Insertion algorithms for network model database management systems

Abdurashid Mamadolimov¹, Khikmat Saburov²

¹,²Mathematical Modelling Laboratory, MIMOS, Technology Park Malaysia, Kuala Lumpur, Malaysia

¹rashid.mdolimov@mimos.my, ²khikmat.saburov@mimos.my

Abstract. The network model is a database model conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, forms partial order. When a database is large and a query comparison is expensive then the efficiency requirement of managing algorithms is minimizing the number of query comparisons. We consider updating operation for network model database management systems. We develop a new sequential algorithm for updating operation. Also we suggest a distributed version of the algorithm.

1. Introduction

A database management system (DBMS) is a software package with computer programs that controls the creation, maintenance, and use of a database. One of the components of DBMS is a Data Manipulation Subsystem. Data manipulation subsystem helps the user to add, change, and delete information in a database and query it for valuable information. Software tools within the data manipulation subsystem are most often the primary interface between user and the information contained in a database. It allows the user to specify its logical information requirements. DBMSs are designed to use one of five database models to provide simplistic access to information stored in databases. The most popular five database models are the followings:

(i) hierarchical model,
(ii) network model,
(iii) relational model,
(iv) multidimensional model, and
(v) object model.

The network model is a database model conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, forms partial order, is not restricted to being a tree or lattice (Figure 1). When a database is large and a query comparison is expensive then the efficiency requirement of managing algorithms is minimizing the number of query comparisons i.e., the efficiency is measured by query complexity of an algorithm. For example, conceptual graphs form a network model structure and query comparison will be a graph homomorphism checking which a very expensive operation is [1]-[9].

In this paper we consider updating operation for network model database management systems. We develop a new sequential algorithm for updating operation. Also we suggest a distributed version of the algorithm.

One of the important components of updating operation is a searching. In [10], Mozes, Onak, and Weimann present a linear-time algorithm that finds the optimal strategy for searching a tree-like partially ordered set. Dereniowski [11] proves that finding an optimal search strategy for general posets is hard and gives a polynomial-time approximation algorithm with sub-logarithmic approximation ratio. In our early works [12], [13] we proposed new sequential and distributed algorithms for searching operation. Another component of updating operation is a finding parents. For
finding parents operation Levinson [14] and Ellis [15] have proposed several modifications of breadth/depth first search technique using some simple properties of partially ordered sets. In this paper we used height of elements of poset for a good ordering of a dataset.

2. Partially ordered set (poset)

Definition. A partial order is a binary relation "≤" over a set P which is reflexive, antisymmetric, and transitive, that is, for all a, b and c in P, we have that:

(i) \( a \leq a \) (reflexivity);
(ii) if \( a \leq b \) and \( b \leq a \) then \( a = b \) (antisymmetry);
(iii) if \( a \leq b \) and \( b \leq c \) then \( a \leq c \) (transitivity).

A set \( P \) with a partial order "≤" is called a partially ordered set or poset \((P, \leq)\). For any two elements \( a \) and \( b \) of a poset \((P, \leq)\) we use \( a \geq b \) if \( b \leq a \). Also we use \( a < b \) \((a > b)\) if \( a \leq b \) \((a \geq b)\) and \( a \neq b \). Let \((P, \leq)\) be a poset. If \( T \in P \) such that \( \forall x \in P, x \leq T \), then \( T \) is called the top element of \( P \) (the bottom element \( \bot \) is dually defined); a poset does not necessarily have a top (bottom) element, but if it has a top (bottom) element then it is unique. Any finite poset can be represented by directed acyclic graph and this representation is called Hasse diagram of a poset (Figure 1). Network model databases have a partially ordered data structure, i.e., data and relations between them form a poset.

Let \((P, \leq)\) be a poset and \( D \) be any non-empty, finite subset of \( P: D \subseteq P, |D| = n \), where \( n \) is a positive integer. We note that \((D, \leq)\) is also a poset. We all times assume that the subset \( D \) has both top and bottom elements and that they are different: \( T \neq \bot \). We call the set \( D \) as a dataset. Let \( x \) be an element of \( P: x \in P \). We call the element \( x \) a query element. The following definitions are useful for future description.

\( \text{Ancestors}(x, D) = \{a \in D: a > x\} \);
\( \text{Descendants}(x, D) = \{d \in D: d < x\} \);
\( \text{Parents}(x, D) = \{p \in \text{Ancestors}(x, D): \exists a \in \text{Ancestors}(x, D), a < p\} \);
\( \text{Children}(x, D) = \{c \in \text{Descendants}(x, D): \exists d \in \text{Descendants}(x, D), d > c\} \);
\( \text{Indeg}(x, D) = |\text{Parents}(x, D)| \);
\( \text{Outdeg}(x, D) = |\text{Children}(x, D)| \).

We use \( \text{Ancestor}(x, D), \text{Descendant}(x, D), \text{Parent}(x, D) \) and \( \text{Child}(x, D) \) for an element of \( \text{Ancestors}(x, D), \text{Descendants}(x, D), \text{Parents}(x, D) \) and \( \text{Children}(x, D) \) respectively.

Figure 1. Example of a graph representation of a poset
Definition. A chain from an element \( a \) to an element \( b \) \((\neq a)\) in a poset \( D \) is a sequence of different elements \( a = x_0, x_1, \ldots, x_l = b \) of a poset \( D \) such that each of its elements, except last one, is a parent for the next element in the sequence; \( l \) is called a length of the chain; A chain from element \( a \) to element \( b \) with maximum length is called maximum chain from \( a \) to \( b \).

Definition. A length of maximum chain from top element \( \top \) to an element \( x \) is called height of \( x \) in a poset \( D \) and denoted as \( \text{Height}(x, D) \). Height of bottom element is called height of poset i.e., \( \text{Height}(D) = \text{Height}(\bot, D) \).

We do not provide here the proofs of the Propositions 1-3.

Proposition 1. For any two elements \( x \) and \( y \) of a poset \( D \), if \( x > y \) then \( \text{Height}(x, D) < \text{Height}(y, D) \).

Proposition 2. For any element \( x \) of a poset \( D \),
\[
\text{Height}(x, D) = \max_{y \in \text{Parents}(x, D)} \text{Height}(y, D) + 1.
\]

Proposition 3. Let \( x, y, z \) be elements of some poset. If \( x \geq y \) is false then for any \( z \) such that \( z \leq x \), we have that \( z \geq y \) is also false.

Also we use the next proposition in our algorithms.

Proposition 4. Let \((P, \leq)\) be a poset and \( D \) be any non-empty, finite subset of \( P \). Let \( q \in P \setminus D \). Let \( L_h \) be a set of elements of \( D \) with fixed height \( h \) i.e., \( L_h = \{x \in D \mid \text{Height}(x, D) = h \} \). Let \( L^+_h \) be a set of elements of \( D \) with height more than \( h \) i.e., \( L^+_h = \{x \in D \mid \text{Height}(x, D) > h \} \). If \( \forall x \in L_h \) \( x > q \) is false, then \( \forall y \in L^+_h \) \( y > q \) is also false.

Proof. Suppose that \( \exists y \in L^+_h \) \( y > q \). Let height of \( y \) be \( l \) i.e., \( \text{height}(y) = l \). Then there exists a (maximum) chain \( y_0 = \top, y_1, y_2, \ldots, y_l = y \) such that \( y_0 > y_1 > y_2 > \cdots > y_l = y \). Since \( l > h \) we have a sequence \( y_0, y_1, y_2, \ldots, y_h, y_{h+1}, \ldots, y_l \) and \( y_0, y_1, y_2, \ldots, y_h \) is a maximum chain from \( y_0 \) to \( y_h \). Therefore \( \text{height}(y_h) = h \) or \( y_h \in L_h \). However \( y_h > y \) and \( y > q \) then \( y_h > q \). This contradicts to the condition of Proposition 3. □

3. Representation of a poset: the extended descendants list data structure

For representation of a poset we use list of descendants or list of ancestors in special form. Let \( c_1, \ldots, c_n \) be elements of poset \( D \) and let \( c_1 \) be top element, i.e., \( c_1 = \top \). We define partial order over integers \( D' = \{1, 2, \ldots, n\} \) in the following way: for any \( x, y \in D', x \leq y \) if and only if \( c_x \leq c_y \). Two lists are associated with an element \( c_i \in D \), \( i = 1..n \): list of descendants \( \text{D}(i) \) and list of ancestors \( \text{A}(i) \). The list of descendants includes \( \text{Height}(i, D) \) as a first component of the list, second component is \( \text{Outdeg}(i, D') \), and first \( \text{Outdeg}(i, D') \) descendants are \( \text{Children}(i, D') \), and other descendants:
\[
\text{D}(i) = [\text{Height}(i, D'); \text{Outdeg}(i, D'); \text{Child}_1(i, D'), \ldots, \text{Child}_{\text{Outdeg}(i, D')}(i, D')];
\]
\[
\text{Descendant}_{\text{Outdeg}(i, D') + 1}(i, D'), \ldots, 1, \quad i = 1..n.
\]

We assume that elements \( c_1, \ldots, c_n \) of poset \( D \) are ordered such that \( \text{Height}(i, D') \) is a non-decreasing function of \( i \). Also we denote the number of elements of dataset \( D' \) with \( \text{Height}(x, D') = h \) as \( n_h \), \( h = 0, \ldots, \text{Height}(D') = H \). It is clear that \( \sum_{h=0}^{H} n_h = n \). The list of ancestors is dually defined.

The following is the list of descendants of the poset given in Figure 1.
\[
\text{D}(1) = [0; 3; 2,3,4; 5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27],
\]
\[
\text{D}(2) = [1; 3; 5,6,16; 10,11,12,16,17,18,21,22,23,24,25,26,27],
\]
\[
\text{D}(3) = [1; 2; 7,8; 12,13,15,18,19,20,22,23,24,25,26,27],
\]
\[
\text{D}(4) = [1; 2; 8,9; 12,13,14,15,18,19,20,22,23,24,25,26,27],
\]
\[
\text{D}(5) = [2; 3; 10,11,12; 16,17,18,21,22,23,24,25,26,27],
\]

\[
\text{D}(25) = [6; 1; 27],
\]
\[
\text{D}(26) = [6; 1; 27],
\]
\[
\text{D}(27) = [7; 0].
\]
4. Updating operation

Updating can be done using Inserting and Deleting of a query element. The Deleting first uses Searching of a query element then deletes it and its links to parents and children. Deleting and Searching have the same query complexity. These two operations are almost same. Since in early works [12], [13] we developed algorithms for Searching operation, in this invention we do not consider the Deleting operation. Inserting can be done using Finding Parents and Finding Children operations. Finding Parents and Finding Children are dual operations, i.e., an algorithm for Finding Parents can be very easily adapted to Finding Children operation and vice versa. Thereby we consider Finding Parents operation in this paper. We assume that a query element is not in a database; otherwise again we have an operation which is equivalent to Searching. Since our original operation is Inserting this assumption is inartificial.

5. Algorithm for finding parents operation

Let \((P, \leq)\) be a poset and \(D\) be any dataset i.e., non-empty, finite subset of \(P: D \subseteq P, |D| = n\), where \(n\) is a positive integer. Let a query element \(q \in P\) be not in \(D\) i.e., \(q \notin D\). Although the query element \(q\) is not in the dataset \(D\), it has parents, children and ancestors, descendants in \(D\) i.e., query element has a virtual address in the dataset (Figure 2). The proposed algorithm takes list of descendants \(D(i), i = 1, \ldots, n\) of a given poset \(D\), the numbers \(n_h, h = 0, \ldots, \text{Height}(D) = H\) of elements of dataset \(D'\) with \(\text{Height}(x, D') = h\) and a query element \(q (\notin D)\) as input and finds \(\text{Parents}(q, D)\).

Data representation way described in section 3 allows us to distribute the elements of dataset in the table (Table 1). The height of the table is \(H + 1\), where \(H\) is height of dataset. Width of the table is equal to \(\max\{n_0, n_1, \ldots, n_H\}\). The top element \(\top\) is only element on the 0-th row of the table, for \(h = 1, \ldots, H\), \(h\)-th row of the table contains elements of dataset from \(\sum_{i=0}^{h-1} n_i + 1\) to \(\sum_{i=0}^{h} n_i\).

Proposed algorithm scans elements of this table from up to down and from left to right. Scanning is a comparison of current element \(c\) and query element \(q\): if \(c > q\) then YES status will be assigned to the element \(c\), else NO. Regarding Proposition 3 if NO status is assigned to the element \(c\) then NO status will be assigned to all its descendants also. Algorithm scans not only current element \(c\) but all its children also. If all children of \(c\) have NO status then \(c\) is a parent of query element \(q\). Regarding Proposition 4 algorithm ends if all elements of some row of the table have NO status.

![Figure 2. Example of a virtual address of a query element](image-url)
In Figure 3 you can see the pseudo-code description of proposed algorithm.

![Algorithm 1: FindingParents(Dataset, query element);](image)

Figure 3. Finding parents algorithm
6. Distributed algorithm

Proposed algorithm can be adapted for distributed computing. For distributed algorithm we use parallel random-access machine (PRAM) model of computation with $P_1, P_2, \ldots, P_m$, $m \geq 2$ processors. In PRAM model of computation all processors can access common RAM in a single algorithm-step. In the distributed algorithm one of the processors starts and ends to compute following Algorithm 1. Scanning of elements in each row will be done parallelly by $m$ processors. Each row of Table 1 i.e., elements of dataset with fixed height, is divided to $m$ equal intervals: 

$$
\left\{ \frac{\sum_{i=0}^{h-1} n_i + 1 + (s-1) \left\lfloor \frac{n_h}{m} \right\rfloor}{m}, \frac{\sum_{i=0}^{h-1} n_i + s \left\lfloor \frac{n_h}{m} \right\rfloor}{m} \right\}, \quad s = 1 \ldots m,
$$

where $\left\lfloor a \right\rfloor$ is the smallest integer not less than $a$.

Each interval is scanned by one processor. It is given the pseudo-code description of distributed algorithm in Figure 4. In Algorithm 2 two or more processors do not check the same query comparison if they do not come to common child of two elements with same height and from different intervals.
Table 1. Distribution of the elements of a dataset by heights

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | - | ... | - | ... | - | ... | - |
| 1 | 2 | 3 | ... | 1+n_i | - | ... | - | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| h | \(\sum_{i=0}^{h} n_i + 1\) | ... | ... | \(\sum_{i=0}^{h} n_i\) | - | ... | - | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| H | n | - | ... | - | ... | - | ... | - |

at exactly the same time. In other words, there is no overlapping of partition of scanning space up to query comparisons. We also note that the processors are almost uniformly tasked.

7. Conclusions

Updating consists of two sub-operations: (1) deleting of an existing object from a database; (2) inserting of a new object to a database. In both case we call of a deleting/inserting object as a query object (element). For deleting/inserting of query object it will be compared with objects of database. This comparison is called a query comparison. It is clear that any updating algorithm uses at worst \(n\) query comparisons, where \(n\) is a database size. If a query comparison is easily computable (for example, comparison of \(d\)-dimensional vectors) then even an algorithm with exactly \(n\) query comparisons is suitable; thereby to develop of an updating algorithm is very easy. However when a query comparison is not easily computable (for example, comparison of graphs) then efficiency of an updating algorithm is required to minimize the number of query comparisons. A query complexity of an algorithm is the number of query comparisons that is used in the algorithm. Our goal was to develop an updating algorithm(s) with a query complexity as less as possible.

We first analyzed an updating operation and picked out an operation Finding Parents as a most considerable sub-operation. We proposed an algorithm for Finding Parents operation. We used a new data structure for posets: it is an extended descendant list of a poset. The main extra parameter of this representation is a height/level of an element of a database. The extended descendant list of poset allows us to distribute the elements of a database in a rectangular table. Then algorithm starts to scan cells of the table. For minimizing of query complexity we have also used the properties of posets. We also proposed a distributed version of the algorithm. Proposed distributed algorithm responds two main requirements for distributed algorithms: uniform distribution of computation among processors and no overlapping computation.

References

[1] Sowa J.F., Conceptual Graphs, IBM Journal of Research and Development, 20(4), 1976, pp336 - 357.
[2] Sowa J.F., Conceptual Structures: Information Processing in Mind and Machine. 1984, Addison- Wesley.
[3] Chein M., Mugnier M.-L., Graph-based Knowledge Representation. Computational Foundations of Conceptual Graphs. 2009, Springer.
[4] Chein M., Mugnier M.-L., Conceptual Graphs: Fundamental Notions, Revue d'Intelligence Artificielle, 6(4), 1992, pp365-406.
[5] Willems M., Projection and unification for conceptual graphs, In Proc. of ICCS’95, Vol. 954 of LNCS, pp 278-292, 1995, Springer.
[6] Mugnier M.-L., On Generalization / Specialization for Conceptual Graphs, Journal of Experimental and Theoretical Artificial Intelligence, 7, 1995, pp325-344.
[7] Baget J.-F., Representer des connaissances et raisonner avec des hypergraphes: de la projection a la derivation sous contraintes. PhD Thesis, 2001, University Montpellier II.
[8] Baget J.-F., *Simple Conceptual Graphs Revisited: Hypergraphs and Conjuctive Types for Efficient Projection Algorithms*. In Proc. of ICCS’03, vol. 2746 of LNAI. 2003, Springer.

[9] Croitoru M. and Compatangelo E., *A tree decomposition algorithm for conceptual graph Projection*. In Proc. of KR’06, pp271-276. 2006, AAAI Press.

[10] Mozes S., Onak K., Weimann O., *Finding an optimal tree searching strategy in linear time*, In: SODA'08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, 2008, pp1096-1105.

[11] Dereniowski D., *Edge ranking and searching in partial orders*, Discrete Appl. Math. 156(13), 2008, pp2493-2500.

[12] Mamadolimov A., *Search algorithms for conceptual graph databases*. International Journal of database Management Systems (IJDMS), Vol.5, No.1, 2013, pp 35-44.

[13] Mamadolimov A., *Searching Algorithms in Partially Ordered Set*, SIAM Conference on Discrete Mathematics, June 18-21, 2012, Dalhousie University, Halifax, Nova Scotia, Canada.

[14] Levinson R.A., *A self-organizing retrieval system for graphs*, In The 3rd National Conference of the American Association for Artificial Intelligence, 1984, pp 203-206, Austin, Texas, AAAI Press, Menlo Park.

[15] Ellis G.R., *Managing Complex Objects*. PhD Thesis. University of Queensland, 1995.