Code Clone Selection Puzzle and Solutions

Abstract

This paper is going to discuss about the problem derived from code clone selection puzzle we encountered in ***Pochoir***'s [1] development, including the problem formulation, possible solution, and complexity bound.

Index Terms

code-clone selection puzzle, algorithm, theory, Pochoir, NP-hardness

Code Clone Selection Puzzle and Solutions [1]

I. Introduction

Problem formulation:

Given a $g \times m$ (g rows, m columns) bit-matrix, each entry of which is either 0 or 1. Suppose that initially all rows and columns are distinct, we can put "x" on some entries of the bit-matrix to make these columns identical. For example, in following 3×3 bit-matrix, if we put "x" on entries $(2,0)$ and $(2,1)$ [1], we can make the first and second column identical like that shown on the right-hand side.

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \Longrightarrow \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ x & x & 0 \end{pmatrix}$$

However, columns such as those in pairs $(\{0,1,x\}^T, \{0,1,1\}^T)$ or $(\{0,1,0\}^T, \{0,1,x\})$ are not considered identical.

The question is: how to find the minimum number of "x" we should place on any $g \times m$ bit-matrix to make the number of distinct columns less than or equal to a given number k ($k \leq m$)

II. A proof of NP-Hardness

From the problem formulation, we can easily see that it's a special case of k-clustering problem. If we denote the input m columns as set $\mathcal{C} = \{c_0, c_1, \ldots, c_{m-1}\}$, the problem is to partition the input set $\mathcal{C}$ into disjoint subsets $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{k-1}$, such that an objective function is minimized or maximized. Three of the most commonly used objective functions include 2-norm distance (the problem of which is known as k-means), 1-norm distance (the problem of which is known as k-medians), radius of maximum cluster (the problem of which is known as k-centers). Our problem differs from all these three problems in that our objective is to minimize the overall summation of Hamming distances.

If we denote the j-th element of column c_i as $c_i[j]$, the Hamming distance between two input columns c_i and c_j is defined as $H(c_i, c_j) = |\{l : c_i[l] \neq c_j[l]\}|$ [2] and the Hamming distance of a set $\mathcal{C}_i$ is defined

[1] all coordinates of bit-matrix follows a convention in programming language C

[2] notation $|C|$ denotes the number of elements of set C

as $H(\mathcal{C}_i) = |\{l : \exists c_{i_1}, c_{i_2} \in \mathcal{C}_i, c_{i_1}[l] \neq c_{i_2}[l]\}|$. We can further denote the summation of Hamming distance of $\mathcal{C}_i$ as $c(\mathcal{C}_i) = |\mathcal{C}_i| \cdot H(\mathcal{C}_i)$, so that our objective is to minimize $\sum_i c(\mathcal{C}_i)$.

Our problem is actually very similar to the problem of k-anonymity, which is proposed by Sweeney [2] as a technique to release public information while ensuring both data privacy and data integrity. A specialization of k-anonymity problem asks that given an $g \times m$ (g rows, m columns) bit-matrix, each entry of which is either 0 or 1, what's the least number of "x" (suppression) we need to put on the bit-matrix to make any column c_{i_0} in the resulting bit-matrix to be identical to at least $k-1$ other columns $c_{i_1}, c_{i_2}, \ldots, c_{i_{k-1}}$.

By comparing our problem with k-anonymity problem, we can see that the input and objective of these two problems are identical, the only difference lies in the constraints. Our problem has a constraint that the number of total clusters can be no more than k, while k-anonymity problem has the constraint that for each cluster, the number of total elements can be no less than k. However, if looking carefully, the constraint of k-anonymity problem can be rephrased as

1) number of elements per cluster be no less than k.

2) total number of clusters be no more than m/k, which is a constraint implied from the first constraint.

So, we can see that the only difference of our problem from k-anonymity problem is that k-anonymity problem has an additional constraint on the number of elements per cluster. That's it! Regarding the similarity of our problem to the k-anonymity problem, we re-name our problem to k-distinct problem.

Since the k-anonymity problem has already been proved NP-hard by numerous ways [], we will try performing a similar reduction from a known NP-hard problem, k-dimensional matching, to our k-distinct problem. Note that a naive reduction from k-anonymity to k-distinct won't work because the global optimum of one instance can not guarantee a global optimum of the other on either direction.

k-dimensional perfect matching: Given a collection $\mathcal{C}$ of k-sets over a universe $\mathcal{U}$, is there a subset $\mathcal{S} \subseteq \mathcal{C}$ such that:

- Every $x \in \mathcal{U}$ is in some k-set s of $\mathcal{S}$
- The sets of $\mathcal{S}$ are disjoint; i.e. $\forall s_1, s_2 \in \mathcal{S}$, $s_1 \cap s_2 = \emptyset$.

Note that when $k = 2$, the k-dimensional perfect matching problem is polynomial time solvable but is NP-hard for $k \geq 3$.

Theorem 1: For $k \leq m/3$, the k-distinct problem is NP-hard.

Proof: Given an arbitrary instance of k-dimensional matching problem, where $\mathcal{U} = \{x_0, x_1, \ldots, x_{m-1}\}$, $\mathcal{C} = \{s_0, s_1, \ldots, s_{g-1}\}$ such that $\forall j \in [0, g-1]$, $s_j \subseteq \mathcal{C}$ and $|s_j| = k$, we construct an $\frac{m}{k}$-distinct problem as

follows:

- rows correspond to $s_j \in \mathcal{C}$.
- columns correspond to $x_i \in \mathcal{U}$.

For any entry of bit-matrix

$$T[j,i] = \begin{cases} 1 & \text{if } x_i \in s_j \\ 0 & \text{otherwise} \end{cases}$$

An example of this construction:

Suppose a 3-dimensional matching problem: $\mathcal{U} = \{1,2,3,4,5,6\}$ and $\mathcal{C} = \{\{1,2,3\},\{1,4,5\},\{4,5,6\},\{2,3,6\}\}$. The constructed bit-matrix is as follows:

	1	2	3	4	5	6
{1,2,3}	1	1	1	0	0	0
{1,4,5}	1	0	0	1	1	0
{4,5,6}	0	0	0	1	1	1
{2,3,6}	0	1	1	0	0	1

TABLE I: Example of reducing k-dimensional matching problem to $\frac{m}{k}$-distinct problem

We claim that k-dimensional matching problem has a perfect matching , i.e. there is a subset $\mathcal{S} \subseteq \mathcal{C}$ of size m/k such that each $x_i \in \mathcal{U}$ belongs to one and only one $s_j \in \mathcal{S}$ $\iff$ the constructed $\frac{m}{k}$-distinct problem has an optimal solution of grouping all columns into $\frac{m}{k}$ clusters and has a cost (the number of "x" put in bit-matrix) less than or equal to $gm - \frac{m^2}{k} - \max\{0, \frac{m}{k} - k\} \cdot (gk - m) = gk^2 - mk$, if we assume $m/k - k \geq 0$.

$\Longrightarrow$: If we have a k-dimensional perfect matching, i.e. a subset $\mathcal{S} \subseteq \mathcal{C}$ of size m/k such that each $x_i \in \mathcal{U}$ belongs to one and only one $s_j \in \mathcal{S}$. Apparently, $\forall s_j, s_i \in \mathcal{S}$, $|s_j| = |s_i| = k$ and $s_i \cap s_j = \emptyset$. So if we group all columns $x_i \in s_j$ together, no entries in row j will be covered by "x". For the m/k rows that corresponds to the sets in perfect matching, there are $m/k \cdot m = m^2/k$ entries that are not covered by "x". For the rest $g - m/k$ rows that corresponds to the sets not in perfect matching, each row has at least $\max\{0, m/k - k\}$ clusters that are not "x" out, each these cluster has k entries, so that the total entries in the rest $g - m/k$ rows that are not "x" out is $\max\{0, (m/k - k)\} \cdot k \cdot (g - m/k) = \max\{0, m/k - k\} \cdot (gk - m)$. So, the overall cost (number of "x") of constructed $\frac{m}{k}$-distinct problem won't be larger than $g \cdot m - m^2/k - \max\{0, \frac{m}{k} - k\} \cdot (gk - m) = gk^2 - mk$.

$\Longleftarrow$: If we can find a $\frac{m}{k}$-clustering of all columns in the constructed bit-matrix with a cost no more than $gk^2 - mk$, let's prove that the solution corresponds to a k-dimensional perfect matching of original problem.

Because $gk^2 - mk = gm - m^2/k - (m/k-k)\cdot k\cdot(g-m/k)$, apparently, it means that at least m/k rows are entirely uncovered by "x". For the rest $g-m/k$ rows, it has $(m/k-k)\cdot k$ in total entries that are not covered by "x". HOW TO PROCEED ALONG THIS DIRECTION??? ■

III. An Exact Algorithm for the code-clone selection puzzle

Now, let's describe an exact algorithm to solve the puzzle. Since if $k = m$, or $k = m-1$, the solution is trivial, in the following algorithm description, let's assume that $k \leq m-2$.

The algorithm can be roughly divided into two stages:

1) Establish a hierarchical DAG out of the m input columns
2) Calculate the k-medians (unified by $\otimes$ operation) [3]and the optimal values out of the hierarchical DAG

Before the description of algorithm, let's define some notations that will be used throughout the text: for the original set of m input columns, let's denote it as $\mathcal{C} = \{c_0, c_1, \ldots, c_{m-1}\}$, for all nodes derived from input columns and construct the hierarchical DAG T, we denote it as $\mathcal{V} = \{v_0, v_1, \ldots, v_{M-1}\}$, where M is the total number of nodes in the DAG. Apparently, $\mathcal{C} \subseteq \mathcal{V}$. For each node $v_i \in \mathcal{V}$, we denote the sub-tree rooted at v_i to be T_{v_i}, the set of original input columns to be $\mathcal{C}_i$, the set of total nodes to be $\mathcal{V}_i$, and $\mathcal{C}_i \subseteq \mathcal{V}_i$ holds. Correspondingly, we denote the set of k-medians as $\mathcal{M} = \{m_0, m_1, \ldots, m_{k-1}\}$. The sub-tree rooted at a median m_i to be T_{m_i} The set of original input columns and total nodes are denoted as $\mathcal{C}_{m_i}$ and $\mathcal{V}_{m_i}$, respectively.

A. Construct a hierarchical DAG

In stage 1, we need to construct a hierarchical DAG, denoted as T out of the m input columns. The entire procedure is a dynamic programming.

- Prepare $g+1$ bins, i.e. $\{0, 1, \ldots, g\}$. Each bin i ($i \in [0, g]$) will correspondingly contains all columns that derived from original m input columns and have i "x"s.
- Prepare a FIFO queue and put all initial m input columns into the FIFO queue.
- Each time, remove a head element (column) from the FIFO queue, apply it with $\otimes$ operation to the rest of elements (columns) in the FIFO and insert resulting unified elements (columns) into the FIFO queue. After applying the head element to all the rest elements in FIFO, insert the head element into bin i according to how many "x" it has in it form.

[3]In the rest of this document, we use the term "median" and "unified column" interchangably

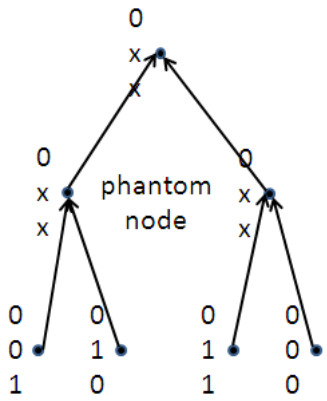

Fig. 1: Phantom Nodes

⊗	0	1	x
0	0	x	x
1	x	1	x
x	x	x	x

Fig. 2: Truth table for ⊗ operation

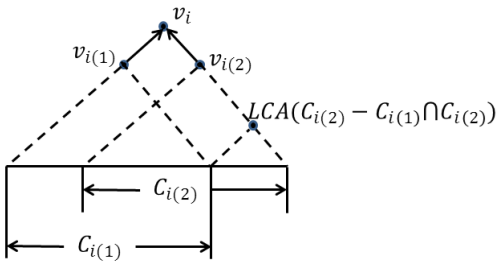

Fig. 3: Illustration of how to handle overlap

Theorem 2: If assuming the number of initial input columns is m, the number of total resulting unified (by $\otimes$ operation) but different columns is M, then the total number of $\otimes$ operation is at most $O(M^2)$.

Proof: Since the total number of elements (columns) in the FIFO queue is at most M at any moment, and each element will $\otimes$ with the rest of elements in FIFO queue, so it's at most $O(M^2)$ times. ■

B. Compute the k-medians out of the hierarchical DAG

In stage 2, we need to compute a k-medians on T.

- Naively, $\binom{bin(0)+bin(1)+...+bin(g)}{k}$ will be our global optimum. But this method will yield complexity bound $\binom{M}{k} = O(M^k)$.
- Another way to compute the k-medians out of the hierarchical DAG is via dynamic programming. Following algorithm to compute k-medians on the hierarchical DAG is largely adapted from the algorithm in [3]. The algorithm in [3] computes k-medians on a tree with n nodes in time $O(kn^2)$. For trees, any two sub-trees don't overlap with each other, but for DAGs, any two sub-trees can overlap. So the innovation of following algorithm lies in mostly how to handle overlaps.
  - Define $d(\mathcal{V}_j) = d(\mathcal{C}_j) = h_j \times \sum_{c_{j,i} \in \mathcal{C}_j} c_{j,i}$, where h_j is the height / bin number of root node v_j, $\mathcal{C}_j$ is the set of all input columns that can be covered in tree T_{v_j}, and $c_{j,i}$ are elements of set $\mathcal{C}_j$. Note that $\mathcal{C}_j$ is also the set of leaves of tree T_{v_j}
  - Define $d(\mathcal{V}_i) \oplus d(\mathcal{V}_j) = d(\mathcal{C}_i) \oplus d(\mathcal{C}_j) = h_i \times \sum_{c_{i,l_i} \in \mathcal{C}_i \wedge c_{i,l_i} \notin \mathcal{C}_j} c_{i,l_i} + \min\{h_i, h_j\} \times \sum_{c_{l_o} \in \mathcal{C}_i \cap \mathcal{C}_j} c_{l_o} + h_j \times \sum_{c_{j,l_j} \in \mathcal{C}_j \wedge c_{j,l_j} \notin \mathcal{C}_i}$.
  - Define $\otimes$ to be a bit-wise operation depending on truth table of Figure 2. So, $\text{LCA}(v_i, v_j) = v_i \otimes v_j$.

- For each node v_i, an integer $q = 1, \ldots, k$, define $G(v_i, q)$ be the optimal value of the subproblem defined on the sub-tree T_{v_i}, given that a total of at least 1 and at most q nodes (medians) are selected in T_{v_j}. Of course, the function $G(v_i, q)$ is computed only for $q \leq |\mathcal{V}_i|$, where $|\mathcal{V}_i|$ is the number of total nodes (not just input columns) contained in sub-tree T_{v_i}.
- Define $d(G(v_i, q)) = d(\mathcal{C}_{m_0}) \oplus \ldots \oplus d(\mathcal{C}_{m_{q-1}})$, where $m_0, \ldots, m_{q-1}$ are the q medians selected in sets $\mathcal{V}_j$

- following procedure employs dynamic programming technique to compute the function $G(v_j, q)$ bottom up the hierarchial DAG T
  - Apparently for every node v_i in T, we define $G(v_i, 0) = \infty$.
  - If v_i is a leaf of T, then $G(v_i, 1) = 0$. $G(v_i, q) = 0$, for any $q \geq 1$.
  - If v_i is a non-leaf of T, since every non-leaf has two and only two children, we denote $v_{i(1)}$ to be the left child and $v_{i(2)}$ to be the right child of v_i respectively. We have recurrence (see Figure 3):

$$
\begin{aligned}
G(v_i, q) = \min\{ & \min_{\substack{q_1+q_2=q \\ q_1 \leq |\mathcal{C}_{i(1)} - \mathcal{C}_{i(1)} \cap \mathcal{C}_{i(2)}| \\ q_2 \leq |\mathcal{C}_{i(2)}|}} \{d(G(\mathrm{LCA}(C_{i(1)} - C_{i(1)} \cap C_{i(2)}), q_1)) \oplus d(G(v_{i(2)}, q_2))\} \\
& , \\
& \min_{\substack{q_1+q_2=q \\ q_1 \leq |\mathcal{C}_{i(1)}| \\ q_2 \leq |\mathcal{C}_{i(2)} - \mathcal{C}_{i(1)} \cap \mathcal{C}_{i(2)}|}} \{d(G(v_{i(1)}, q_1))) \oplus d(G(\mathrm{LCA}(C_{i(2)} - C_{i(1)} \cap C_{i(2)}, q_2))\} \\
& , \\
d(\mathcal{V}_i) \oplus & \min_{\substack{q_1+q_2=q-1 \\ q_1 \leq |\mathcal{C}_{i(1)} - \mathcal{C}_{i(1)} \cap \mathcal{C}_{i(2)}| \\ q_2 \leq |\mathcal{C}_{i(2)}|}} \{d(G(\mathrm{LCA}(C_{i(1)} - C_{i(1)} \cap C_{i(2)}), q_1)) \oplus d(G(v_{i(2)}, q_2))\} \\
& , \\
d(\mathcal{V}_i) \oplus & \min_{\substack{q_1+q_2=q-1 \\ q_1 \leq |\mathcal{C}_{i(1)}| \\ q_2 \leq |\mathcal{C}_{i(2)} - \mathcal{C}_{i(1)} \cap \mathcal{C}_{i(2)}|}} \{d(G(v_{i(1)}, q_1)) \oplus d(G(\mathrm{LCA}(C_{i(2)} - C_{i(1)} \cap C_{i(2)}), q_2))\} \\
& \}
\end{aligned}
\tag{1}
$$

  In recurrence 1, terms "$\min_{\substack{q_1+q_2=q \\ q_1 \leq |\mathcal{C}_{i(1)} - \mathcal{C}_{i(1)} \cap \mathcal{C}_{i(2)}| \\ q_2 \leq |\mathcal{C}_{i(2)}|}} \{d(G(\mathrm{LCA}(C_{i(1)} - C_{i(1)} \cap C_{i(2)}), q_1)) \oplus d(G(v_{i(2)}, q_2))\}$" and "$\min_{\substack{q_1+q_2=q \\ q_1 \leq |\mathcal{C}_{i(1)}| \\ q_2 \leq |\mathcal{C}_{i(2)} - \mathcal{C}_{i(1)} \cap \mathcal{C}_{i(2)}|}} \{d(G(v_{i(1)}, q_1))) \oplus d(G(\mathrm{LCA}(C_{i(2)} - C_{i(1)} \cap C_{i(2)}, q_2))\}$" represent the case that all q medians are selected in two sub-trees of the tree T_i.

Terms "$d(\mathcal{V}_i) \oplus \min_{\substack{q_1+q_2=q-1 \\ q_1 \le |\mathcal{C}_{i(1)} - \mathcal{C}_{i(1)} \cap \mathcal{C}_{i(2)}| \\ q_2 \le |\mathcal{C}_{i(2)}|}} \{d(G(\mathrm{LCA}(C_{i(1)} - C_{i(1)} \cap C_{i(2)}), q_1)) \oplus d(G(v_{i(2)}, q_2))\}$" and "$d(\mathcal{V}_i) \oplus \min_{\substack{q_1+q_2=q-1 \\ q_1 \le |\mathcal{C}_{i(1)}| \\ q_2 \le |\mathcal{C}_{i(2)} - \mathcal{C}_{i(1)} \cap \mathcal{C}_{i(2)}|}} \{d(G(v_{i(1)}, q_1)) \oplus d(G(\mathrm{LCA}(C_{i(2)} - C_{i(1)} \cap C_{i(2)}), q_2))\}$" represent the case that we select node v_i as one of the medians, and select the rest $q-1$ medians from two sub-trees.

C. Proof of Global Optimality

In this section, we are trying to prove that the k-medians got from above algorithm are the global optimum.

Theorem 3: Let's denote v_0 to be the root of the DAG T, global k-median optimum $G(v_0,k)$ can be given by any two non-overlapping sub-trees which covers the entire set of $\mathcal{C}_0$ by recurrence 1. That is, if T_i and T_j are two sub-trees of T, $(\mathcal{C}_i \cup \mathcal{C}_j = \mathcal{C}_0) \wedge (\mathcal{C}_i \cap \mathcal{C}_j = \emptyset)$, then $G(v_0,k) = \min\{\min_{\substack{k_1+k_2=k \\ k_1 \le |\mathcal{V}_i| \\ k_2 \le |\mathcal{V}_j|}} \{d(G(v_i,k_1)) \oplus d(G(v_j,k_2))\}, d(\mathcal{V}_0) \oplus \min_{\substack{k_1+k_2=k-1 \\ k_1 \le |\mathcal{V}_i| \\ k_2 \le |\mathcal{V}_j|}} \{d(G(v_i,k_1)) \oplus d(G(v_j,k_2))\}\}$. In short, let's denote $G(v_0,k) = \min_{k_1+k_2=k}\{\mathcal{C}_i \odot \mathcal{C}_j\}$

Proof: Let's prove the theorem by induction on the height h of the DAG:

- When height of DAG equals 0, i.e. v_0 is a leaf node, apparently $G(v_0,q)$, where $q \in [0,k]$ gives the global optimum.
- When height of DAG equals 1, which means, for non-leaf node v_0, its only two sub-trees are leaves. Since it's binary tree and both its sub-trees are leaves and non-overlapping, $G(v_0,q)$, where $q \in [0,3]$ gives the global optimum.
- Assuming that for any DAG of height h, the conclusion holds.
- For DAG T_0 of height $h+1$. If we divide $\mathcal{C}_0$ into two disjoint sub-sets $\mathcal{C}_i$ and $\mathcal{C}_j$, apparently, the corresponding $G(v_i,k_1)$, where $k_1 \le |\mathcal{V}_i|$, and $G(v_j,k_2)$, where $k_2 \le |\mathcal{V}_j|$ gives corresponding global optimum of two sub-trees T_i and T_j. Now, we are going to prove that $G(v_0,k) = \min_{k_1+k_2=k}\{\mathcal{C}_i \odot \mathcal{C}_j\}$ gives the global optimum of tree T_0.

  Suppose it's not, which means that at least there is one median m_k not in sub-trees T_i or T_j. Since $\mathcal{C}_i \cup \mathcal{C}_j = \mathcal{C}_0$, so we can divide the input column set $\mathcal{C}_{m_k}$ of T_{m_k} into two disjoint sets $\mathcal{C}_{m_k,i}$ and $\mathcal{C}_{m_k,j}$, such that $(\mathcal{C}_{m_k,i} \cup \mathcal{C}_{m_k,j} = \mathcal{C}_{m_k}) \wedge (\mathcal{C}_{m_k,i} \subseteq \mathcal{C}_i) \wedge (\mathcal{C}_{m_k,j} \subseteq \mathcal{C}_j)$. So, we can further divide $\mathcal{C}_i$ into two sub-sets $\mathcal{C}_{m_k,i} \cup (\mathcal{C}_i - \mathcal{C}_{m_k,i})$, and divide $\mathcal{C}_j$ into two sub-sets $\mathcal{C}_{m_k,j} \cup (\mathcal{C}_j - \mathcal{C}_{m_k,j})$, respectively.

  According to the inductive hypothesis, $G(v_{m_k},k) = \min_{k_1+k_2=k}\{\mathcal{C}_{m_k,i} \odot \mathcal{C}_{m_k,j}\}$, $G(v_i,k) = \min_{k_1+k_2=k}\{\mathcal{C}_{m_k,i} \odot (\mathcal{C}_i - \mathcal{C}_{m_k,i})\}$, $G(v_j,k) = \min_{k_1+k_2=k}\{\mathcal{C}_{m_k,j} \odot (\mathcal{C}_j - \mathcal{C}_{m_k,j})\}$. So, any benefits can be

got from median m_k must be already covered by two non-overlapping sub-trees T_i and T_j.

■

IV. Example

Assuming we have input bit-matrix as follows (3 rows, 5 columns):

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

So it has 5 input columns, $g = 3$.

After stage 1, it has following bins:

bin 1	column 0	column 1	column 2	column 3
00x	1	1	∞	∞
0x0	1	∞	1	∞
x00	1	∞	∞	1
bin 2	column 0	column 1	column 2	column 3
0xx	2	2	2	∞
x0x	2	2	∞	2
xx0	2	∞	2	2
xx1	∞	2	∞	∞
x1x	∞	∞	2	∞
1xx	∞	∞	∞	2
bin 3	column 0	column 1	column 2	column 3
xxx	3	3	3	3

Fig. 4: Generated data structure in bins

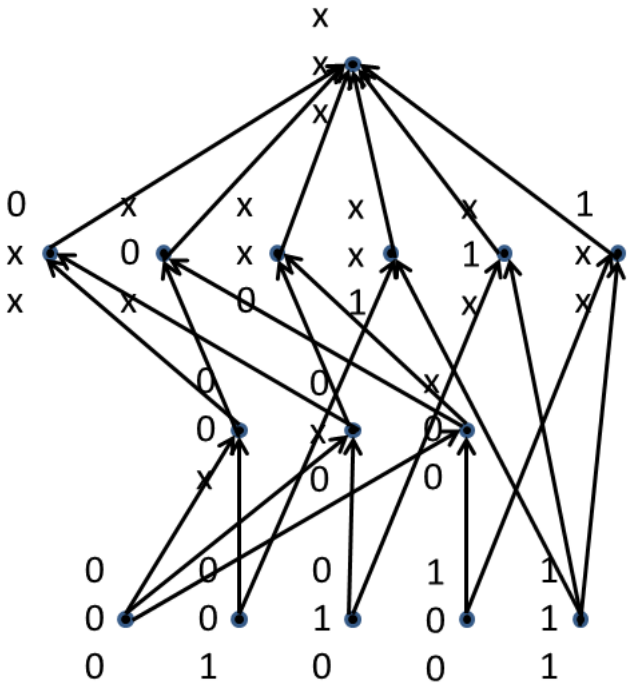

Fig. 5: Generated DAG data structure

From Figure 4, we can see that from 5 input columns, the algorithm will generate 10 new nodes spreaded in 3 different bins. If including the input 5 columns, the DAG has 15 total nodes as illustrated in Figure 5.

V. The Analysis of Complexity Bound

A. The complexity bound of stage 1

From Theorem 2 we can see that if we assume the DAG has M nodes, the algorithm to construct the DAG is no more than M^2. In this section, we are going to clarify the relation of M and the number of input columns m.

Intuitively, $m \in [2, 2^g]$, when $m = 2$, $M = 3 \in O(m^2)$, when $m = 2^g$, $M = 3^g = O(m^2)$, also, if $M = f(m)$, this f seems an monotonically increasing function. However, it turns out that in the worst case, M can be an exponential of m. Following theorem tries to prove it by induction.

Theorem 4: If there are m input columns, each of which is of g bit wide. If we assume the output DAG has $M = f(m)$ nodes in total (including the input m columns), then in the worst case $M = f(m) = O(2^g)$.

Proof: Actually, the worst case comes when the input bit matrix is diagonal as follows:

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

If we induct on the number of input columns, we have:

First, for $m = 2$, $M = 3$.

Second, if assuming that we already constructed a DAG from m input columns, $c_0, c_1, \ldots, c_{m-1}$, and bin 0 has $b_0 = m$ nodes, bin 1 has b_1 nodes, …, bin i has b_i nodes. When we add in one more node c_m, let's count how many nodes will be increased in each bin, and total.

- for bin 0, apparently, $b'_0 = b_0 + 1$.
- for bin 1, $b'_1 = b_1 + b_0$. The new nodes in bin 1 are of the form $c_i c_m$, with $c_i \in$ bin 0. Because it's diagonal matrix, so the newly add-in node c_m has a 1 set at different position from any of existing old nodes, so there is no collision of newly merged nodes with old nodes.
- for bin 2, $b'_2 = b_2 + b_1$. Note that the new nodes in bin 2 are of the form $c_i c_j c_m$, with $c_i, c_j \in b_1$.
- for bin i, $b'_i = b_i + b_{i-1}$.
- Overall, $\sum_i b'_i = 2\sum_i b_i$, i.e. the summation of number of nodes in all bins increases exponentially in this case. Because the number of columns of diagonal matrix is at most g, so it has at most 2^g nodes in all bins after the construction.

■

Theorem 4 is about the complexity of M in the worst case, following calculation yields the expected number of M.

$$
\begin{aligned}
E[bin(0)] &= m \\
E[bin(1)] &= \binom{m}{2} \cdot Pr\{I_{i,j}\} \quad \text{where } I_{i,j}=1 \text{ if column } i \text{ and } j \text{ differs by only 1 bit, 0 otherwise} \\
&= \binom{m}{2} \cdot \frac{\binom{g}{1}}{2^g} \\
&= O(gm^2/2^g) \\
E[bin(2)] &= \binom{E[bin(1)]}{2} \cdot Pr\{I_{i,j}\} \quad \text{where } I_{i,j}=1 \text{ if column } i \text{ and } j \text{ differs by only 1 non-“x” position, 0 otherwise} \\
&= (\frac{gm^2}{2^g})^2 \cdot \frac{g-1}{2^{g-1}} \\
&= O(\frac{g^3m^4}{2^{3g}}) \\
\dots &= \dots \\
E[bin(i)] &= \binom{E[bin(i-1)]}{2} \cdot Pr\{I_{i,j}\} \quad \text{where } I_{i,j}=1 \text{ if column } i \text{ and } j \text{ differs by only 1 non-“x” position, 0 otherw} \\
&= O(\frac{g^{2i-1} \cdot m^{2^i}}{2^{(2i-1)g}})
\end{aligned}
$$

By non-“x” position, we distinguish two cases:

- columns $(0,1,0)^T$ and $(0,1,x)^T$ differ by a “x” position.
- columns $(0,0,0)^T$ and $(0,1,x)^T$ differ by a non-“x” position.

So, $E[\sum_{i=0}^{g}|bin(i)|] = O(\frac{g^{2g} \cdot m^{2^g}}{2^{2g^2}})$

B. The complexity bound of stage 2

Apparently, in recurrence 1, operation “LCA” has complexity $O(m)$, set operation “$-$” and “$\cap$” has complexity $O(m)$, “$\oplus$” has complexity $O(m)$. The recurrence 1 takes a minimum of four terms, each of which has complexity $O(qm)$. If we assume there are total M nodes in DAG T, in total, the complexity is $O(M\sum_{q=1}^{k} qm) = O(k^2Mm)$.

VI. Extension the code-clone selection puzzle from point to region

The extension of code-clone selection puzzle from point to region is straightforward. The differences are

- The entry in bit-matrix can be 'x'. $\{0,1,x\}$
- The 'x' in bit-matrix comes from contraction of a whole region to one bit-vector via bit-wise operation $\otimes$
- After this contraction, each bit-vector will be associated with a weight w_i to denote the number of points it represents.
- In computing the k-medians out of the DAG, we also need to take the weights of each node into account.

But the basic algorithm and complexity bound shouldn't change.

References

[1] Y. Tang, R. A. Chowdhury, B. C. Kuszmaul, C.-K. Luk, and C. E. Leiserson, "The Pochoir stencil compiler," in *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures*, ser. SPAA '11. New York, NY, USA: ACM, 2011, pp. 117–128. [Online]. Available: http://doi.acm.org/10.1145/1989493.1989508

[2] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[3] A. Tamir, "An $o(pn^2)$ algorithm for the p-median and related problems on tree graphs," *Operations Research Letters*, vol. 19, pp. 59–64, 1996.