Efficient Query Evaluation for DAG-shaped Hierarchies

En Cheng
Case Western Reserve University
Cleveland, OH, 44106, USA
en.cheng@case.edu

Ali Cakmak
Case Western Reserve University
Cleveland, OH, 44106, USA
ali.cakmak@case.edu

Z. Meral Ozsoyoglu
Case Western Reserve University
Cleveland, OH, 44106, USA
meral@case.edu

ABSTRACT

This paper focuses on the use of labeling schemes for evaluating queries on DAG structured data, such as pedigrees and ontologies that are stored in a relational database. We compare using Dewey$^+$ labeling, NodeCodes and its variants for the evaluation of ancestor/descendant queries on ontologies and inbreeding coefficient calculation on pedigrees. Ancestor/descendant queries can be answered based on the existence of the paths between nodes, while inbreeding coefficient calculations require the complete path information. While Dewey$^+$ performs slightly better for descendant queries for DAGs with low selectivity, it cannot be used to evaluate queries requiring path information, e.g. inbreeding coefficient queries for pedigrees. NodeCodes enable evaluation of both types of queries (requiring path information, and ancestor/descendant queries) efficiently.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval]:** Content Analysis and Indexing – *Indexing methods*.

1. INTRODUCTION

In biomedicine and bioinformatics, ontologies are nowadays commonly used to annotate objects of interest, such as biological samples, clinical pictures, or species in a standardized way. For example, the Gene Ontology (GO) [1] is the result of an effort aimed at providing a controlled vocabulary for describing roles of genes and gene products in any organism. In these applications, an ontology is merely a structured vocabulary in the form of a directed acyclic graph (DAG) of concepts. Typically, ontologies are stored together with the data they annotate in relational databases.

In past years, some web-based tools for GO have been developed to help researchers browse and search for GO terms. e.g., Ontology Search and Browser provided by WormBase [2]. A user can submit a GO term through the user interface, and then the browser returns all the paths from the root terms, *biological process*, *cellular component* and *molecular function*, to the input term. There are also some online tools for measuring the semantic similarities of GO terms. For instance G-SESAME [3] contains a tool for measuring the semantic similarity of GO terms. In G-SESAME, the semantics of a GO term are determined based on its relationship with the other terms, and its location in the entire GO graph. According to the functionalities of the GO-term -based tools, there are some frequently used queries for GO, such as ancestor query, descendant query, and path query. In existing tools, these queries are processed by iteratively traversing the GO graph, which lacks scalability. With the increasing use of large ontologies, such as SNOMED [4], it is even more important to have efficient and scalable methods for evaluating relationship-related queries on large DAG-shaped hierarchies.

In addition to GO, pedigrees are typically represented as DAGs as well. More precisely, a pedigree can be defined as "a simplified diagram of a family's genealogy that shows family members' relationships to each other and how a specific trait, abnormality, or disease has been inherited" [5] in human genetics. Pedigrees are hierarchical hereditary structures which are utilized to trace the inheritance of a specific trait or disease, calculate genetic risk ratios, and identify individuals at risk. Efficiently evaluating descendant/ancestor, as well as, finding nearest common ancestors are among the frequently run queries in human genetics. Furthermore, computing inbreeding coefficients is a significant and practical issue in modern genetics [5, 6]. The inbreeding coefficient measures the probability that the two alleles of a gene are received from the same ancestor [12]. It is zero if the individual is not inbred. Traditional calculation method utilizes recursive formula, which is quite time-consuming and cannot scale very well on large pedigrees. In this paper, our interests are not only efficiently computing inbreeding coefficients on large pedigrees, but also efficiently evaluating frequent query set, such as descendant/ancestor queries on large ontologies that are stored in a relational database.

In this paper, we present Dewey$^+$ labeling, NodeCodes, and two variants of NodeCodes by incremental and aggregative labeling, for DAG-shaped hierarchies. We then demonstrate the efficiency of descendant/ancestor query evaluation by using only the generated labels. In addition, we show NodeCodes can be used for evaluating more complex queries, such as inbreeding coefficient calculation. We also present experimental results evaluating the performance of four labeling methods for descendant/ancestor query as well as the inbreeding coefficients calculation.

2. FAMILY OF LABELING SCHEMES

In this paper, we focus on labeling schemes for DAG-shaped hierarchies, because most commonly used ontologies and all pedigrees are DAG-shaped.

2.1 Dewey$^+$ Labeling

For a tree T, the Dewey labeling [7] directly encodes the parent of a node in T, as a prefix of its label using for instance a depth-first tree traversal. More precisely, the label of a node v in T is $l(u)l(v)$ where $l(u)$ is the label of its parent u, $l(v)$ is the sibling order of v among u's children.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM-BCB 2010, August 2–4, 2010, Niagara Falls, NY, U.S.A.
Copyright © 2010 ACM ISBN 978-1-4503-0438-2... $10.00.

For a directed acyclic graph G, a spanning tree T of the graph G is constructed in the first place. As a result, the edges are divided into tree-edges and non-tree edges. The spanning tree T is encoded using a depth-first tree traversal. Therefore, each node in T is assigned a unique label. Then, given a non-tree edge from u to v, the label of u is propagated to v and all the descendants of v by storing an entry $[id(d), l(u)]$ where d is either v or a descendant of v, $id(d)$ is the id of d. This extension of the Dewey labeling for DAGs [8] is called Dewey$^+$ labeling in this paper.

2.2 NodeCodes

NodeCodes is a graph encoding scheme originally proposed for encoding single source directed graphs [9]. First the roots (nodes with in-degree 0) are labeled (we may consider adding a virtual source node s and making all roots children of s). For each node u in the graph, the set of NodeCodes of u, denoted NC(u), are assigned using a breadth-first-search traversal starting from the source node as follows: if u is the virtual source node s, then NC(u) contains only one element, the empty string. Let u be a node with a set of NodeCodes, NC(u), and $v_0, v_1, \ldots v_k$ be u's children in sibling order, then for each x in NC(u), a code xi is added to NC(v_i), where $0 \le i \le k$.

2.3 Variants of NodeCodes

A variant of NodeCodes for pedigrees called *Family NodeCodes* is introduced in [5]. It enables evaluating pedigree queries efficiently. Here, we consider partitioning ontologies to trees and present two labeling schemes based on this tree partitioning.

The idea behind tree-based partitioning for arbitrary taxonomies is based on the observation that, in a tree, each node has a single NodeCode. The automated identification of local tree structures in a taxonomy can be performed as follows: i) starting from level 0, traverse the taxonomy in breadth-first manner; ii) whenever a node v with multiple parents is encountered start a new tree with n as its root; iii) repeat this procedure until all the nodes in the taxonomy are exhausted.

2.3.1 Incremental Labeling

Root node n of a local tree contains two sets of NodeCodes: (i) *global NodeCodes*: standard NodeCodes that are inherited from n's immediate parents in other trees, and (ii) *local NodeCodes*: an independent newly generated local NodeCode, one per node in a tree. The other nodes in a tree are labeled using the standard NodeCode approach, where only the local NodeCode of the root node is considered for labeling other nodes in the same tree. Since each root node has a single local NodeCode, in each tree, it is guaranteed that all nodes (except the root node) have a single node code.

2.3.2 Aggregative Labeling

In this labeling scheme, global NodeCode set of the root node n in a tree T_i includes global NodeCodes of roots in those trees that are above T_i according to the hierarchical organization of the taxonomy, as well as the NodeCodes that are inherited from n's direct parents using standard NodeCode labeling scheme. The internal nodes in each tree are labeled in the same way as done in incremental labeling approach as illustrated in Section 2.3.1.

3. QUERY EVALUATIONS

3.1 Processing Descendant Query

Given a term t in a taxonomy T, the goal of this query is to find the set of all descendant terms of t in T.

1) Using standard NodeCodes: obtain the terms whose NodeCodes start with t's NodeCode.

2) Using incremental NodeCodes: first obtain the terms whose NodeCodes start with t's local NodeCode. Then, from each local tree, we can iteratively use the root of the local tree to obtain all its descendant terms.

3) Using aggregative NodeCodes: first obtain the terms whose NodeCodes start with t's local NodeCode. Then, from each local tree, we can use the root of the local tree to obtain all its descendant terms.

4) Using Dewey$^+$ labels: first get the descendants of t that are reachable to t by tree edges. Then, we process the (*TermID*, *ancestorLabel*) entries to get the descendants that are reachable to t with at least one non-tree edge.

3.2 Processing Ancestor Query

Given a term t in a taxonomy T, the goal of this query is to find the set of all ancestor terms of t in T.

1) Using standard NodeCodes: obtain the prefixes of all t's NodeCodes, which each unique prefix corresponds to an ancestor of t.

2) Using incremental NodeCodes: first obtain the prefixes of t's local NodeCode, plus the prefixes of t's global NodeCodes, if exist. Then, from each local tree, we can iteratively use the root of the local tree to obtain all its ancestor terms.

3) Using aggregative NodeCodes: first obtain the prefixes of t's local NodeCode, plus the prefixes of t's global NodeCodes, if exist. Then, from each local tree, we can use the root of the local tree to obtain all its ancestor terms.

4) Using Dewey$^+$ labels: get the ancestors of that are reachable to t by tree edges. Then, we process the (*TermID*, *ancestorLabel*) entries to return the labels of the ancestors of t that are reachable to t with at least one non-tree edge.

3.3 Processing Inbreeding Query

If an individual has inbreeding, the inbreeding coefficient is calculated using Wright's formula [10], where the paths from an ancestor to a given individual are required. First of all, NodeCodes has the capability for directly identifying all paths from a progenitor to a given individual. Although incremental and aggregative NodeCodes can iteratively identify the paths using both local NodeCodes and global NodeCodes, they are less efficient than standard NodeCodes. While, Dewey$^+$ labels cannot fully obtain the path information from the non-tree edges propagation table, which cannot contribute to the inbreeding query. Below presents the general outline for calculating the inbreeding coefficient of an individual p using NodeCodes.

CalculatingInbreedingCoefficients

Input: NodeCodes NC(p)
Output: Inbreeding coefficient of p

1. Find the NodeCodes of mother m and father f of individual p.
2. Identify common ancestors of mother m and father f.
3. For each common ancestor c
   a. Find the set of pairs of paths from c to m and f.
   b. Identify non-overlapping pairs of paths
   c. Find the number of generations between the common ancestor and the individual for the non-overlapping pairs of paths.
   d. Compute the inbreeding coefficient.

The details of using NodeCodes for inbreeding coefficients computation can be found in [5]. In addition, we also demonstrated the efficiency of using NodeCodes for kinship coefficients, and generalized kinship coefficients which are important genetic measures in modern genetics as well [6].

4. EXPERIMENTS

In this section, we show the efficiency of four labeling methods for ancestor and descendant queries on GO. The total number of labels for GO is listed in Table 1. We tested the effectiveness of four labeling methods using C# 2005 and SQL Server 2005. All queries were run on cold cache and the test machine was a 2.0Ghz Pentium 4 with 1GB RAM running Windows XP. For GO, we compare 5 methods: *Iterative*, *StandardNC*, *IncrementalNC*, *AggregativeNC*, and *Dewey*$^+$ *Labeling*. *Iterative* method refers to the method which the descendants/ancestors of a given term by iteratively traversing the GO graph.

Table 1. The number of labels using four labeling schemes

Methods	# of Labels	AVG (labels per term)
StandardNC	330,802 13	
IncrementalNC	52,126 2	
AggregativeNC	213,725 8	
Dewey$^+$ Labeling	129,557 5	

First, we run descendant queries on 5 different cases: each case corresponds to a different choice of input node and therefore of query selectivity. In this paper, for any query, its *selectivity* is defined as the proportion of terms that it returns. In this experiment, we choose the selectivity ranging from 58.7% to 0.01%. We run these five queries 50 times each and the average query processing time is shown in Table 2.

Table 2. Execution time (ms) of descendant query for 5 cases (% selectivity)

Methods	Case1	Case2	Case3	Case4	Case5
	58.7% 22.	1% 4.	96% 1.	27%	0.01%
Iterative	125031 47687		10359	2656	31
StandardNC	656 484 609 437				531
IncrementalNC	35453 2937		984	156	62
AggregativeNC	19875 1718		500	171	31
Dewey$^+$Labeling	28187 3343		140	125	31

According to Table 2, we can recommend the use of a particular labeling scheme according to the characteristics of a taxonomy. In GO, the average selectivity of all terms is 0.04%, which is close to Case5. Considering the space cost and time cost, *Dewey*$^+$ *Labeling* is a good choice for GO.

Then, we run ancestor queries for 5 cases having different selectivity. In this experiment, we choose the selectivity ranging from 2.13% to 0.118%. We run these five queries 50 times each and the average query processing time is shown in Table 3.

Table 3. Execution time (ms) of ancestor query for 5 cases (% selectivity)

Methods	Case1	Case2	Case3	Case4	Case5
	2.13% 1.	06%	0.552%	0.276%	0.118%
Iterative	468 234 125			78	31
StandardNC	484 500 765 671				296
IncrementalNC	359 109		31	31	31
AggregativeNC	156	93 62 62			31
Dewey$^+$Labeling	234 171		46	46	31

As can be seen, for Case1 and Case2, *AggregativeNC* performs best. As the selectivity becomes lesser, *IncrementalNC* performs best for Case3 and Case4. For Case5, *Iterative*, *IncrementalNC*, *AggregativeNC*, and *Dewey*$^+$ *Labeling* have the same query processing time.

In this experiment, we show the effectiveness of NodeCodes for inbreeding query by comparing the use of NodeCodes with a recursive method used in existing systems [11]. First, we show the effect of pedigree size on the query processing time. The results are shown in Figure 4 for the average time per query for each pedigree (as the pedigree size doubles).

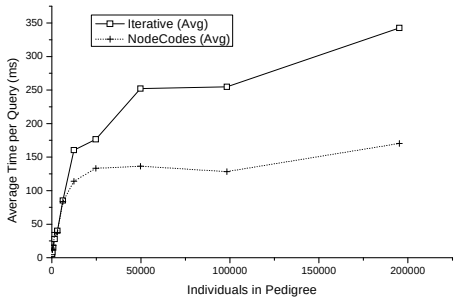

Figure 4: Effect of Pedigree Size on Average Query Time in Synthetic Pedigrees (100 random individ. each)

As can be seen, the expected time per query grew increasingly longer on iterative compared to NodeCodes as the pedigree size increased, from a comparable amount of time on the small pedigrees (<3000 individuals) to twice as much time per query on the largest.

5. ACKNOWLEDGMENTS

This work is partially supported by the US National Science Foundation grants DBI-0218061, ITR-0312200 and CNS-0551603.

6. REFERENCES

[1] Michael Ashburner et al. Gene Ontology: tool for the unification of biology, *Nature Genetics* 25, pp.25-29, 2000.
[2] WormBase Site Map, http://www.wormbase.org/db/misc/site_map?format=searches
[3] Zhidian Du et al., G-SESAME: web tools for GO-term-based gene similarity analysis and knowledge discovery, *Nucleic Acids Research*. 2009, Vol. 37, Web Server issue, *W345-349*.
[4] SNOMED, http://www.nlm.nih.gov/research/umls/Snomed/snomed_main.html
[5] B. Elliott, E. Cheng, S. Mayes, Z. M. Ozsoyoglu, Efficiently Calculating Inbreeding on Large Pedigrees Databases, *Information Systems*, 34(6):469-492, 2009.
[6] E. Cheng, B. Elliott, Z. M. Ozsoyoglu, Efficient Computation of Kinship and Identity Coefficients on Large Pedigrees, *Journal of Bioinformatics and Computational Biology*, 7(3):429-453, 2009.
[7] Online Computer Library Center, Dewey decimal classification, http://www.oclc.org/dewey
[8] V. Christophides et al., Optimizing taxonomic semantic web queries using labeling schemes, Web Semantics: Science, Services and Agents on the World Wide Web 1 (2004) 207-228.
[9] L. Sheng, Z. M. Ozsoyoglu, G. Ozsoyoglu, A Graph Query Language and Its Query Processing. *Proc. Of ICDE Conference*, 1999.
[10] Sewall Wright. Coefficients of Inbreeding and Relationship. In *The American Naturalist*, Vol. 56, No. 645, 1922.
[11] R. Agarwala et al. Software for Constructing and Verifying Pedigrees Within Large Genealogies and an Application to the Old Order Amish of Lancaster County. *Genome Research*, 98(8):211-221, 1998.
[12] Malécot, G. (1948) Les mathématiques de l'hérédité, Masson et Cie, Paris. Lange, K. (1997) Mathematical and statistical methods for genetic analysis, Springer-Verlag, New-York.