Search-Based Test Data Generation for SQL Queries

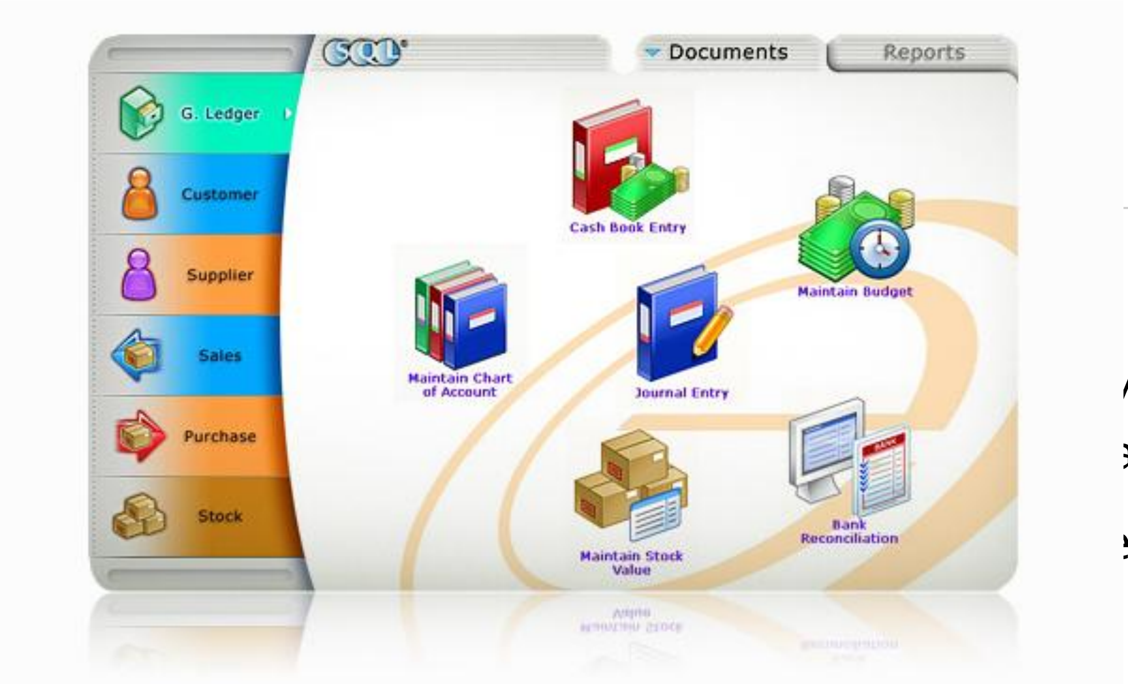
Jeroen Castelein[1], Maurício Aniche[1], Mozhan Soltani[1], Annibale Panichella[1,2], Arie van Deursen[1]
[1]Delft University of Technology, [2]Snt-University of Luxembourg (ICSE 2018)

20184228 Seah Kim
20184400 Jeonggwan Lee
20186473 Liu Lingjun
20186505 Nick Heppert

Table of Contents

SQL is everywhere!

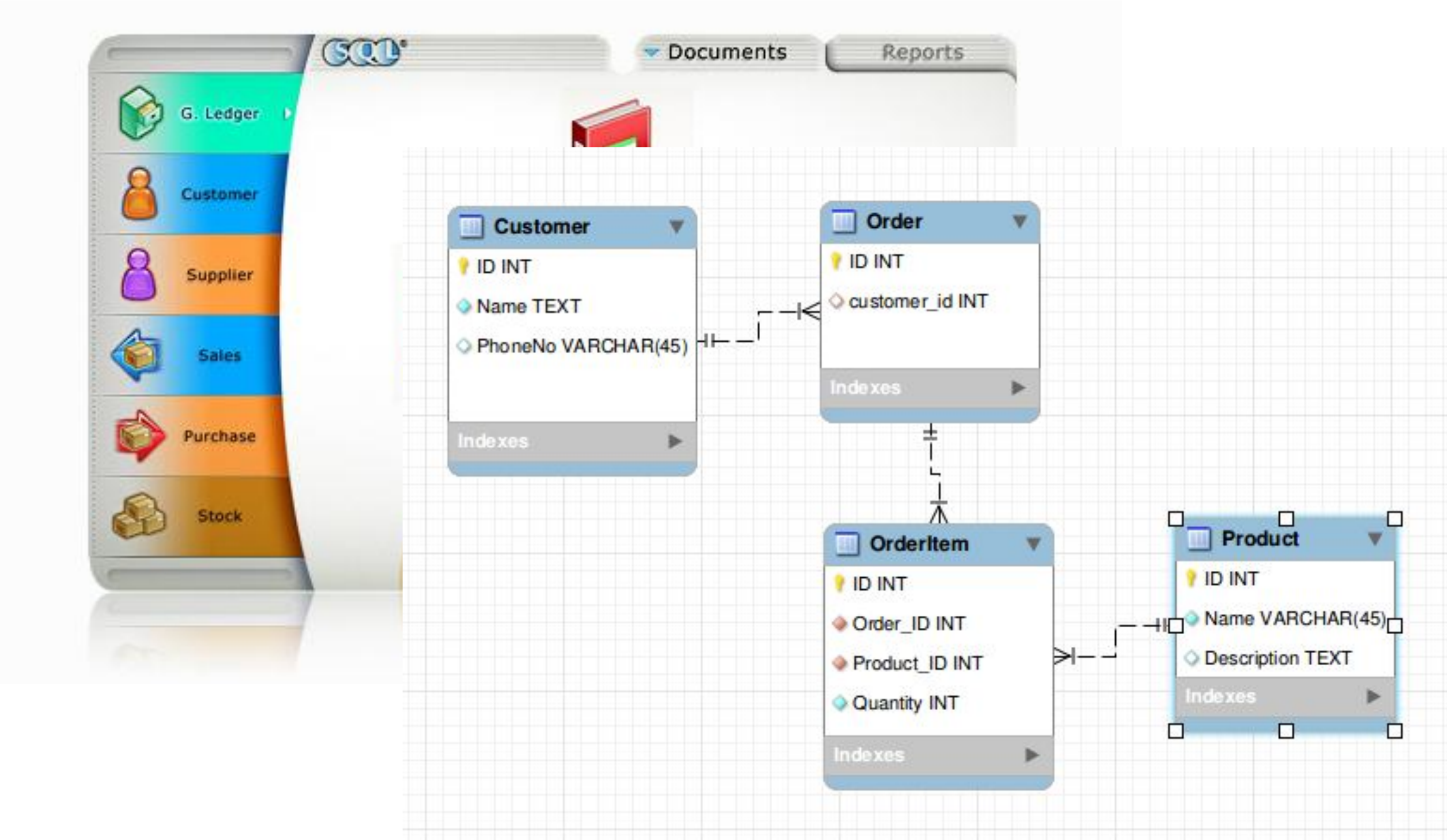
- Database-centric systems strongly rely on *SQL Queries* to manage and manipulate their data.
- Accounting, Customer management, Medical Information etc.



/ rely on *SQL Queries* to

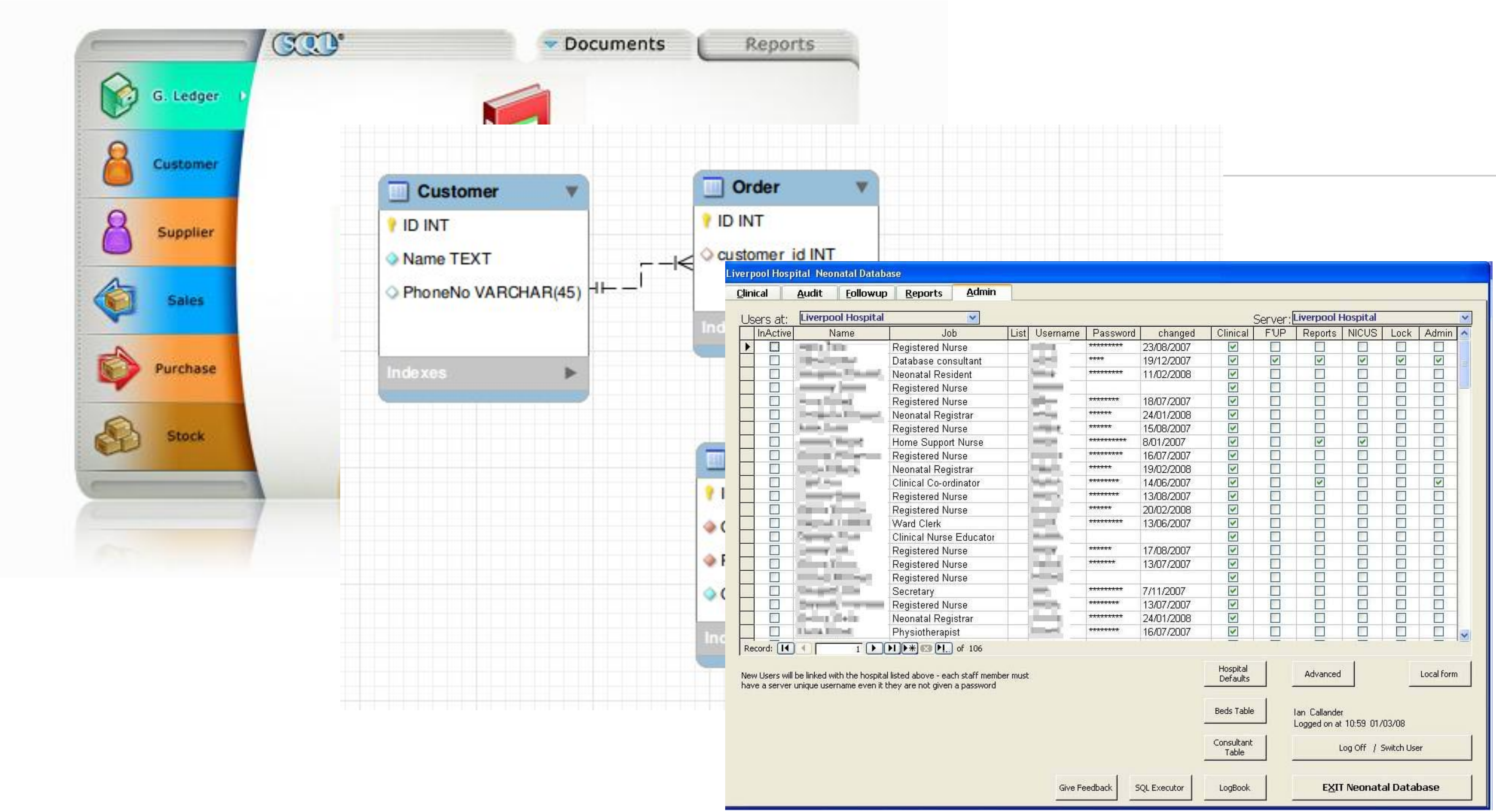
a.

ent, Medical Information



ueries to

ormation


SQL
Documents
Reports
G. Ledger
Customer
Supplier
Sales
Purchase
Stock
Customer
ID INT
Name TEXT
PhoneNo VARCHAR(45)
Indexes
Order
ID INT
customer_id INT
Liverpool Hospital Neonatal Database
Clinical
Audit
Followup
Reports
Admin
Users at:
Liverpool Hospital
Server:
Liverpool Hospital
InActive
Name
Job
List
Username
Password
changed
Clinical
F'UP
Reports
NICUS
Lock
Admin
Registered Nurse
Database consultant
Neonatal Resident
Neonatal Registrar
Home Support Nurse
Clinical Co-ordinator
Ward Clerk
Clinical Nurse Educator
Secretary
Physiotherapist
Record:
of 106
New Users will be linked with the hospital listed above - each staff member must have a server unique username even it they are not given a password
Hospital Defaults
Advanced
Local form
Beds Table
Ian Callander
Logged on at 10:59 01/03/08
Consultant Table
Log Off / Switch User
Give Feedback
SQL Executor
LogBook
EXIT Neonatal Database

Related Work

- 5 similar tools in total
- Problems:
  - Constraint Solver
  - Restriction to certain constraints
    - Subqueries
    - String predicates

Goal: Generate Data for SQL-Queries in a Search-Based Manner

- Limits of constraints solvers:
  - No support for strings
  - Mapping from SQL to Constraints highly demanding (so no advanced SQL)
- Solution:
  - Use a search based method to move through solution space
  - Use fully functioning SQL database engine → support of whole standard SQL syntax

Problem Definition

Given

- Query
- Database Scheme
- Time Budget

Use

- Random (as a baseline)
- Biased Random
- Genetic Algorithm

Database that covers the query sufficiently

Problem Space

- Assume a database scheme DS
- Given a query Q
- We want to cover all of its coverage targets (by SQLFpc*)

$$CT = \{ct_1, ct_2, \ldots, ct_n\}$$

- Example: $DS = \{T_1 = \{c_1 = \text{ID}, c_2 = \text{Category}\}\}$

> - Q = "**SELECT** * **FROM** Product **WHERE** (Category = 'Toy')"
> - CT = {"**SELECT** * **FROM** Product **WHERE** (Category = 'Toy')",
>   "**SELECT** * **FROM** Product **WHERE** **NOT**(Category = 'Toy')"}

[SQLFpc*]Tuya, Javier, María José Suárez-Cabal, and Claudio De La Riva. "Full predicate coverage for testing SQL database queries." *Software Testing, Verification and Reliability* 20.3 (2010): 237-288.

Solution Space

- Solution $S = \{T_1, T_2, \ldots, T_m\}$
- Table $T_i = \{R_1, R_2, \ldots, R_k\}$
- Row $R_i = < V_1, V_2, \ldots, V_c >$
  - c is the number of columns given by the DB scheme
- Example $DS = \{T_1 = \{c_1 = \text{ID}, c_2 = \text{Category}\}\}$

ct_1 = “ `SELECT * FROM Product WHERE (Category = 'Toy')`”

$S = \{T_1 = \{< 1, \text{"}Toy\text{"} >\}\}$ ⟺

id	category
1	Toy

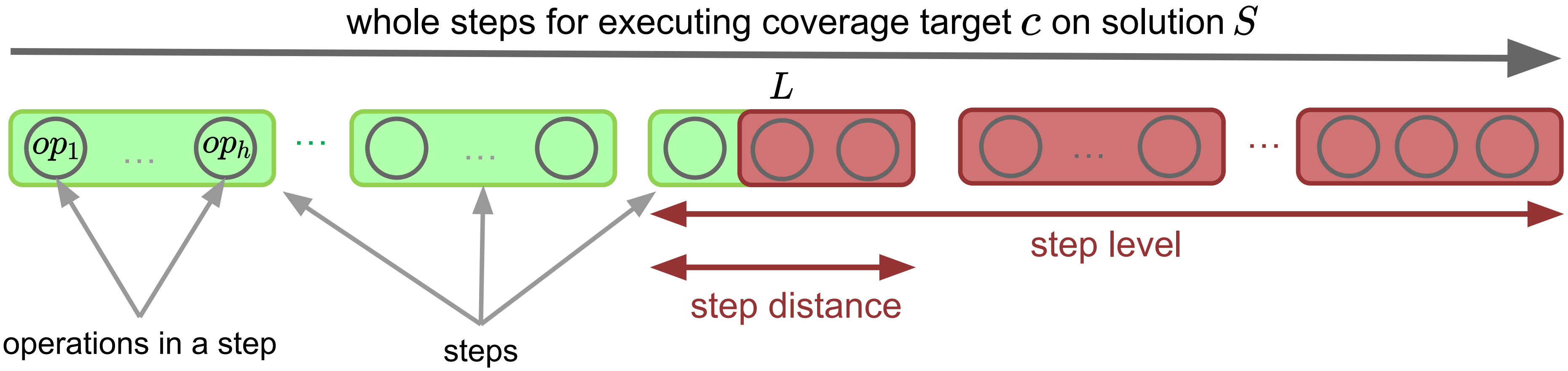
Fitness Function
(Query Execution Plan)

```
SELECT *
FROM Cars
JOIN Tires
ON Cars.tire_id = Tires.id
WHERE model = ‘Ferrari’ and color = ‘red
```

Step 1

Step 2

Fitness Function
(High level view)



$$F(c, S) = step\_level(c, S) + step\_distance(S, L)$$

Fitness Function
(step_distance)

- $$\text{step\_distance}\,(S, L) = \phi\left[\sum_{i=1}^{h} \phi\left[dist(S, op_i)\right]\right]$$

  where $\phi(x) = \frac{x}{1+x}$
- Various distance operators op_i possible:
  - Comparison operators (=, <>, >, >=, < and <=)
  - Logical operators (e.g. **`AND`**)
  - SQL operators (**`BETWEEN`**, **`IS (NOT) NULL`**, **`IN`**, **`LIKE`** and **`EXISTS`**)

Genetic Algorithm

- Generate population (randomly generated solutions)

repeat

- Tournament selection based on fitness function for next generation
- Uniform crossover with two solutions
- Mutation on one table when a solution is mutated

→ Selection-reproduction cycle ends if one achieves the coverage target

Mutation

- Delete
- Insert
  - duplicate one existing row
  - add a newly generated row
- Change
  - integers: delta mutation
  - booleans: flipping values
  - If the column is nullable, one of the value is set to NULL with some probability

id	category
1	Toy
2	Food
3	Car

↓

id	category
1	Toy
2	Food

Mutation

- Delete
- Insert
  - duplicate one existing row
  - add a newly generated row
- Change
  - integers: delta mutation
  - booleans: flipping values
  - If the column is nullable, one of the value is set to NULL with some probability

id	category
1	Toy
2	Food

↓

id	category
1	Toy
2	Food
2	Food

Mutation

- Delete
- Insert
  - duplicate one existing row
  - add a newly generated row
- Change
  - integers: delta mutation
  - booleans: flipping values
  - If the column is nullable, one of the value is set to NULL with some probability

id	category
1	Toy
2	Food

↓

id	category
1	Toy
2	Food
3	Car

Mutation

- Delete
- Insert
  - duplicate one existing row
  - add a newly generated row
- Change
  - integers: delta mutation
  - booleans: flipping values
  - If the column is nullable, one of the value is set to NULL with some probability

id	category
1	Toy
2	Food
3	Car

↓

id	category
1	Toy
2	Food
11	Car

Seeding Strategy

- Insert values from a seeding pool into population with some probability

- Seeding pool contains all constants in the query

```
SELECT * FROM Cars WHERE model = ‘Ferrari’ AND id = 10
```

<Random Search>

generate random solutions and repeat until the coverage target is satisfied

<Biased Random Search>

random search with seeding strategy

“

EvoSQL

Input

- Query
- Database Scheme
- Time Budget

Output

- Test data for SQLFpc* coverage target

[SQLFpc*]Tuya, Javier, María José Suárez-Cabal, and Claudio De La Riva. "Full predicate coverage for testing SQL database queries." *Software Testing, Verification and Reliability* 20.3 (2010): 237-288.

Collected queries

totally 2,135 queries (Alura, EspCRM, SuiteCRM, ERPNext)

Application	Total Queries	Queries w/o bad syntax	Unique Queries	Final Queries
Alura	554	494	258	**249**
EspoCRM	151	149	40	**40**
SuiteCRM	709	704	280	**279**
ERPNext	18,454	17,761	1,631	**1,567**
Total	19,868	19,108	2,209	**2,135**

Table 1: Queries collected per application

Configuration of the Search Parameters

The exercised configurations (in a total of 108 different combinations)

(1) NULL mutation (p_{null}) = {0.01, 0.10, 0.50},

(2) Inserting, deleting mutation (p_m) = {⅓, ⅙}

(3) Changing mutation (p_c) = {1/n, 1},

where n is the number of rows in the mutated table.

(4) seeding = {0.01, 0.10, 0.50}

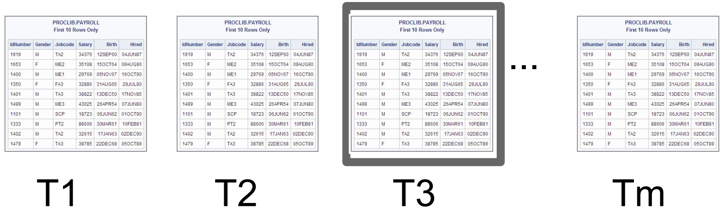
(5) crossover = {0.0, 0.6, 0.75}

We selected the configuration with the ***smallest execution time.***

Configuration of the Search Parameters

- Population size = 50
- Tournament size = 4
- NULL mutation (p_{null}) = 0.1
- Inserting, deleting mutation (p_m) = ⅓
- Changing mutation (p_c) = 1
- Seeding = 0.5
- Crossover = 0.75
- Cloning from previous target population = 0.6

$$S = \{T_1, T_2, \ldots, T_m\}$$



$p_{table} = 1/K$, (*K tables*)

Experimental Procedure

- executed the three approaches(Random, Biased, GA) across the 10 executions.
- manually removed infeasible coverage targets(*"A > 10 and A < 10"?*) (127 out of 12,991)
- set time budget as 30 minutes for one query

RQ1 : What is the coverage achieved?

	fully covered	less than 10 cover. target, fully covered	10~20 cover. target, fully covered	more than 20 cover. target, fully covered	achieved 0% coverage
Random Search	6.5%	7.3%	0%	0%	28.33%
Biased search	90%	96.54%	49.36%	7.04%	0.05%
GA	98.64%	99.999%	94.93%	74.64%	0%

fully covered : covered in all 10 executions

RQ2 : What is the performance(speed)?

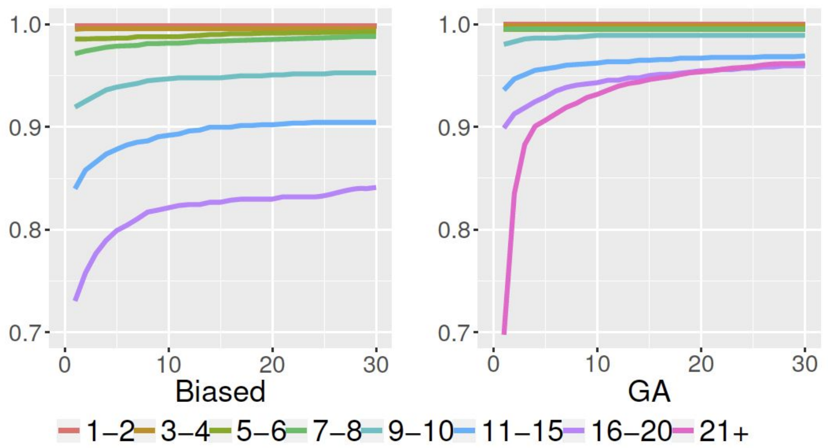


Figure 3: The average coverage of a SQL query (Y axis) in a given time budget (X axis, in minutes) controlled by the number of coverage targets. Figure better visualized in colors.

	1st quantile		Median		3rd quantile	
	Biased	GA	Biased	GA	Biased	GA
1-2	0.03	0.15	0.05	0.22	0.07	0.37
3-4	0.09	0.2	0.15	0.28	0.36	0.46
5-6	0.17	0.3	0.34	0.41	0.56	0.58
7-8	1.23	0.59	2.25	0.88	7.64	1.45
9-10	1.26	0.9	5.95	1.48	100.80	2.49
11-15	7.47	2.21	74.04	3.65	844.10	8.80
16-20	83.1	7.33	844.00	15.69	1539.00	41.79
21+	1027	81.36	1688.00	155.90	1800.00	581.90

Table 4: Descriptive statistics of the biased and GA approaches' average runtime (in seconds).

RQ3 : Why not 100% coverage?

- ✓ How?
  - generated a decision tree model
  - classified whether coverage target is likely to be covered or not
- ✓ Results (accuracy - random : 85.93% / BS : 90.27% / GA : 91.88%)
  - The random search can not deal with JOINs and strings.
  - The biased search suffers from queries with many predicates.
  - The size of the query impacts the GA.

Analysis : BS vs GA

Biased Search, showed **higher efficiency** for simple queries

✓ Limit on Biased Random Search : complex string manipulation functions

```
●  SELECT * FROM product WHERE length(name) = 12 AND
   left(name, 5) = 'REFRI' AND right(name, 7) = 'GERATOR'
●  SELECT * FROM product WHERE reverse(name) = 'ROTAREGIRFER'
```

✓ Additional steps in for GA are an overhead for simple queries

Threats to Validity

- ✓ Limited diversity of queries
- ✓ Manual removal of infeasible targets
- ✓ Empirical selection of configurations for GA*

Gordon Fraser and Andrea Arcuri. 2013. Whole test suite generation. IEEE Transactions on Software Engineering 39, 2 (2013)
José Miguel Rojas, Gordon Fraser, and Andrea Arcuri. 2016. Seeding strategies in search-based unit test generation. Software Testing, Veriication and Reliability 26, 5 (2016),

Threats to Validity

Property	0	1 - 2	3 - 4	5 - 6	7 - 8	9 - 10	11 - 15	16 - 20	21+
Predicates	58	1389	495	100	33	11	27	16	6
Joins	1890	189	32	3	17	2	-	1	1
Subqueries	2052	78	3	1	-	-	1	-	-
Functions	1796	291	12	16	2	6	12	-	-
Used columns	60	1369	457	127	43	26	20	13	20
Coverage rules	-	656	382	408	346	114	107	51	71

✓ Limited diversity of queries (not a threat)

✓ Manual removal of infeasible targets → part of our project

✓ Empirical selection of configurations for GA

→ Hyperparameter optimization (e.g. Bayesian Optimization)

Summary

- ✓ This paper, modeled the problem of test data generation for SQL queries as a **search-based problem.**
- ✓ Implemented three different search approaches in tool, named ***EvoSQL*** (random search, biased random search, genetic algorithm)
- ✓ Executed on 2,135 queries extracted from 4 software systems
- ✓ Achieved 98.64% Coverage with GA

Castelein, Jeroen, et al. "Search-based test data generation for SQL queries." *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018.

Backups

“

Distance Operators
(Comparison Operator)

Datatype	Distance
Numbers, Boolean	Standard Branch Distance
Strings	Enhanced Edit Distance (Standard + Character Distance
Dates	Sum of differences for each numerical calendar part

Distance Operators
(SQL Operators 1/2)

Operator	Distance
`BETWEEN`	$lb <= v <= ub$ $\rightarrow lb <= v \text{ AND } v <= ub$
`IS NULL`	eq. to boolean $v = \text{NULL}$
`IN`	$dist(S, op_i) = \min_{e \in \text{list}} \text{abs}\,(v - e)$
`LIKE`	Branch distance for pattern matching

Distance Operators
(SQL Operators 2/2)

Operator	Distance
`EXISTS`	Operation distance of subquery
`JOIN`	Branch distance for equality operator between columns

Mutation

Change operation

- Floating-point numbers → polynomial mutation
- Integers -> delta mutation
- Date→ delta mutation to all its calendar parts
- Strings→ adding, removing or replacing characters
- Booleans→ flipping values
- If the column to mutae is nullable, one of the values is set to NULL with some probability

RQ3 : Why not 100% coverage?

- ✓ J48 decision tree
  - classifies whether a coverage target is likely to be covered or not
  - features to the model
    - #of tables
    - SQL query's total # of coverage targets
    - # of SQL predicates or functions

Search-based Test Data Generation for SQL Queries

Jeroen Castelein et al. ICSE 18’

Problem Let R = $\{r_1, \ldots, r_m\}$ *be the set of coverage targets for a query Q according to a coverage criterion. Find test data S that satisfies all coverage targets in R.*

“

Solution $T = \{T1, \ldots, Tn\}$: set of tables

each table $Ti = \{R1, \ldots, Rk\}$: composed of rows

each row $Rj = \{V1, \ldots, Vc\}$

(c = number of columns in Ti)

```
SELECT items .* FROM invoice
JOIN items ON invoice.id =
items.invoiceid
WHERE amount > 1000 OR taxFree =
true
```

```
SELECT items .* FROM invoice
JOIN items ON invoice.id =
items.invoiceid
WHERE amount > 1000 OR taxFree =
true
```

How should we test this query?

“

Fitness Function

Step 1

```
SELECT *
FROM Cars
JOIN Tires
ON Cars.tire_id = Tires.id
WHERE model = 'Ferrari'
```

Step 2

If Step 1 doesn't produce an output,
database stops its execution before proceeding to Step 2

Castelein, Jeroen, et al. "Search-based test data generation for SQL queries." *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018.

Fitness Function

- ✓ Satisfying coverage target means solution *T* yields a non-empty result. ($T = \{T1, \ldots, Tn\}$ *:* set of tables)
- ✓ If *T* does not cover the given target, measure the distance of *T* to cover the target
  - Counting the number of yet unsatisfied steps (*step level*)
  - Measuring how far T is to satisfy the step where execution stopped (*step distance*)

Castelein, Jeroen, et al. "Search-based test data generation for SQL queries." *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018.

Genetic Algorithm

- ✓ For single coverage target from a query, implement standard GA(with Crossover, Mutation).
- ✓ Using seeding strategies, it uses a seeding pool containing all constants appearing in the query.
- ✓ Post Processing for readability of the generating data.

There are three approaches, Random Search, Biased Random Search(seeding strategy) and, GA.

What can we develop?

“

Inefficiency of GA in EvoSQL

calculating the fitnesses & applying the search operators

Inefficiency of GA in EvoSQL

calculating the fitnesses & applying the search operators

Re-*executed multiple times, once for* ***each coverage target***

Problem: Inefficiency of single-target strategy

1. The order of each coverage target is not optimized

2. Inefficient allocation of the budget might happen, such as infeasible coverage target.

Problem Let R = $\{r_1, \ldots, r_m\}$ *be the set of coverage targets for a query* Q *according to a coverage criterion. Find test data* S *that satisfies all coverage targets in* R*.*

Solution $T = \{T1, \ldots, Tn\}$: set of tables

each table $Ti = \{R1, \ldots, Rk\}$: composed of rows

each row $Rj = \{V1, \ldots, Vc\}$

(c = number of columns in Ti)

Fitness Function

Distance

Operations required to process the query and the order by which they needed to be performed

Function

Operations required to process the query and the order by which they needed to be performed

Testing SQL Queries

- Database-centric systems strongly rely on *SQL Queries* to manage and manipulate their data.
- How developers test SQL queries?

  => Actually, they need to create test data *by hand*.

  But, this can be challenging and time-consuming for complex queries.

Solution

Multi-Objective Optimization!

Multi-Objective Optimization

SELECT * FROM Product WHERE (Category = ‘Toy’)

Multi-Objective Optimization

SELECT * FROM Product WHERE (Category = ‘Toy’)

id	category
1	Toy
2	Food
3	Car

Product

Multi-Objective Optimization

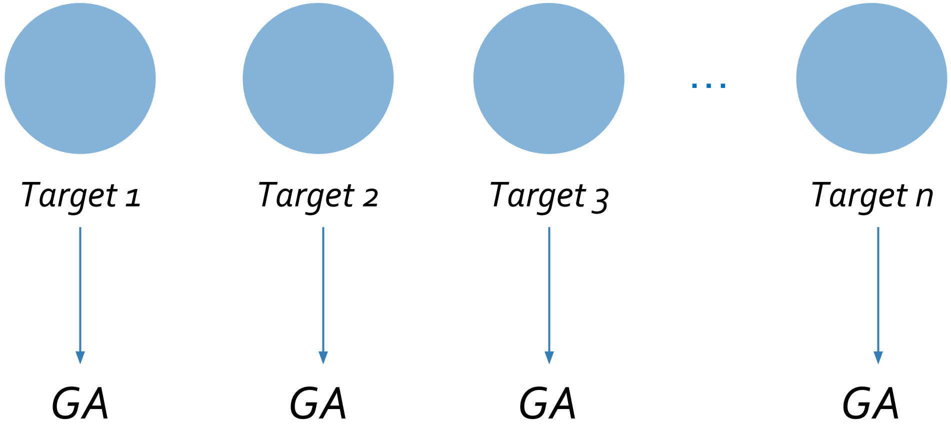
SELECT * FROM Product WHERE (Category != 'Toy')

id	category
1	Toy
2	Food
3	Car

Product

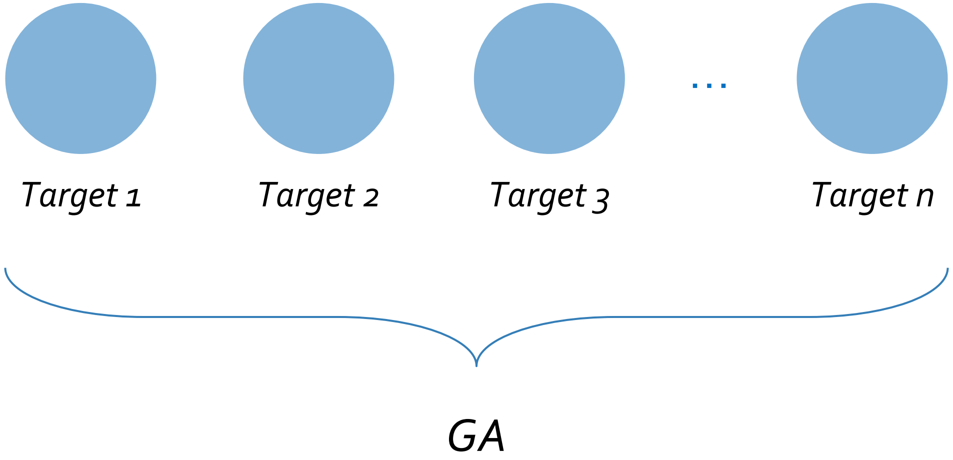
Multi-Objective Optimization

Challenge 1. The order of each coverage target being executed is not optimized



Multi-Objective Optimization

Challenge 1. The order of each coverage target being executed is not optimized

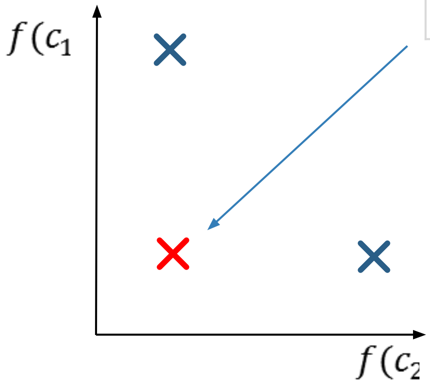


Multi-Objective Optimization

SELECT * FROM Product WHERE (Category = 'Toy')

id	category
1	Car

id	category
1	Toy
2	Car



id	category
2	Toy

$$\begin{cases} \min & f(c_1, t) & c_1 : WHERE\ Category = \text{'Toy'} \\ \min & f(c_2, t) & c_2 : WHERE\ Category\ != 'Toy' \end{cases}$$

Multi-Objective Optimization

Challenge 2. Inefficient allocation of the budget might happen, such as infeasible coverage target.

"A > 10 and A < 10"? → infeasible, no solution from GA

$$\begin{cases} \min & f(c_1, t) & f(c_1, t) \Rightarrow \textit{feasible} \\ \min & f(c_2, t) & f(c_2, t) \Rightarrow \textit{Infeasible} \\ & \dots & \dots \\ \min & f(c_m, t) & f(c_m\ t) \Rightarrow \textit{feasible} \end{cases}$$

Solution from EvoSQL is to manually remove infeasible coverage targets.

RQ1 : What is the coverage achieved?

	all targets, fully covered	less than 10 predicates, fully covered	more than 20 predicates, fully covered	all targets, achieved 0% coverage	partial median coverage
Random Search	6.5%			28.33%	33.75%
Biased search	90%	96.54%	7.04%	0.05%	79.76%
GA	98.64%	99.999%	74.64%	0%	86.66%

fully covered : covered in all 10 executions