Prime Number-Based Hierarchical Data Labeling Scheme for Relational Databases

Serhiy Morozov

Submitted to the Department of Electrical Engineering and Computer Science and the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master's of Science

Committee Members:

_____
Dr. Hossein Saiedian
Professor and Thesis Adviser

_____
Dr. Arvin Agah
Associate Professor

_____
Dr. James Sterbenz
Associate Professor

Date Defended_____

The thesis committee for Serhiy Morozov certifies that this is the approved version of the following thesis:

Prime Number-Based Hierarchical Data Labeling Scheme for Relational Databases

Committee Members:

_____

Dr. Hossein Saiedian
Professor and Thesis Adviser

Dr. Arvin Agah
Associate Professor

Dr. James Sterbenz
Associate Professor

Date Approved: _____

Abstract

Hierarchical data structures are an important aspect of many computer science fields including data mining, terrain modeling, and image analysis. A good representation of such data accurately captures the parent–child and ancestor–descendent relationships between nodes. There exist a number of different ways to capture and manage hierarchical data while preserving such relationships. For instance, one may use a custom system designed for a specific kind of hierarchy. Object oriented databases may also be used to model hierarchical data. Relational database systems, on the other hand, add an additional benefit of mature mathematical theory, reliable implementations, superior functionality and scalability.

Relational databases were not originally designed with hierarchical data management in mind. As a result, abstract information can not be natively stored in database relations. Database labeling schemes resolve this issue by labeling all nodes in a way that reveals their relationships. Labels usually encode the node's position in a hierarchy as a number or a string that can be stored, indexed, searched, and retrieved from a database. Many different labeling schemes have been developed in the past. All of them may be classified into three broad categories: recursive expansion, materialized path, and nested sets. Each model has its strengths and weaknesses. Each model implementation attempts to reduce the number of weaknesses inherent to the respective model.

One of the most prominent implementations of the materialized path model uses the unique characteristics of prime numbers for its labeling purposes. However, the performance and space utilization of this prime number labeling scheme could be significantly improved. This research introduces a new scheme called reusable prime number labeling (rPNL) that reduces the effects of the mentioned weaknesses. The proposed scheme advantage is discussed in detail, proven mathematically, and experimentally confirmed.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real world information often consists of multiple pieces that are somehow related to each other. As a result, there exists a great demand for data management systems that can easily store, retrieve and search this kind of information. One type of such abstract data is hierarchy. Hierarchical structures are a very common representation of business organization, work breakdown, or any data that can be organized in a tree. Hierarchical data representations are often referred to as trees because of their similarity in shape. The root node is an ancestor of all other nodes, and the entire hierarchy is composed of branches of nodes starting from the root.

Hierarchical data management is not a new concept. In fact, hierarchical and network databases like IMS, MRI and TOTAL were quite popular during mainframe computing before relational databases took over (Haigh 2006). Hierarchical relationships within given data provide a very interesting insight into how the information is organized in real life. Hierarchical models are especially useful for organizing large amounts of data into related categories. The most common application of a hierarchical model is the file system on any modern operating system.

It allows thousands of files to be neatly organized into appropriate folders, subfolders, etc. Another popular hierarchical model is the Domain Name System (DNS) which organizes server names based on predefined structure: top level domain (e.g. edu, com, net), second level domain (e.g. wikipedia.com, google.com, yahoo.com), and multiple sub-domains (e.g. maps.google.com, mail.google.com, tv.yahoo.com). Both of these labeling schemes are variations of the materialized path model discussed in section 2.3. Each one specifies the path from the root of the hierarchy to a specific node. This results in an accurate representation of node relationships in a tree that is so difficult to recreate in relational databases.

A defining distinction between hierarchical and relational data management is the way each method locates data. Hierarchical systems are best suited for gradual refinement of the search criteria or limiting the search to a specific category, subcategory, etc. Due to their advanced indexing ability, relational database systems excel at searches based on exact criteria. Both kinds of functionality are very useful; however, no one system can provide both of them. In fact, this is why modern operating systems generate a flat file system index, an optimized inventory of system files, in addition to maintaining all files in a hierarchy. As an alternative, attempts have been made to add the hierarchical functionality to an already existing relational database. This research is focused on the latter topic.

1.1 Justification

Florescu and Kossmann discuss three classical approaches to managing hierarchical data. The most apparent approach is to build a custom system specifically designed and optimized towards handling this kind of information. The authors discuss some of the most prominent research prototypes such as Rufus, Lore, and Strudel (Florescu and Kossmann 1999b).

Rufus is a system based on an object-oriented database with extensible class hierarchy and text search functionality (Shoens, Luniewski, Schwarz, Stamos, and Thomas 1993). Lightweight Object Repository (Lore) is a database management system (DBMS) designed specifically for managing semistructured data that uses DataGuides instead of conventional database schema. The DataGuides are essentially structural summaries of the data used to maintain the hierarchical relationships (McHugh, Abiteboul, Goldman, Quass, and Widom 1997). Strudel is a website content and structure management system that supports abstract data management by maintaining the hierarchical relationships separately from externally stored data (Fernández, Florescu, Kang, Levy, and Suciu 1998). Deutsch, Fernandez, and Suciu presented a semistructured to relational data (STORED) query language and storage schema. STORED performs various data mining operations in order to extract the scheme from existing data and then build the appropriate relations. This custom solution is very similar to other products such as Lorel (Quass, Rajaraman, Sagiv, Ullman, and Widom 1995), UnQL (Buneman, Davidson, Hillebrand, and Suciu

1996), MSL (Papakonstantinou, Abiteboul and Garcia-Molina 1996), and StruQL (Fernandez, Florescu, Levy, and Suciu 1997, Fernández et al. 1998).

A similar approach involves using an object-oriented database system that models the nodes in a hierarchy as objects and edges as properties. Object data types allow flexible storage capabilities and easily updatable trees. Florescu and Kossmann focus on two commercial products that implement this method, O_2 and Objectsore (Deux et al. 1990). Atkinson, DeWitt, Maier, Bancilhon, Dittrich, and Zdonik outlined the main features and desired characteristics of object oriented database management systems (OODBMS). They also performed a comprehensive survey of many existing products including Gemstone, Vision, Orion, Flavors, Lore, Simula, Vbase and O_2. DeWitt et al. compared these systems in regards to extensibility, data persistence, concurrency and recovery functionality. The authors concluded that many of the considered products had satisfactory results, which made those OODBMS a viable solution for storing hierarchical data.

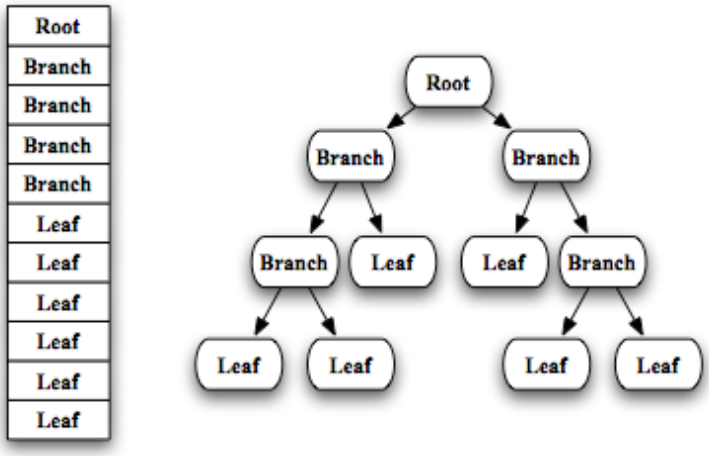
The third approach is to use an existing relational database system and map the semistructured data onto the database tables. Florescu and Kossmann focused on the performance of all of the mentioned solutions. The authors concluded that modeling hierarchical data in a relational database is the most favorable. In fact, the Florescu and Kossmann demonstrated that relational data management solutions could outperform other approaches, especially when given complex queries on large datasets (Florescu and Kossmann 1999b). A supporting argument by Jiang, Lu,

Wang, and Yu states that relational database solutions can outperform special purpose XML repositories such as Lore (Jiang et al. 2002a).

1.2 RDBMS Hierarchical Labeling Problem

The goal of hierarchical labeling schemes is to capture structured data into relational databases while maintaining the accuracy of the real world relationships. The reason such abstract data is not stored in its raw format is because relational databases offer greater flexibility, performance and scalability. Figure 1.1 shows an example hierarchy that needs to be stored in a flat database table.

Figure 1.1: RDBMS Hierarchical Labeling Problem



As one can see, certain relationship information is lost. That is why labeling schemes are needed. They record additional information that captures the relationships among nodes. The goal of each labeling scheme is to minimize the space required to record this information while maximizing the performance and available functionality. Different labeling schemes employ various encoding techniques to achieve this goal. Each label encoding varies in size and ability. Some tree models allow fast searches,

while others facilitate simple updates without the need for re-labeling. Label size is also an issue when modeling hierarchies. Physical computer limitations put a definite limit on the maximum label size that can be managed with the necessary precision. Theoretically, there is no fundamental limit on label size. Modern computers can successfully determine relationships between labels of arbitrary length. However, that kind of computations would take up a lot of time and resources. As a result, there are limits on the label sizes that allow acceptable response time.

This research is focused on improving labeling scheme performance by reducing the label size. Smaller labels result in faster computations that improve overall performance. Additionally, smaller labels are located closer to each other (e.g. numeric labels) so overall label size grows slowly. As a result, labeling schemes with small labels produce more compact model representations and are capable of capturing more complex hierarchies because they do not run out of space as quickly. This research introduces a new labeling scheme that is able to harness all of the benefits associated with small labels.

1.3 Significance

Each labeling model implementation can be optimized by introducing more clever ways of encoding the label information. This thesis is focused on the performance and space utilization problem of the prime number labeling (PNL) scheme, introduced by Wu, Lee, and Hsu. This scheme attempts to reduce the model size by storing aggregate information from which the original labels can be inferred. PNL scheme uses consecutive prime numbers and their products to label each node and its

ancestors. Since self-labels use unique prime numbers, their products (i.e. the ancestor labels) grow exponentially. Figure 4.2 demonstrates this problem graphically. The performance of the PNL scheme decreases rapidly when the products of continuously growing labels become so large that they can no longer be managed with the necessary precision. This shows inefficient utilization of the available number space.

Theoretically, the PNL scheme does not lack anything. It is able to accurately record and retrieve hierarchical information from a relational database. However, as the size of the hierarchy increases, the space and processing requirements grow accordingly. These two requirements are limited by the physical characteristics of the hardware. Even though modern technology has drastically improved the storage and processing capabilities, there are still distinct limitations on the numbers that can be manipulated. For example, MySQL v5.0.45 database can only handle integers up to 64 bits long or 1.84×10^{19}. Besides using very big labels, which take longer to process, the PNL scheme does not utilize the available number space efficiently, which in turn limits the model utility. A desired improvement would decrease the size of the labels, thus making the computations easier, improving performance, and increasing model capacity.

1.4 Expected Contributions

We propose a more capable labeling scheme that improves upon the PNL model. Our reusable prime number labeling (rPNL) scheme is able to reuse small prime numbers throughout the tree, which decreases the label size and improves performance. The

scheme also inherits all of the strengths of the PNL model such as fast descendant searches and simple ancestor determination. The proposed scheme is especially suited for deep hierarchies up to 15 levels. It generates parent labels that are approximately half the size of the PNL scheme. Additionally, the proposed method has superior model capacity and a label recycling functionality that is not present in the PNL model. In fact, the rPNL model can successfully record over 91 million maximum depth paths whereas the PNL scheme can only handle one.

1.5 Evaluation Criteria

Initially, we introduce the rPNL labeling scheme and the mathematical rules and concepts that it is based on. We then prove that the proposed solution is, in fact, capable of accurately capturing and retrieving hierarchical data. We compare PNL and rPNL label size growth patterns and determine the effect they have on each model's capacity. Finally, we perform benchmark testing of PNL, rPNL, and other representative models on several hierarchies with different depth and fan-out. We measure performance of each scheme against the most common functional requirements: tree labeling, direct child lookups, descendent searches, ancestor determination, and overall model update flexibility. All experiments are conducted five times, and an average measurement is noted in order to decrease the effect of any interfering software processes. Trial testing has shown that there are no significant changes in experiment results when they were ran more than five times. The results are presented as graphs and discussed in detail.

1.6 Thesis Organization

The thesis is organized into the following chapters:

- **Chapter 1: Introduction** – The background of the problem, significance and a justification of a solution.
- **Chapter 2: Previous Work** – The current state of the art in RDBMS labeling schemes in tree categories: recursive expansion, nested set, and materialized path.
- **Chapter 3: Prime Number Labeling Scheme** – The prime number labeling scheme and some of its limitations.
- **Chapter 4: Reusable Prime Number Labeling Scheme** – The proposed reusable prime number labeling scheme and calculations and proofs that demonstrate the validity of the suggested model.
- **Chapter 5: Evaluation and Analysis** – The experimental results, their explanations and analysis.
- **Chapter 6: Conclusions and Future Work** – The conclusions and future research direction in the field.

Chapter 2

Previous Work

In order for any labeling scheme to be successful, it should ensure that the parent–child relationship among the nodes is readily available or easily computable. A number of techniques are used by different schemes in order to accurately model a tree. As Joe Celko specified, inheritance is another very important property of any hierarchical model (Celko 2004). Therefore, a good labeling scheme should support multigenerational ancestor–descendent relationships.

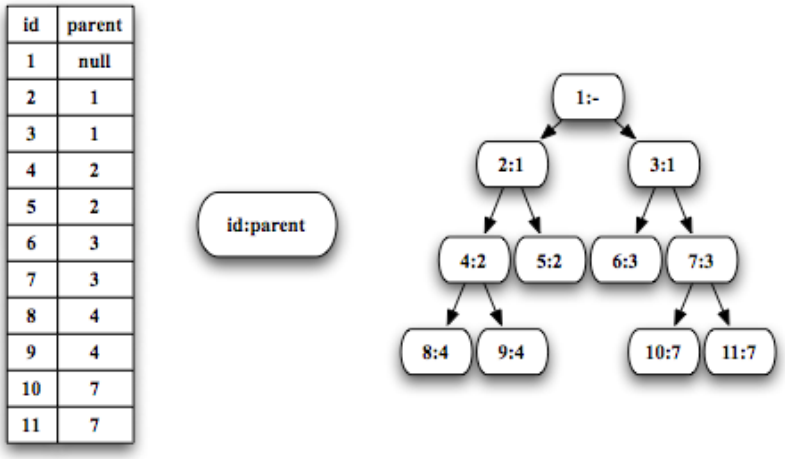
Vadim Tropashko identified two major categories in SQL representation of hierarchies: recursive expansion and tree encodings. Tree encodings are further divided into two groups: materialized path and nested sets (Tropashko 2005). This chapter covers examples of all three models and outline their benefits and drawbacks.

2.1 Recursive Expansion Model

The recursive expansion model allows access to only one node at a time. In order to expand one's view of the tree, additional requests must be performed and intermediate results saved. The adjacent list method is an example of a recursive expansion model. It is probably the most natural way to store hierarchical data,

especially for procedural programming language developers who are used to the concept of recursion. Each record contains a self-label and a label of a direct parent. Oracle was the first commercial database to use such an approach (Celko 2004). Storing hierarchical data by shredding it into rows of a relational database table is still a widely used technique (Shanmugasundaram, Tufte, Zhang, He, DeWitt, & Naughton 1999). Figure 2.1 shows the adjacent list database table as well as the actual hierarchy it models.

Figure 2.1: Adjacent List Labeling Scheme



Given a node, its direct parent-label is available. Since all siblings share the same parent-label, sibling queries become trivial. Adding a node to an existing tree requires no additional operations. Ancestor queries are much more difficult. A number of requests must be performed, each retrieving the label of a previous parent in the hierarchy. Recursion is an extremely powerful concept, but it may require significant computer resources even if the computations are very simple. Certain programming languages such as Lisp and Prolog were specifically designed with

recursion in mind. However, the majority of other programming languages are not as fit for recursion and, as a result, recursive expansion model implementations are usually quite slow and resource intensive (Celko 2004). Additionally, descendent searches are extremely inefficient, especially in large trees, in which intermediate results must be stored in temporary tables or kept in the memory.

The adjacent list method uses consecutive integers as its labels. It is a very compact model because every possible number is likely to be utilized. Reusing deleted labels, however, is not a default behavior of this scheme so an uneven number distribution is possible if the hierarchy is modified frequently. Several papers have been written about successfully using this model through recursive queries (Brandon 2005) and multiple self-joins (Shui, Lam, Fisher, and Wong 2005, Florescu and Kossmann 1999a, David 2003).

The Edge approach presented by Florescu and Kossmann implements the adjacency list model. It involves only one normalized self-referencing table that stores pointers to source and destination nodes. A similar approach, called Monet, stores the pointers to source and destination nodes across multiple small, semantically homogeneous relations. In other words, all nodes on the same level are placed in the same relation. For example, a path $a_1 - a_2 \ldots a_n$ will result in $n+1$ relations. As a result, the individual relations are much smaller, but there are a great number of them. In fact, there must be a relation for each possible path (Schmidt, Kersten, Windhouwer, and Waas 2000). Yet another related approach, called XMLEase, introduces redundant links between each node and all of its ancestors. The entire

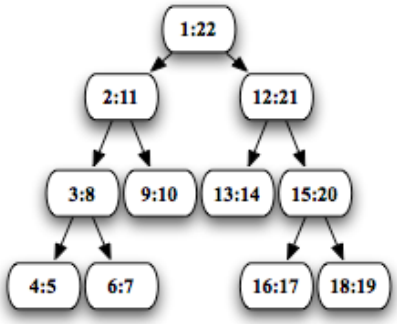
hierarchy resides in single relation, where the number of ancestor attributes determines the maximum tree depth. For example, a tree with a maximum path $node_1$–$node_2$…$node_n$ will require *n* ancestor attributes (Elçi and Rahnama 2006). Clearly this kind of labeling scheme wastes a lot of space and is not very well fit for dynamic trees with varying depth.

2.2 Nested Set Model

The majority of performance issues in hierarchical models are related to descendent searches. In particular, it is difficult to quickly determine all nodes that are ancestors of a given parent. This task is especially difficult for recursive expansion model. Ideally, this function should be very simple, similar to determining if one number is bigger than the other. The interval based labeling scheme, called Range, does just that. Each node receives a number range as a label and then an ancestor–descendent relationship may be calculated by determining if one number range is contained within another. This technique is called Dietz's numbering scheme (Dietz 1982). Figure 2.2 shows the interval based database table as well as the actual hierarchy it models.

Figure 2.2: Interval Labeling Scheme

id	head	tail
1	1	22
2	2	11
3	12	21
4	3	8
5	9	10
6	13	14
7	15	20
8	4	5
9	6	7
10	16	17
11	18	19



Each node receives two numbers as a label. The numbers represent the beginning and the end of a number range. For example, the node with 3:8 range is a parent of all labels starting with 4 or more and ending with 7 or less. Interval based labeling scheme is the fastest way to do descendent search, which is difficult for other schemes to accomplish (Tropashko 2005). Another advantage of the nested set method is that labels may be assigned a fixed size, which allows database optimization and improves performance (Shui et al. 2005). The performance advantage of this method is strictly in descendent searches, as it is computationally easy to locate all numbers within a range. Tropashko stated that ancestor searches would be especially slow for this model, because it is considerably more difficult to search all the ranges that contain a specific number (Tropashko 2005).

A major disadvantage of this scheme is that frequently changing tree structures will require a considerable number of label adjustments as the changes will stretch/shrink multiple number ranges. Assuming that any node has an equal chance

on being changed, an average update will cause half of the tree structure to be re-labeled. In order to avoid this, a labeling scheme that allows new labels to be inserted or removed without re-labeling is needed. Böhme and Rahm were able to achieve this with a dynamic level numbering (DLN) scheme by padding the existing container so that the new labels will have enough space. This requires anticipating the number of future nodes, which is not very reliable.

The dyadic rational number encoding scheme and its Farey fractions alternative are also capable to reducing the re-labeling issue (Tropashko 2005). Both methods use fraction properties to reduce the need to re-label, as there always exists a third fraction that is between the two existing ones. The resulting hierarchy may be quickly searched and easily updated. However, it does not utilize the number space efficiently and does not scale well. A similar scheme called Quartering-Regions Scheme (QRS) was developed by Amagasa, Yoshikawa, and Uemura. It uses floating point numbers and their binary equivalents as self labels that allow new nodes to be inserted without re-labeling. This approach does not completely eliminate the problem of re-labeling, but it does improve it significantly.

The XML indexing and storage system (XISS) is another variant of interval encoding (Li & Moon 2001). Instead of head and tail labels, there are head and size labels. The tail label is calculated, which reduces the space required for the labeling scheme and improves the update flexibility. Additionally, this approach uses the concept of extended preorder in order to handle future node insertions. In other words, extra space is reserved at each node region in order to avoid future re-labeling.

A similar labeling scheme is called BIRD - Balanced Index-based numbering scheme for Reconstruction and Decision (Weigel, Schulz, & Meuss 2005). This approach follows the same labeling technique, but does not utilize consecutive numbers. Both schemes simply delay the need for re-labeling as they allow only a limited number of new nodes to be inserted before a global re-labeling must occur.

In response to this issue, a few authors proposed schemes specifically designed to allow unlimited node insertions without the need for re-labeling. The quaternary encoding for dynamic XML data (QED) scheme supports label insertion without re-labeling by utilizing the lexicographical and not numerical ordering (Li & Ling 2005a). Four numbers are used to encode each node's region. As a result, this scheme minimizes the individual label size while supporting infinite inserts between any two existing labels (Li & Ling 2005a). LSDX, a labeling scheme for dynamically updating XML data, uses both letters and numbers to describe the depth of the node as well as its order (Duong & Zhang 2005). The root node receives `0a` as a label because its depth is `0` and it is the first node in its generation. The first child of the root node will be labeled `1a.b`, second `1a.c`, etc. These labels uniquely identify each node and allow additional nodes to be inserted easily. For instance, a new node between `1a.z` and `1a.zb` would receive a label `1a.zbb` according to the lexicographical ordering.

Khaing and Thein pointed out a problem with LSDX labeling scheme. The authors consider a case when one node must be inserted between `1a.z` and `1a.zb` and another between `1a.zb` and `1a.zc`. In both cases the same label `1a.zbb` will be

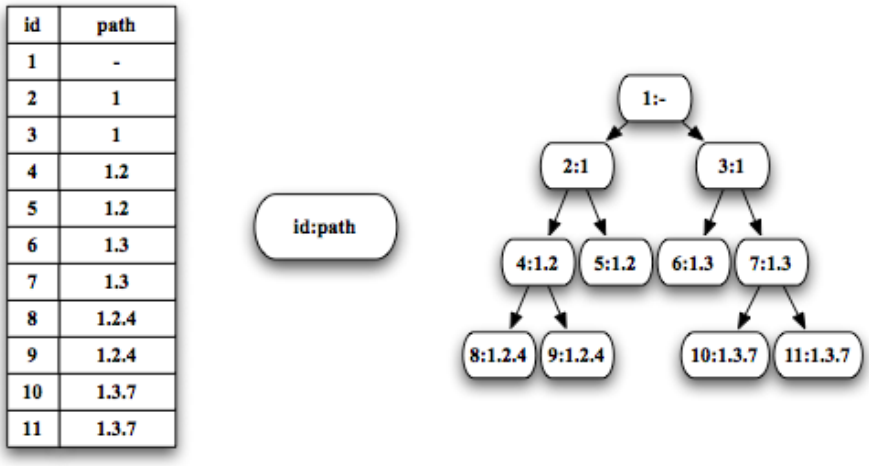
generated. Because of such collisions, the authors conclude that LSDX scheme does not allow arbitrary node insertions. Khaing and Thein also propose a solution to the label collision problem that occurs when the self-label of each node is limited to numbers or digits only. The authors developed a labeling scheme for dynamic trees that is very similar to LSDX, but uses a combination of numbers and digits as self labels.

2.3 Materialized Path Model

The prefix-based labeling scheme proposed by Cohen, Kaplan and Milo is a typical example of materialized path model. It is very simple to understand as each new node inherits its parent's path and appends its own label to it. This makes determining the parent–child relationship a matter of comparing label prefixes. Unlike the nested sets approach, this labeling scheme allows inserting new nodes without any re-labeling. The great benefit of path enumeration models is that parent information is encoded in the node's label itself. In fact, node relationships are usually clearly visible to a human scanning though a list of nodes. This also eliminates the need to make costly database requests to determine the node's ancestors. Additionally, searching a materialized path tree does not involve any kind of recursion or expensive joins.

The path from the node to root is usually enumerated with numbers of a specified length or encoded with delimited strings. Figure 2.3 shows the prefix-based database table as well as the actual hierarchy it models.

Figure 2.3: Prefix-Based Labeling Scheme



The Dewey decimal system, which is standard in library catalogs, uses numbers as well as periods and letters to categorize books. For example *PHP Hacks: Tips & Tools For Creating Dynamic Websites* by Jack Herrington has 005.133 Dewey classification number associated with it. This means it can be categorized under the following subjects: "Computer programming, programs, data" (005), "Programming" (005.1), "Programming languages" (005.13), and "Specific programming languages" (005.133).

The ORDPATH labeling scheme is an improved version of the Dewey decimal system for storing hierarchical data in a relational database (O'Neil & O'Neil 2004, Leonard 2006). Since label growth is an issue with materialized path schemes, this approach utilizes few optimizations such as assigning odd-number labels to newly inserted nodes thus leaving even number labels for future additions (Leonard 2006).

Another very common example of path enumeration labeling model is the US Postal Service ZIP code. This label is structured in such a way that each digit carries some geographical information. The information ranges from more general, such as postal region and state, to more specific, such as city and post office location (Celko 2004, Böhme and Rahm 2004). Such a five-digit label can handle up to 100,000 unique values, which is sufficient for relatively small hierarchies. However, if the tree grows, label size must also increase. The ZIP code decimal scheme also produces a strictly balanced structure with limited fan-out, which means that there are at most ten root branches that must be equal in size. Because only ten digits may be used, any node may have at most ten children. The problems occur if a node has more than ten children, e.g. densely populated state, or if some nodes only have a few children, they are wasting the allocated space.

In deep hierarchies, some paths may be lengthy and their encodings take up a significant amount of space. Using more compact numerical labels instead of character based ones has additional advantage in which some queries may be sped up by using fast numeric comparisons. Scanning character labeled paths usually involves complex pattern matching, which is slow and inefficient. However, if numbers are used (e.g. ZIP code encoding), a mathematical function may be used to quickly locate and update the necessary nodes. Path enumeration label length usually increases linearly as the depth of the hierarchy grows. As a result, performance is negatively affected because no fixed amount of space may be allocated for hierarchical

information (Shui et al. 2005). This makes path enumeration model best suited for relatively small, balanced, and static hierarchies.

The XParent approach implements a materialized path model (Jiang, Lu, Wang, & Yu 2002b). Unlike Edge, this scheme explicitly stores available paths in a separate relation. Unlike Monet, the path information is contained in a single table. This data may be materialized into a new table to support ancestor–descendant relationships (Jiang et al. 2002b). An alternative to XParent, called XRel, is able to model hierarchical information in terms of a combination of path and region (Yoshikawa & Amagasa 2001). Similarly to Monet, a separate relation is created for each node type. Unlike Monet, XRel stores all existing paths in a separate relation. Since this scheme does not maintain edge information, sibling nodes must be uniquely identified. In order to preserve the ordering and containment relationship among nodes, XRel records the region (start and end position) of each node. Scheme combinations such as this one often introduce better functionality at the cost of increased complexity and size.

A very interesting encoding technique is discussed by Tropashko. Given two co-prime numbers a and b such that $1 \le b \le a$ and certain information may be encoded and decoded. For example, to encode a path 1.2.3.4.5 one would simplify the following continued fraction.

$$1+\cfrac{1}{2+\cfrac{1}{3+\cfrac{1}{4+\cfrac{1}{5}}}}=\frac{225}{157}$$

The Euclidean algorithm is used to decode the path.

$$225 = 68 + 1 \times 157$$
$$157 = 21 + 2 \times 68$$
$$68 = 5 + 3 \times 21$$
$$21 = 1 + 4 \times 5$$
$$5 = 0 + 5 \times 1$$

This kind of encoding is not very computationally intensive and it accurately captures the path with relatively small labels. The resulting labels are unique and could be indexed for improved performance. However, ancestor information is not easily accessible without actually decoding the labels, which makes descendant searches extremely slow.

2.4 Other Approaches

Provided there is a well-known hierarchy structure (maximum fan-out, depth, etc), an optimized database schema may be generated. A table may be created for every level in a hierarchy and then easily searched using existing one-to-many relationships between the tables. This technique may be used to create an entire database schema based on document type definition (DTD) to store documents of previously known structure (Shanmugasundaram et al. 1999, Christophides, Abiteboul, Cluet, & Scholl 1994). This approach is clearly the fastest, because it takes full advantage of the database optimization algorithms, indexes, etc. However, the structure of the hierarchy is rarely known ahead of time, which limits the utilization of this approach.

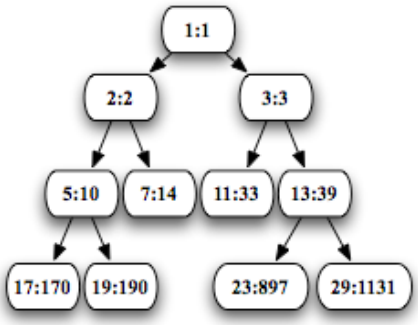
Chapter 3

Prime Number Labeling Scheme

A recent work by Wu, Lee, and Hsu introduces a new way to encode the hierarchy information with prime number labeling scheme. In this top-down scheme each node receives two numbers: a unique prime number called *self-label* and another number called *parent-label*. Each parent-label is divisible by all of its ancestors' self-labels, because the label is in fact a product of all ancestor self-labels and the self-label of the node. Figure 3.1 shows the prime number labeling database table as well as the actual hierarchy it models.

Figure 3.1: Prime Number Labeling Scheme

id	prime	label
1	1	1
2	2	2
3	3	3
4	5	10
5	7	14
6	11	33
7	13	39
8	17	170
9	19	190
10	23	897
11	29	1131



This labeling scheme allows determining the relationship between two nodes by simply comparing two numbers. If the self-label of node X divides node Y's parent-label, then node X is considered to be a parent of node Y. Likewise, all nodes whose parent-labels are divisible by prime P are descendants of the node with P as a self-label. A lightweight modulo function may be utilized for this purpose. A modulo function is a way to determine if a given number is divisible by another number without a remainder. It is not computationally intensive and can quickly operate on very large numbers.

The PNL scheme inherits all the benefits of the materialized path model while introducing much smaller, numeric labels that can be managed by fast and lightweight mathematical functions. Adding a node to such a tree is very simple. A self-label is assigned a value of any unused prime number and a parent-label is simply a product of this prime with a parent-label of the parent node (Wu, Lee, and Hsu 2004). Unlike the rigid nested-sets method, this approach is very flexible as no re-labeling is required when new nodes are added to the tree.

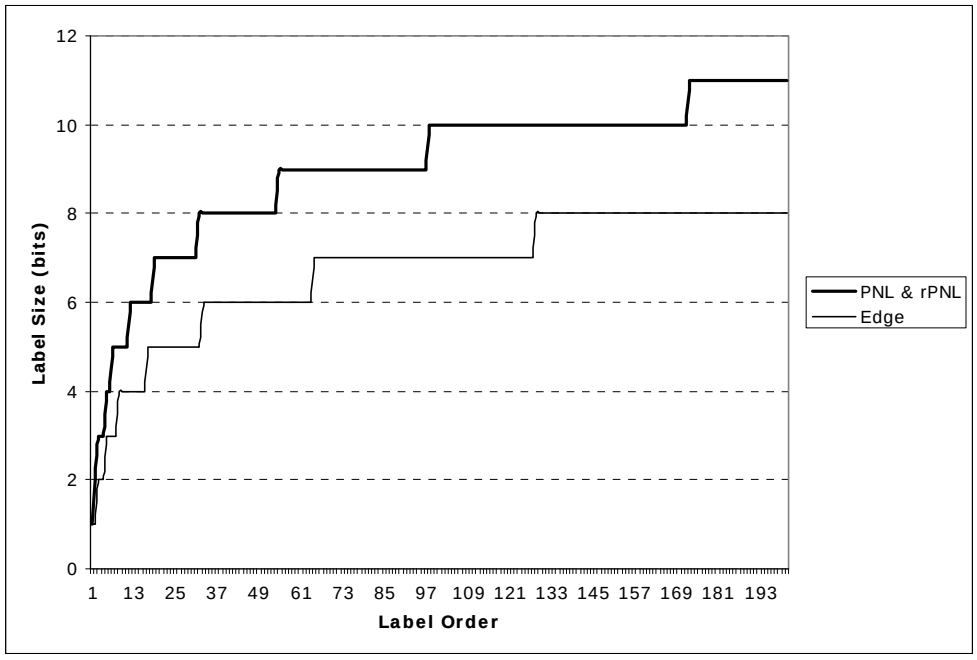
One valid disadvantage of PNL scheme is the fact that each descendant search must go through the entire dataset in order to determine the parent–child relationships. This may be particularly slow on very large datasets. However, this method of searching the hierarchy is a better alternative to multiple joins especially in very deep hierarchies as implemented in the recursive expansion model. Parent-labels in this scheme cannot be indexed or sorted in any particular way to minimize the number of operations needed for each scan. Another major disadvantage of PNL

scheme is that each prime number may only be used once. This helps establish the uniqueness of the labels but also causes the magnitudes of each subsequent parent-label to increase rapidly. This shortcoming is especially apparent in deep hierarchies. Even though the authors propose a number of optimizations to improve the label space usage, these improvements provide only a limited result.

3.1 PNL Label Size Issues

The authors also show that the size of the label grows mostly due to the increasing depth of the tree, which requires multiple prime numbers to be multiplied. A more detailed discussion of this issue is covered in section 4.2. Fan-out, on the other hand, affects the label size very slightly as the increase is due to relative difference between consecutive prime numbers. Figure 3.2 shows the label size requirements for a number of nodes on the same level.

Figure 3.2: Effect of Fan-Out on Label Size in PNL



The label length is measured in the minimum number of bits required to represent the label. The graph shows the fan-out of the tree at level one, which means that the label is the same as the prime assigned to the node. In other words, this graph models the size requirements for storing consecutive prime numbers (PNL & rPNL) and consecutive integers (Edge). This graph clearly shows that prime number labels are much bigger and grow faster than Edge labels. Li, Ling and Hu did multiple comparisons of this scheme to two variants of the nested set model and a Dewey prefix scheme. The PNL scheme required considerably more storage and had a much longer response time.

After comparing their prime number labeling scheme to two other prefix-based dynamic labeling schemes Wu, Lee, and Hsu concluded that when the hierarchy has a large fan-out but limited depth their method consumes less storage

space. However, when the hierarchies are very deep with limited fan-out, the prime number labeling scheme is not the best option. The authors believe that the PNL scheme is appropriate, as the majority of analyzed XML documents have less than 8 levels of nesting and fan-outs up to 10,000 (Mignet, Barbosa, and Veltri 2003). Even though this encoding scheme is not the most compact, it is least affected by the structure of the hierarchy (Wu, Lee, and Hsu 2004). PNL scheme scalability is limited by the label size restrictions. However, it uses numerical labels, which allows taking advantage of standard relational database optimizations. Härder, Haustein, Mathis, and Wagner performed benchmark experiments with PNL scheme modeling trees up to 37 levels deep and a maximum fan out of several millions. The author concluded that PNL scheme was not the most optimal solution for such complicated hierarchies.

3.2 Problem Statement

There are multiple areas of improvement in PNL model. For instance, the prime number labeling scheme does not allow self labels to be reused. This causes parent-labels to grow exponentially, which significantly limits the model capacity, increases overall model size, and slows down performance. In fact, there is a discernible limit on the maximum depth and fan-out dictated by the hardware limitations. This issue has been identified by the authors and confirmed by independent research. Additionally, this approach does not natively support label recycling. Ability to reuse deleted labels results in much higher number space utilization and improves all of the mentioned drawbacks. This functionality may be implemented with PNL at the cost of decreased performance. This scheme may be an excellent solution in the future,

when hardware limitations are no longer an issue. However, prime number labeling scheme is not the optimal solution for the currently available resources. This research introduces a new labeling scheme called rPNL that is a better alternative to PNL.

Chapter 4

Reusable Prime Number Labeling Scheme

A number of different approaches have been developed attempting to improve the shortcomings of PNL scheme. For instance, Li, Ling and Hu propose a new algorithm that allows reusing deleted labels in order to control the label size increase rate. Davy Preuveneers and Yolande Berbers recommended decreasing the label size by labeling each node with two different parent-labels that could then be factorized into a single set of parent self-labels. The major contribution of this research is a new reusable prime number labeling scheme called rPNL. The reusable prime number labeling scheme attempts to improve on the same problem, the label size. If prime number self-labels are reused, the resulting parent-labels will be considerably smaller. This should increase model capacity and improve performance. Additionally, the rPNL scheme uses the available number space much more efficiently by utilizing labels that are located close to each other.

By definition, prime numbers are numbers that are not divisible by anything except 1 and the number itself. This means that every non-prime number may be expressed as a product of one or more prime numbers. According to the fundamental

theorem of arithmetic also known as the *unique factorization theorem*, every natural number n greater than 1 can be written as a unique product of prime numbers p_k. $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_n}$ (Lindemann 1933). This formula is very helpful if n and p_k are used as labels. Given n, factorization will reveal the list of all parent-labels. This process may be done algorithmically, without the costly database requests. Note that factorization is a computation intensive operation. In fact, modern cryptography methods rely on the fact that factorization of very large numbers is computationally infeasible. However, if n is a relatively small number, factorization costs are negligible relative to the cost of multiple database requests. A simple experiment on GNU/Linux *factor* command shows that the longest time to factor a 64-bit integer is just over a tenth of a second. The largest integer most of the current databases can handle is 64 bits long. Therefore, factoring integers of that size is in fact a more efficient alternative to multiple queries. The reusable prime number labeling scheme attempts to minimize parent-label n to take advantage of the factorization as a method of deriving parent information.

Wu, Lee, and Hsu use the Chinese remainder theorem to record the global order of the nodes. The proposed method uses the same idea to record the order of the parents' self-labels used. If prime numbers are allowed to be reused throughout the hierarchy, repeating labels are bound to be created. Reusable prime number labeling scheme distinguishes between the order of the prime numbers as well as their product.

The Chinese remainder theorem states that there exists a number n that satisfies k simultaneous congruencies

$$n = n_1 \bmod m_1, n = n_2 \bmod m_2 \ldots n = n_k \bmod m_k$$

if $n_i = n_j \bmod \gcd(m_i, m_j)$ for all i and j (Howard 2002). The solution n is then congruent to the least common multiple of all m_i. In other words, $n = n \bmod lcm(m_1, m_2 ... m_3)$. Because every modulo used is always prime, the following holds true regardless of the prime numbers chosen.

$$lcm(n_1, n_2 \ldots n_k) = \prod_{i=1}^{k} n_i$$

$$\gcd(n_1, n_2 \ldots n_k) = 1$$

The above solution proves that, because only prime numbers are allowed to be self-labels, a simultaneous congruence (SC) number is guaranteed to exist for any combination of the prime numbers. The SC number is also guaranteed to be less than the product of all the prime numbers used. Therefore, as long as there is space for the parent-label, there will be enough space to record the order of the prime numbers used.

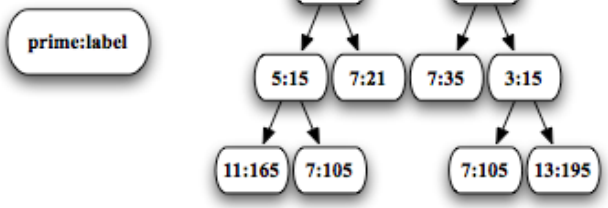
The reusable prime number labeling scheme uses SC number to encode the position of the prime numbers used on the path such that $SC = i \bmod p_i$. In order to maintain functionality with this method, three rules must be enforced. First, only unique prime numbers may be allowed on each individual path. In other words, it is impossible to have two simultaneous congruencies with the same modulo but different remainders. Second, only unique self-labels may be allowed among siblings. In order to uniquely identify the siblings, they must have distinct self-labels. Third,

each self-label must be larger than the level at which it resides in order to avoid confusion. For example, two different paths could generate identical numbers, e.g. $SC = 1 \bmod 3 = 4 \bmod 3$.

Figure 4.1 shows rPNL database table as well as the actual hierarchy it models. Note that according to the three rPNL rules, self-label 2 should have been used for one of the first-generation nodes. However, two different self-labels were deliberately chosen to demonstrate the fact that none of the three rPNL labels can uniquely identify a node.

Figure 4.1: Reusable Prime Number Labeling Scheme

id	prime	label	sc
1	1	1	1
2	3	3	1
3	5	5	1
4	5	15	7
5	7	21	-5
6	7	35	16
7	3	15	-4
8	11	165	322
9	7	105	52
10	7	105	-4
11	13	195	-439



With this approach self-labels and parent-labels are assigned similarly to PNL scheme. However, according to the three rules mentioned above, self-labels are not required to be globally unique prime numbers. In fact, this scheme ensures that the prime number chosen is the smallest possible number that is 1) bigger than its position on the path 2) unique within the given path and 3) unique within the siblings. This reduces the label size growth that is so problematic with PNL scheme. Also it

forces the deleted labels to be automatically reused which results in a much more efficient use of the number space.

There is some redundancy in a way that rPNL scheme stores the labels. The parent-label and SC number are based on the same set of prime numbers. However, the parent-label does not contain ordering information and the SC number cannot be uniquely factored. For example, a SC number 38 could mean self-label 5 in a third position ($38 \bmod 5 = 3$) as well as a self-label 7 in the same position ($38 \bmod 7 = 3$). Because of these imperfections, both of the numbers must be used. The general rule is that if a prime divides the parent-label, it can be trusted that the SC number accurately captured its position on the path.

All necessary parent information is encoded in two labels. Prime factors of the parent-label represent self-labels of parent nodes and the SC number encodes their order. Furthermore, the gathered information may be combined together to calculate the parent-label and SC number of any parent node on the path. Then each parent node may be retrieved because the self-label, parent-label and SC number uniquely identify all nodes. This gives rPNL method the advantage of having no costly database requests to determine ancestor information.

Let's consider an example leaf node with parent-label=165 and SC=322. Factoring 165 shows that the prime numbers used to compose that number are 11, 5, and 3. Besides 11, which is the self-label of the node, applying each prime to the SC number reveals their order: $322 \bmod 11 = 3$, $322 \bmod 5 = 2$, and $322 \bmod 3 = 1$. Note that node's depth is encoded in the SC number as well. This information may be used

to calculate the parent-labels as well as SC numbers for the parent node(s): first parent, parent-label=3, SC number=1; second parent, parent-label=15, SC number=7. This example shows how all ancestor information may be calculated rather then retrieved.

The reusable prime number labeling scheme offers a more flexible alternative to PNL scheme. It is deterministic, which means that relationships can be easily identified by scanning all nodes. It is dynamic, as it allows adding new nodes to the hierarchy without major re-labeling. All label changes are computationally light as they rely on simple mathematical functions such as multiplication and division. The rPNL scheme is proven to be more compact as it uses smaller labels by reusing the prime numbers and is, therefore, more capable.

4.1 rPNL Label Relationship

There exists an interesting pattern between rPNL labels. The SC number of the child nodes is congruent to the parent's SC number modulo the parent's parent-label. For example, a node with a self-label=7 and ancestor path of 2.3.5 has a parent-label=30 and SC number=23. The child of this node, with a path 2.3.5.7 would have a parent-label=210 and SC number=53. It is evident that $53 \bmod 30 = 23$, which is also the parent node's SC number.

The proof of this pattern is relatively simple. Assuming X_{child} is the SC number of the child, X_{parent} is the SC number of the parent, and i is the position of the prime, the following must be true for all prime numbers on the parent's path.

$$X_{child} \bmod p_i = i = X_{parent} \bmod p_i$$

$$X_{child} \bmod p_i = X_{parent}$$

Since the last formula holds true for all prime number on the parent's path, it must hold true for their product as well. In other words

$$X_{child} \bmod \prod p_i = X_{parent}$$

This property of rPNL labels may be used for both direct child and descendent node searches.

4.2 PNL and rPNL Capacity Comparison

It is difficult to model the exact label size requirements for the rPNL model as the label size depends on the structure of the hierarchy. The general rule is that the deeper the tree is the more labels are reused. Assuming a two-level hierarchy with only one parent node and all its children at the first level, rPNL label size requirements will be identical to PNL's. The major advantage of rPNL is that it may reuse more labels at the higher levels. In fact, the number of possible reusable labels is a little less than $n!$, where n is the hierarchy depth. Because a different prime is used for a node at each level, there will be $n!$ possible combinations that result in the same product.

The actual number will be slightly less because certain small prime numbers may not be used at the level that is equal to or more than the prime itself. When modeling the two labeling schemes, we used MySQL v5.0.45 database. It can handle integers up to 64 bits long or 1.84×10^{19}, which limits the biggest label possible. This influences the number of self-labels/levels any one branch may have. By definition, a

primorial (*n*#) is the product of all prime numbers less than or equal to *n* (Dubner 1987). The biggest value of the primorial that fits into the allocated number space is $47\# = 6.15 \times 10^{17}$. This means that there are 15 prime numbers that may be used to describe the hierarchy: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, and 47. Assuming that the prime number position must be less than the prime number itself, there are nine prime numbers that are greater than 15. They may be organized in any order. The first six prime numbers are smaller so some positions may be unavailable. Table 4.1 outlines the availability of each of the 15 positions relative to each prime.

Table 4.1: rPNL Self-Label Availability

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Prime Number	2	3	5	7	11	13	17	19	23	29	31	37	41	43	47
Possible Positions	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Unsuitable Positions	14	13	11	9	5	3	0	0	0	0	0	0	0	0	0
Available Positions	1	1	2	3	6	7	9	8	7	6	5	4	3	2	1

This table shows the first 15 prime numbers in order (position and prime number rows). Each prime number may be located in any position, but only once (possible positions row). Smaller prime numbers (2-13) may not be used at a position that is greater than the prime number itself (unsuitable positions row). As a result, there is a fixed amount of possible positions each prime number may assume. In fact, there are $1 \times 1 \times 2 \times 3 \times 6 \times 7 \times 9! = 91{,}445{,}760$ total possible combinations of the first 15 prime numbers. The difference between PNL and rPNL schemes is that the discussed path is the only 15-level path PNL scheme may have. It also must be the first path in the tree assuming depth-first approach. Reusable prime number labeling scheme may have

over 91 million of such 15-level paths due to different combinations of the prime numbers. Figure 4.2 shows PNL label size requirements for a symmetric tree with fan-out and depth between 1 and 15 nodes. Figure 4.3 shows rPNL label size requirements for the same tree.

Figure 4.2: PNL Parent-Label Size Growth

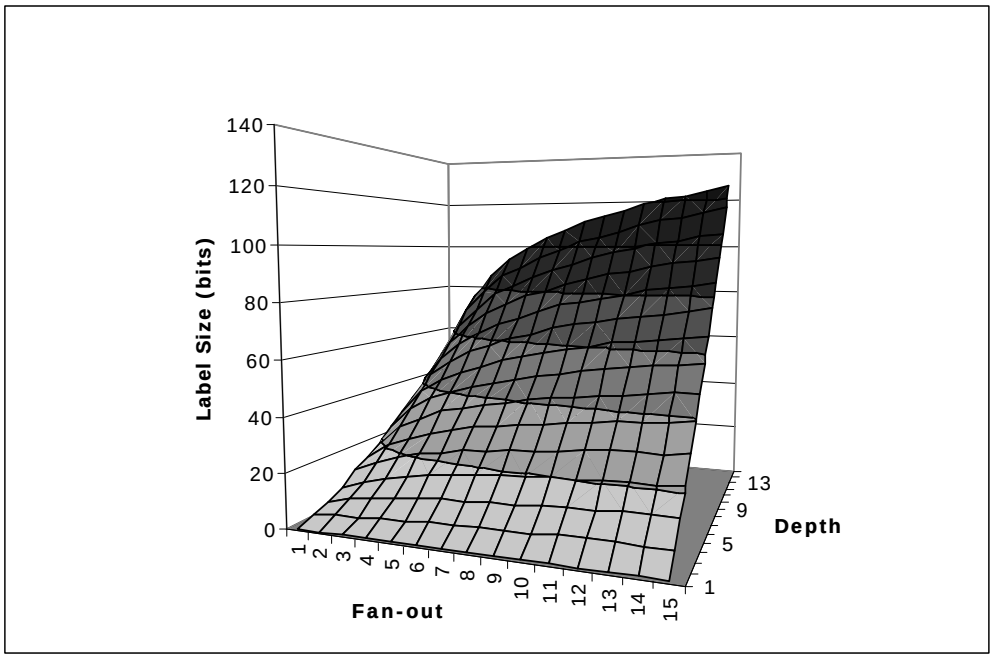
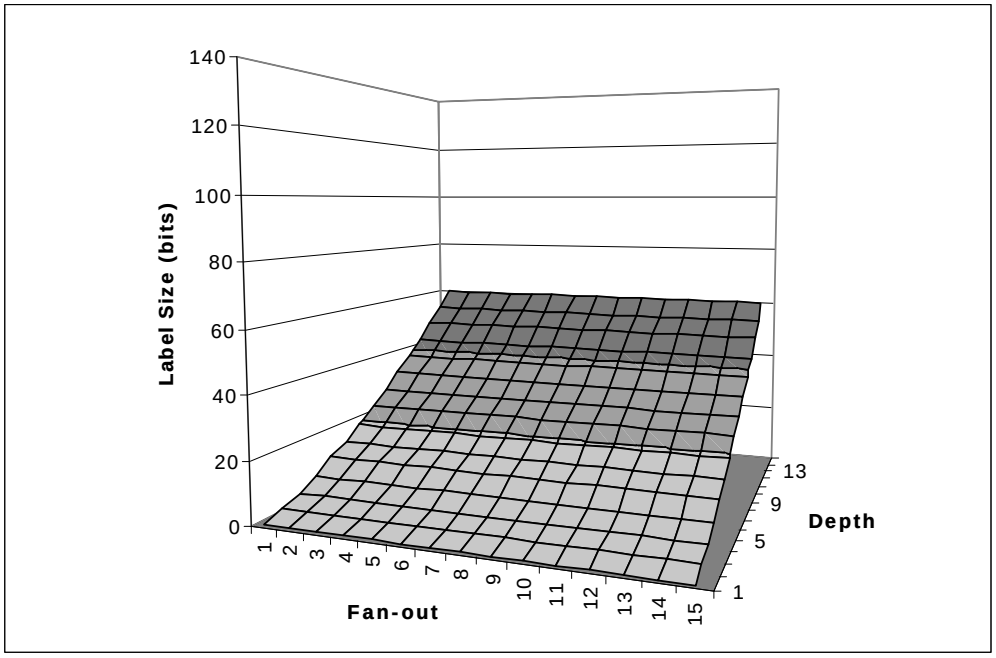


Figure 4.3: rPNL Parent-Label Size Growth



The label size is measured in the minimum number of bits required to represent it. Assuming breadth first approach, the scheme proposed by Wu, Lee, and Hsu uses every 15th prime at the beginning of each level as the rest of the prime numbers are used by other nodes on that level. The proposed scheme allows prime number reuse, so consecutive prime numbers will be used at the beginning of each level. The label at the beginning of each level is guaranteed to be the smallest one on that level, which makes the two graphs a best-case scenario for both labeling schemes. Reusable prime number labeling scheme produces much smaller parent-labels, which requires roughly half the space needed by PNL graph. The advantage of the rPNL scheme is that the label size requirement grows very slowly. If the fan-out was larger than 15, the difference would be even more dramatic.

4.3 Update Flexibility

There is a way to shift a branch up or down by updating the parent-label (division or multiplication by a prime) and SC number (addition or subtraction of the SC number components). Below are the detailed explanations of how it can be done. Table 4.2 shows the two nodes before and after their branch has been updated. $N = \prod p_i$ is the product of all participating prime numbers p. N_i is the product of all prime numbers except p_i. R_i is the reciprocal of the prime number p_i in ith position. X_i is the simultaneous congruence component for each prime number and $X = \sum X_i$ is the simultaneous congruence number.

Table 4.2: rPNL Hierarchy Update

Node Before Update				**Node After Update**			
	N_i	R_i	X_i		N_i	R_i	X_i
$X = 1 \bmod 2$	105	1	$1 \times 105 \times 1 = 105$	$X = 1 \bmod 2$	21	1	$1 \times 21 \times 1 = 21$
$X = 2 \bmod 3$	70	1	$2 \times 70 \times 1 = 140$	$X = 2 \bmod 3$	14	-1	$2 \times 14 \times (-1) = -28$
$X = 3 \bmod 5$	42	-2	$3 \times 42 \times (-2) = -252$				
$X = 4 \bmod 7$	30	-3	$4 \times 30 \times (-3) = -360$	$X = 3 \bmod 7$	6	-1	$3 \times 6 \times (-1) = -18$
$X = 105 + 140 - 252 - 360 \bmod 210 = 53$				$X = 21 - 28 - 18 \bmod 42 = 17$			

When shifting the branch down, a new parent-label is easily calculated by dividing the old parent-label by the self-label of the node being removed. For the example above the new parent-label would be 210/5=42. To generate a new SC number, we need to remove the X_3 of the respective prime and reduce any of the following SC number components.

An interesting property of all reciprocals in a SC number is that their sum is always $1 \bmod N$, where N is the product of all prime numbers. This can be proven by simply noting that, by definition, $N_i * R_i = 1 \bmod p_i$ and $N_k * R_k = 0 \bmod p_i$ where $i \neq k$ for all prime numbers. Therefore $\sum N_i R_i = 1 \bmod N$. This formula may be decomposed into two parts, the stable part (before the node being deleted) and remainder part (after the node being deleted). The stable part varies from node to node as prime numbers and reciprocals change. However, there is a way to find a reciprocal for each p_i.

According to the Euler's theorem $a^{\varphi(n)} = 1 \bmod n$ if and only if $\gcd(a, n) = 1$ (Guderson 1943). Thus, there is a way to find a reciprocal for a prime regardless of the parent-label or other prime numbers used. In other words, $N_i^{\varphi(p_i)} = 1 \bmod p_i$ if and only if $\gcd(N_i, p_i) = 1$ which is always true because, by definition, N_i is the product of all prime numbers other than p_i. Also, according to the definition of the Euler's totient function, it returns the number of positive integers less than or equal to n such that each one is relatively prime to n. If n is prime, there will be $n-1$ of such integers (all numbers between 1 and the prime itself). As a result, we get $N_i^{p_i - 1} = 1 \bmod p_i$. The SC number component related to p_3 may be calculated as

$$X_3 = 3 * (210/5) * (210/5)^{5-1} \bmod 210 = 126$$

Then the remainder of the SC number components will need to be modified by

$$\sum_{i=3}^{4} N_i R_i = 1 - (210/2)^1 - (210/3)^2 \bmod 210 = 36$$

As a result the new SC number will be $(53 - 126 - 36) \bmod 42 = 17$.

In order to shift a branch down, the same actions will have to be performed in reverse. New parent-label would be 42*5=210. The SC number component related to p_3 may be calculated as

$$X_3 = 3 * (42) * (42)^{5-1} \bmod 42 * 5 = 126$$

Then the remainder part of the SC number components will need to be modified by

$$\sum_{i=3}^{4} N_i R_i = 1 - (42 * 5/2)^1 - (42 * 5/3)^2 - 42^4 \bmod 42 * 5 = 120$$

The new SC number will be $(17 + 120 + 126) \bmod 42 * 5 = 53$

A major issue with this kind of update is that the new prime must be unique along the entire path to the nodes to be updated as well among the sibling nodes. There is no way to tell which prime numbers are available for this update so a globally unique prime should be used. Note that both kinds of updates could be performed globally, regardless of the affected nodes location, based strictly on the information available i.e. self-label of the parent node being added/removed and previous ancestor information.

Chapter 5

Evaluation and Analysis

This section presents the experimental results of the benchmark comparison between the original PNL scheme, the proposed the rPNL scheme and three other representative schemes. All three hierarchical models are represented: Edge scheme from the Recursive Expansion model, Range from the Nested Sets model, Path, PNL and rPNL from the Materialized Path model.

5.1 Benchmark Setup

In order to conduct performance comparison between PNL scheme and the proposed method we needed a highly structured hierarchy with considerable depth and fan-out. The CNN website matches this description because it has multiple nested tables, which adds to the depth of the tree. It is also saturated with links, which adds to the fan-out. Additionally, because it is a web page, it has a highly unbalanced structure i.e. body tag contains the majority of the data whereas head element has only a few children. The original document did not validate as XHTML 1.0 so certain tags were modified to improve readability and facilitate parsing process. Figure 5.1 shows the XHTML tree used in the comparison (Aharef 2007). Yahoo website is represented by

very similar but simpler hierarchy. It is a much smaller tree, with fewer generations but similar fanout. Figure 5.2 shows the XHTML tree used in the comparison (Aharef 2007). Also, in order to test the fan-out effects we needed a relatively shallow hierarchy with very large fan-out. We used an XML encoded play "Hamlet" by William Shakespeare (Bosak 1999). This specific document was often used by other authors in their benchmark experiments. Figure 5.3 shows the XML tree used in the comparison (Aharef 2007). A much simpler document, yet still with considerable fan-out is "The Comedy of Errors" by William Shakespeare (Bosak 1999). Figure 5.7 shows the XML tree used in the comparison (Aharef 2007). Like in any other XML compatible documents, each tag may have multiple children but at most one parent. If there are inconsistencies with this rule in the graphs, it is purely for spacing purposes. Table 5.1 outlines the important hierarchical properties of each tree.

Table 5.1: Test Tree Properties

Tree	Max Depth	Max Fan-Out	Nodes	Paths
CNN	14	20	840	444
Hamlet	6	174	6,636	1,205
Yahoo	10	21	473	248
Comedy	4	98	959	162

Such distinct tree compositions have been chosen purposefully to determine if there is a relationship between the schemas' performance and the structure of the tree being modeled. Note that a wide range of depths and fan-outs is represented in order to thoroughly test the performance of each labeling scheme.

Figure 5.1: CNN Tree Visualization (Aharef 2007)

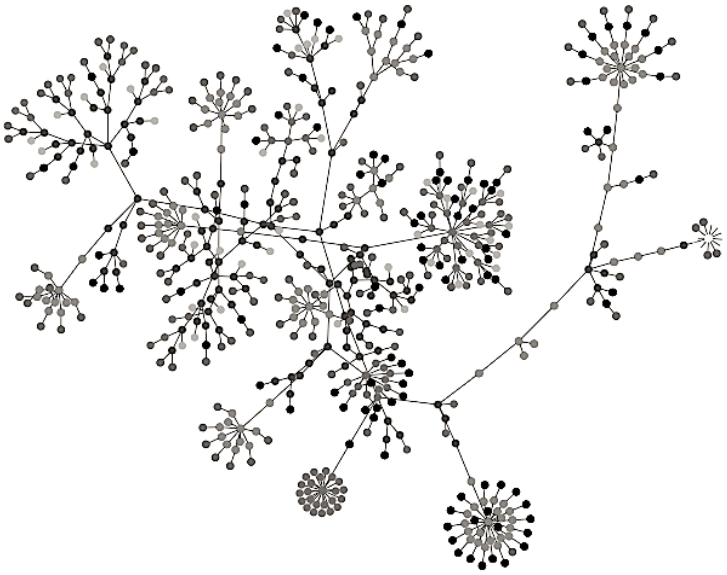


Figure 5.2: Yahoo Tree Visualization (Aharef 2007)

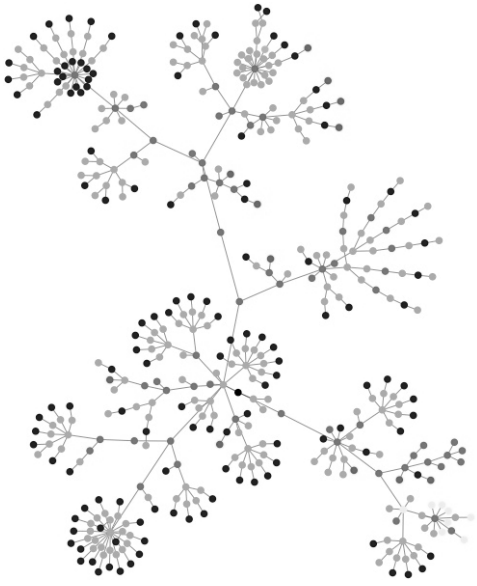


Figure 5.3: Hamlet Tree Visualization (Aharef 2007)

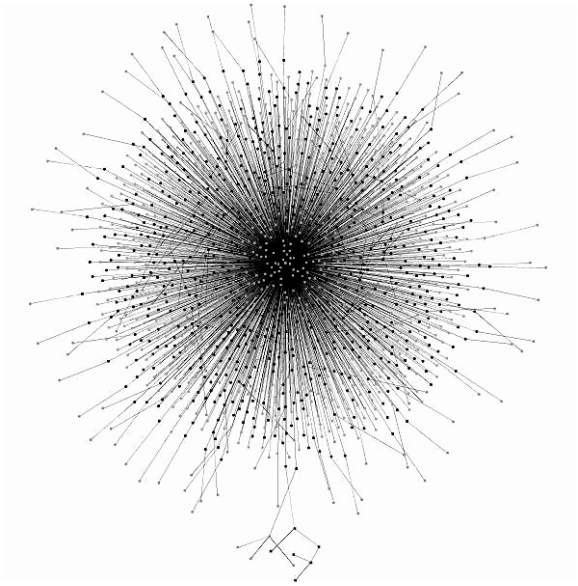
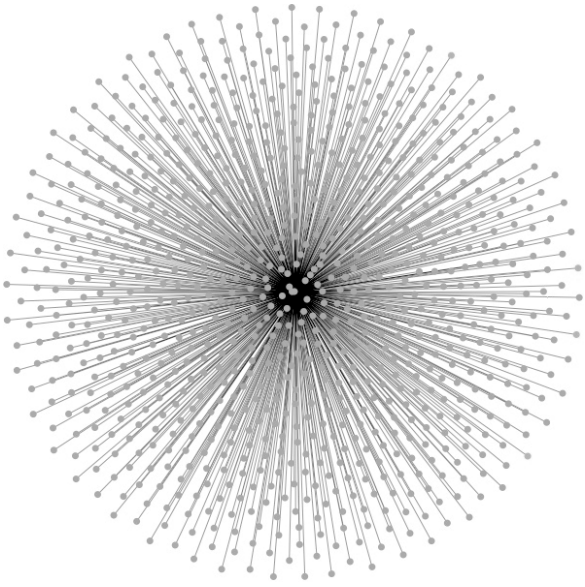


Figure 5.4: Comedy Tree Visualization (Aharef 2007)



Hardware resources play a very important role in each model's performance. Some of the experiments were run on a slower hardware with a considerable performance drop off. The experimental system configuration is outlined in Table 5.2.

Table 5.2: Test System Configuration

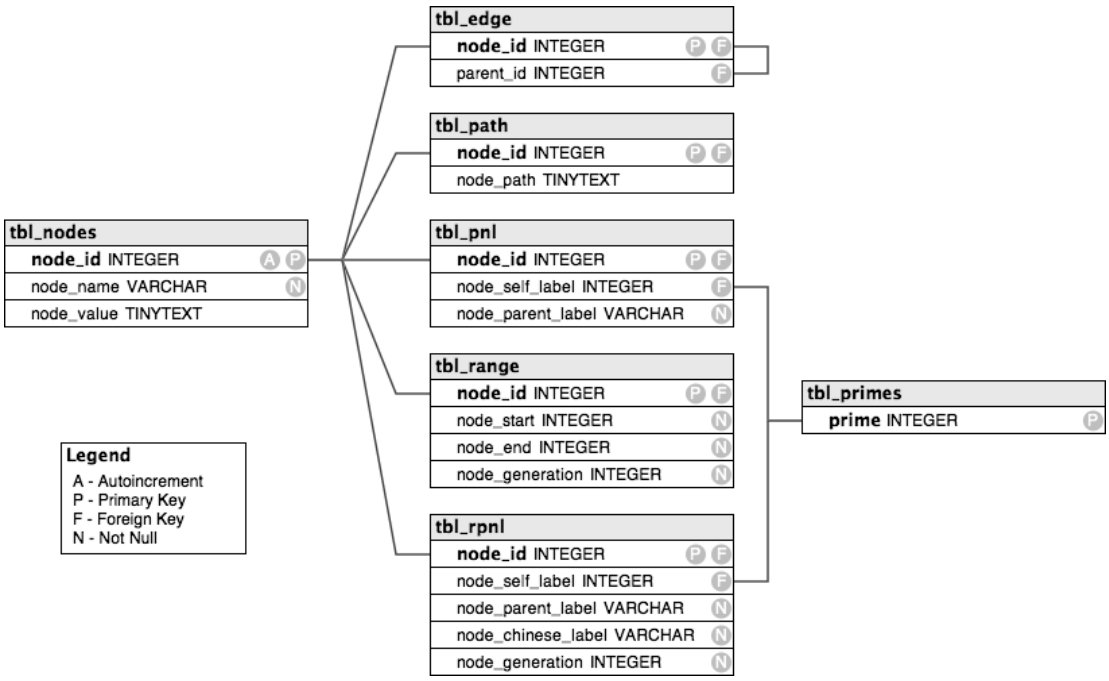
CPU	Intel Pentium D 2.66GHz
RAM	2GB
OS	MS Windows XP Professional
Database	MySQL v5.0.45
Code	PHP v.5.2.3 DOM/XML enabled

Additionally, network connection speed may have a great impact on the performance on certain schemes. If a scheme relies on extensive computation on the client side that results in very few queries being sent to the database server then network delay effects are minimal. Conversely the Edge approach relies on recursive queries being sent to the server. If the network connection is slow, this model's performance will suffer. Note that the code in all experiments accesses the database through a direct socket connection, so no network delay is recorded.

The database schema composition may also influence the performance of labeling schemes. Figure 5.5 shows the database structure used. Every labeling scheme table refers to the *tbl_nodes* table to ensure consistency among multiple tree representations. We did not implement any indexing or other optimization techniques in order to test the true performance of each labeling scheme. The large numeric labels in the PNL and rPNL models were stored as strings and converted to floating point numbers on demand, as specified by the IEEE 754 standard, in order to

maintain the necessary precision. Otherwise the least relevant bit would be truncated, which is catastrophic for number based labeling schemes.

Figure 5.5: Relational Database Setup



5.2 Tree Labeling

After modeling each tree, we compared the PNL and rPNL label sizes. Table 5.3 shows the PNL and rPNL maximum label sizes for all experiments as well as their prime number usage.

Table 5.3: PNL and rPNL Maximum Label Sizes

	Max PNL Label	Max rPNL Label	PNL Primes	rPNL Primes
CNN	6.1×10^{41} (139 bits)	9.91×10^{10} (37 bits)	840	28
Hamlet	2.5×10^{19} (65 bits)	3.38×10^{6} (22 bits)	6,636	178
Yahoo	3.11×10^{28} (95 bits)	8.21×10^{11} (40 bits)	473	30
Comedy	5.54×10^{7} (26 bits)	5.41×10^{3} (13 bits)	959	101

In addition to a much smaller parent-label that is much easier to factor, the rPNL scheme used only 28 prime numbers to label the entire CNN tree. For the Yahoo tree, the rPNL labeling scheme was forced to use 30 different prime numbers to label this small tree because it has fewer paths and thus less opportunity to reuse existing self-labels. However, it is better than the original prime number labeling scheme that has reserved 473 unique primes for this tree. Note that for every tree, except the smallest Yahoo tree, the PNL scheme produced very large labels that are very difficult to work with. The rPNL scheme consistently produced small labels, which easily fit into the 64 bits available to the default modulo function, can be factorized, divided and otherwise manipulated with a reasonable latency.

We also compared the tree representation sizes of each labeling scheme. Table 5.4 shows the comparison between the size of the original documents and each scheme representation. The XML documents are the most compact of all representations, which makes them an excellent format for transportation. However, they can not be easily searched or analyzed and that is what labeling schemes are especially good at.

Table 5.4: Tree Model Size Comparison

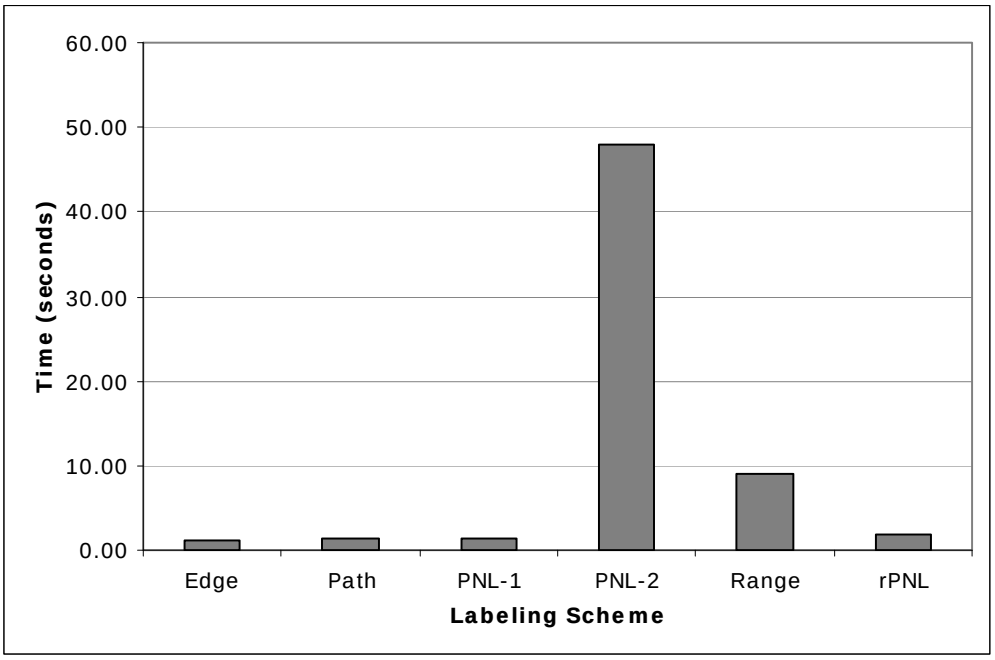
	XML	Edge	Path	PNL	Range	rPNL
CNN	41,550	82,376	98,760	115,144	82,376	98,760
Hamlet	286,022	637,836	703,372	752,524	686,988	719,756
Yahoo	12,288	16,384	49,152	49,152	16,384	49,152
Comedy	49,152	114,688	147,456	163,840	131,072	163,840

The size, measured in bytes, is approximated from the database tables that were created by each scheme. Note that many schemes generate representations of the

same size, especially in small trees e.g. Yahoo and CNN. This is because in small trees the labels do not grow large enough to exceed the space allocated to each label type by default. However, in large trees e.g. Comedy and Hamlet, each scheme's label size growth effects are more evident. Overall, the Edge approach produced the most compact representation, followed by the Range labeling scheme. Reusable prime number scheme was equal or better than PNL method regardless of the kind of hierarchy being modeled. The rPNL method should be more compact than the Path approach in especially deep tress because numeric multiplication increases label size slower than string concatenation and because the proposed scheme has more opportunities to reuse existing labels.

We also recorded the time it took each scheme to model a tree. Figure 5.6 shows that Range labeling scheme was the slowest of all models to record CNN tree of only 840 nodes. The reason Range scheme was so slow is because a lot of relabeling had to occur when a new node was inserted deep in the hierarchy. The Edge and Path schemes recorded the tree very quickly since new label generation did not require any computation or database requests. The Reusable prime number labeling scheme was a little behind Edge and Path models because it had to request a new label from the database before each new node was saved. Note that, depending on the implementation, PNL scheme took 48 seconds, if the next available prime must be requested each time (PNL-1), or 1.28 seconds, if consecutive prime numbers are guaranteed to be unique with each new node being added (PNL-2).

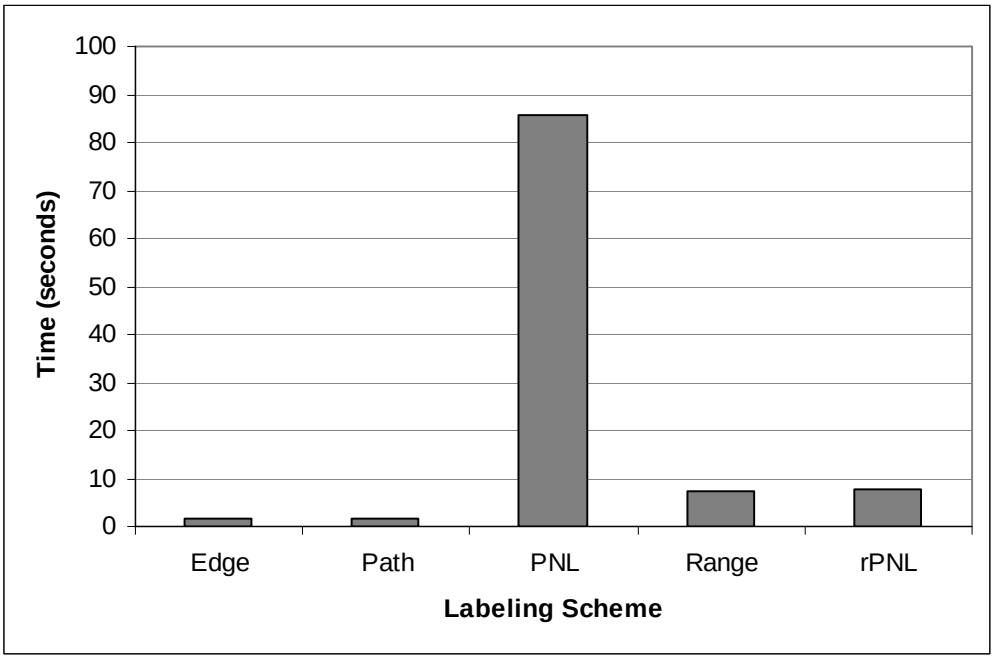
Figure 5.6: CNN Tree Labeling Time



This experiment assumed that a new prime label must be requested from the database to make sure it is unique relative to the schema's constraints. This is usually the case when there are multiple sources that populate the hierarchy or if it is built up at different points of time. If, however, the entire tree is available from the start, the overhead of requesting the next available prime may be eliminated completely. In fact, our experiments have shown that PNL scheme may record the same hierarchies 40 times faster if it uses consecutive prime numbers from a pre-sorted source. PNL scheme is a valid scheme that can successfully record hierarchical data. However, it performs much worse than the rPNL scheme in a very likely scenario when the entire tree to be modeled is not readily available.

The same experiment on the shallow hierarchies produced similar results. Figure 5.7 shows the time it took to model the Comedy tree for each of the five schemes.

Figure 5.7: Comedy Tree Labeling Time



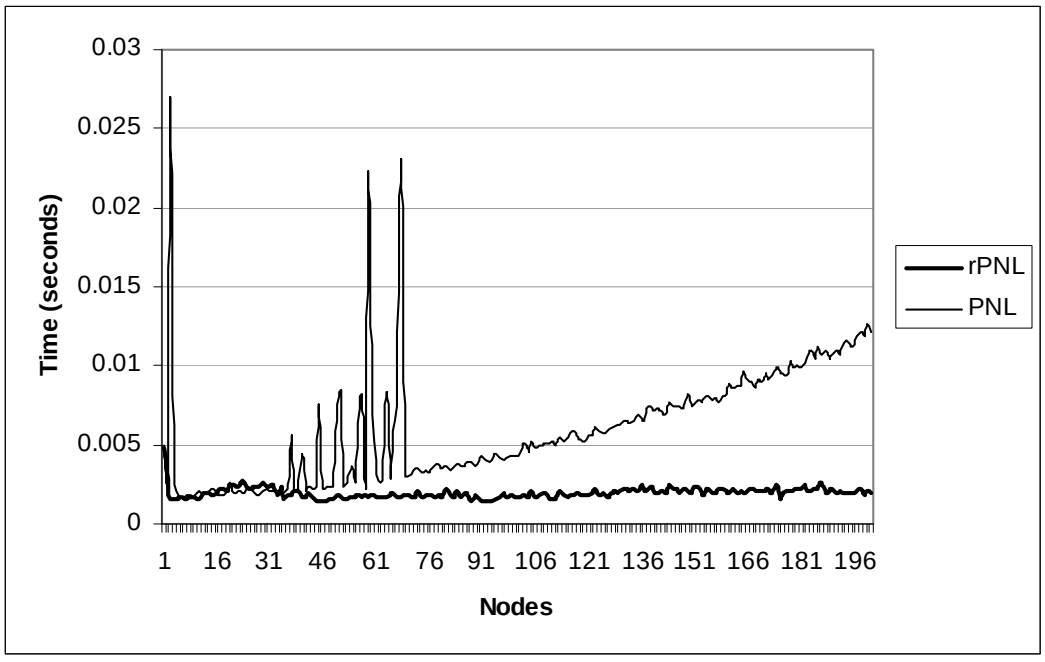
Again, the Edge and Path schemes had consistently good results as their label generation algorithms are not influenced by the structure of the hierarchy. Range labeling scheme has much better results recording this shallow tree. Assuming a depth first approach, very little relabeling was needed. As a result, Range labeling scheme was faster than the proposed reusable prime number labeling scheme. The original prime number labeling scheme was the slowest in all experiments. The reason PNL scheme took so long is not because it is time consuming to generate a label, but because it takes long to ensure this label is globally unique. Figure 5.6 showed that if PNL scheme is guaranteed to have reliable labels, its performance

could be greatly increased. The proposed scheme, on the other hand, produced good results without making this assumption for all hierarchies being modeled.

5.3 Node Labeling

We also compared the time it took to record each individual node in a hierarchy. Both PNL and rPNL have strict rules about the uniqueness of the labels used. For PNL scheme global uniqueness is required, which results in increasing delays as the tree gets bigger. For the rPNL scheme only local uniqueness is needed. As a result, very few records are referenced with every new node, which increases performance. Figure 5.8 shows the time it took to record each node in CNN tree.

Figure 5.8: CNN Tree Labeling with PNL and rPNL



The two labeling schemes are very close at the beginning because only a few node labels must be checked for uniqueness. However, as the tree expands, the

number of labels increases and PNL scheme must check them all to ensure global uniqueness. Reusable prime number labeling scheme, however, is only concerned with relative/local uniqueness. The labels being checked with this scheme include only the nodes on the path to the root, and sibling nodes. As a result, the delay is relatively constant. The proposed scheme would not outperform PNL method in a very shallow hierarchy with large fan-out. In fact, in this case the performance would be almost identical because relative uniqueness would also be global.

This claim is supported by the experimental results shown on Figure 5.9. Both schemes have nearly identical results at the beginning of the Comedy tree experiment when the number of labels being checked is the same. The gradual increases and drops in the rPNL scheme graph clearly show different paths being recorded. The increase in the delay for both schemes is proportional to the number of labels that must be checked. For rPNL scheme it is proportional to the fanout of a current generation on a given path. For PNL it is proportional to the total number of already labeled nodes. As more nodes are recorded, the PNL scheme falls behind rPNL. In fact, the rPNL delay gets reset with every new path, whereas PNL delay grows continuously.

Figure 5.9: Comedy Tree Labeling with PNL and rPNL

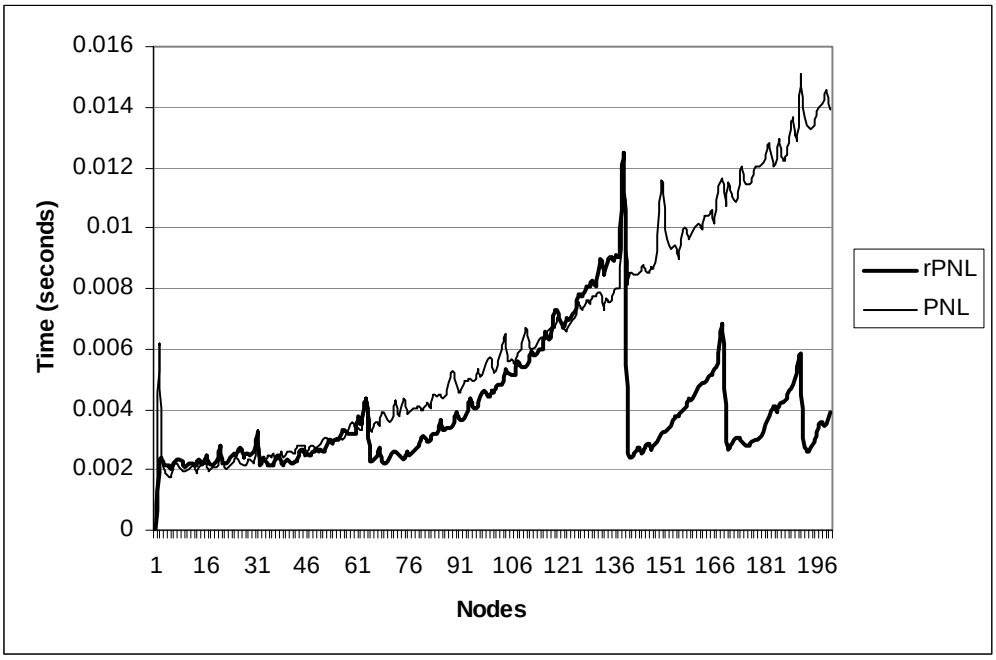


Figure 5.10 and Figure 5.11 show a similar comparison between rPNL, Edge and Path schemes for CNN and Comedy trees. The Edge and Path labeling schemes are not influenced by the structure of the hierarchy. Therefore each node insertion should take the same amount of time and result in a straight-line graph. The small discrepancy in the Edge and Path graphs is due to hardware resource availability, hard disk access times, etc. During the CNN tree experiment, the proposed scheme differs from the Edge approach only slightly whenever the tree's depth increases rapidly. For example, the depth of the CNN tree depth goes from 4 to 13 and back to 4 generations between the 10^{th} and 40^{th} nodes. In fact, the average difference between the two schemes across the entire experiment was only 0.0007 seconds.

Figure 5.10: CNN Tree Labeling with rPNL and Edge

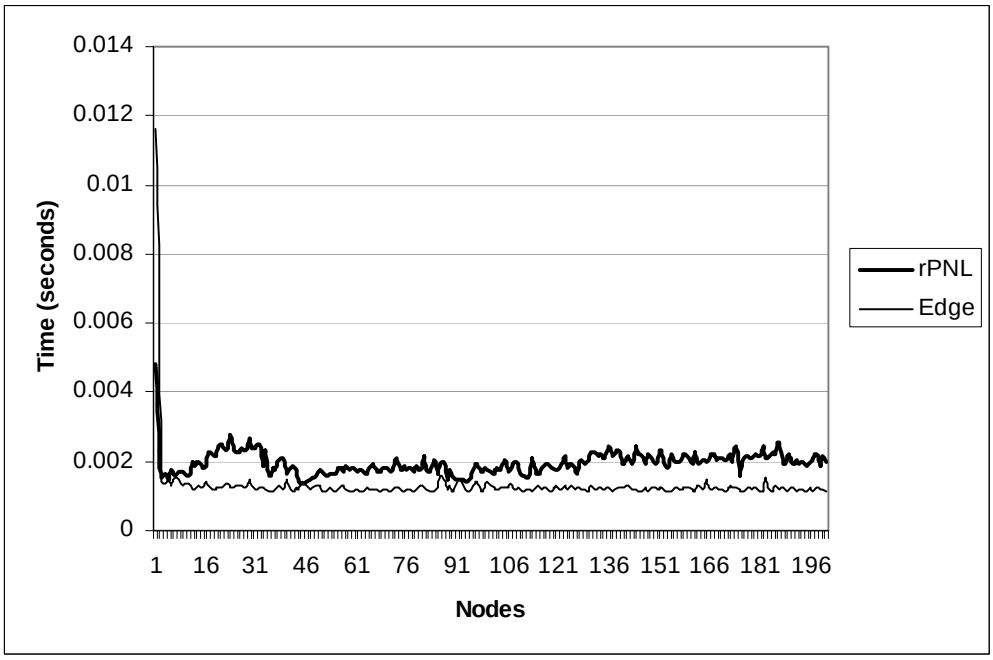
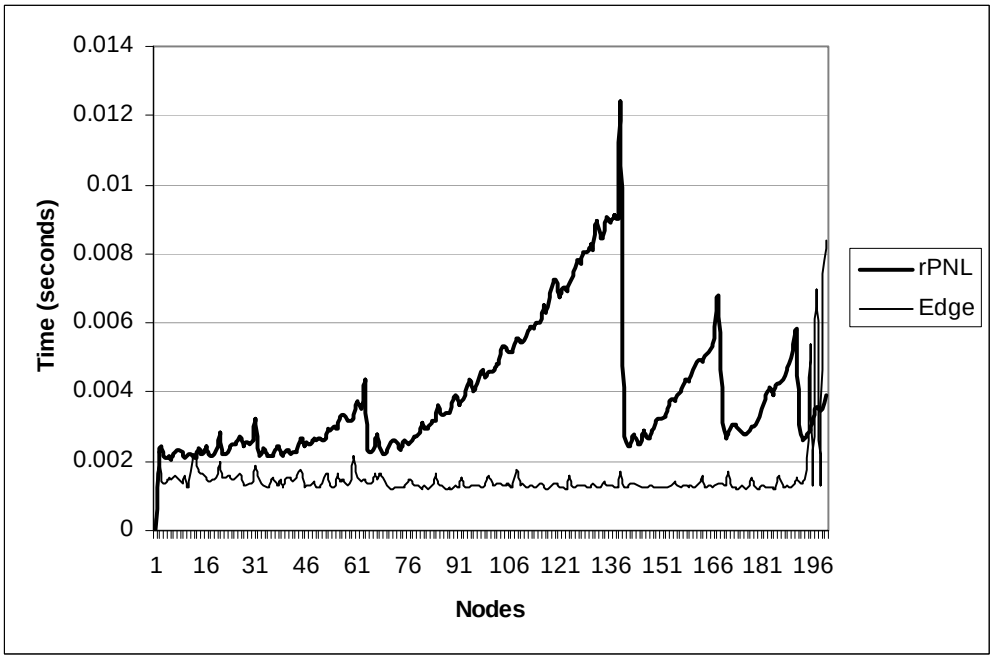


Figure 5.11: Comedy Tree Labeling with rPNL and Edge



Likewise the proposed scheme falls behind the Path approach whenever it has to deal with a large number of siblings. For instance, Comedy tree nodes 70 to 140 are siblings. Unlike the Range approach, the reusable prime number labeling scheme maintains its performance across the entire tree without the need to re-label parts of it as the new nodes get added.. A shallow Hamlet hierarchy does not have as many relabeling incidents as the CNN tree experiment. However, they are much more extreme because many more nodes must be referenced to determine if they need to be relabeled.

Figure 5.12 and Figure 5.13 show that the proposed scheme does get affected by the structure of the tree, but not as much as the Range model. The spikes in the Range labeling scheme occur mostly when there is a number of sibling nodes to be inserted deep inside the tree hierarchy (CNN nodes 18-24, 68-74, 137-141, and 177-192). If the boundaries of the parent container are no longer large enough, the entire branch must be re-labeled and ancestor boundaries expanded. Additionally, any of the ancestor siblings' containers must be shifted in order to prevent boundary overlapping. This kind of re-labeling is very resource intensive and that is why the peaks are so extreme. However, if enough adjustment has been provided, no further updates should be necessary. This is why the peaks are very narrow. A shallow Hamlet hierarchy does not have as many relabeling incidents as the CNN tree experiment. However, they are much more extreme because many more nodes must be referenced to determine if they need to be relabeled.

Figure 5.12: CNN Tree Labeling with rPNL and Range

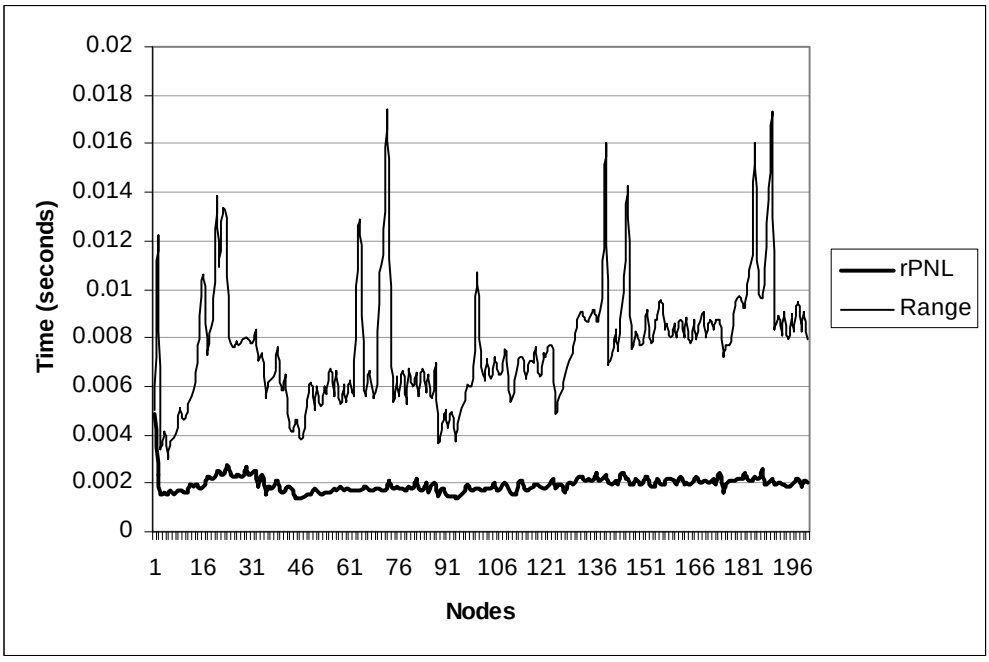
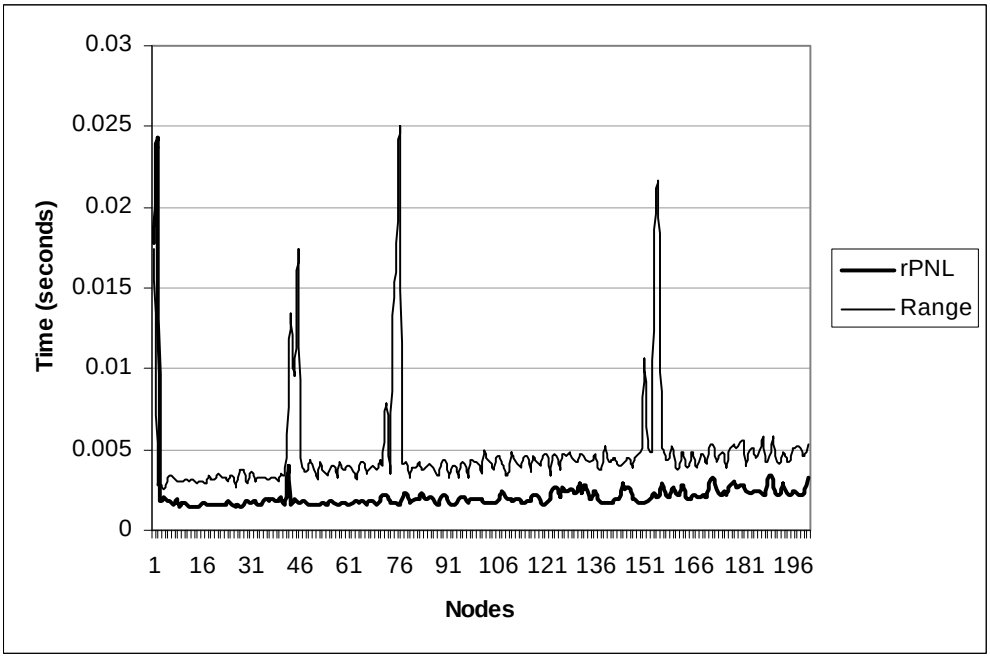


Figure 5.13: Hamlet Tree Labeling with rPNL and Range



5.4 Direct Children Lookup

The goal of this experiment was to traverse the entire tree from top to bottom one level at a time. For each node in a tree only its direct children were located. The results are displayed in Figure 5.14, Figure 5.15, and Figure 5.16.

Figure 5.14: CNN Tree Top-Down Tree Traversal

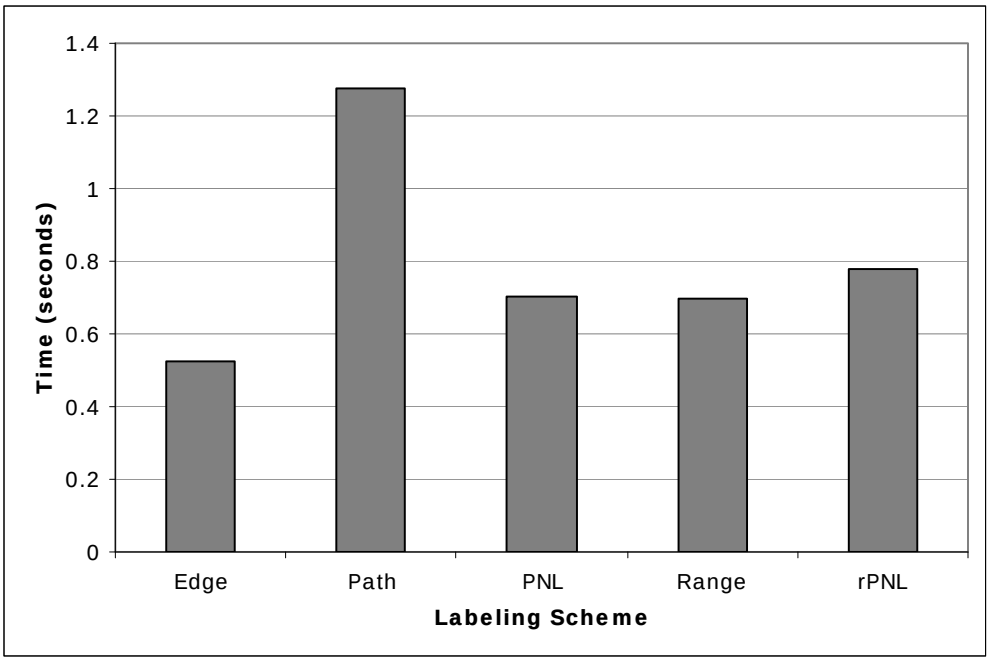


Figure 5.15: Yahoo Tree Top-Down Tree Traversal

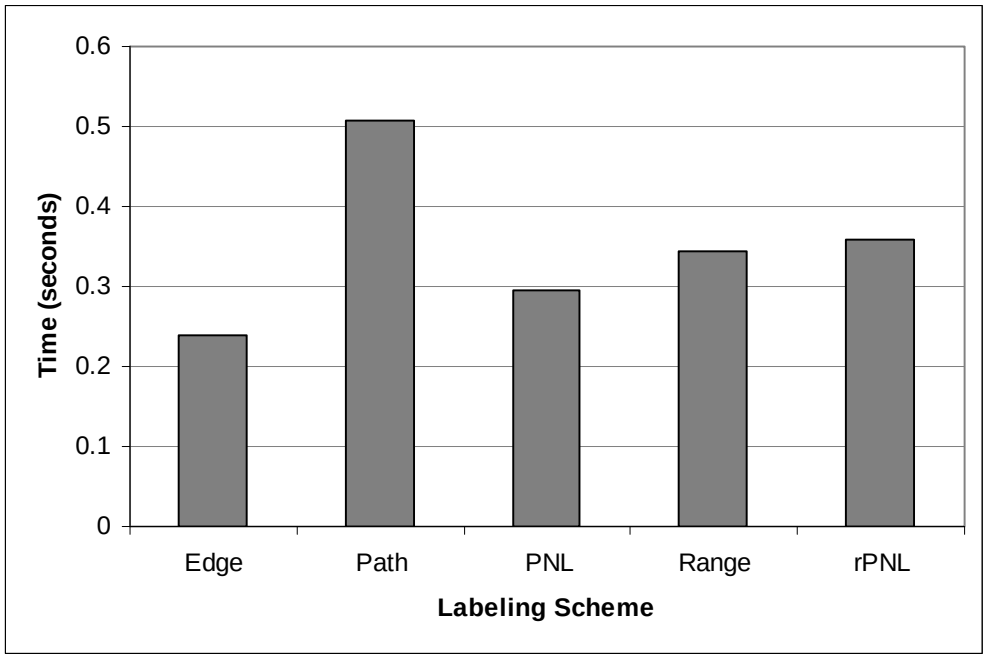
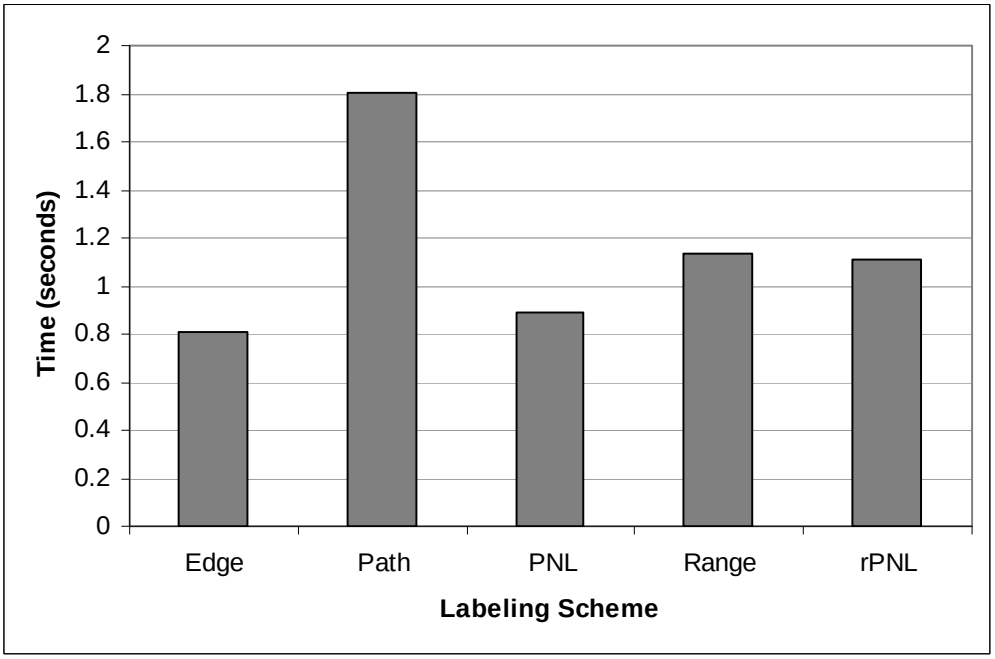


Figure 5.16: Comedy Tree Top-Down Tree Traversal



Since the experiment focused on searching for nodes only one generation down, the Edge model performed best as it involved fast number lookups. The PNL scheme also used numerical lookups but it took some time to generate the necessary labels. The Range approach performed well, because it best suited for descendent searches with a small correction for depth in this case. The Reusable prime number labeling scheme was a little behind PNL. Unlike PNL, rPNL parent-labels are not globally unique, so fast number lookups were not possible. Instead a descendent search, similar to Range scheme, was performed. The proposed scheme did far better than Path approach because regular expression matching is much more complex and resource intensive than modulo function.

5.5 Descendent Search

Descendent search is the most common requirement of the hierarchical models. It is a very frequent practice to search only a specific branch or sub-tree of the hierarchy. In fact, the rPNL scheme was developed for this specific purpose in mind. Figure 5.17 shows the results of the CNN descendent search experiment. The nodes being searched are located on different depths between 5 and 11 generations. There are a total of 71 paths being tested with 135 matching records. As expected from section 2.2 discussion, Range descendent search produced the best results because the only function needed to identify the resulting nodes was a simple number comparison. The Path scheme was also able to successfully locate all of the matching nodes; however, regular expression matches fall behind a lightweight modulo function. The Reusable prime number labeling scheme was able to locate the same nodes in considerably less

time. Both prime number based models had similar results on smaller depths and smaller hierarchies, as demonstrated on Figure 5.18 and Figure 5.19. However, the PNL scheme was not able to successfully complete the CNN tree experiment as it failed to compute the relationships among nodes with such large labels. Edge scheme, used recursive queries to traverse the entire sub-tree, analyzing each node individually to determine if it matches the criteria. As a result, it took 0.09471 seconds to complete. This is almost forty times as slow as the rPNL scheme.

Figure 5.17: CNN Tree Descendent Search

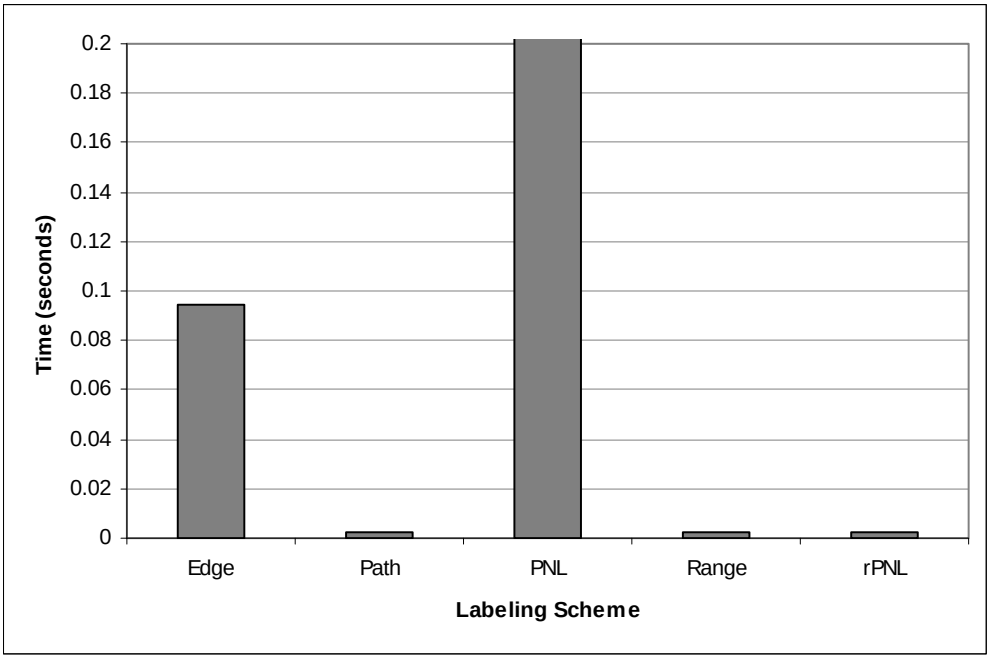


Figure 5.18: Yahoo Tree Descendent Search

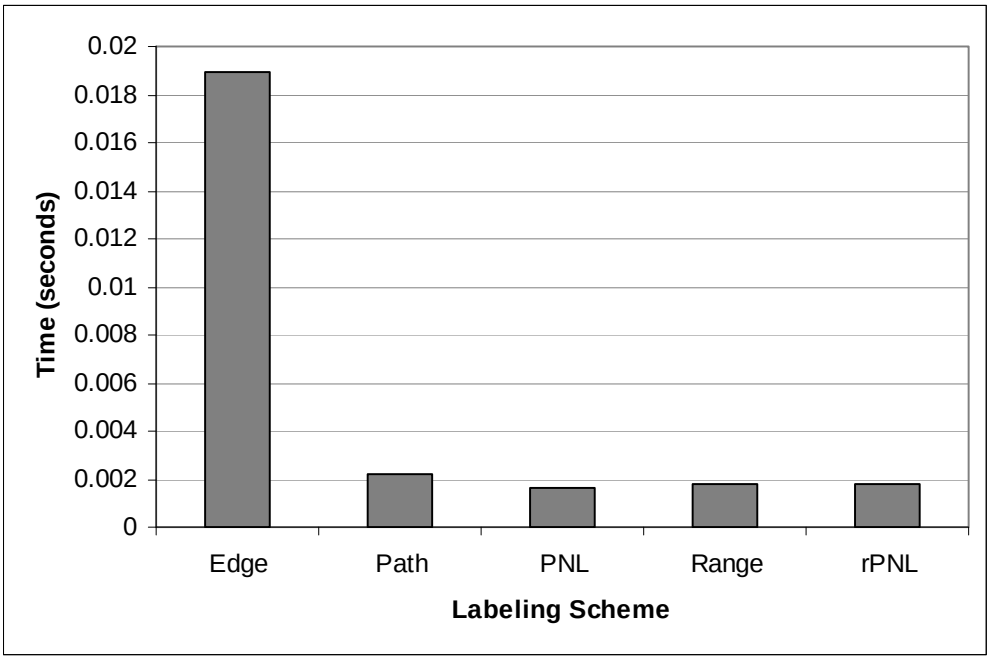
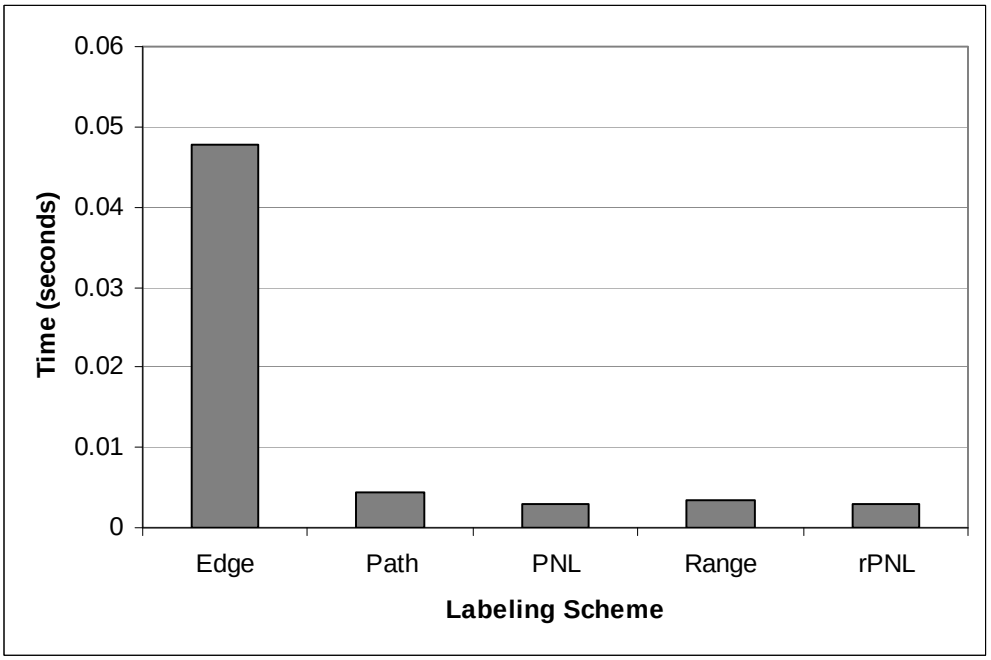


Figure 5.19: Comedy Tree Descendent Search



5.6 Ancestor Determination

Another very common functionality of hierarchal labeling schemes is ancestor determination. It is used to establish a path from the current node to the root of the hierarchy. This experiment measured the performance of this functionality among different schemes. The results of the experiment are shown in Figure 5.20, Figure 5.21, Figure 5.22 and Figure 5.23. When the experiment was performed on CNN tree, the Range model was able to locate all ancestors the fastest. This contradicts with the prediction made by Tropashko in section 2.2. This is due to the fact that the tree was relatively small, with limited fan-out and considerable depth.

Figure 5.20: CNN Tree Ancestor Determination

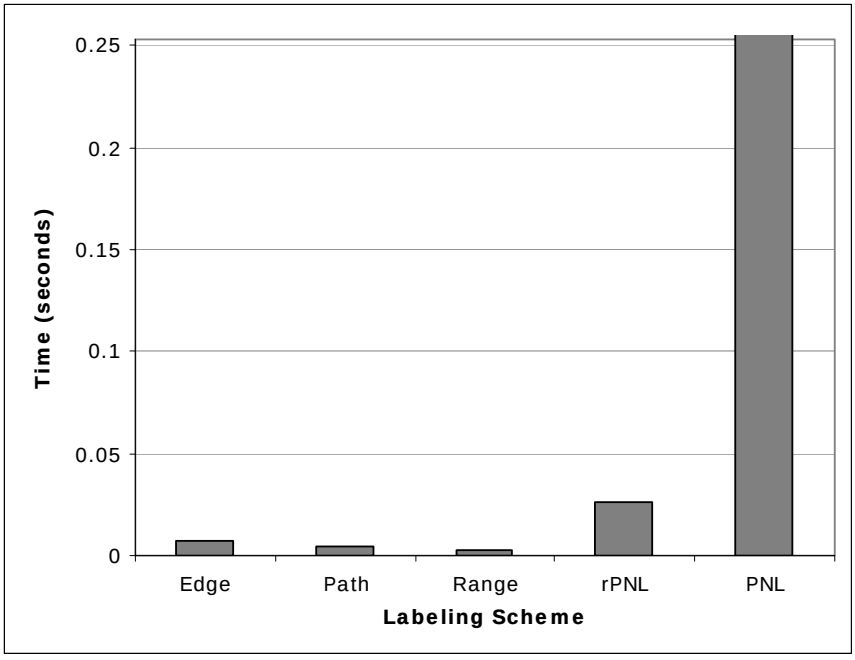


Figure 5.21: Hamlet Tree Ancestor Determination

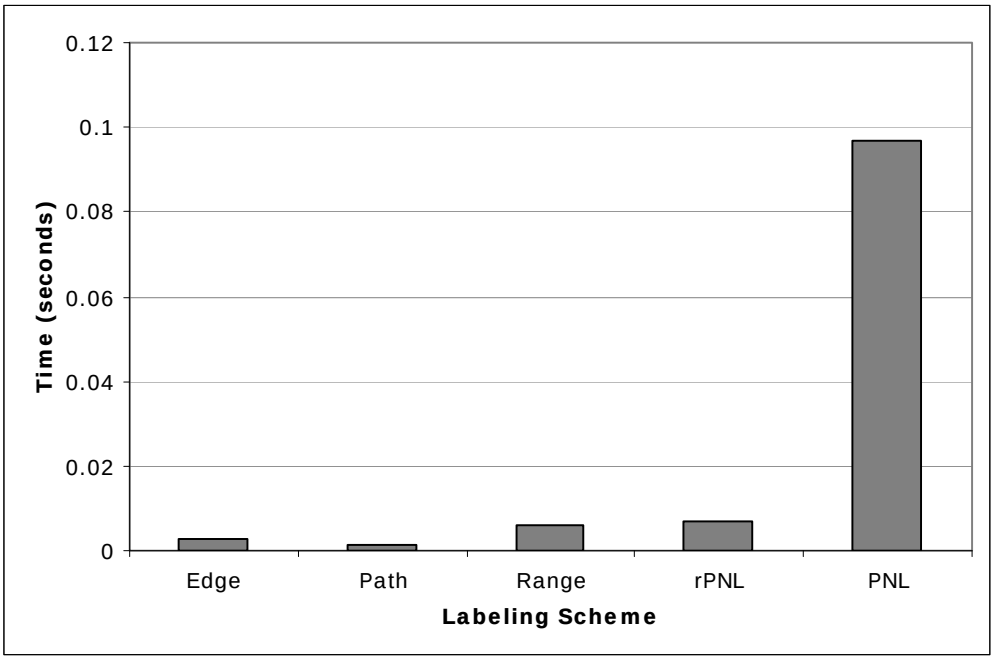


Figure 5.22: Yahoo Tree Ancestor Determination

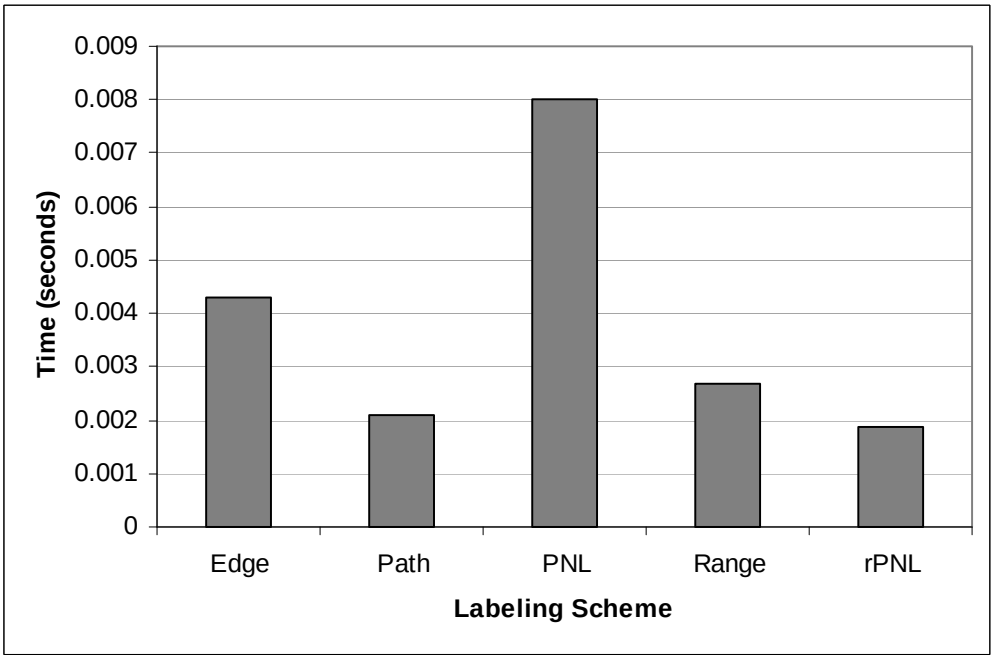
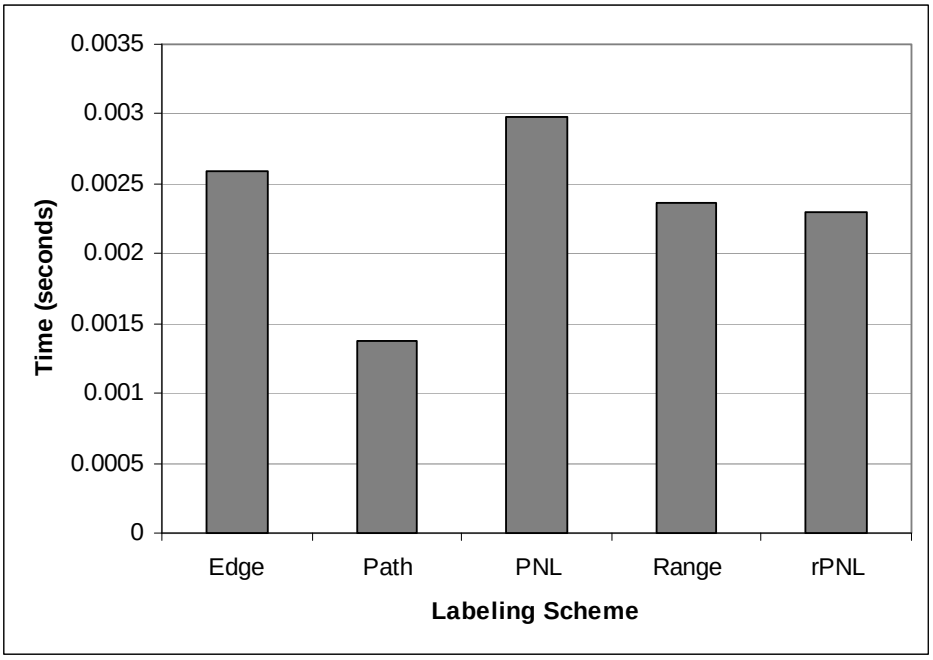


Figure 5.23: Comedy Tree Ancestor Determination



When the same experiment was performed on Hamlet tree (Figure 5.21), the Range model performance did in fact suffer due to an increase in overall tree size and fan-out. A similar phenomenon occurred during the experiments on the Comedy tree (Figure 5.23). In fact, Range scheme ancestor determination in a shallow hierarchy is more difficult than in a deep hierarchy.

The Path scheme did not involve any database requests regarding the parent-labels. In fact, all necessary information was retrieved from the node's path label. This is the reason this model showed consistent results regardless of the hierarchy being modeled. The Edge scheme involved a lot of recursive queries each one performing a simple numeric label lookup. This is the reason Edge scheme was always slower than Path model. The proposed reusable prime number labeling

scheme had consistently good results across all experiments. In fact, it was the fastest of the five schemes during the Yahoo experiment (Figure 5.22).

The rPNL algorithm which is reverse of the one described in section 4.1 could have been used for this experiment. However, due to the rounding error in MySQL software a different approach has been implemented. With this approach, each parent-label is factorized and the appropriate parent SC number is generated. Then each parent node is located. This method is obviously much slower as it involves factorization and multiple database requests. However, the factorization of smaller rPNL labels is much faster than factorization of PNL labels. The native rPNL ancestor determination algorithm involves a single query submitted to the database with no factorization or SC number calculations. If the rounding error could be eliminated, the rPNL scheme performance would be improved even more.

The results of this experiment have shown that both prime number schemes were considerably slower than the rest, especially in larger hierarchies. Their performance was proportional to the depth of the tree because both scheme implementations relied on prime factorization functionality. For instance, in the CNN tree experiment the PNL scheme was 604 times slower than Edge model and in Hamlet tree experiment it was 16 times slower than Range scheme. However, even with a slower algorithm implementation the rPNL scheme was on average 150 times faster than PNL scheme. The overall performance of prime number scheme may be greatly improved by introducing a more efficient prime factorization algorithm like the ones presented by Connelly Barnes (Barnes 2004). However, because both

models rely on the same prime factorization function, their relative performance ratio will remain constant.

5.7 Update Flexibility

Tree updates are one of the most complex operations available. They usually require a number of nodes to be re-labeled in order to reflect ancestor removal/addition and branch movement. The Edge labeling scheme is best suited for this purpose because only a single node needs to be updated to in order to execute any kind of tree update. The Range model is also quite flexible in this regard. It will require multiple unrelated nodes to be relabeled in order to accommodate the changes. However, each kind of update is relatively simple and could be performed globally. Additionally, this scheme design assumes frequent updates. Therefore, if a good scheme implementation is available, the tree can be successfully updated. Materialized path models such as Path and PNL may also be updated. For Path labeling scheme each individual update is not very complex. Even though string operations are much more resource intensive than math ones, they are guaranteed to work at all depths of the tree. PNL scheme can accommodate various types of updates as well. However, due to extremely large labels that are common in deep hierarchies, some of the updates might not be able to propagate through the entire sub-tree. The reusable prime number scheme is especially difficult to update. It relies on relative label uniqueness so branch movement is especially complicated. Vertical branch movement algorithm has been described in section 4.3. Unfortunately, there is no simple way to move an entire sub-tree within a hierarchy because there is no reliable way to ensure that the

prime numbers used in the destination branch are not also used in the sub-tree being moved. The only way to accomplish this would be through individual node re-labeling.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

The purpose of hierarchical data modeling is a quick determination of relationships among the nodes in a tree. In order to do that efficiently, a labeling scheme must be in place that supports fast, computationally light queries. We have proposed a new prime number labeling scheme that utilizes the unique characteristics of prime numbers to encode the node position in a hierarchy. The rPNL scheme allows labels to be reused throughout the tree while still being unique at the sibling level and along the leaf-root path. This keeps the label size minimal, which in turn dramatically improves performance. The reusable prime number labeling scheme allows capturing larger hierarchies by encoding the order of the prime numbers with a simultaneous congruence number. Section 4.2 discussed the mathematical reasoning behind label size growth in both schemes and section 5.2 provided experimental evidence of the rPNL scheme producing a much more compact tree representation comparable to the path model. As shown in section 5.5, rPNL scheme is capable of searching deep hierarchies better than PNL. The proposed scheme also showed a significant

improvement in performance relative to the original approach as demonstrated in sections 5.5 and 5.6. Table 6.1 and Table 6.2 outline the results of the experiments performed. Overall, the rPNL scheme is not the best performer. However, it is better than PNL scheme for deep and shallow hierarchies.

Table 6.1: Scheme Comparison Summary for Deep Trees

	Edge	Path	PNL	Range	rPNL
Model Size	1	4	5	2	3
Tree Labeling	1	2	5	4	3
Direct Children Lookup	1	5	3	2	4
Descendent Search	5	3	4	1	2
Ancestor Determination	3	1	5	2	4
Update Flexibility	1	3	4	2	5

Table 6.2: Scheme Comparison Summary for Shallow Trees

	Edge	Path	PNL	Range	rPNL
Model Size	1	3	5	2	4
Tree Labeling	1	2	5	3	4
Direct Children Lookup	1	5	2	3	4
Descendent Search	5	4	3	1	2
Ancestor Determination	2	1	5	3	4
Update Flexibility	1	2	4	3	5

6.2 Summary of Contributions

This thesis presented a more capable labeling scheme that improves upon PNL model. We described the reusable prime number labeling scheme in detail, outlined its governing rules, and made conclusions regarding its expected capacity and overall size. Additionally, we discussed the update limitations that our scheme has and proposed an algorithm for vertical branch movement that does not require individual node label re-generation. A number of experiments, comparing the reusable prime

number scheme to others representative methods, have been carried out. The results of each experiment were analyzed and explained. We were able to demonstrate that the proposed scheme is capable of accurately modeling hierarchies of high depth and high fan-out within the available space. The proposed labeling scheme includes label recycling functionality which is not natively supported by any other labeling scheme. This research has shown that the rPNL labeling scheme is a valid method for encoding hierarchical information into a relational database.

6.3 Future Work

This research focused on labeling scheme comparison with no optimizations of any kind. This allowed true model performance to be isolated. In the future we would like to research various model optimizations applicable to the rPNL scheme. Some of the optimizations have been outlined by Wu, Lee, and Hsu in the context of PNL scheme. One of the major disadvantages of both prime number labeling schemes is that all searches must go through the entire tree. A further research into tree partitioning and caching would reduce the issue of full table scans. Additionally, some of the issues with both schemes were hardware limitations of the testing systems. Currently all computations were done within the 32-bit computer architecture. Further research could focus on scheme performance in 64-bit environments, on other operating systems, more efficient programming languages, and other relational databases like Oracle and MS SQL Server.

Bibliography

Abiteboul, S., Kaplan, H., and Milo, T. (2001), "Compact Labeling Schemes for Ancestor Queries", In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, Society for Industrial and Applied Mathematics, pp. 547-556, Washington, DC.

Aharef, S. (2007), *Websites as Graphs - An HTML DOM Visualizer Applet*, Java Applet, http://www.aharef.info/static/htmlgraph/

Amagasa, T., Yoshikawa, M., and Uemura, S. (2003), "QRS: A Robust Numbering Scheme for XML Documents", In *Proceedings of the 19th International Conference on Data Engineering*, pp. 705-707.

Atkinson, M., DeWitt, D., Maier, D., Bancilhon, F., Dittrich, K., and Zdonik, S. (1992), "The Object-Oriented Database System Manifesto", *Building an Object-Oriented Database System: The Story of 0_2,* Morgan Kaufmann Publishers.

Barnes, C. (2004), *Integer Factorization Algorithms*, Technical Report, Department of Physics, Oregon State University.

Böhme, T. and Rahm, E. (2004), "Supporting Efficient Streaming and Insertion of XML Data in RDBMS", In *Proceedings of the 3rd DIWeb Workshop,* CAiSE, pp. 70-81, Riga, Latvia.

Bosak, J. (1999), *The Plays of Shakespeare in XML*, http://www.oasis-open.org/cover/bosakShakespeare200.html

Brandon, D. (2005), "Recursive Database Structures", *Journal of Computing Sciences in Colleges*, vol. 21, no. 5, pp. 295-304.

Buneman, P., Davidson, S., Hillebrand, G., and Suciu, D. (1996), "A Query Language and Optimization Techniques for Unstructured Data", In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pp. 505-516, Montreal, Quebec.

Celko, J. (2004), *Joe Celko's SQL for Smarties: Trees and Hierarchies*, Morgan Kaufmann Publishers.

Christophides, V., Abiteboul, S., Cluet, S., and Scholl, M. (1994), "From Structured Documents to Novel Query Facilities", In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pp. 313-324, Minneapolis, MN.

Cohen, E., Kaplan, H., and Milo, T. (2002), "Labeling Dynamic XML Trees", In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 271-281, Madison, WI.

David, M. (2003), "ANSI SQL Hierarchical Processing Can Fully Integrate Native XML", In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, vol. 32, no. 1, pp. 41-46.

Deutsch, A., Fernandez, M., and Suciu, D. (1999), "Storing Semistructured Data With STORED", In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pp. 431-442, Philadelphia, PA.

Deux, O. et al. (1990), "The Story of O_2", *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 1, pp. 91-108.

Dietz, P. (1982), "Maintaining Order in a Linked List", In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pp.122-127, San Francisco, CA.

Dubner, H. (1987), "Factorial and Primorial Primes", *Journal of Recreational Mathematics*, vol. 19, pp. 197–203.

Duong, M. and Zhang, Y. (2005), "LSDX: A New Labeling Scheme for Dynamically Updating XML Data", In *Proceedings of the 16th Australasian Database Conference - Volume 39*, vol. 103, pp. 185-193, Newcastle, Australia.

Elçi, A. and Rahnama, B. (2006), "XMLEase: A Novel Access and Space-Efficiency Model for Maintaining XML Data in Relational Databases", In *Proceedings of the International Conference on Semantic Web and Web Services*, pp. 186-192, Las Vegas, NV.

Fernandez, M., Florescu, D., Levy, A., and Suciu, D. (1997), "A Query Language for a Web-Site Management System", *ACM SIGMOD Record*, vol. 26, no. 3, pp. 4-11.

Fernández, M., Florescu, D., Kang, J., Levy, A., and Suciu, D. (1998), "Catching the Boat With Strudel: Experiences With a Web-Site Management System", In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pp. 414-425, Seattle, WA.

Florescu, D. and Kossmann, D. (1999a), "Storing and Querying XML Data Using an RDBMS", *IEEE Data Engineering Bulletin*, vol. 22, no. 3, pp. 27-34.

Florescu, D. and Kossmann, D. (1999b), *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*, Research Report no. 3680, INRIA, Rocquencourt, France.

Guderson, N. (1943), "Some Theorems of Euler's Function", *Bulletin of the American Mathematical Society*, vol. 49, pp. 278-280.

Härder, T., Haustein, M., Mathis, C., and Wagner, M. (2007), "Node Labeling Schemes for Dynamic XML Documents Reconsidered", *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 126-149.

Haigh, T. (2006), "A Veritable Bucket of Facts: Origins of the Data Base Management System", *ACM SIGMOD Record*, vol. 35, no. 2, pp. 33-49.

Howard, F. (2002), "A Generalized Chinese Remainder Theorem", *The College Mathematics Journal*, vol. 33, no. 4, pp. 279-282.

IEEE Standards Committee 754, (1985), *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Standard 754-1985, Institute of Electrical and Electronics Engineers, vol. 22 no. 2, pp. 9–25.

Jiang, H., Lu, H., Wang, W. and Yu, J. (2002a), "Path Materialization Revisited: An Efficient Storage Model for XML Data", In *Proceedings of the Thirteenth Australasian Conference on Database Technologies*, Australian Computer Society, Inc., pp. 85–94, Melbourne, Victoria, Australia.

Jiang, H., Lu, H., Wang, W. and Yu, J. (2002b), "Xparent: An Efficient RDBMS-Based XML Database System", In *Proceedings of the 18th International Conference on Data Engineering*, IEEE Computer Society, p.335, Washington, DC.

Khaing, A. and Thein, N. (2006), "A Persistent Labeling Scheme for Dynamic Ordered XML Trees", In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web intelligence*, IEEE Computer Society, pp. 498-501.

Leonard, J. (2006), *Strategies for Encoding XML Documents in Relational Databases: Comparisons and Contrasts*, Master Thesis, East Tennessee State University.

Li, C. and Ling, T. (2005a), "QED: A Novel Quaternary Encoding to Completely Avoid Re-Labeling in XML Updates", In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, pp. 501-508, Bremen, Germany.

Li, C. and Ling, T. (2005b), "An Improved Prefix Labeling Scheme: A Binary String Approach for Dynamic Ordered XML", In *Proceedings of the 10th International Conference on Database Systems for Advanced Applications*, pp. 125-137.

Li, C., Ling, T., and Hu, M. (2006), "Reuse or Never Reuse the Deleted Labels in XML Query Processing Based on Labeling Schemes", In *Proceedings of the 11th International Conference on Database Systems for Advanced Applications*, DASFAA, pp. 659-673, Singapore.

Li, C., Ling, T. W., Lu, J., and Yu, T. (2005), "On Reducing Redundancy and Improving Efficiency of XML Labeling Schemes", In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management* pp. 225-226, Bremen, Germany

Li, Q. and Moon, B. (2001), "Indexing and Querying XML Data for Regular Path Expressions", In *Proceedings of the 27th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers, pp. 361-370, San Francisco, CA.

Lindemann, F. (1933), "The Unique Factorization of a Positive Integer", *Quarterly Journal of Mathematics*, vol. 4, pp. 319-320.

McHugh, J., Abiteboul, S., Goldman, R., Quass, D., and Widom, J. (1997), "Lore: A Database Management System for Semistructured Data, *ACM SIGMOD Record*, vol. 26, no. 3, pp. 54-66.

Mignet, L., Barbosa, D., and Veltri, P. (2003), "The XML Web: A First Study", In *Proceedings of 12th International World Wide Web Conference*, ACM Press, pp. 500-510, Budapest, Hungary.

O'Neil, P. and O'Neil, E. (2004), "ORDPATHs: Insert-Friendly XML Node Labels", In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1-6, Paris, France.

Papakonstantinou, Y., Abiteboul, S., and Garcia-Molina, H. (1996), "Object Fusion in Mediator Systems", In *Proceedings of the 22nd International Conference on Very Large Data*, Morgan Kaufmann Publishers, pp. 413-424, San Francisco, CA.

Preuveneers, D. and Berbers, Y. (2006), *Prime Numbers Considered Useful: Ontology Encoding for Efficient Subsumption Testing*, Technical Report, Department of Computer Science, K.U.Leuven, Leuven, Belgium.

Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., and Widom, J. (1995), "Querying Semistructured Heterogeneous Information", In *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases*, Lecture Notes In Computer Science, vol. 1013, pp. 319-344, Springer-Verlag, London.

Schmidt, A., Kersten, M. L., Windhouwer, M., and Waas, F. (2001), "Efficient Relational Storage and Retrieval of XML Documents", In *Selected Papers From the Third International Workshop WebDB 2000 on the World Wide Web and Databases*, Lecture Notes In Computer Science, vol. 1997, pp. 137-150, Springer-Verlag, London.

Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D., and Naughton, J. (1999), "Relational Databases for Querying XML Documents: Limitations and Opportunities", In *Proceedings of the 25th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers, pp. 302-314, San Francisco, CA.

Shoens, K., Luniewski, A., Schwarz, P., Stamos, J., and Thomas, J. (1993), "The Rufus System: Information Organization for Semi-Structured Data", In *Proceedings of the 19th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers, pp. 97-107, San Francisco, CA.

Shui, W., Lam, F., Fisher, D., and Wong, R. (2005), "Querying and Maintaining Ordered XML Data Using Relational Databases", In *Proceedings of the 16th Australasian Database Conference - Volume 39*, ACM International Conference Proceeding Series, vol. 103, pp. 85-94, Newcastle, Australia.

Tatarinov, I., Viglas, S., Beyer, K., Shanmugasundaram, J., Shekita, E., and Zhang, C. (2002), "Storing and Querying Ordered XML Using a Relational Database System", In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pp. 204-215, Madison, WI.

Tropashko, V. (2005), "Nested Intervals Tree Encoding in SQL", In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, vol. 34, no. 2, pp. 47-52.

Weigel, F., Schulz, K., and Meuss, H. (2005), "The BIRD Numbering Scheme for XML and Tree Databases -- Deciding and Reconstructing Tree Relations Using Efficient Arithmetic Operations", In *Proceedings of the 3rd International XML Database Symposium*, pp. 49-67, Trondheim, Norway.

Wu, X., Lee, M., and Hsu, W. (2004), "A Prime Number Labeling Scheme for Dynamic Ordered XML Trees", In *Proceedings of the 20th International Conference on Data Engineering*, p. 66, Washington, DC.

Wunderlich, M (1983), "Recent Advances in the Design and Implementation of Large Integer Factorization Algorithms", In *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, pp. 67-71, Washington, DC.

Yoshikawa, M. and Amagasa, T. (2001), "Xrel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases", In *ACM Transactions on Internet Technology*, vol. 1, no. 1, pp. 110-141.