DBDM – Data Bases and Data Mining
Closure Algorithm for Functional Dependencies

`http://liris.cnrs.fr/ecoquery/dokuwiki/doku.php?id=enseignement:dbdm:start`

March 20, 2017

Abstract

The objective of this lab session is to implement two closure algorithms for functional dependencies given in pseudo-code and to experimentally compare their performances. The programming language may be Java, C++, OCaml, Haskell or Python. Up to two students, but no more, may collaborate together on this project.

1 Introduction

1.1 Delivery

An email addressed to Emmanuel Coquery with a `zip` file including all your source files, the `readme.txt` file, your experimental figures in a `tsv` or `csv` file, the image depicting the later as well as relevant supplementary material has to be sent by:

Monday, April 3rd, 2017, 23:59, Paris time.

The `readme.txt` report, written in French or in English, must include the full names of students involved and will contain your answers to the open questions (justification for the data structures you selected, a discussion on the experimental results etc.).

1.2 Main Program

A set of sample files is provided in the `examples` folder. Each file contains a set of Functional Dependencies (FDs). Your main program `closure` has three options, where `input` is either a file containing FDs. or `-` (the hyphen character) if the FDs are to be read on the standard input (`cin`):

- `closure -naive input atts`: with Σ the set of FDs read on `input` and X the set of attributes in `atts`, computes $Closure(\Sigma, X)$ with the *naive algorithm*, see Algorithm 1, and prints it on the standard output (`cout`);
- `closure -improved input atts`: with Σ the set of FDs read on `input` and X the set of attributes in `atts`, computes $Closure'(\Sigma, X)$ with the *improved algorithm*, see Algorithm 2, and prints it on the standard output (`cout`);
- `closure -generate n`: with n an integer, generates a particular set of FDs of the form $\{n-1 \to n, n-2 \to n-1, \ldots, 0 \to 1\}$, shuffles it and prints it on the standard output (`cout`);
- `closure -normalize input`: with Σ the set of FDs read on `input`, computes $Reduce(Minimize(\Sigma)$, see Algorithms 3 and 4, and prints it on the standard output (`cout`);
- `closure -decompose input`: with Σ the set of FDs read on `input`, supposed to be obtained after normalization, decomposes the (implicit) relation into a BCNF schema, and prints it on the standard output (`cout`).

Algorithm 1: $Closure(\Sigma, X)$

Data: Σ a set of FDs, X a set of attributes.
Result: X^+, the closure of X w.r.t. Σ

```
1 Cl := X
2 done := false
3 while (¬done) do
4 |   done := true
5 |   forall the W → Z ∈ Σ do
6 |   |   if W ⊆ Cl ∧ Z ⊈ Cl then
7 |   |   |   Cl := Cl ∪ Z
8 |   |   |   done := false
9 return Cl
```

Algorithm 2: $Closure'(\Sigma, X)$

Data: Σ a set of FDs, X a set of attributes.
Result: X^+, the closure of X w.r.t. Σ

```
1  for W → Z ∈ Σ do
2  |   count[W → Z] := |W|
3  |   for A ∈ W do
4  |   |   list[A] := list[A] ∪ W → Z
5  closure := X, update := X
6  while update ≠ ∅ do
7  |   Choose A ∈ update
8  |   update := update \ {A}
9  |   for W → Z ∈ list[A] do
10 |   |   count[W → Z] := count[W → Z] − 1
11 |   |   if count[W → Z] = 0 then
12 |   |   |   update := update ∪ (Z \ closure)
13 |   |   |   closure := closure ∪ Z
14 return closure
```

2 Closure algorithms for functional dependencies

2.1 Basic Data Structure

Exercise 1 input and output

Your program needs to read a set of FDs from a file and to write sets of FDs and sets of attributes on the standard output (`cout`).

1. Define data structures for attributes, sets of attributes, sets of sets of attributes, FDs and sets of FDs.
2. Define input/read and output/write for the data structures of the previous question. If needed by your favorite set library, define a lexical (total) order on FDs.
3. Define a main program with the requested command line options.

2.2 Closure algorithms

The closure X^+ of a set of attributes by a set of FDs Σ is defined as $X^+ = \{A \mid F \vdash X \to A\}$. Algorithm 1 gives a procedure to compute X^+. As discussed during the lecture[1] Algorithm 1 can be improved to Algorithm 2 that does exactly the same job.

Exercise 2 naive closure

1. Implement Algorithm 1.
2. Provide unit tests for your implementation.

Exercise 3 improved algorithm

1. Define data structures for the *count* and *list* indices of Algorithm 2.
2. Implement the construction of indices of Algorithm 2, Lines 2 to 4 as a separate function.
3. Implement Algorithm 2.
4. Provide unit tests for your implementation.

Exercise 4 open questions

1. Justify your implementation choices for the data structures, in the appropriate section of the `readme.txt` file.
2. Discuss your strategy that implements the Choose A instruction of Algorithm 2, Line 7.
3. There is a small corner case in Algorithm 2 with non-standard DFs. Find it.

3 Experimental Comparison

At this point you have two algorithms solving the same problem. The goal is to experimentally compare the practical performance gain.

Exercise 5 performance comparison

1. Implement the `-generate n` command-line option that produces the set of DFs $\{n-1 \to n, n-2 \to n-1, \ldots, 0 \to 1\}$. Shuffle the DFs of this set before it is output.
2. Inspire yourselves from the `./perf.sh` script to obtain raw execution times to be stored in a Tab/Comma Separated Value (`tsv`/`csv`) file.
3. Use your favorite spreadsheet, plot or statistics software to graphically depict your results. Figure 1 is provided for inspiration.

[1] `http://liris.cnrs.fr/ecoquery/dokuwiki/lib/exe/fetch.php?media=enseignement:dbdm:dbdm-02.pdf`

Exercise 6 open questions

1. Explain why the set $\{n-1 \to n, n-2 \to n-1, \ldots, 0 \to 1\}$ is "interesting" when evaluating the closure algorithms.
2. Detail your experimental setup and methodology.
3. Analyze your experimental results.

4 Normalization

Algorithm 3: $Minimize(F)$

Data: F a set of FDs
Result: G a *minimal* (in cardinality) cover of F

$G := \emptyset$
for $X \to Y \in F$ **do**
  $G := G \cup \{X \to X^+\}$;
for $X \to X^+ \in G$ **do**
  if $G \setminus \{X \to X^+\} \vdash X \to X^+$ **then**
    $G := G \setminus \{X \to X^+\}$;
return G;

Algorithm 4: $Reduce(F)$

Data: F a set of FDs
Result: G a cover of F with reduced left-hand sides

$Min := F$
for $X \to Y \in Min$ **do**
  $W := Y$
  for $A \in Y$ **do**
    $G := (Min - \{X \to Y\}) \cup \{X \to (W - A)\}$
    if $G \models X \to Y$ **then**
      $W := W - \{A\}$;
  $Min := (Min - \{X \to Y\}) \cup \{X \to W\}$;
return Min;

Algorithm 5: $Decompose(\Sigma, U)$

$F := Reduce(Minimize(\Sigma))$
$\mathbf{R} = \{U\}$;
`/* While` Σ `not in Boyce-Codd Normal Form */`
while $(\exists R \in \mathbf{R}.\neg BCNF(R))$ **do**
  `/* Find a non-trivial non-key FD */`
  let $X \to Y \in F$ with $Y \not\subseteq X$ and $F \not\models X \to U$;
  `/* Replace` R `by` $R_1 = X^+$ `and` $R_2 = (R \setminus X^+) \cup X$ `*/`
  $\mathbf{R} := \mathbf{R} \setminus \{R\} \cup \{X^+, (R \setminus X^+) \cup X\}$;
return R

Exercise 7 decomposition

The decomposition pipeline is given by Algorithm 5. It uses two subroutines Algorithm 3 and Algorithm 4. The function $Reduce \circ Minimize$ is said to *normalize* a set of FDs.

1. Implement a tool function that checks if $\Sigma \models X \to Y$ by testing if $Y \subseteq Closure(\Sigma, X)$ or $Y \subseteq Closure'(\Sigma, X)$.

2. Implement the `-normalize` command-line option that runs Algorithms 3 and 4. Mind that the set of FDs is modified along the process, so be careful when using Algorithm 2.
3. Implement a tool $Schema(\Sigma)$ function that gather all attributes that appears in a set of FDs, for instance $Schema(\{AB \to C, D \to C\}) = ABCD$.
4. Implements a tool function that given a set of FDs Σ and a set of attributes X checks if X is a key of $Schema(\Sigma)$. Use this function to implement the BCNF test that appears at Line 3 of Algorithm 5.
5. Implement the `-decompose` command-line option that computes $Decompose(\Sigma, Schema(\Sigma))$

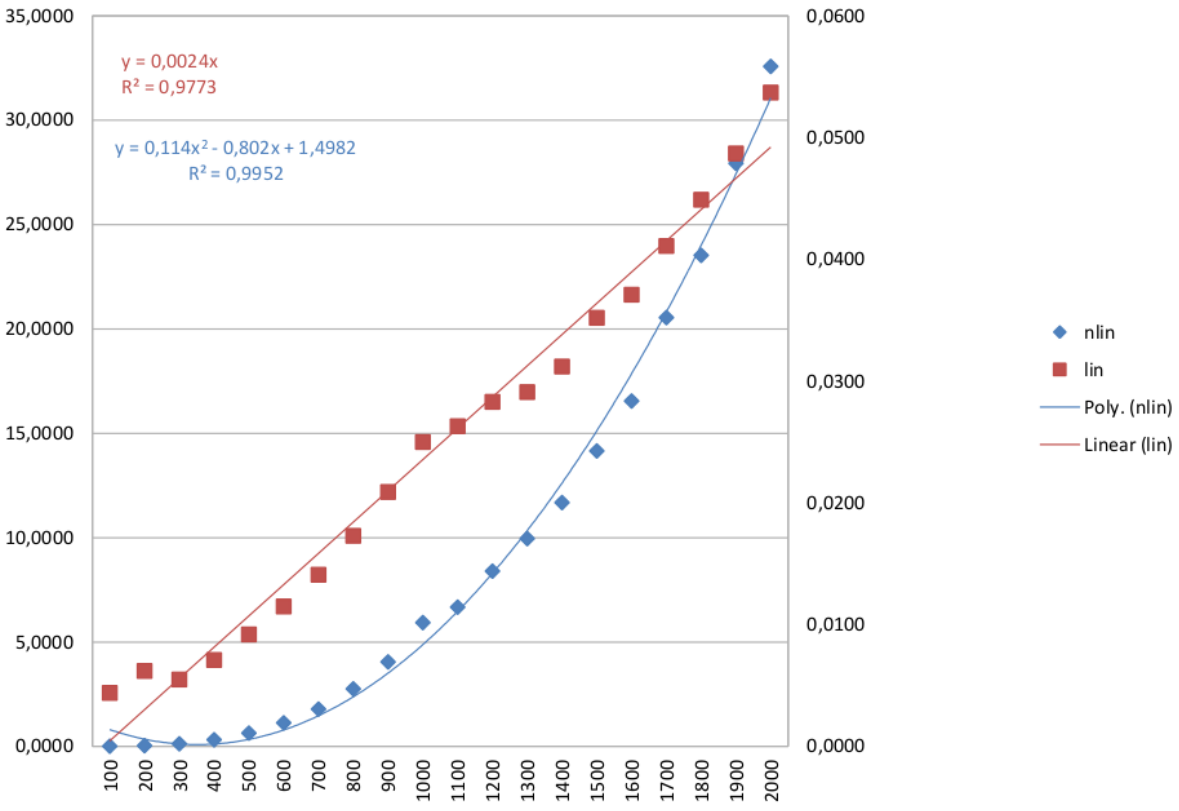

Figure 1: Sample experimental evaluation