Efficiently Bounding Cardinality Ratios through Database Constraints

Paolo Ciaccia, Matteo Golfarelli, and Stefano Rizzi

DEIS, University of Bologna, Italy

Abstract. Numerical dependencies (NDs) are a type of database constraints in which one limits the number of distinct Y-values that can appear together with any X-value, where both X and Y are sets of attributes. The seminal work by Grant and Minker has shown that NDs are not finitely axiomatizable, which has cut further investigation on this kind of constraints. In this paper we show that, given a set of sound inference rules similar to those used for functional dependencies, the membership problem for NDs is NP-hard, and propose a branch & bound algorithm for efficiently solving the problem. The algorithms adopts a suite of optimization strategies that make it applicable in practice, providing considerable speed-up over a naïve approach.

1 Introduction

Reasoning with database constraints has a huge number of practical applications. These include database design, query processing and optimization, schema matching, data lineage and repair, to name just a few. Consequently, properly understanding the properties of a given type of constraints has always been a major topic in database theory. Cardinality constraints are a remarkable class that has been investigated in a variety of specific settings. For instance, in the context of the Entity-Relationship model, [8] studies the problem of determining when a set of cardinality ratio constraints, imposing restrictions on the mappings between entities and relationships, are consistent, i.e., no entity or relationship is compelled to be empty in all the legal instances of the schema. In [6] the focus is on cardinality constraints that impose restrictions on the number of relationships an object can be involved in. The entailment problem (i.e., checking whether a given constraint set entails further constraints) is faced, and combinatorial methods for reasoning about sets of cardinality constraints are proposed. Similarly, in [7] the satisfiability and implication problems for numerical constraints are faced with reference to the XML language. A numerical constraint is defined in terms of path expressions, and restricts the number of nodes that have the same values on some selected subnodes.

In this paper we consider a specific type of cardinality constraints, called *numerical dependencies* (NDs), which were introduced by Grant and Minker in [5]. Intuitively, given two sets of attributes X and Y, there is an ND from X to Y if each value of X can never be associated to more than k distinct values of Y,

$k \geq 1$. NDs are a natural generalization of functional dependencies, which are obtained when $k = 1$. Reasoning with NDs has a number of applications in the database field. For instance, estimating the projection size of a relation, i.e., the number of distinct values over a subset of its attributes, is a frequent problem in database applications [4]. When data are not available (e.g., at design time), statistical techniques like those based on histograms or sampling cannot be used to this end, so probabilistic approaches must be followed. These approaches rely on the assumption that the relation attributes are independent of each other; when this is not the case, NDs enable inter-attribute cardinality constraints to be effectively captured, thus noticeably improving the accuracy in projection size estimation. Similarly, estimating the cardinality of aggregate views has a crucial importance in the field of data warehousing, with reference to logical and physical design as well as query processing and optimization [10]. In particular, view materialization may significantly benefit from using NDs since the algorithms that select the best aggregate views to be materialized are based on view cardinality estimates [9, 3].

Unfortunately, NDs are not finitely axiomatizable, thus no finite set of sound and complete rules exists for them [5]. This negative result has prevented further research aiming to deepen the understanding of NDs. In this paper we first prove that, given a set of sound inference rules similar to those used for functional dependencies, the membership problem for NDs is NP-hard. Thus, even reasoning on derivable NDs only is a complex task. Then we propose a branch & bound algorithm, based on a graph-based characterization of NDs, for efficiently solving the problem. The algorithm adopts a suite of optimization strategies that make it applicable in practice, providing considerable speed-up over a naïve approach.

2 Numerical Dependencies

In this section we provide the necessary background and prove basic facts about NDs that are needed for the paper.

Let $R(U)$ be a relation schema,[1] and let r be an instance of R, i.e., a finite set of tuples over U. For any $X \subseteq U$, $dom(X)$ is the Cartesian product of the domains of the attributes in X and an X-value is any value from $dom(X)$.

Given a set of constraints $\Delta = \{\delta_1, \ldots, \delta_n\}$, r is *legal* w.r.t. Δ iff r satisfies all the δ_i's in Δ. A set of constraints Δ *entails* δ if when r is legal w.r.t. Δ then r also satisfies δ. The (semantic) closure Δ^+ of Δ is the set of all the constraints entailed by Δ. A constraint δ is *derivable* from Δ using a set of inference rules I if there exists a finite derivation of δ from Δ using the rules I. The (syntactic) closure Δ_I^+ of Δ is the set of constraints that are derivable from Δ using I. Rules I are *sound* if $\Delta_I^+ \subseteq \Delta^+$ and *complete* if $\Delta^+ \subseteq \Delta_I^+$. Given a set of constraints Δ of a given type $\mathcal{T}$, the *membership problem* for $\mathcal{T}$ is to determine if $\delta \in \Delta^+$.

[1] We use uppercase letters from the beginning (ending) of the alphabet to denote single (respectively, sets of) attributes. We use concatenation for forming sets of attributes, thus writing ABC for $\{A, B, C\}$, and for denoting union, thus XY stands for $X \cup Y$.

Functional dependencies (FDs) are among the most common types of relational database constraints. The FD $X \rightarrow Y$ is satisfied by an instance r of $R(XYZ)$ if any two tuples with the same X-value also have the same Y-value. It is known that the membership problem for FDs can be easily solved in linear time in len(Δ), the total number of non-distinct attribute symbols in Δ [2].

NDs [5] are a generalization of FDs, in which one limits the number of Y-values that can be associated with any X-value.

Definition 1 (Numerical Dependencies). *Given $R(XYZ)$ and a finite integer $k \geq 1$, we say the* numerical dependency $X \xrightarrow{k} Y$ *is satisfied by an instance r of $R(XYZ)$ if for any $k+1$ tuples $t_1, \ldots, t_{k+1}$ in r, if $t_1[X] = \ldots = t_{k+1}[X]$, then there are at least two of these tuples, t_i and t_j, $i \neq j$, such that $t_i[Y] = t_j[Y]$. For an ND $\delta : X \xrightarrow{k} Y$ we say that k is the* weight *of δ, also denoted as $w(\delta)$.*

Clearly, an FD is just a particular case of ND in which $k = 1$, i.e., $X \xrightarrow{1} Y \equiv X \rightarrow Y$. NDs also allow to specify *cardinality constraints* over sets of attributes [5]. For this one considers NDs of the form $\perp \xrightarrow{k} Y$, where $\perp$ is the empty set of attributes. This is equivalent to say that the cardinality of projection on Y can never exceed k: $|\pi_Y(r)| \leq k \ \forall r$. More in general, if $X \xrightarrow{k} Y$ holds, then for any legal instance r of $R(XYZ)$ it is $|\pi_{XY}(r)| \leq k \cdot |\pi_X(r)|$. For instance, if $BC \xrightarrow{5} DE$, then each BC-value can never appear with more than 5 distinct DE-values. Thus, $|\pi_{BCDE}(r)| \leq 5 \cdot |\pi_{BC}(r)|$ in each legal instance r.

Unlike FDs, NDs do not admit a finite set of sound and complete rules [5]. A sound set of inference rules, that generalizes that of FDs, is the following one:[2]

$$\begin{aligned} \textit{Reflexivity}\ (R): &\quad \vdash X \xrightarrow{1} X \\ \textit{Extended transitivity}\ (E): &\quad X \xrightarrow{k} YW \ \wedge Y \xrightarrow{l} Z \ \vdash X \xrightarrow{k \cdot l} YWZ \\ \textit{Decomposition}\ (D): &\quad X \xrightarrow{k} YZ \ \vdash X \xrightarrow{k} Y \end{aligned}$$

Given the above *RED rules*, our aim is to understand if a given ND δ is derivable using them, i.e., if $\delta \in \Delta^+_{RED}$. Unlike FDs, this can be a complex task for NDs.

Theorem 1. *Given a set of NDs Δ over a schema $R(U)$, determining if the ND $\delta : X \xrightarrow{k} Y$, $XY \subseteq U$, is derivable from Δ using the RED rules is NP-hard.*

Note that Δ^+_{RED} has infinite size even when Δ consists of a single ND $X \xrightarrow{k} Y$, because from $X \xrightarrow{k} Y$ one can derive $X \xrightarrow{k^n} Y$, for any $n > 1$.[3] Although one can easily remove such uninteresting NDs by considering only derivations that

[2] This is equivalent to the set of rules proposed by Grant and Minker if one also adds the so-called *Successor* rule, $X \xrightarrow{k} Y \ \vdash \ X \xrightarrow{k+1} Y$, which we have intentionally omitted since it has no influence on the problem we deal with.

[3] From $X \xrightarrow{1} X$ and $X \xrightarrow{k} Y$ it is derived, using (E), $X \xrightarrow{k} XY$; applying again rule (E) to $X \xrightarrow{k} XY$ and $X \xrightarrow{k} Y$ yields $X \xrightarrow{k^2} XY$, hence $X \xrightarrow{k^2} Y$ by (D); and so on.

use at most once any ND in Δ, Δ^+_{RED} would still contain several NDs with the same left- and right-hand sides, as the following example shows.

Example 1. Let $U = ABCD$ and $\Delta = \{A \xrightarrow{k_1} B, B \xrightarrow{k_2} C, B \xrightarrow{k_3} D, D \xrightarrow{k_4} C\}$. All the following NDs relating $X \equiv A$ and $Y \equiv BCD$ are in Δ^+_{RED}:

$$\delta_1 : A \xrightarrow{k_1 k_2 k_3} BCD \qquad \delta_2 : A \xrightarrow{k_1 k_2 k_3 k_4} BCD \qquad \delta_3 : A \xrightarrow{k_1 k_3 k_4} BCD$$

Also observe that, while it is impossible to say which ND, between δ_1 and δ_3, has the lowest weight without knowing the values of the k_i's involved, one can immediately notice that δ_2 is "loose", since it is always true that $w(\delta_2) \geq w(\delta_1)$ and $w(\delta_2) \geq w(\delta_3)$, *regardless of k_i's values.*

Given the above observations, we consider the following problem:

Given a set of NDs Δ over $R(U)$ and two sets of attributes X and Y, $XY \subseteq U$
Determine $k(X, Y)$, i.e., the minimal value of k such that $X \xrightarrow{k} Y \in \Delta^+_{RED}$.

For the purpose of determining $k(X, Y)$, an ND like δ_2 in Example 1 is uninteresting at all since, for *any* assignment of values to the k_i's, its weight will not be lower than those of δ_1 and δ_3. This observation is crucial for characterizing the "interesting" part of the RED closure, which consists of all and only those NDs for which it is necessary to look at the values of the k_i's to determine which one has the lowest weight.

Definition 2 (Tight RED Closure). *For an ND $\delta : X \xrightarrow{k_1 \cdot \ldots \cdot k_n} Y$, let $K(\delta)$ denote the bag $\{\{k_i\}\}$, where each k_i appears as many times as it appears in δ. An ND $\delta : X \xrightarrow{w(\delta)} Y \in \Delta^+_{RED}$ is* tight *if, for any other ND $\delta' : X \xrightarrow{w(\delta')} Y \in \Delta^+_{RED}$, it is not $K(\delta') \subset K(\delta)$,* loose *otherwise. The* tight (RED) closure *of Δ is the set $\Delta^*_{RED} \subset \Delta^+_{RED}$ of all the tight NDs in Δ^+_{RED}.*

3 A Graph-Based Characterization

In this section we show that the tight closure of a set of NDs Δ can be precisely characterized in graph-theoretical terms. Without loss of generality we assume that Δ does not contain two NDs with the same left- and right-hand sides.[4] We represent a set of NDs through an *ND-graph* defined as follows:

Definition 3 (ND-graph). *Given a set Δ of NDs over $R(U)$, the* ND-graph $G_\Delta = (\mathcal{V}, \mathcal{E})$ induced by Δ *is the directed graph with nodes $\mathcal{V} \subseteq 2^U$, arcs $\mathcal{E} = \mathcal{E}^f \cup \mathcal{E}^d$ ($\mathcal{E}^f \cap \mathcal{E}^d = \emptyset$), and an arc labeling function $\omega : \mathcal{E} \to \mathbb{N}$ (weight) such that:*

1. *For every ND $\delta : X \xrightarrow{k} Y \in \Delta$ there are in $\mathcal{V}$ two nodes X and Y, and there is in $\mathcal{E}^f$ a* full *arc $\langle X, Y\rangle$ (oriented from X to Y) such that $\omega(\langle X, Y\rangle) = k$. When needed we write $\langle X, Y\rangle_k$ to denote that $\omega(\langle X, Y\rangle) = k$.*

[4] If this is the case, the one with the higher weight can be safely removed.

2. *For every node $X \in \mathcal{V}$, $X = A_1, \ldots, A_r$, $r > 1$, there are r nodes $A_1, \ldots, A_r$ in $\mathcal{V}$ and r* dotted *arcs $\langle X, A_1\rangle, \ldots, \langle X, A_r\rangle$ in $\mathcal{E}^d$ with $\omega(\langle X, A_i\rangle) = 1$.*
3. *If the empty set of attributes $\perp$ is in $\mathcal{V}$, then for each node $A_i \in \mathcal{V}$ there is in $\mathcal{E}^d$ a dotted arc $\langle A_i, \perp\rangle$ with $\omega(\langle A_i, \perp\rangle) = 1$.*

In the particular case $\Delta = \emptyset$, it is conventionally assumed that, for any *choice of $X \subseteq U$, the graph including node X, and completed respecting the above rule 2 if X consists of more than one attribute, is also an ND-graph. With a slight abuse of terminology we say that this is the* ND-graph induced by X. *If $\Gamma \subseteq \Delta$, then G_Γ is also called an ND-subgraph of G_Δ, and $Attr(G_\Gamma)$ denotes the set of all attributes appearing in at least one node of G_Γ.*

Example 2. The ND-graph for $U = ABCDE$ and $\Delta = \{G \xrightarrow{k_0} A, A \xrightarrow{k_1} B, A \xrightarrow{k_2} BC, E \xrightarrow{k_3} BC, A \xrightarrow{k_4} CD, A \xrightarrow{k_5} DE, E \xrightarrow{k_6} F\}$ is shown in Figure 1.

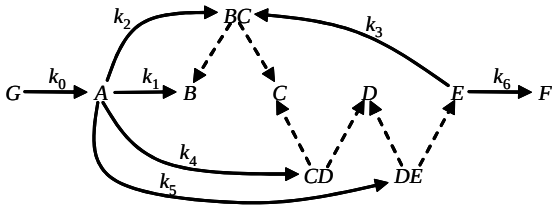

Fig. 1. The ND-graph for the set of NDs in Example 2

The ND-graph represents in a compact form all the relevant information needed to derive tight NDs. The following defines when a node Y is *reachable* from a node X, i.e., how one can navigate an ND-graph.

Definition 4 (Reachability). *Given the ND-graph $G_\Delta = (\mathcal{V}, \mathcal{E})$ and a node $X \in \mathcal{V}$, the following rules define which nodes are* reachable *from X:*

1. *If $X = A_1, \ldots, A_r$, then any A_i is reachable from X.*
2. *If W is reachable from X and $\langle W, Z\rangle_k \in \mathcal{E}^f$, then Z is reachable from X.*
3. *If $Z = B_1, \ldots, B_m$, $Z \in \mathcal{V}$, and each B_i is reachable from X, then Z is reachable from X.*

The preconditions of all the above rules can be uniformly expressed in terms of which are the attributes appearing in the nodes of a suitably defined ND-subgraph of G_Δ, which justifies the following definition.

Definition 5 (ND-path from X to Y). *Given the ND-graph $G_\Delta = (\mathcal{V}, \mathcal{E})$ and $X \in \mathcal{V}$, an* ND-path from X *is any ND-subgraph of G_Δ that can be inductively obtained as follows:*

1. *The ND-subgraph induced by X is an ND-path from X.*
2. *If G_Π is an ND-path from X induced by a set of NDs $\Pi \subset \Delta$, and $\langle W, Z\rangle_k \in \mathcal{E}^f$, with $W \subseteq Attr(G_\Pi)$ and $Z \not\subseteq Attr(G_\Pi)$, the ND-subgraph induced by $\Pi \cup \{W \xrightarrow{k} Z\}$ is also an ND-path from X.*

3. No other ND-subgraph of G_Δ is an ND-path from X.

In order to emphasize that an ND-subgraph G_Π is also an ND-path from X, the notation G^X_Π will also be used. The weight *$\omega(G^X_\Pi)$ of G^X_Π is the product of the weights in its* full *arcs; by definition, the weight of $G^X_\emptyset$ is 1.*

Given an ND-path G^X_Π from X and a set of attributes $Y \subseteq Attr(G^X_\Pi)$, G^X_Π is also called an ND-path from X to Y. *G^X_Π is Y-*minimal *iff there is no other ND-path G^X_Γ from X to Y such that $\Gamma \subset \Pi$.*

Notice that, although any subset of NDs $\Gamma \subseteq \Delta$ induces an ND-subgraph, due to rule 2 in the above definition not all of them are also ND-paths.

Example 3. With reference to the ND-graph introduced in Example 2, Figure 2 shows two ND-paths from A. It is $Attr(G^A_{\Pi_1}) = Attr(G^A_{\Pi_2}) = ABCD$; the weights are $\omega(G^A_{\Pi_1}) = k_1k_2k_5$ and $\omega(G^A_{\Pi_2}) = k_1k_2$, respectively. Both $G^A_{\Pi_1}$ and $G^A_{\Pi_2}$ are ND-paths to BC and BCD, but only $G^A_{\Pi_2}$ is both BC- and BCD-minimal. Finally, the ND-subgraph G_{Π_3} is *not* an ND-path, since there is no way to incrementally build it by respecting rule 2 in Definition 5.

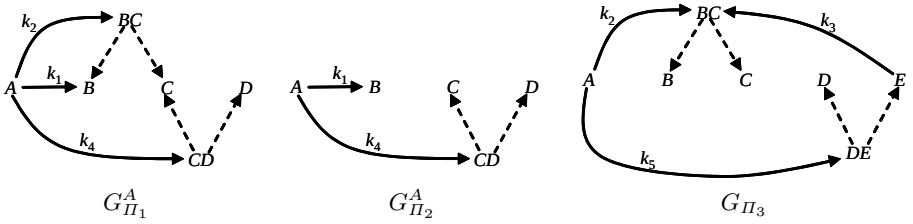

Fig. 2. $G^A_{\Pi_1}$ and $G^A_{\Pi_2}$ are ND-paths from A obtained from the ND-graph in Figure 1; G_{Π_3} is not an ND-path

It is quite simple to show that if there exists an ND-path from X to Y with weight $\omega(G^X_\Pi)$, then the ND $\delta : X \xrightarrow{\omega(G^X_\Pi)} Y$ is in Δ^+_{RED}. However, not all such NDs are necessarily tight. This leads to the following main result.

Theorem 2. *Let G_Δ be the ND-graph induced by a set of NDs Δ, and let $X \in \mathcal{V}$. There exists in G_Δ a Y-minimal ND-path G^X_Π from X to Y having weight $\omega(G^X_\Pi)$ iff $\delta : X \xrightarrow{\omega(G^X_\Pi)} Y \in \Delta^*_{RED}$.*

So far, we have used G_Δ to reason on NDs whose left-hand side is a node in G_Δ. In order to generalize to arbitrary NDs, it is sufficient to properly extend G_Δ by adding the required node X; after that, all above results apply unchanged. For this reason in the following, without losing in generality and to avoid unnecessarily complicating the description, we will always assume that X is a node in G_Δ.

4 Efficiently Finding the Minimal Weight

A Naïve algorithm (not detailed here for lack of space) would compute $k(X, Y)$ starting with the ND-graph $G_{\emptyset}^{X}$ induced by X, and then progressively extending it by adding full arcs, until *all* possible ND-paths from X to Y (*solutions*) are generated.

Example 4. Figure 3 shows the complete search space of the Naïve algorithm for the ND-graph in Figure 1, assuming $X \equiv A$ and $Y \equiv BE$. Each node corresponds to an ND-path from A and is represented by its set of attributes; all leaves represent possible solutions. Each edge is labeled with the weight of the new full arc being added. The tree includes three distinct BE-minimal ND-paths (boxed in Figure 3), whose weights are k_1k_5, k_2k_5, and k_3k_5 respectively.

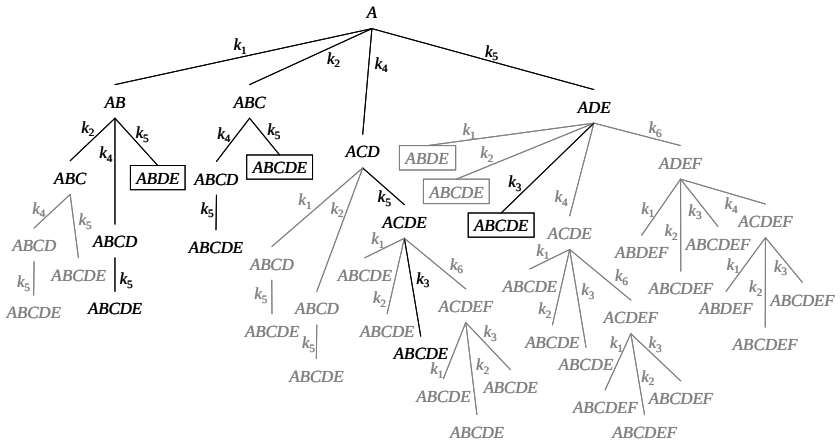

Fig. 3. The complete search space of the Naïve algorithm applied to the ND-graph in Figure 1 with $X \equiv A$ and $Y \equiv BE$; boxed nodes represent BE-minimal ND-paths. In grey, the part that is pruned by the non-numerical optimization techniques of the BBND algorithm

As Example 4 makes evident, the Naïve algorithm has several drawbacks. First of all, it creates some solutions using arc $\langle E, F \rangle_{k_6}$, that obviously can never contribute to reach BE. Besides, it generates twice or more the same ND-paths by adding the same set of full arcs in different orders. Furthermore, it extends an ND-path even if its weight is higher than that of the current solution. Finally, it does not detect non-minimal ND-paths, so it wastes time in extending them.

In order to obviate all these problems, we introduce a branch & bound algorithm, called BBND, whose major features are described in the following.

Removing Useless NDs. The ND-graph G_{Δ} includes *all* the full arcs corresponding to NDs defined in the application domain, but only some of them might be

Algorithm 1 The BBND algorithm

Input: G_{Δ}, X, Y
Output: $k(X, Y)$

1: $k(X,Y) \leftarrow \infty$
2: $ActiveNDPaths \leftarrow \{G^X_{\emptyset}\}$ ▷ ND-paths to be extended
3: **if** $Y \not\subseteq Attr(G_{\Delta})$ **then return** $k(X,Y)$ ▷ Y is not reachable from X
4: RemoveUselessNDs$(G_{\Delta}, X, Y)\}$ ▷ Remove useless NDs from G_{Δ}
5: **while** $ActiveNDPaths \neq \emptyset$ **do**
6: $\quad G^X_{\Pi} \leftarrow$ Pop$(ActiveNDPaths)$
7: $\quad$ **for all** $G^X_{\Gamma_i} \in$ SmartExtensions(G^X_{Π}) **do**
8: $\quad\quad$ **if** $Y \subseteq Attr(G^X_{\Gamma_i})$ **then** ▷ found a solution...
9: $\quad\quad\quad$ **if** $\omega(G^X_{\Gamma_i}) < k(X,Y)$ **then** ▷ ... better than the current one
10: $\quad\quad\quad\quad k(X,Y) \leftarrow \omega(G^X_{\Gamma_i})$
11: $\quad\quad$ **else if** $\omega(G^X_{\Gamma_i}) < k(X,Y) \wedge \neg$IsDominated$(G^X_{\Gamma_i}, ActiveNDPaths) \wedge$ IsMinimal$(G^X_{\Gamma_i})$ **then**
12: $\quad\quad\quad$ Push$(ActiveNDPaths, G^X_{\Gamma_i})$ ▷ $G^X_{\Gamma_i}$ must be further extended
13: **return** $k(X,Y)$

relevant in determining $k(X, Y)$. For instance, with reference to the ND-graph in Figure 1, full arcs $\langle G, A\rangle_{k_0}$ and $\langle E, F\rangle_{k_6}$ are obviously useless to compute $k(A, BE)$ and should be removed to improve performance.

Removal of useless NDs is carried out by the RemoveUselessNDs method (line 4), that works in two phases. In the first phase it navigates the ND-graph forward starting from X, so as to mark all the nodes that can be reached from X, and consequently a set of full arcs; in our example, all full arcs are marked except $\langle G, A\rangle_{k_0}$, that is removed. In the second phase, RemoveUselessNDs navigates the so-reduced ND-graph backward starting from the nodes corresponding to each $A_i \in Y$, and marks all the full arcs from which A_i can be reached; in our example, only the full arc $\langle E, F\rangle_{k_6}$ is not marked and consequently removed.

Avoiding Repeated ND-Paths. Different sequences of extension steps might lead to ND-paths sharing the same set of full arcs. To avoid generating repeated ND-paths, it is necessary to be able to recognize if a given set of full arcs has already been obtained in the enumeration process. Clearly, keeping trace of all the sets of full arcs generated so far would require an exponential effort in terms of space. A simpler alternative is to impose an ordering criterion on the set of NDs, such as a lexicographically ordering of their right-hand sides. This can be easily implemented as follows.

Let $\Delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$ be the given set of NDs, where subscripts reflect the chosen ordering criterion. Consider an ND-path $G^X_{\Pi_i}$, $\Pi_i \subset \Delta$, that has been obtained by extending the ND-path G^X_{Π} with (the full arc corresponding to) δ_i. Furthermore, let $\Gamma \subseteq \Delta \setminus \Pi$ be the set of NDs that could have been used to extend G^X_{Π} (these are precisely all the NDs in $\Delta \setminus \Pi$ that satisfy rule 2 in Definition 5) and $\Gamma_i \subseteq \Delta \setminus (\Pi \cup \{\delta_i\})$ be similarly defined. The Naïve approach would extend $G^X_{\Pi_i}$ by picking *all* the NDs in Γ_i. In order to guarantee that the

same ND-path is not generated more than once, method SmartExtensions (line 7) does *not* pick from Γ_i those NDs δ_j such that $\delta_j \in \Gamma$ and $j < i$, i.e., those NDs that lexicographically precede δ_i and that could have been added at the previous extension step.

Pruning Non-Minimal ND-Paths. The Naïve algorithm is likely to waste a lot of work in generating non-minimal ND-paths. As the following result shows, these can be safely dropped.

Lemma 1. *Let G^X_Π be an ND-path from X, and G^X_Γ be any ND-path such that $\Gamma \subset \Pi$. If G^X_Γ is not $Attr(G^X_\Gamma)$-minimal, then G^X_Π is not Y-minimal for all $Y \subseteq Attr(G^X_\Pi)$.*

Detection of non-minimal ND-paths is based on Definition 5. Given an ND-path G^X_Γ, if all the NDs included in Γ are removed one at a time and the corresponding ND-subgraphs G_{Γ_i} are generated, then if at least one of such G_{Γ_i}'s is indeed an ND-path from X with $Attr(G_{\Gamma_i}) = Attr(G^X_\Gamma)$ it follows that G^X_Γ is not minimal and can be discarded. Method IsMinimal (line 11) runs in $\mathcal{O}(n\text{len}(\Gamma))$ time, where n is the number of NDs in Γ. The $\text{len}(\Gamma)$-time complexity of ND-subgraph generation follows after observing that just checking that G_{Γ_i} is an ND-path from X with $Attr(G_{\Gamma_i}) = Attr(G^X_\Gamma)$ is analogous to the problem of computing the FD-closure of a set of attributes X, i.e., the set of all attributes A_i such that the FD $X \to A_i$ holds. Since this can be done in linear time in the length of the given set of constraints [2], the same result also applies to NDs.

Method IsMinimal adopts two complementary strategies for reducing the actual number of ND-subgraphs to be tested. The first of them is based on the concept of *essential* NDs. Intuitively, these are those NDs whose removal makes some attributes not reachable anymore from X, thus method IsMinimal needs not try to remove them.

Lemma 2. *Given an ND-path G^X_Γ, let $\delta \in \Gamma$, $\delta : W \xrightarrow{k} Z$, be an* essential *ND for G^X_Γ, i.e., there exists $A \in Z \setminus X$ such that no other ND in G^X_Γ includes A in its right-hand side. Let G_{Γ_i} be the ND-subgraph of G^X_Γ induced by $\Gamma_i = \Gamma \setminus \{\delta\}$; then it is $Attr(G^X_{\Gamma_i}) \subset Attr(G^X_\Pi)$.*

A second way to reduce the actual running time of the minimality check is based on the following lemma.

Lemma 3. *Let G^X_Γ be an ND-path from X, obtained by extending a minimal ND-path G^X_Π with the ND $\delta : W \xrightarrow{k} Z$. Let $\delta_i : W_i \xrightarrow{k_i} Z_i \in \Pi$ be a non-essential ND for G^X_Π. If $Z \cap Z_i = \emptyset$, then the ND-subgraph G_{Γ_i} induced by $\Gamma_i = \Gamma \setminus \delta_i$ is not an ND-path from X.*

Figure 3 shows in grey the portion of the search space that is not explored when methods RemoveUselessNDs, SmartExtensions, and IsMinimal are applied. Note that all these optimization techniques are non-numerical, i.e., their application is independent of the numerical values of the weights.

Exploiting Weights. A first, simple way to exploit the ND weights for reducing the search space is to prune all the branches that originate from an ND-path whose weight is not lower than that of the best solution found so far. A more effective way to exploit weights is based on the concept of *domination*.

Definition 6 (Domination). *Let G^X_Π and G^X_Γ be two ND-paths from X. G^X_Π is said to* dominate *G^X_Γ if $Attr(G^X_\Pi) \supseteq Attr(G^X_\Gamma)$ and $\omega(G^X_\Pi) \leq \omega(G^X_\Gamma)$.*

Lemma 4. *Let G^X_Π and G^X_Γ be two ND-paths from X. If G^X_Π dominates G^X_Γ then, for any ND-path $G^X_{\Gamma_i}$ obtained by extending G^X_Γ with an ND δ_i, either (1) G^X_Π dominates $G^X_{\Gamma_i}$, or (2) the ND-path $G^X_{\Pi_i}$ that extends G^X_Π with δ_i dominates $G^X_{\Gamma_i}$.*

Lemma 4 proves that dominated ND-paths can be safely discarded. This kind of pruning is highly effective in reducing the size of the search space because it can be applied to generic ND-paths and even because it works in both directions, i.e., method IsDominated (line 11) not only checks if the newly generated ND-path G^X_Γ is dominated by an ND-path in $ActiveNDPaths$, but, if the test fails, it also verifies if the opposite holds, in which case dominated ND-paths are removed from the queue.

Unlike the previous optimization strategies, the influence that numerical domination has on the search space depends on the specific strategy adopted for enumerating ND-paths. In BBND we implemented a best-first enumeration strategy for choosing which ND-path has to be extended.[5] The specific heuristic function we adopt attempts to predict how close the ND-path is to a solution. In particular, we extend the ND-path G^X_Π for which the cardinality of $Attr(G^X_\Pi) \cap Y$ is maximum, i.e., the ND-path that is "nearest" to Y, first. In case of ties, the ND-path with the lowest weight is chosen.

5 Experimental Results and Conclusions

In this paper we have shown how, given a set of sound inference rules for NDs, one can efficiently determine the best possible (minimal) value $k(X, Y)$ such that the ND $X \xrightarrow{k(X,Y)} Y$ is derivable. The branch & bound algorithm we have proposed is based on a graph characterization of the NDs and adopts a suite of optimization strategies. Our ND-graph representation is largely inspired to the one proposed in [1] for FDs; however, the major difference with [1] is that, in the worst case, we have to generate *all* possible minimal paths from X to Y, which is not an issue for FDs.

We extensively tested the BBND algorithm against different sets of NDs. Since deriving tight bounds is crucial in physical design of data warehouses, the relation schema for our tests was inspired by a star schema. We created a relation schema R including 24 attributes, on which we defined 40 "basic"

[5] Other strategies we tested, such as breadth-first and depth-first, lead to considerable higher running times.

NDs: 16 FDs that model multidimensional hierarchies, plus 24 NDs of the form $\perp \xrightarrow{k_i} A_i$ that put a cardinality constraint k_i on each simple attribute A_i in R. Note that, using only this data, the cardinality bound of each projection Y_m of R is simply the product of the cardinalities of the attributes in Y_m. Then, we randomly generated three sets of additional NDs (Δ_1, Δ_2, and Δ_3, with $\Delta_1 \subset \Delta_2 \subset \Delta_3$) over R. Finally, we randomly generated 300 projections Y_m (each including from 2 to 8 attributes); for each set Y_m and for each Δ_j, we applied the BBND algorithm to compute the minimal weight $k_{j,m}$ such that $\perp \xrightarrow{k_{j,m}} Y_m \in \Delta_j^+$, which corresponds to finding the best cardinality bound for Y_m given Δ_j. Thus, 900 optimization problems were solved overall.

Table 1. Results of BBND tests; execution times are in seconds

	Δ_1	Δ_2	Δ_3
♯ NDs	(40+)10	(40+)25	(40+)50
% bound	43%	11%	4%
BBND exec. time	11.81	14.62	62.09
$\mathsf{BBND}_{\mathsf{useless}}$ exec. time	79.8	15.63	90.97
$\mathsf{BBND}_{\mathsf{minimal}}$ exec. time	15.54	15.57	39.7
$\mathsf{BBND}_{\mathsf{numerical}}$ exec. time	152.85	593.27	$> 5\ 000$
♯ useless NDs	6	2	5
♯ ND-paths generated by BBND	316 417	572 061	1 952 367

Table 1 summarizes the results. The second row gives a measure of effectiveness of NDs in reducing cardinality bounds by showing the average ratio between the cardinality bound for Y_m obtained using NDs and the one computed as the product of the cardinalities of the attributes in Y_m. The following four rows address efficiency by reporting the average execution times (on a Pentium 4, 3.4 GHz, 2 GB RAM) of four variants of our algorithm: BBND (the complete algorithm shown in Section 4), $\mathsf{BBND}_{\mathsf{useless}}$ (useless NDs not removed), $\mathsf{BBND}_{\mathsf{minimal}}$ (no minimality check executed), and $\mathsf{BBND}_{\mathsf{numerical}}$ (no numerical domination check executed). The last two rows report the average number of useless NDs and the average number of ND-paths generated by BBND. Finally, Figure 4 shows how effectiveness and efficiency depend on the number of additional NDs used in the optimal solution.

It is apparent from these results that: (1) NDs are highly effective in reducing cardinality bounds: even having a single additional ND in the optimal solution reduces the cardinality bound by about 80%; (2) although the problem is NP-hard, our algorithm finds the solution within a reasonable time even for a very large state space thanks to its optimization strategies; (3) removing useless NDs highly improves efficiency, because detecting these NDs is a low-cost operation and may drastically cut the search space; (4) checking ND minimality may significantly reduce the search space size, but for large sets of NDs the cost for checking overcomes the benefit; (5) the most effective pruning strategy is the one based on numerical domination, in fact the execution times of $\mathsf{BBND}_{\mathsf{numerical}}$ are (at least) one order of magnitude higher than those of BBND. Of course, the

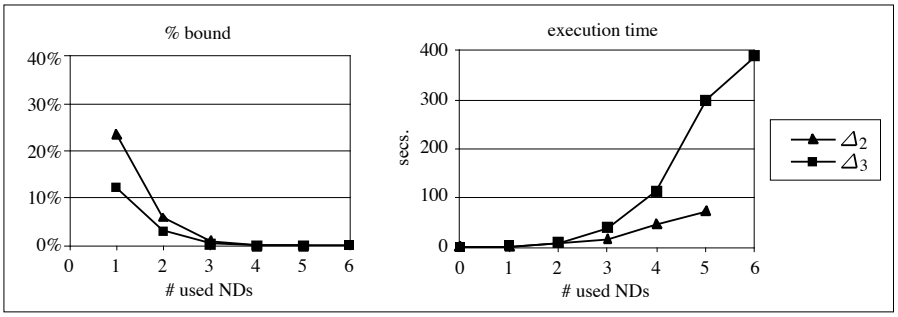

Fig. 4. Average effectiveness and efficiency of BBND in function of the number of additional NDs used in the optimal solution

impact of optimization strategies also depends on the order in which they are evaluated (in our implementation, low-cost strategies are applied first) and on the topology of the ND-graph (for instance, for Δ_2 the impact of useless NDs removal is low because there are only two useless NDs).

We finally remark that, though the detailed results are not reported for lack of space, the tests also show that, on the average, the bound found by BBND in 3 seconds differs from the optimal bound by less than 10%.

References

1. G. Ausiello, A. D'Atri, and D. Saccà. Graph algorithms for functional dependency manipulation. *Journal of the ACM*, 30(4):752–766, 1983.
2. C. Beeri and P.A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM TODS*, 4(1):30–59, 1979.
3. P. Ciaccia, M. Golfarelli, and S. Rizzi. Bounding the cardinality of aggregate views through domain-derived constraints. *DKE*, 45(2):131–153, 2003.
4. P. Ciaccia and D. Maio. Domains and active domains: What this distinction implies for the estimation of projection sizes in relational databases. *IEEE TKDE*, 7(4):641–655, 1995.
5. J. Grant and J. Minker. Numerical dependencies. In H. Gallaire, J. Minker, and J.-M. Nicolas, editors, *Advances in Database Theory*. Plenum Publ. Co., 1984.
6. S. Hartmann. On the implication problem for cardinality constraints and functional dependencies. *Ann. Math. Artif. Intell.*, 33(2-4):253–307, 2001.
7. S. Hartmann and S. Link. Numerical constraints for XML. In *Proc. WoLLIC*, pages 203–217, 2007.
8. M. Lenzerini and P. Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. In *Proc. VLDB*, pages 147–154, 1987.
9. D. Theodoratos and M. Bouzeghoub. A general framework for the view selection problem for data warehouse design and evolution. In *Proc. DOLAP 2000*, pages 1–8, Washington, DC, 2000.
10. P. Vassiliadis. Gulliver in the land of data warehousing: practical experiences and observations of a researcher. In *Proc. DMDW'00*, pages 12/1–12/16, 2000.