A Regular Expression-based DGL for Meaningful Synthetic Data Generation

Kai Cheng
Department of Information Science
Kyushu Sangyo University, Japan
chengk@is.kyusan-u.ac.jp

Outline

1. Background
2. Basics of Regular Expression
3. Extended Regular Expression as a DGL
4. Case Study: TPC-H
5. Implementation
6. Performance Evaluation
7. Conclusion

Synthetic Data Generation

- In development of big data applications, tools for populating the database with meaningful data that satisfy database constraints and statistical distributions are becoming increasingly important

- Real data is typically subject to privacy regulations, synthetic data generation is the only feasible solution in the development phrase and a number of previous work have been done on automated data generation

Synthetic Data

- Dummy Data
  - Random texts that are meaningless or not understandable for human users, for example, F7sA4LK as supplier name
  - Used for performance and/or function tests
- Fake Data
  - Synthetic data that are understandable for human users, for example, Douglas Mitchell as customer name
  - Used both for system test as well as for prototyping

Synthetic Data Generators

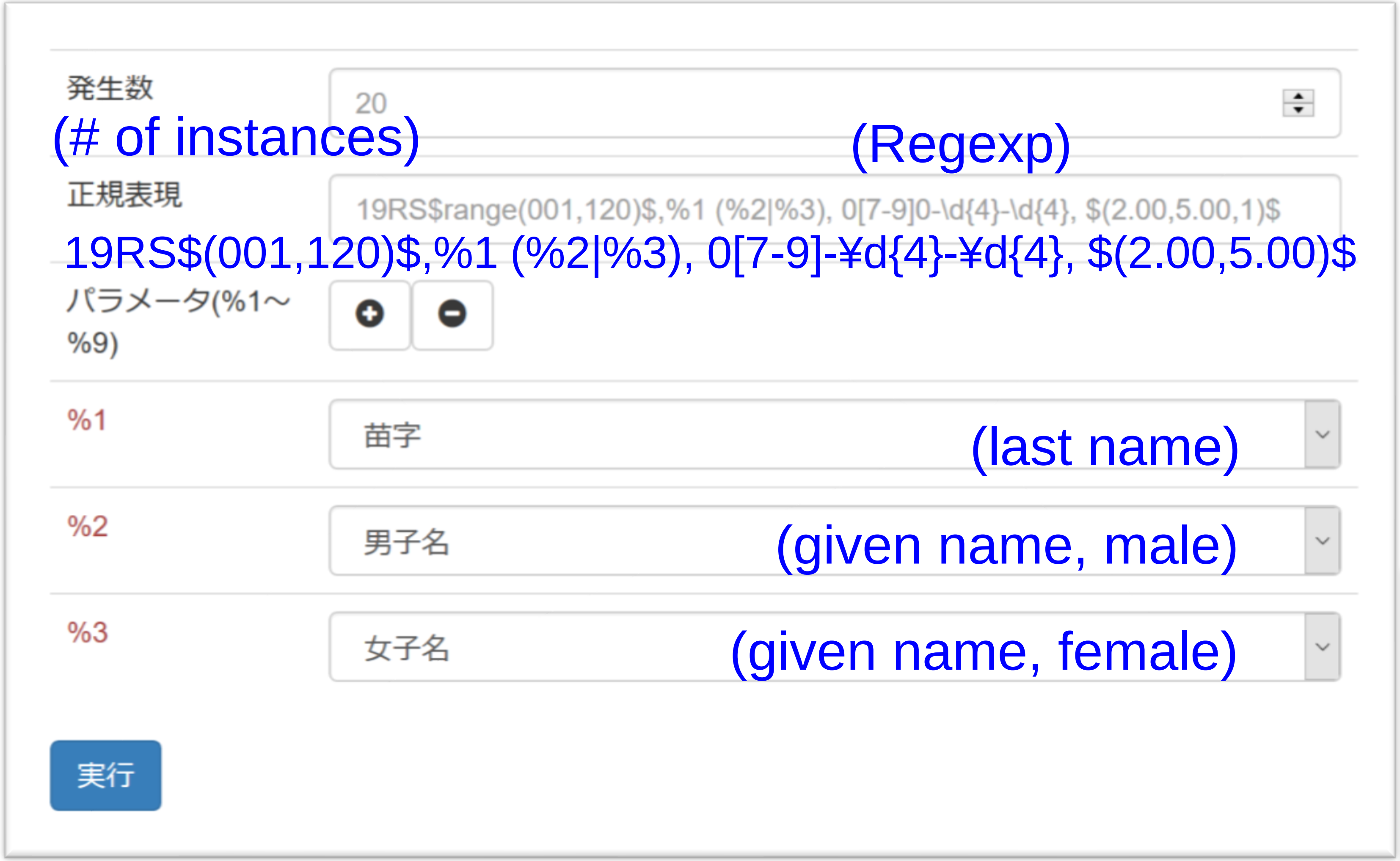
- Ad-hoc Generation Tools
  - Tools or programs that are developed for a specific system, for example, dbgen tool in TPC benchmarks
  - Only a limited types of data can be generated and specific constraints be satisfied

- General-Purpose Generators
  - Data generators with a data generation language（DGL）to define what kind of data to be generated (specification)
  - The generator then automatically produces the required data that satisfy the specification

Contributions

- Propose a data generation language (DGL) based on Regexp for synthetic test data generation
  - Given a regular expression, a set of strings that exactly match it will be generated
- Introduce important extensions to enhance Regexp as a powerful tool for generating realistic datasets
  - Sequential/random number, dictionary etc.

Regexp as DGL



発生数 20

(# of instances)

(Regexp)

正規表現 19RS$range(001,120)$,%1 (%2|%3), 0[7-9]0-\d{4}-\d{4}, $(2.00,5.00,1)$

19RS$(001,120)$,%1 (%2|%3), 0[7-9]-¥d{4}-¥d{4}, $(2.00,5.00)$

%1 苗字 (last name)

%2 男子名 (given name, male)

%3 女子名 (given name, female)

(Ex.) synthetic data for 20 students with

1. name (male / female)
2. mobile numbers, and
3. GPA

(Results)

```
19RS001,松岡 ありさ, 090-3487-8001, 2.69
19RS002,杉本 友菜, 070-5646-8120, 3.01
19RS003,小田 陽貴, 080-8711-1330, 2.04
19RS004,高野 和彦, 070-2916-8370, 4.39
19RS005,前田 優香, 090-4317-1579, 3.60
19RS006,矢野 勘太, 070-0399-9795, 4.65
19RS007,西川 佑樹, 070-4793-0696, 2.25
19RS008,馬場 龍太, 090-8314-3237, 2.77
```

Regular Expression (Regexp)

- A regular expression is a sequence of characters that define patterns in text
- A pattern consists of one or more character literals, operators, or constructs
- Each character in a Regex is either a meta-character, having a special meaning, or a regular character that has a literal meaning
- Regex is often used for pattern-matching, searching pattern in text

Basics of Regexp

Character Classes		Quantifiers	
.	Any char	*****	0 or more times
[ ]	Char in []	**+**	1 or more times
[^]	Char not in [ ]	**?**	0 or 1 time
[a-g]	Char from ‘a’ to ‘g’	**{n}**	Exactly n times
¥d	Digit, char in [0-9]	**{m,n}**	Between m and n
¥w	Word char	**{m,}**	At least m times

Patterns in Regexp

	Regexp	Matched Strings
Tel	090-¥d{4}-¥d{4}	090-6043-4759
Student ID	1[7-9]RS0¥d{2}	17RS066,19RS023
Password	[0-9a-zA-Z]{5,6}	uHiNG,4w7Skv

Limitations / Constraints of Regex

- Limitations
  - Data types and formats not supported
  - Character classes not sufficient for real-world applications
  - Statistical distributions and database constraints

- Constraints
  - Upper bounds of quantifier (「*」, 「+」) should be definite
  - Anchors (「^」, 「$」) only useful in pattern-matching

EXTENDED REGEXP AS DGL

Extended Regexp

- Data Types / Formats
- Primitive Generators
  1. Sequential number generator, *range(min, max, step)*
  2. Random number generator, *random(min, max, dist)*
  3. Random sample generator, *random*$(s_1{:}p_1,..s_n{:}p_n)$
- Dictionary, *Dict.FamilyName*
- Reference, *%1~%9, %a - %z*

Data Types and Formats

- Date Types
  - Integer, e.g., 23000, 001
  - Decimal, e.g., 0.24, 1.00
  - Date, e.g., 2019-12-01, 2019-12
  - Time, e.g., 12:30:00, 12:30
  - Datetime, e.g., 2019-12-01 12:30

- Type and Format Inference
  - Inferring data types and formats from data instances.
  - 001 => Integer with leading 0s, %03d ,
  - 2020-02-19 => Date,

Sequential Number Generator

- A sequential number generator (or sequential generator) produce arithmetic sequence to support identifiers as in most database systems.
- A sequential number generator is defined as follows.

$$range(min, max, step = 1)$$

  - – *min* and *max* specify the range
  - – *step* is the increment to next number, default 1

Examples of Sequential Number

- *range(1,255)* : *Integer*
  - Outputs *1,2,3,4,…,255*
- *range(001,255)* : *Integer with Format ‘%03d’*
  - Outputs *001, 002, 003,…, 255*
- *range(‘2019-1-1’,’2019-8-31’, 7)*: *Dates*
  - Outputs *2019-1-1, 2019-1-8, 2019-1-15*, …

Random Number Generator

- A random number generator (or random generator) generate uniformly at random or follow some statistical distribution

- A random number generator can be any of the following forms

$$random(min, max, dist = 0)$$

  - *min* and *max* define the range where number is generated
  - *dist* indicates the distribution, 0: uniform distribution (default)
    1: normal distribution 2: exponential distribution;
    3: Poisson distribution 4: Zipfian distribution

Examples of Random Number

- *random(1,255)*: *Integer*
  - – Outputs *57,2,124,…,208*

- *random(001,255)*: *Integer with format '%03d'*
  - – Outputs 057, *002, 124,…,208*

- *random('9:00','12-10')*: *Time*
  - – Outputs *10:24, 11:30, 9:40, …*

Random() as a Step Function

- Another form of random generator is a step function defined as follows

```
random(
    [min_1, max_1] : prob_1,
    [min_2, max_2] : prob_2,
      :
    [min_n, max_n] : prob_n,
  )
```

With probability *prob_i,* a number is drawn from [*min_i, max_i*], *prob_1+prob_2+…prob_n = 1*.

Example of Step Function

```
random(
        [0, 59]:  0.3,
        [60, 69]: 0.2,
        [70, 79]: 0.25,
        [80,100]: 0.25
 )
```

- Outputs *74, 23, 89, 66, 95, …*

Random Sample Generator

- A random sample generator (or sample generator) is a more general and powerful generator which takes any sets as input, instead of ranges as in random generator

```
random(
   set_1 : prob_1,
   set_2 : prob_2,
     :
   set_n : prob_n,
 )
```

with probability prob_i, a number drawn from set_i, (i=1, 2,…)

Example of Sample Generator

```
random(
        [S]:  0.10,
        [AB]: 0.35,
        [C]:  0.40,
        [DE]: 0.15
 )
```

Outputs A, C, S, D, B, …

Dictionary(1/2)

- In standard regular expressions, only ASCII characters can be used. We introduce *dictionary* to extend the vocabulary.
- A dictionary is simply a named list of values
  - ***Color***: red, green, blue, yellow, pink, gray, white, black,…
  - ***FamilyName***: Stewart, Morgan, Trump, Bush, Scott, …
  - ***GirlsName***: Lucy, Lily, Sophia, Isabella, Oliva, Alice, …
  - ***BoysName***: James, Oliver, Benjamin, Jackson, Henry, …
  - ***StateName***: Arizona, California, Florida, Illinois, Iowa, …
  - ***StateAbbr***: AZ, CA, FL, IL,IA, KY, MD, …

Dictionary(2/2)

- Dictionaries are objects maintained externally in databases or files and can be referenced by **Dict** wrapper,
  - For example, *Dict.Color, Dict.FamilyName*.
- Dictionaries can be imported by wrapping the values in an existing database, which is useful in generating foreign keys in a table
- Dictionaries are used with the reference mechanism

Reference

- References are special symbols in an extended Regex, to refer to predefined dictionary items

- The following symbols are used as references

  %1 ~ %9, %a ~ %z

Example of References

/(%1|%2) %3 , %a/

%1 := Dict.BoysName

%2 := Dict.GirlsName

%3 := Dict.FamilyName

%a := random('1980-1-1’, '1999-12-31')

Outputs:

```
Douglas Mitchell, 1982-4-3
 Jennifer Stewart, 1990-3-14
 Ernest Morgan, 1984-7-4
 Isla Scott, 1999-1-15
 Jessica Simmons, 1988-6-8
 Sophia Moore, 1998-4-17
 Susan Smith, 1997-5-1
```

CASE STUDY: TPC-H

DGL for TPC-H

- We demonstrate that the proposed DGL can be used to generate TPC-H benchmark
- TPC-H consists of separate and individual tables and relationships between columns in these tables
  - CUSTOMER, ITEMS, PART, SUPPLIER, REGION, NATION
  - ORDER, PARTSUPP, LINEITEM
- The data types in TPC-H include
  - Identifier, Integer, [Big] Decimal, Fixed / Variable Text, Date

Dictionaries in TPC-H

- P_TYPE

  (part type, combination of size, coat, material)
  - PartSize={Standard, Small, Medium, Large, Economy, Promo},
  - PartCoat={Anodized, Burnished, Plated, Polished, Brushed}
  - PartMaterial={Tin, Nickel, Brass, Steel, Copper}

/%1¥s%2¥s%3/

%1:=Dict.PartSize
%2:=Dict.PartCoat
%2:=Dict.ParMaterial

Small Plated Brass,
Economy Burnished Copper

Populating Tables in TPC-H

- PART Table (SF: Scaling Factor)
  - – **P_PARTKEY**: identifier, ***range(1, SF*200000)***
  - – **P_NAME**: variable text, size 55, ***/¥w{10,55}/***
  - – **P_MFGR**: fixed text, size 25***,*** ***/¥w{25}/***
  - – **P_BRAND**: fixed text, size 10***,*** ***/¥w{10}/***
  - – **P_TYPE**: variable text, size 25***, /%1¥s¥2¥s%3/,***
    - ***%1:=Dict.PartSize, %2:=Dict.PartCoat, %3:=Dict.PartMaterial***
  - – **P_SIZE:** integer, ***random(1,100)***

Populating Tables in TPC-H

- Foreign Keys:

PS_PARTKEY, PS_SUPPKEY

random(

%1: 35%

%2: 65%

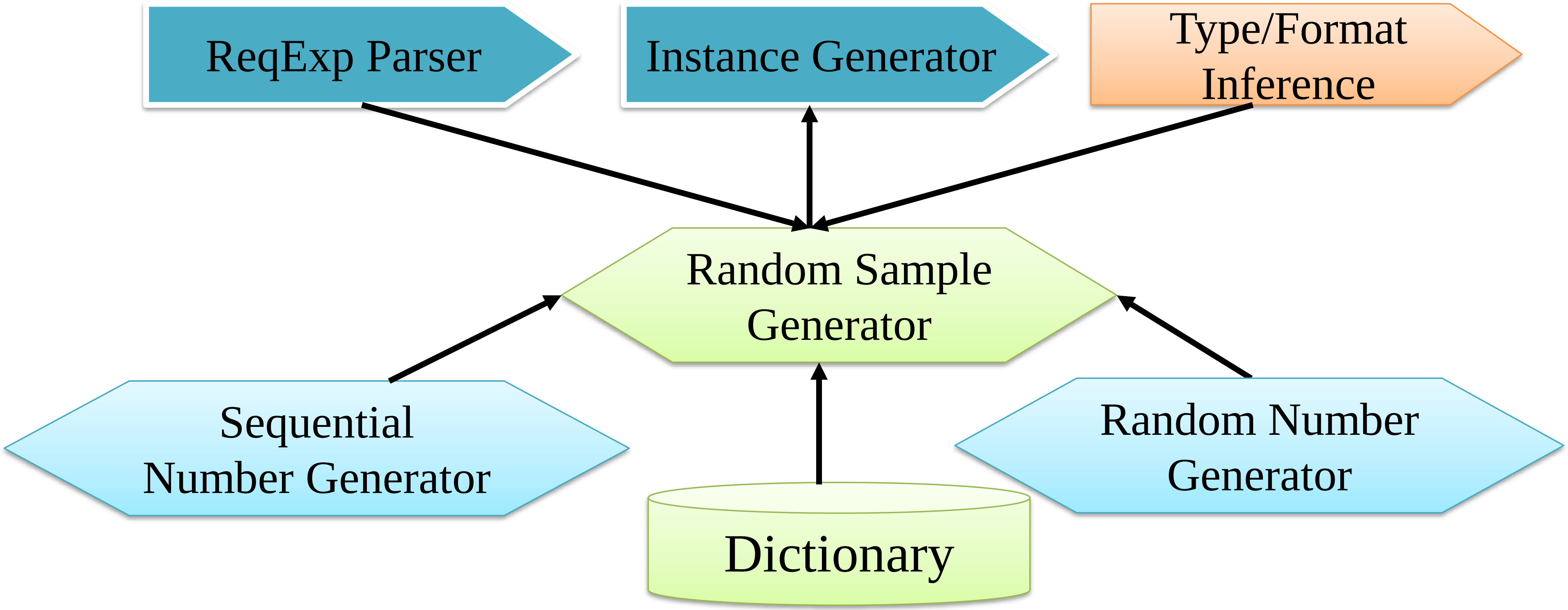
),

%1:=Dict.P_PARTKEY,

%2:=Dict.S_SUPPKEY

IMPLEMENTATION AND PERFORMANCE

Architecture of DGL Implementation



ReqExp Parser

Instance Generator

Type/Format Inference

Random Sample Generator

Sequential Number Generator

Dictionary

Random Number Generator

Efficiency of Data Generation

(1) Partial regular expression caching

- To reduce the cost of parsing (compiling) regular expressions, caching the internal form once compiled can save time cost

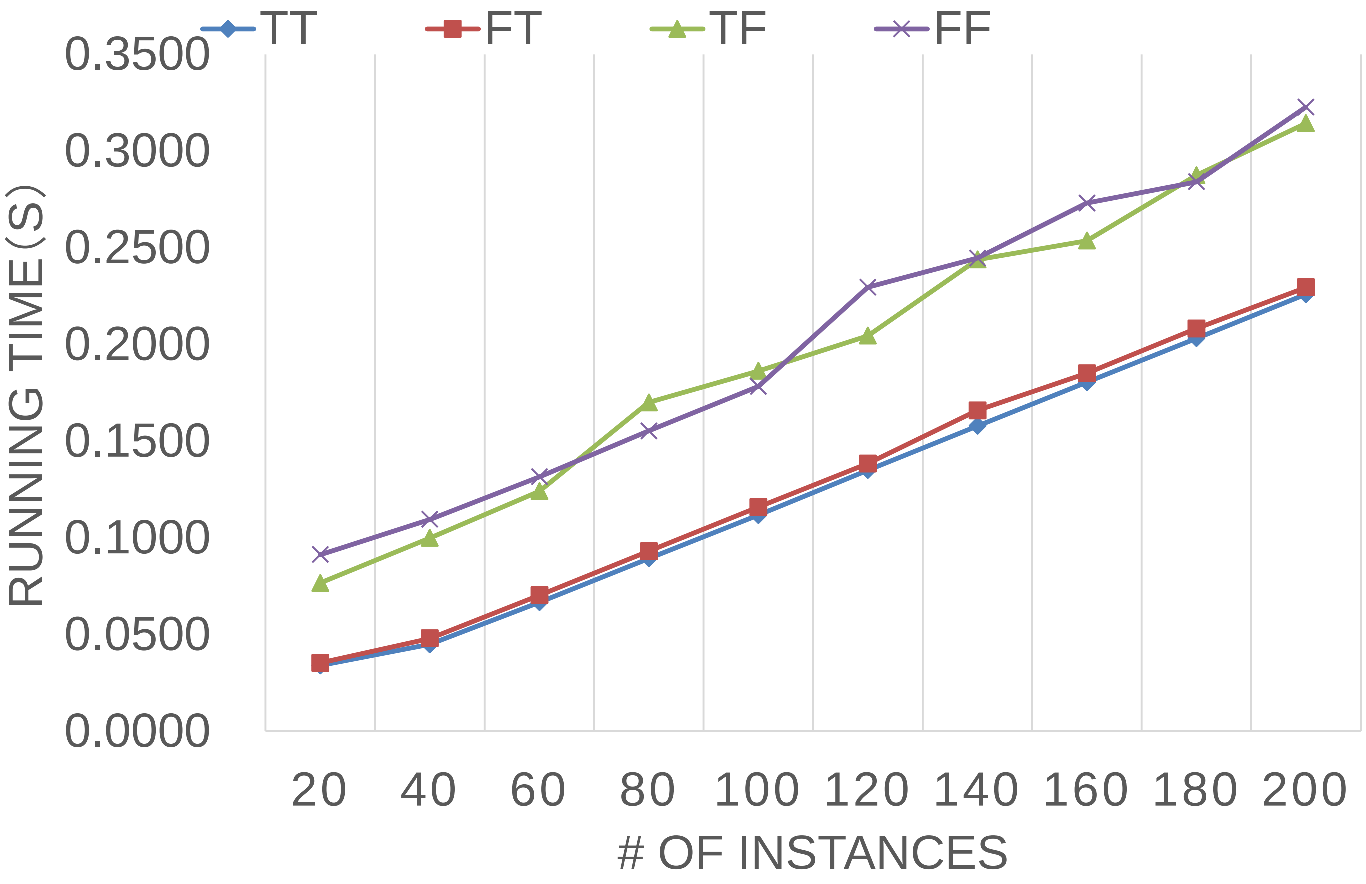
(2) DB Caching

- The cost of sample extraction from user-defined character classes can be reduced by DB caching

Experimental Evaluation

	Regexp Cache	DB Cache
TT	Yes	Yes
FT	No	Yes
TF	Yes	No
FF	No	No

Average Time of 10 Executions



Conclusion

- In this work, we have proposed a Regexp-based DGL for database populating
- By introducing type/format inference, dictionary and the extended reference mechanism, various kinds of meaningful pseudo data can be generated.
- We have demonstrated the strength of the proposed language by showing how to specify and populate database of TPC-H benchmark
- The performance improvement by the cache was considered and verified by preliminary experiments
- The future work includes learning regular expressions from instances, where regular expressions for synthetic data can be learned from positive instances

Thank you for attention!

Q&A

Kai Cheng
chengk@is.kyusan-u.ac.jp

SANGYO UNIVERSITY

The First International Workshop on Conceptual Modeling for Big Data and Smart Computing (CMComp2020), February 19th, 2020, Busan, Korea

In conjunction with IEEE BIGCOMP 2020

Regular Expression-based DGL
or Meaningful Synthetic Data
Generation

Kai Cheng
Department of Information Science
Kyushu Sangyo University, Japan
chengk@is.kyusan-u.ac.jp

CMCOMP2020@Busan Korea

1

COMP2020@