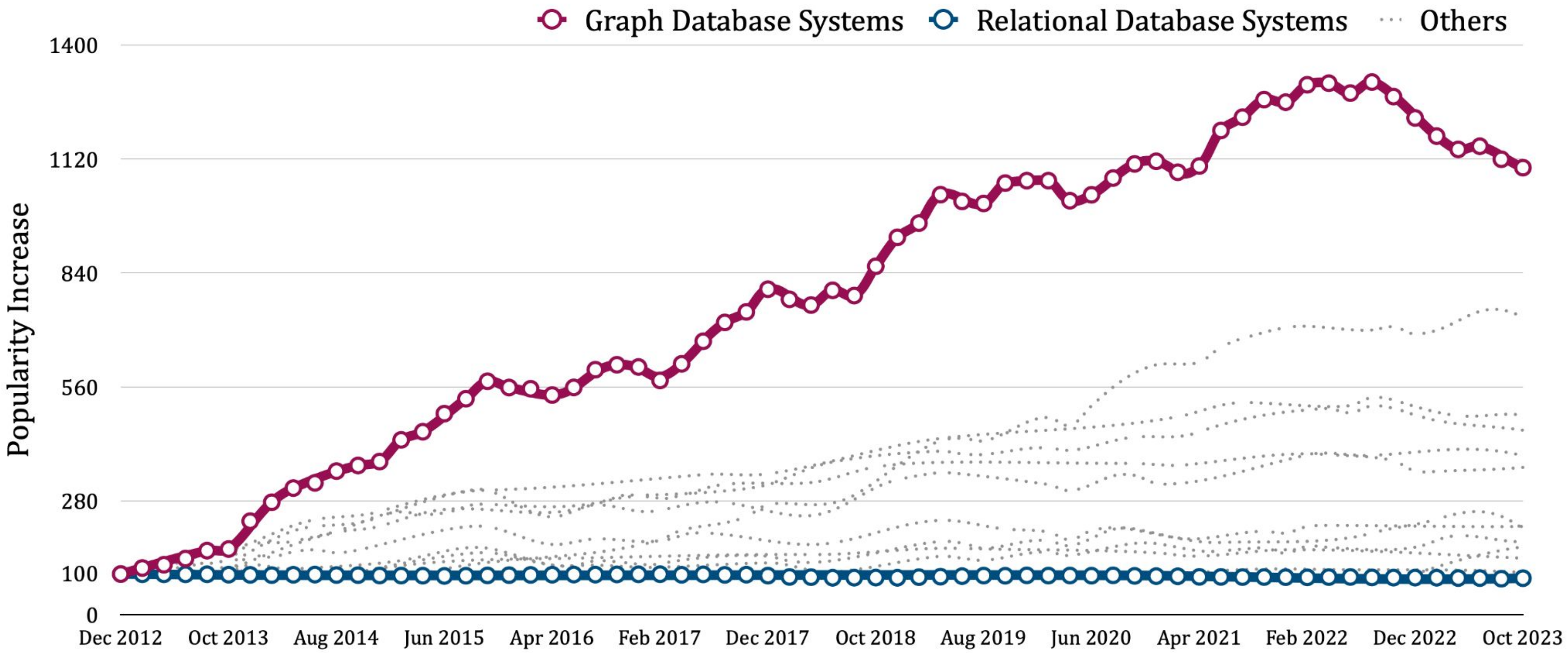


The Fastest Growing Model in Past Decade*



*: Statistics collected at db-rank: https://db-engines.com/en/ranking_categories

The Fastest Growing Model in Past Decade*



Graph Database Testing becomes essential to improve Robustness and Accuracy

*: Statistics collected at db-rank: https://db-engines.com/en/ranking_categories

Bug Categories in Database Testing

Internal Errors (Crash): out of service (segfault, unexpected exception)

- Oracle: sanitizers, unexpected exception handlers
- Difficulty: low (users can help to submit issues when noticed)

Bug Categories in Database Testing

Internal Errors (Crash): out of service (segfault, unexpected exception)

- Oracle: sanitizers, unexpected exception handlers
- Difficulty: low (users can help to submit issues when noticed)

Logic Bugs: outputs incorrect results (wrong count, inaccurate data)

- Oracle: test oracles like differential or metamorphic testing
- Difficulty: **high** (logic bugs often go unnoticed by users without alerts)

Bug Categories in Database Testing

Internal Errors (Crash): out of service (segfault, unexpected exception)

- Oracle: sanitizers, unexpected exception handlers
- Difficulty: low (users can help to submit issues when noticed)

Logic Bugs: outputs incorrect results (wrong count, inaccurate data)

- Oracle: test oracles like differential or metamorphic testing
- Difficulty: **high** (logic bugs often go unnoticed by users without alerts)

Performance Issues: unreasonable query computing time

- Oracle: test oracles like differential or metamorphic testing
- Difficulty: **high** (hard to determine time bug without references)

Bug Categories in Database Testing

Internal Errors (Crash): out of service (segfault, unexpected exception)

- Oracle: sanitizers, unexpected exception handlers
- Difficulty: low (users can help to submit issues when noticed)

Logic Bugs: outputs incorrect results (wrong count, inaccurate data)

- Oracle: test oracles like differential or metamorphic testing
- Difficulty: **high** (logic bugs often go unnoticed by users without alerts)

Performance Issues: unreasonable query computing time

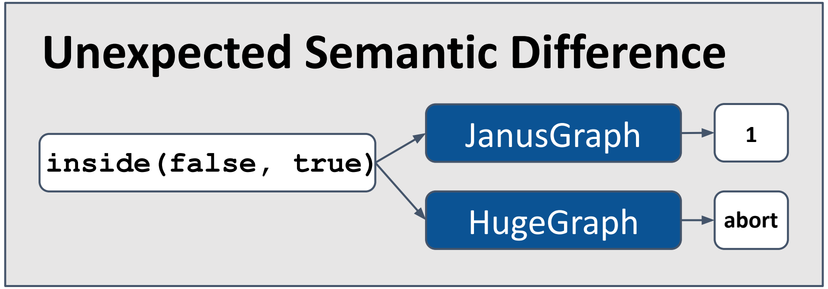
- Oracle: test oracles like differential or metamorphic testing
- Difficulty: **high** (hard to determine time bug without references)

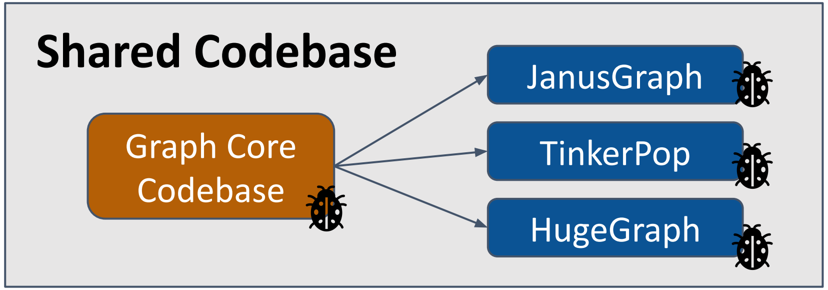
Existing Solutions

Cypher, Gremlin are the top two popular graph query languages

Grand (ISSTA'22): differential testing on *Gremlin* graph databases

☹ high false alarm rate ☹ many missed bugs ☹ unsupport of *Cypher*



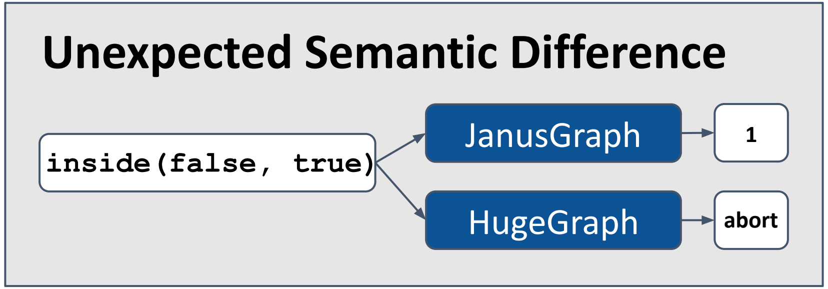


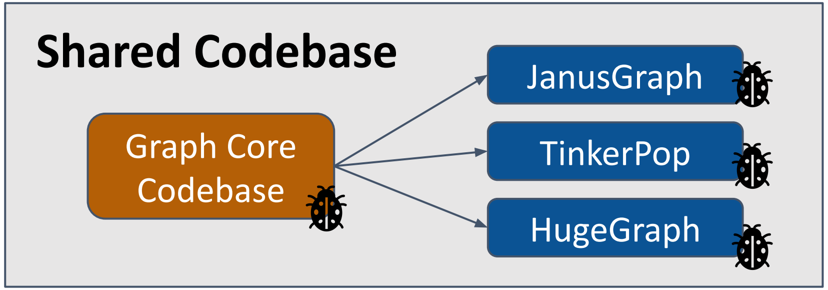
Existing Solutions

Cypher, Gremlin are the top two popular graph query languages

Grand (ISSTA'22): differential testing on *Gremlin* graph databases

☹ high false alarm rate ☹ many missed bugs ☹ unsupport of *Cypher*





GDBMeter (ISSTA'23): metamorphic testing (TLP*)

☹ without considering graph patterns

*: Ternary Logic Partitioning (TLP) was first introduced for testing relational database systems and based on the insight that a boolean predicate p can yield one of three possible outcomes: True, False, or Null.

Consider Graph When Testing Graph

Additional Constraints in Graph Queries

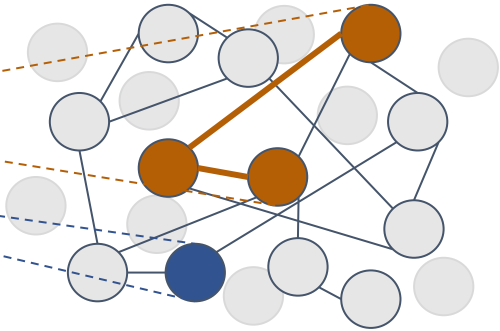
Graph Query: **graph patterns** + predicates + others

Consider Graph When Testing Graph

Additional Constraints in Graph Queries

Graph Query: graph patterns + predicates + others

```
MATCH (a:Movie)--(b)--(c)
    WHERE a.year=2012
  RETURN count(a) LIMIT 1
```

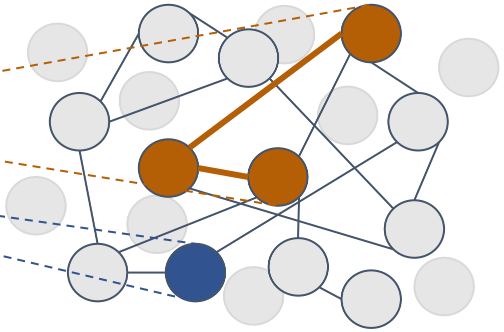


Consider Graph When Testing Graph

Additional Constraints in Graph Queries

Graph Query: graph patterns + predicates + others

```
MATCH (a:Movie)--(b)--(c)
  WHERE a.year=2012
RETURN count(a) LIMIT 1
```



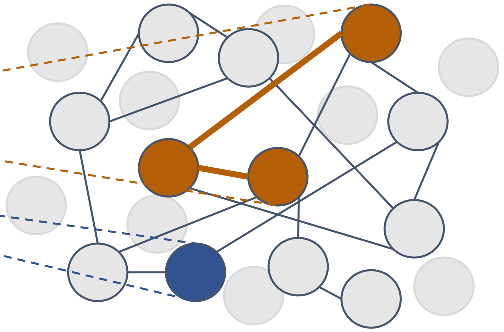
In contrast to mutating **predicates** like existing works, we aim to mutate the **graph patterns** to generate new comparable queries

Consider Graph When Testing Graph

Additional Constraints in Graph Queries

Graph Query: graph patterns + predicates + others

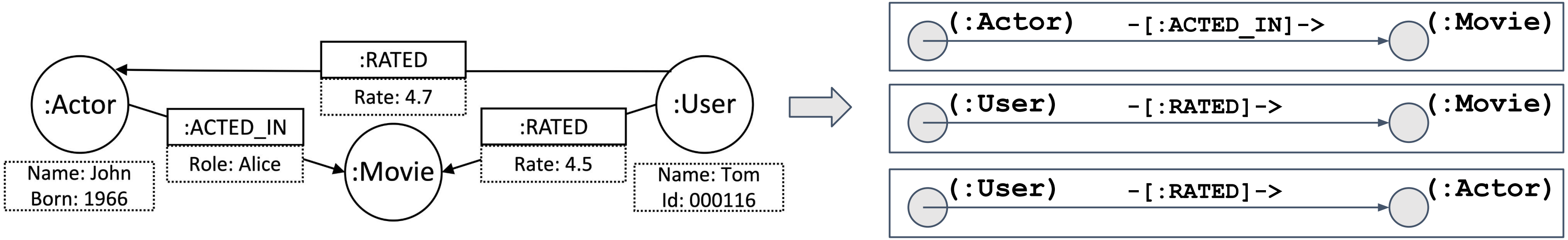
```
MATCH (a:Movie)--(b)--(c)
    WHERE a.year=2012
  RETURN count(a) LIMIT 1
```



In contrast to mutating **predicates** like existing works, we aim to mutate the **graph patterns** to generate new comparable queries

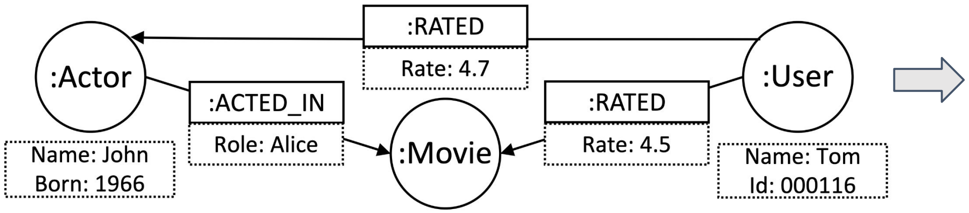
Specifically, we generate follow-up queries via systematic query transformations derived from injective and surjective mappings among **directed edge sets** of **graph patterns**

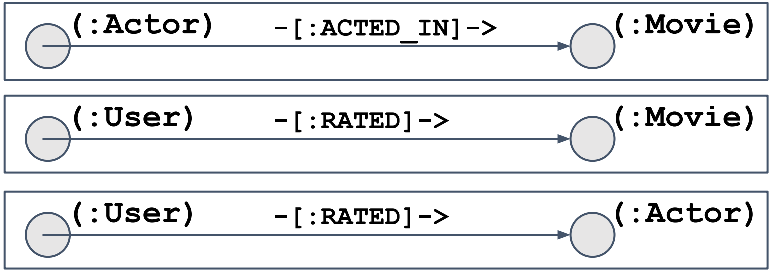
From Graph to Directed Edge Sets



Directed Edge Sets: set of edge with its head, tail, and edge information

From Graph to Directed Edge Sets

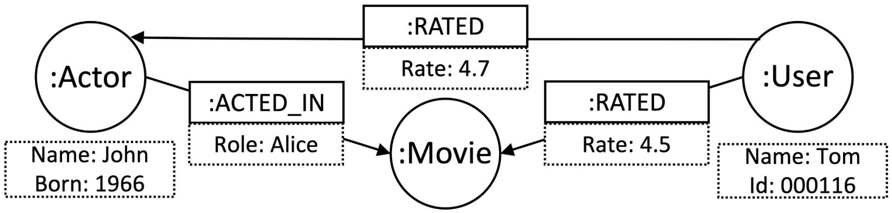


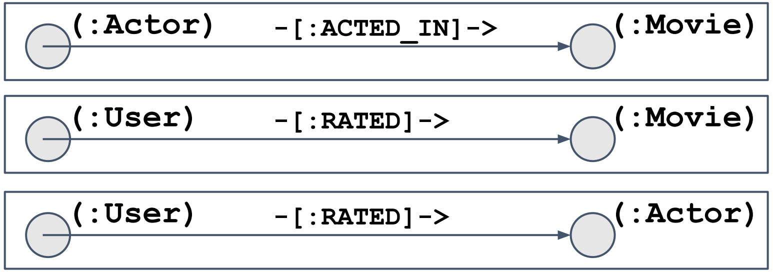


Directed Edge Sets: set of edge with its head, tail, and edge information

Graph Query Mutation => Graph Pattern Mutation => Sets Mapping

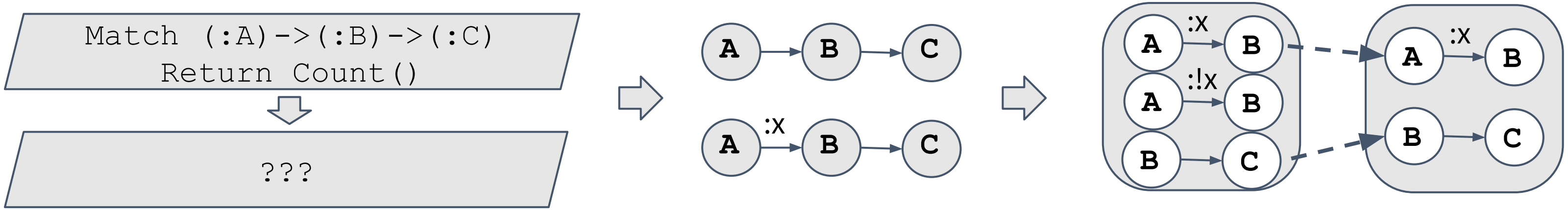
From Graph to Directed Edge Sets





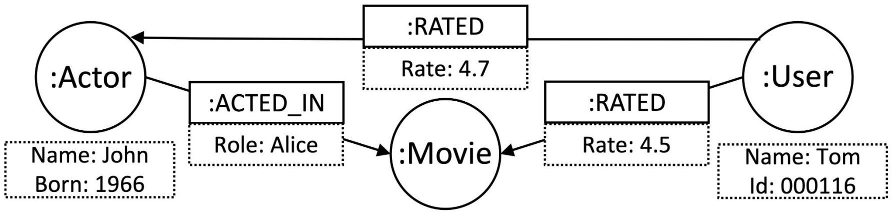
Directed Edge Sets: set of edge with its head, tail, and edge information

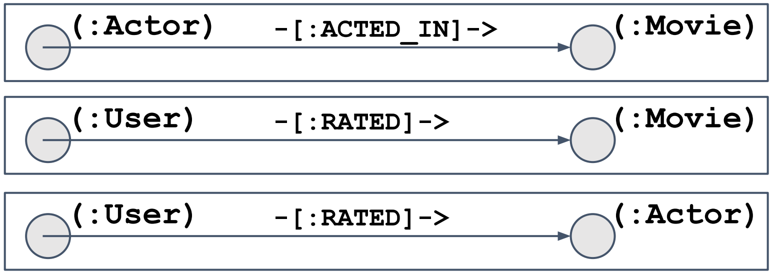
Graph Query Mutation => Graph Pattern Mutation => Sets Mapping



Restricted Query Mutation **Restricted Pattern Mutation** **Injective Mapping**

From Graph to Directed Edge Sets



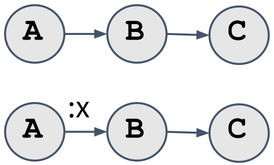


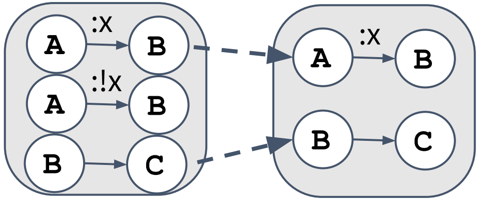
Directed Edge Sets: set of edge with its head, tail, and edge information

Graph Query Mutation => Graph Pattern Mutation => Sets Mapping

```
Match (:A)->(:B)->(:C)
Return Count()
```

```
Match (:A)-[:x]->(:B)
->(:C) Return Count()
```





Restricted Query Mutation **Restricted Pattern Mutation** **Injective Mapping**

GraphGenie*:

first metamorphic testing approach considering graph pattern mutations

*: GraphGenie is available at https://github.com/YuanchengJiang/GraphGenie

GraphGenie*:

first metamorphic testing approach considering graph pattern mutations

Testing Process: (1) Query Generation (2) Query Mutation (3) Result Analysis

- Query Generation: focus on **Cypher**, diverse in graph patterns, incremental
- Transformation Combinations: helps to generate more complex graph queries

*: GraphGenie is available at https://github.com/YuanchengJiang/GraphGenie

GraphGenie*:

first metamorphic testing approach considering graph pattern mutations

Testing Process: (1) Query Generation (2) Query Mutation (3) Result Analysis

- Query Generation: focus on **Cypher**, diverse in graph patterns, incremental
- Transformation Combinations: helps to generate more complex graph queries

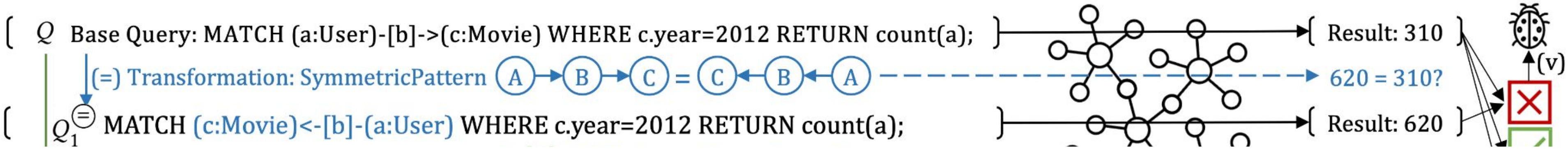
*: GraphGenie is available at https://github.com/YuanchengJiang/GraphGenie

GraphGenie*:

first metamorphic testing approach considering graph pattern mutations

Testing Process: (1) Query Generation (2) Query Mutation (3) Result Analysis

- Query Generation: focus on **Cypher**, diverse in graph patterns, incremental
- Transformation Combinations: helps to generate more complex graph queries



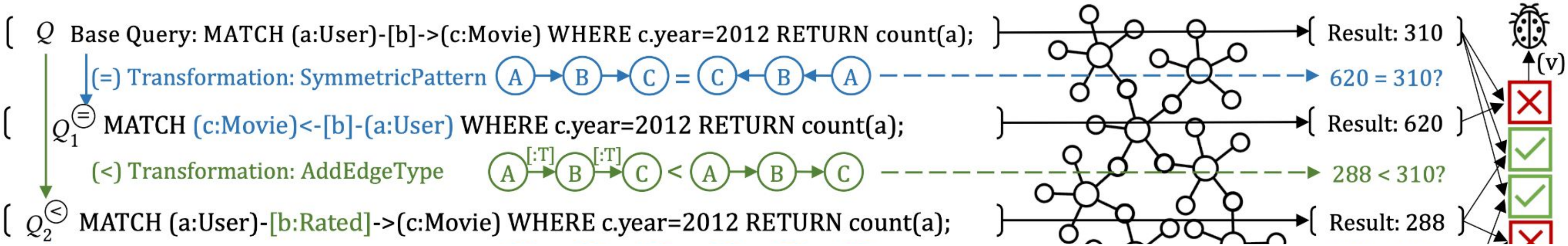
*: GraphGenie is available at https://github.com/YuanchengJiang/GraphGenie

GraphGenie*:

first metamorphic testing approach considering graph pattern mutations

Testing Process: (1) Query Generation (2) Query Mutation (3) Result Analysis

- Query Generation: focus on **Cypher**, diverse in graph patterns, incremental
- Transformation Combinations: helps to generate more complex graph queries



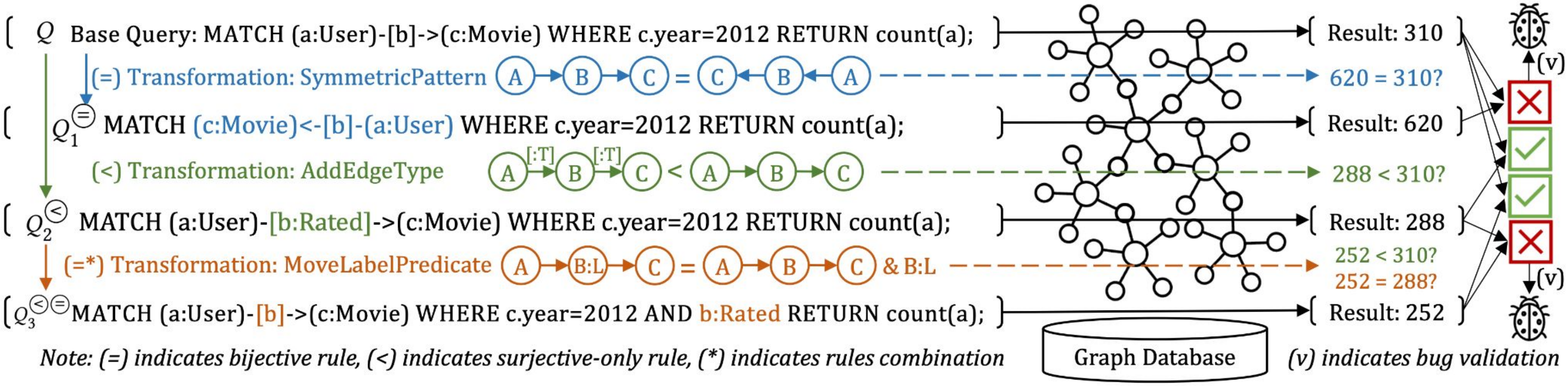
*: GraphGenie is available at https://github.com/YuanchengJiang/GraphGenie

GraphGenie*:

first metamorphic testing approach considering graph pattern mutations

Testing Process: (1) Query Generation (2) Query Mutation (3) Result Analysis

- Query Generation: focus on **Cypher**, diverse in graph patterns, incremental
- Transformation Combinations: helps to generate more complex graph queries



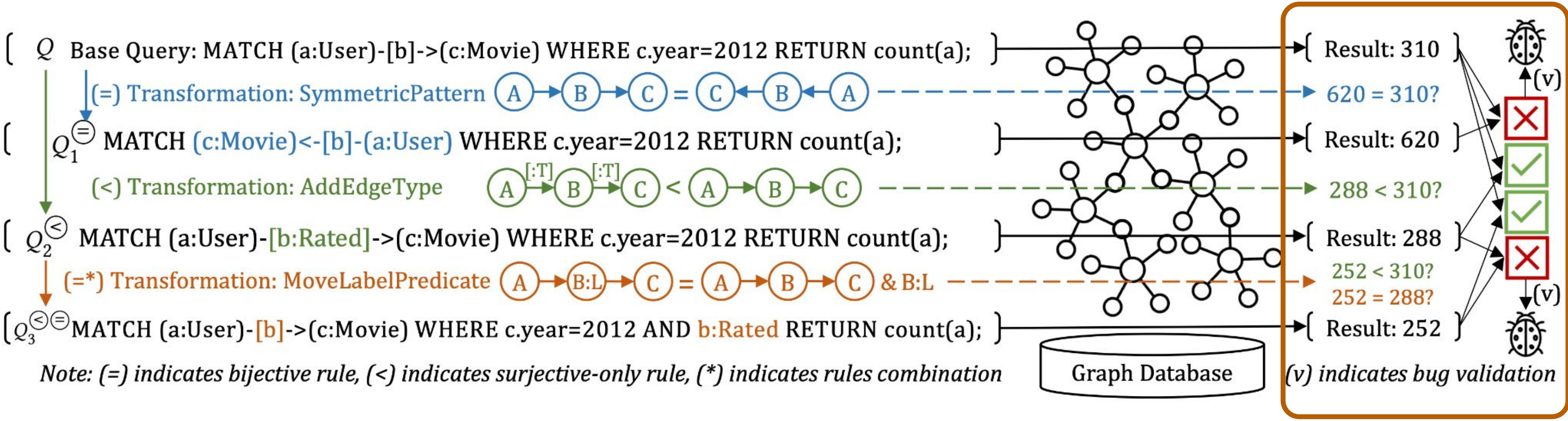
*: GraphGenie is available at https://github.com/YuanchengJiang/GraphGenie

GraphGenie*:

first metamorphic testing approach considering graph pattern mutations

Testing Process: (1) Query Generation (2) Query Mutation (3) Result Analysis

- Query Generation: focus on **Cypher**, diverse in graph patterns, incremental
- Transformation Combinations: helps to generate more complex graph queries



*: GraphGenie is available at https://github.com/YuanchengJiang/GraphGenie

Graph Query Transformations (GQT)

- ◐ Structure-GQT: mutations considering graph patterns

ID	Rule Name	Class	Type	Transformation	Example (In *Cypher*)
01	SymmetricPattern	◐	Equivalent	Replace graph pattern with a symmetric one	MATCH (A~~:MOVIE~~)--(B**:MOVIE**) RETURN COUNT(~~A~~**B**);
02	UnfoldCyclicPattern	◐	Equivalent	Unfold cyclic pattern via adding predicate	MATCH (A)--(B:MOVIE)--(C~~A~~) **WHERE A=C** RETURN COUNT(A);
03	PatternPartition	◐	Equivalent	Split graph pattern to disjoint sub-patterns	MATCH (A)-->(B:MOVIE)**, (B:MOVIE)**-->(C) RETURN COUNT(A);
04	AddEdgeDirection	◐	Variant	Add edge direction to undirected edge	MATCH (A)--**>**(B:MOVIE) WHERE B.YEAR=2012 RETURN COUNT(A);
05	SpanningSubgraph	◐	Variant	Spanning subgraph by deleting edges	MATCH (A)-->(B:MOVIE)-->(C)~~, (A)-->(C)~~ RETURN COUNT(A);
06	InducedSubgraph	◐	Variant	Induced subgraph by deleting vertices	MATCH (A)--(B:MOVIE)--(C)~~--(D:ACTOR)~~ RETURN COUNT(A);
07	ExpandPattern	◐	Variant	Expand graph pattern by adding nodes	MATCH (A)--(B:MOVIE)--(C:MOVIE)**--(D)** RETURN COUNT(A);

Graph Query Transformations (GQT)

- ◐ Structure-GQT: mutations considering graph patterns
- ◑ Property-GQT: mutations considering graph properties

ID	Rule Name	Class	Type	Transformation	Example (In *Cypher*)
01	SymmetricPattern	◐	Equivalent	Replace graph pattern with a symmetric one	MATCH (A~~:MOVIE~~)--(B**:MOVIE**) RETURN COUNT(~~A~~**B**);
02	UnfoldCyclicPattern	◐	Equivalent	Unfold cyclic pattern via adding predicate	MATCH (A)--(B:MOVIE)--(C~~A~~) **WHERE A=C** RETURN COUNT(A);
03	PatternPartition	◐	Equivalent	Split graph pattern to disjoint sub-patterns	MATCH (A)-->(B:MOVIE)**, (B:MOVIE)**-->(C) RETURN COUNT(A);
04	AddEdgeDirection	◐	Variant	Add edge direction to undirected edge	MATCH (A)--**>**(B:MOVIE) WHERE B.YEAR=2012 RETURN COUNT(A);
05	SpanningSubgraph	◐	Variant	Spanning subgraph by deleting edges	MATCH (A)-->(B:MOVIE)-->(C)~~, (A)-->(C)~~ RETURN COUNT(A);
06	InducedSubgraph	◐	Variant	Induced subgraph by deleting vertices	MATCH (A)--(B:MOVIE)--(C)~~--(D:ACTOR)~~ RETURN COUNT(A);
07	ExpandPattern	◐	Variant	Expand graph pattern by adding nodes	MATCH (A)--(B:MOVIE)--(C:MOVIE)**--(D)** RETURN COUNT(A);
08	AddNodeLabel	◑	Variant	Add node label to existing node	MATCH (A**:USER**)--(B:MOVIE) WHERE NOT A=B RETURN COUNT(A);
09	AddEdgeType	◑	Variant	Add edge type to existing edge	MATCH (A:USER)-**[R:RATED]**-(B:MOVIE) RETURN COUNT(A);
10	MoveLabelPredicate	◑	Equivalent	Move node label to the predicate	MATCH (A~~:USER~~)--(B:MOVIE) **WHERE A:USER** RETURN COUNT(A);
11	CountIdProperty	◑	Equivalent	Count the node id property	MATCH (A:USER)--(B:MOVIE)-->(C) RETURN COUNT(**ID(**A**)**);
12	CountOtherName	◑	Equivalent	Count other name in the same path	MATCH (A:USER)--(B:MOVIE)-->(C) RETURN COUNT(~~A~~**C**);

Graph Query Transformations (GQT)

- ◐ Structure-GQT: mutations considering graph patterns
- ◑ Property-GQT: mutations considering graph properties
- ○ Non-GQT: mutations on other parts of graph queries

ID	Rule Name	Class	Type	Transformation	Example (In *Cypher*)
01	SymmetricPattern	◐	Equivalent	Replace graph pattern with a symmetric one	MATCH (A~~:MOVIE~~)--(B:MOVIE) RETURN COUNT(~~A~~B);
02	UnfoldCyclicPattern	◐	Equivalent	Unfold cyclic pattern via adding predicate	MATCH (A)--(B:MOVIE)--(C~~A~~) WHERE A=C RETURN COUNT(A);
03	PatternPartition	◐	Equivalent	Split graph pattern to disjoint sub-patterns	MATCH (A)-->(B:MOVIE), (B:MOVIE)-->(C) RETURN COUNT(A);
04	AddEdgeDirection	◐	Variant	Add edge direction to undirected edge	MATCH (A)-->(B:MOVIE) WHERE B.YEAR=2012 RETURN COUNT(A);
05	SpanningSubgraph	◐	Variant	Spanning subgraph by deleting edges	MATCH (A)-->(B:MOVIE)-->(C)~~, (A)-->(C)~~ RETURN COUNT(A);
06	InducedSubgraph	◐	Variant	Induced subgraph by deleting vertices	MATCH (A)--(B:MOVIE)--(C)~~--(D:ACTOR)~~ RETURN COUNT(A);
07	ExpandPattern	◐	Variant	Expand graph pattern by adding nodes	MATCH (A)--(B:MOVIE)--(C:MOVIE)--(D) RETURN COUNT(A);
08	AddNodeLabel	◑	Variant	Add node label to existing node	MATCH (A:USER)--(B:MOVIE) WHERE NOT A=B RETURN COUNT(A);
09	AddEdgeType	◑	Variant	Add edge type to existing edge	MATCH (A:USER)-[R:RATED]-(B:MOVIE) RETURN COUNT(A);
10	MoveLabelPredicate	◑	Equivalent	Move node label to the predicate	MATCH (A~~:USER~~)--(B:MOVIE) WHERE A:USER RETURN COUNT(A);
11	CountIdProperty	◑	Equivalent	Count the node id property	MATCH (A:USER)--(B:MOVIE)-->(C) RETURN COUNT(ID(A));
12	CountOtherName	◑	Equivalent	Count other name in the same path	MATCH (A:USER)--(B:MOVIE)-->(C) RETURN COUNT(~~A~~C);
13	DisjointPredicate	○	Equivalent	Split predicate into disjoint parts	MATCH (A) WHERE A.P>0 ~~AND~~WITH * WHERE A.Q>0 COUNT(A);
14	RedundantPredicate	○	Equivalent	Append alway-true condition to predicate	MATCH (A:USER)–(B:MOVIE) WHERE NOT A=B RETURN COUNT(A);
15	RenameVariables	○	Equivalent	Rename node or edge variables	MATCH (~~A~~N)--(~~B~~M:MOVIE) WHERE ~~A~~N:USER RETURN COUNT(~~A~~N);
16	AddCallWrapper	○	Equivalent	Return results by calling the function	CALL { MATCH (A:USER) RETURN COUNT(A) AS X } RETURN X;

Effectiveness

Effectiveness in discovering unknown bugs in mature graph database systems?

	Logic Bugs			Internal Errors	
GDBMS	**Unconfirmed**	**Confirmed**	**Fixed**	**Fixed**	**Total**
Neo4j	0	0	2	3	5
RedisGraph	1	3	1	0	5
AgensGraph	0	0	3	0	3
Gremlin-DBs	6	0	0	0	6
Total	7	3	6	3	19

Effectiveness

Effectiveness in discovering unknown bugs in mature graph database systems?

	Logic Bugs			Internal Errors	
GDBMS	**Unconfirmed**	**Confirmed**	**Fixed**	**Fixed**	**Total**
Neo4j	0	0	2	3	5
RedisGraph	1	3	1	0	5
AgensGraph	0	0	3	0	3
Gremlin-DBs	6	0	0	0	6
Total	7	3	6	3	19

Logic Bug via Symmetric Pattern in RedisGraph

```
Q: MATCH (a:A)-[*1..2]-(b:B) return count(1);
   // Result: 204    Response Time: 0.76ms
Q⊜: MATCH (b:B)-[*1..2]-(a:A) return count(1);
   // Result: 238    Response Time: 0.75ms
```

https://github.com/RedisGraph/RedisGraph/issues/2865

Graph Pattern: viariable length patterns having endpoints with (a:A) and (b:B)

Fixed. Caused by incorrect logic to stop expanding a path upon detecting a cycle.

Effectiveness

Effectiveness in discovering unknown bugs in mature graph database systems?

	Logic Bugs			Internal Errors	
GDBMS	**Unconfirmed**	**Confirmed**	**Fixed**	**Fixed**	**Total**
Neo4j	0	0	2	3	5
RedisGraph	1	3	1	0	5
AgensGraph	0	0	3	0	3
Gremlin-DBs	6	0	0	0	6
Total	7	3	6	3	19

Logic Bug via Symmetric Pattern in RedisGraph

```
Q: MATCH (a:A)-[*1..2]-(b:B) return count(1);
   // Result: 204    Response Time: 0.76ms
Q⊜: MATCH (b:B)-[*1..2]-(a:A) return count(1);
   // Result: 238    Response Time: 0.75ms
```

https://github.com/RedisGraph/RedisGraph/issues/2865

Graph Pattern: viariable length patterns having endpoints with (a:A) and (b:B)

Fixed. Caused by incorrect logic to stop expanding a path upon detecting a cycle.

Graph Pattern: partition the pattern (a)->(b) into two paths (a) and (a)->(b)

Fixed. Caused by columns not visible when involving variable length edge

Logic Bug via Pattern Partition in AgensGraph

```
Q: MATCH (a)-[*1..1]->(b) RETURN count(a);
   // Result: 100
Q⊜: MATCH (a),(a)-[*1..1]->(b) RETURN count(a);
   // ERROR:  column "a" does not exist
```

https://github.com/bitnine-oss/agensgraph/issues/609

Does Graph Pattern Work?

Q:
```
MATCH (a)-[]-(a) RETURN count(a);
      // Base Query Result: 200
MATCH (a)-[]-(a) WHERE id(a)>=1.0 RETURN count(a);
      // (TLP-True) Result: 200
MATCH (a)-[]-(a) WHERE NOT id(a)>=1.0 RETURN count(a);
      // (TLP-False) Result: 0
MATCH (a)-[]-(a) WHERE id(a)>=1.0 IS NULL RETURN count(a);
      // (TLP-Null) Result: 0
```
$Q^{\ominus}$:
```
MATCH (a)-[]-(b) WHERE a=b RETURN count(a);
      // (GraphGenie) Result: 16
```

GDBMeter

We analyze fixed bugs found by us and use GDBMeter's approach to detect them. **Out of 9 bugs that are applicable to Ternary Logic Partition, GDBMeter was able to detect only 3 bugs.**

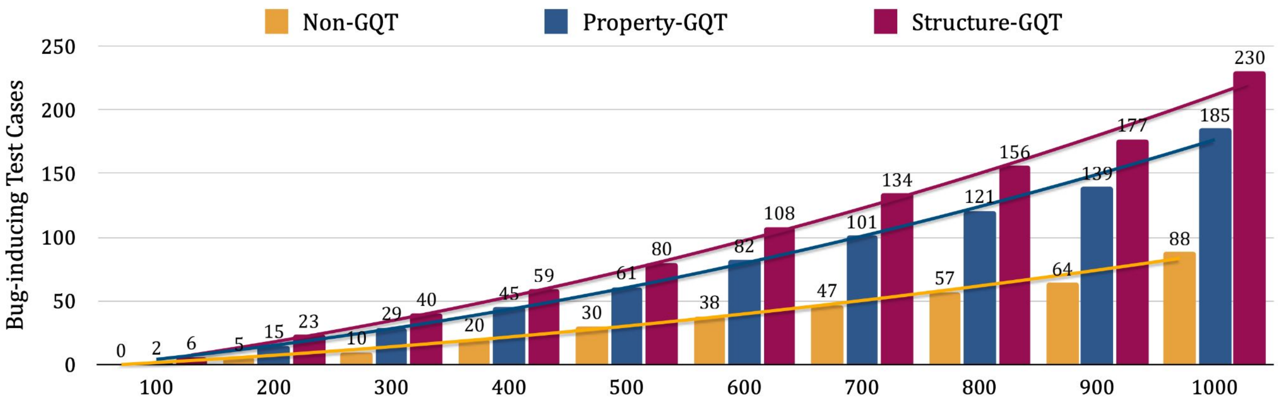
Does Graph Pattern Work?

Q:
```
MATCH (a)-[]-(a) RETURN count(a);
     // Base Query Result: 200
MATCH (a)-[]-(a) WHERE id(a)>=1.0 RETURN count(a);
     // (TLP-True) Result: 200
MATCH (a)-[]-(a) WHERE NOT id(a)>=1.0 RETURN count(a);
     // (TLP-False) Result: 0
MATCH (a)-[]-(a) WHERE id(a)>=1.0 IS NULL RETURN count(a);
     // (TLP-Null) Result: 0
```
$Q^{\ominus}$:
```
MATCH (a)-[]-(b) WHERE a=b RETURN count(a);
     // (GraphGenie) Result: 16
```

GDBMeter

We analyze fixed bugs found by us and use GDBMeter's approach to detect them. **Out of 9 bugs that are applicable to Ternary Logic Partition, GDBMeter was able to detect only 3 bugs.**

Non-GQT rules are effective in finding bug-inducing test cases while **using GQT rules facilitates uncovering more bug-inducing cases in testing GDBMS.**



Performance Issues? No Problem!

Graph Query Transformations:

We reuse transformations for logic bugs, then redesign the test oracles

Test Oracle (e.g. for equivalent mutated queries):

The difference of execution time should less than the threshold T[=].

$$max(time(Q), time(Q^{\boxed{=}})) \leq min(time(Q), time(Q^{\boxed{=}})) \times T^{\boxed{=}}$$

(T[=] is customizable, we set it as 5x)

Performance Issues? No Problem!

Graph Query Transformations:

We reuse transformations for logic bugs, then redesign the test oracles

Test Oracle (e.g. for equivalent mutated queries):

The difference of execution time should less than the threshold T[=].

$$max(time(Q), time(Q^{\boxed{=}})) \leq min(time(Q), time(Q^{\boxed{=}})) \times T^{\boxed{=}}$$

(T[=] is customizable, we set it as 5x)

Positive Feedbacks about Performance Issues from Neo4J Developers

Bug ID	Status	Time(Q)	Time(Q')	Developer Feedback
12973	Fixed	4642011ms	5984ms	A fix will come with the next release
13034	Fixed	100ms	201384ms	A fix will come with the next release
13010	Confirmed	77ms	12147ms	Bad plan but low priority to optimize
12957	Confirmed	13933ms	22ms	A suboptimal plan in old version
13003	Intended	165547ms	332ms	Query plan is suboptimal but intended
13033	Intended	1402ms	16585ms	Inaccurate estimated rows and bad plan

Thank You!

Check Our Paper:

https://yuanchengjiang.github.io/docs/GraphGenie-ICSE24.pdf

Try GraphGenie:

https://github.com/YuanchengJiang/GraphGenie

NUS National University of Singapore