

Visual Representation of Database Queries using Structural Similarity

Dennis P. Groth
School of Informatics
Indiana University
Bloomington, IN 47405, USA
dgroth@indiana.edu

Abstract

It is often useful to get high-level views of datasets in order to identify areas of interest worthy of further exploration. In relational databases, the high-level view can be described using Entity-Relationship diagrams, which identify relationships between entities in the data model. Such high-level views are useful for database design activities, and can be used to generate user interfaces for constructing queries. This research introduces techniques for visualizing structural similarity of database queries. We demonstrate that individual queries can be visualized using graph visualization techniques. A distance measure based on query structure is proposed that provides database designers and administrators with a high-level perspective of relationships in the underlying data.

1 Introduction

Developing an understanding of the underlying structure of information is a key element of information visualization research. Typically, the mapping of data to the visualization space is based on the user's model of the structure of the data. For example, mappings that plot datapoints close to each other based on their similarity provide can be used to provide an overall view of the structure.[10, 11]

In this research we focus on the *questions* about data, in the form of database queries, as the information visualization problem. We defer the formal definition of query structure to a later section. However, to motivate the problem consider Figure 1. Although these two simple queries are clearly not equivalent, they are similar. In particular, they share the same input table.

This simple example illustrates the intuition behind

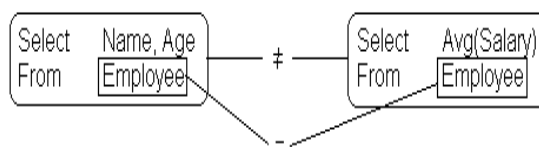


Figure 1. Two database queries with similar, but not identical structure.

our approach, in which the structure of queries (i.e. attributes, tables, join conditions) can be graphically depicted. This work represents a new approach to database query comparison that can be used for database design activities, query optimization, and improved user search interfaces. We utilize graph-based techniques for visualizing queries based on straightforward structural equivalences. In addition, we define a straightforward distance measure that allows collections of database queries to be visualized according to their structural similarity.

This work differs from traditional query optimization strategies that transform queries using algebraic identities. Instead, our intention is to provide a user with tools that will allow them to develop queries faster and easier.

The remainder of this paper is structured as follows. We outline previous work in Section 2 Section 3 describes the structural components of queries that we consider, as well as graph-based techniques for visualizing queries. In Section 4 we define a distance measure for query similarity and demonstrate its application to collections of queries. Lastly, Section 5 concludes with a description of future work.

2 Related Work

Visual representations of database structure are most frequently used for communicating the design of the database. Entity-relationship diagrams [5] and Bachman diagrams [3] are the most frequent styles used. Large databases cannot be concisely presented on a single diagram. Instead, decomposition of a data model based on functional area of the application yields more understandable information.

Our earlier work in this area includes an information theoretic approach for visualizing relationships in data.[9] We also introduced the initial concept of query structure visualization using graph visualization techniques.[8]

Clearly, work on visual query specification is relevant to our approach. The goal of visual query development, in contrast to ours, is to assist the user in specifying queries - not uncovering information about the underlying information structure. Graphical representations of queries based on data model diagrams has been shown to improve user's abilities to formulate queries.[1, 2, 4] Some of these approaches include graph-based presentations.[12, 7]

3 Query Structure

The underlying basis of our approach is the relational model.[6] For sake of clarity we identify the key constructs of the relational model that are employed. First, let $\mathbf{R} = \{R, S, T\}$ be a database schema with a finite set of relations. For each relation $R \in \mathbf{R}$, we define a finite set of attributes, $R = \{A_1, \dots, A_n\}$. For each attribute A_i we associate a set of possible values $\mathbf{dom}(A_i)$, called the domain of A_i . When considering an instance \mathbf{r} we define $\mathit{adom}(A_i)$, the active domain of A_i , to be the set of values existing in \mathbf{r} . Note that, while $\mathit{adom}(A_i)$ is finite, $\mathbf{dom}(A_i)$ may be infinite. A tuple t of \mathbf{r} is a function $\mathbf{dom}(A_i) \rightarrow \mathit{adom}(A_i)$. Define $\mathbf{dom}(R) = \mathbf{dom}(A_1) \cup \dots \cup \mathbf{dom}(A_n)$.

A query q is a function mapping from instances of relations to instances. We focus on the structural components of such queries for this research. In particular, we consider relational algebra (RA) queries involving relation symbols, attributes, and the following relational operators:

Selection: This operator, written $\sigma_\theta(R)$, selects tuples from its input that satisfies the boolean condition θ . The θ expression is typically of the form $\alpha\phi\beta$, where $\alpha, \beta \in R \cup \mathbf{dom}(R)$ and ϕ is a binary relation. For instance \mathbf{r} , $\sigma_\theta(\mathbf{r}) = \{t | t \in \mathbf{r} \wedge \theta\}$.

Projection: This operator, written $\pi_X(R)$, projects attributes of R . For instance \mathbf{r} , $\pi_X(\mathbf{r})$

$= \{t.X | t \in \mathbf{r} \wedge X \subseteq R\}$.

Cartesian Product: This operator, written $R \times S$, combines relations. For example, if $R = \{A, B, C\}$ and $R = \{D, E\}$, $R \times S = \{A, B, C, D, E\}$. For instances \mathbf{r} and \mathbf{s} , $\mathbf{r} \times \mathbf{s} = \{t | \langle t.A, t.B, t.C \rangle \in \mathbf{r} \wedge \langle t.D, t.E \rangle \in \mathbf{s}\}$.

In practice, the Structured Query Language (SQL) is used to specify queries against relational database systems. As the name implies, each query is structured with the following basic syntax:

```
select attribute_list
from table_list
where boolean condition
```

With respect to the relational algebra, the list of attributes is equivalent to the project operator, the list of tables is equivalent to the cartesian product operator, and the boolean condition is equivalent to the select operator. We assume some familiarity with the relational model and relational query languages.[6]

In order to visualize query structure we must first identify the structural elements that are contained within a query. In this section we define the elements that are contained within Project-Select-Join (PSJ) queries. In particular, we focus on relations, attributes, and join conditions.

Let \mathbf{R} be a database schema, and q be a PSJ query, we define:

Definition 3.1 Relation Schema: $Rel(q) = \{(R, c) | R \in \mathbf{R} \wedge R \text{ occurs } c \text{ times in } q\}$

It is sometimes useful to consider the set of relations without their occurrence count:

Definition 3.2 Relation: $Relation(q) = \{R | (R', c) \in Rel(q) \wedge R = R'\}$

Let q be a query and π_θ be a project expression in q , we define:

Definition 3.3 Project Schema: $Sch(q, \pi) = \{(R, A) | R \in \mathbf{R} \wedge A \in \theta\}$

Let q be a query, $\beta = \{=, \neq, <, \leq, >, \geq\}$, and $\sigma_{A'\beta A''}$ be a select expression in q , we define:

Definition 3.4 Select Schema: $Sch(q, \sigma) = \{(R, A) | R \in \mathbf{R} \wedge R \in q \wedge A \in R \wedge [A = A' \vee A = A'']\}$

Building upon the select schema, we define the join conditions in a query:

Definition 3.5 Join Condition: $Join(q) = \{ \langle R, A, \beta, S, B \rangle | (R, A) \in Sch(q, \sigma) \wedge (S, B) \in Sch(q, \sigma) \}$

In summary, we consider these elements to be the *base* structure of a query. Consider the following example:

Example 3.1

Consider the following relations:

$Emp = \{EmpID, EmpName, Salary, DeptID\}$

$Dept = \{DeptID, DeptName, ManagerID\}$

We'll use the following query: *List employee names and the names of departments that each employee works for.* The relational algebra expression for this query is:

$\pi_{EmpName, DeptName}(\sigma_{Emp.DeptID=Dept.DeptID}(Emp \times Dept))$

The SQL query is:

```
select E.EmpName, D.DeptName
from   Emp E, Dept D
where  E.DeptID = D.DeptID
```

The structural elements for this query are:

$Relation(q) = \{Emp, Dept\}$

$Sch(q, \pi) = \{(Emp, EmpName), (Dept, DeptName)\}$

$Sch(q, \sigma) = \{(Emp, DeptID), (Dept, DeptID)\}$

$Join(q) = \{(Emp, DeptID, =, Dept, DeptID)\}$

□

3.1 Structural Domination and Equivalence

Using the base structural elements defined in the previous section we define in this section the concept of structural equivalence. Structural equivalence provides the basis for comparing queries, which we use for visualization purposes.

Given two queries q_1 and q_2 we say that q_1 *alpha*-dominates q_2 , written $q_1 \succeq_{\alpha} q_2$, if q_1 dominates q_2 relative to some property α . The properties that we consider for α are the structural elements of the query, delineated below:

Relation Domination:

$Rel(q_1) \supseteq Rel(q_2) \rightarrow q_1 \succeq_{Rel} q_2$

Project Schema Domination:

$Sch(q_1, \pi) \supseteq Sch(q_2, \pi) \rightarrow q_1 \succeq_{Sch(\pi)} q_2$

Select Schema Domination:

$Sch(q_1, \sigma) \supseteq Sch(q_2, \sigma) \rightarrow q_1 \succeq_{Sch(\sigma)} q_2$

Join Domination:

$Join(q_1) \supseteq Join(q_2) \rightarrow q_1 \succeq_{Join} q_2$

Given these structural domination properties we can define the concept of structural equivalence. Again, given two queries q_1 and q_2 we say that q_1 is *alpha*-equivalent to q_2 , written $q_1 \equiv_{\alpha} q_2$, if $q_1 \succeq_{\alpha} q_2$ and $q_2 \succeq_{\alpha} q_1$, again relative to some property α . Accordingly, we define the following *alpha*-equivalent properties:

Relation Equivalent: $q_1 \equiv_{Rel} q_2$

Project Schema Equivalent: $q_1 \equiv_{Sch(\pi)} q_2$

Select Schema Equivalent: $q_1 \equiv_{Sch(\sigma)} q_2$

Join Equivalent: $q_1 \equiv_{Join} q_2$

In cases where queries may be *alpha*-equivalent in more than one way we extend *alpha* to be a logical expression

of the properties. For example, if q_1 and q_2 are Join and Select Schema equivalent we denote the equivalence by $q_1 \equiv_{Sch(\sigma), Join} q_2$.

3.2 Visualizing Query Structure as Graphs

We utilize a graph-based model for depicting query structure. A graph is represented by $G = (V, E)$. $V = \{v_1, \dots, v_n\}$ is the set of vertices. $E = \{(v_i, v_j) | v_i, v_j \in V\}$ is the set of edges in the graph. We consider two types of vertices in the graphs we draw - relations and queries. Edges are drawn between each query and the relations contained in the query. In addition, edges are drawn between the relations if a join condition exists for some query involving those relations. Queries are drawn radially according to their time of occurrence, with the 12 o'clock position being the starting point.

When more than one query is visualized the similarity between the queries is graphically derived from the graph structure in two ways. First, queries that are join equivalent are drawn as the same graph, with color being used to show an increased frequency for those queries. Second, relations are drawn only once, which allows for non-equivalent queries that share some (but not all) of the join conditions to become connected. Example 3.2 shows the graph generated for three queries. The relations are drawn to the midpoint between all queries that refer them.

Example 3.2

Let $q_1 = \sigma_{R.A=S.A}(R \times S)$, $q_2 = \sigma_{R.A=S.A \wedge R.B=T.B}(R \times S \times T)$, and $q_3 = \sigma_{R.A=S.A \wedge R.B=U.B}(R \times U)$. Figure 2 shows the graph for these queries. Note that the graph demonstrates the dominance relation $q_1 \preceq_{Join} q_2$. □

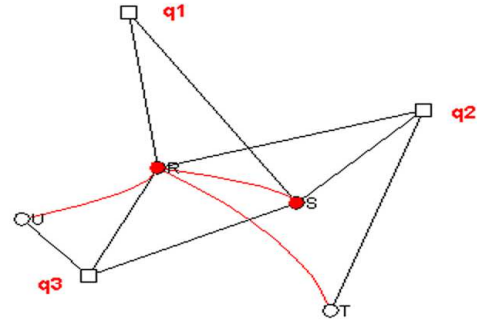


Figure 2. A graph showing three queries.

The technique can scale up to support a modest number of queries. For example, Figure 3 shows the graph for 100 randomly generated queries. We use random queries in order to evaluate the basic properties of our approach and to not be biased by any one set of real-life data. As expected, the relations used in the queries are clustered towards the center of the

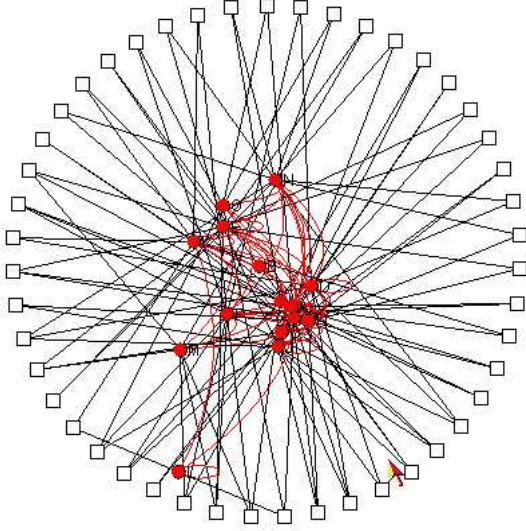


Figure 3. A graph showing 100 randomly generated queries.

graph. We would expect that, in the limit, all relations would converge to a single, central point for random data.

The graph-based visualizations we presented in this section are useful due to their one-to-one correspondence to the underlying model of the data. However, comparisons across all of the structural equivalences while maintaining the correspondence to the model is not likely to yield a meaningful global view due to the locality of the query to query connectivity. In essence, the graphs appear more useful for local, small scale comparisons of the query structure. We expand upon this technique in the next section by developing a distance measure that combines the structural equivalences.

4 Query Similarity

In this section we develop a framework for combining the structural equivalence mechanism into a distance measure that provides an overall description of similarity between queries. The framework is flexible, and allows the user to manipulate the influence of the structures according to a set of defined weights.

We utilize a straightforward distance measure based on the similarity of the sets defined by the structures defined in Section 3. Given any two sets A and B we define the similarity between the sets:

Definition 4.1 $Sim(A, B) = |A \cap B| / |A \cup B|$

We also define a corresponding dissimilarity measure:

Definition 4.2 $Dis(A, B) = 1 - Sim(A, B)$

Note that the similarity measure is not a distance metric, since $Sim(A, A) = 1$. However, the dissimilarity measure, $Dis(A, B) = 1 - Sim(A, B)$, does satisfy the following properties, and is a distance metric:

$$Dis(A, A) = 0, Dis(A, B) \geq 0$$

$$Dis(A, B) = Dis(B, A)$$

$$Dis(A, C) \leq Dis(A, B) + Dis(B, C)$$

Using this measurement we can define the similarity between queries based on their structure. The distance between two queries is defined by two vectors. The first is a vector of dissimilarities and the second is a vector of weights.

Definition 4.3 $\overline{Dis} = \langle Dis_1, \dots, Dis_k \rangle$

Each Dis refers to a specific dissimilarity measure parameterized by two particular sets. For example, given two queries q and q' , $\overline{Dis} = \langle Dis(Relation(q), Relation(q')) \rangle$ illustrates the approach.

Definition 4.4 $\overline{W} = \langle w_1, \dots, w_k \rangle$, such that $w_i \in [0 \dots 1]$ and $\sum_{i=1}^k w_i = 1$.

The distance between two queries is defined:

Definition 4.5 Given two queries q and q' ,
 $D(q, q') = \sum_{i=1}^k Dis_i \cdot w_i$

The weights are used to control the influence of the structure comparisons. For example, a user may be more interested in finding queries that are more similar in their use of relations than in the attributes being projected. Example 4.1 demonstrates the use of this framework.

Example 4.1

Consider the same relations from Example 3.1:
 $Emp = \{EmpID, EmpName, Salary, DeptID\}$
 $Dept = \{DeptId, DeptName, ManagerID\}$

We'll use the following queries:

- q_1 = List employee names and the name of their department
- q_2 = List employee names and the name of their manager
- q_3 = List employee names

The SQL syntax for these queries is:

```
select E.EmpName, D.DeptName
from Emp E, Dept D
where E.DeptID = D.DeptID
```

```
select E1.EmpName, E2.EmpName
from Emp E1, Dept D, Emp E2
where E1.DeptID = D.DeptID AND
E2.EmpId = D.ManagerID
```

```
select E.EmpName
from Emp E
```

For each pair of queries we will use the following dissimilarity measures:

$$Dis_1 = Dis(Relation(q), Relation(q'))$$

$$Dis_2 = Dis(Sch(q, \pi), Sch(q', \pi))$$

$$Dis_3 = Dis(Sch(q, \sigma), Sch(q', \sigma))$$

We summarize the dissimilarities in the following table:

	Dis_1			Dis_2			Dis_3		
	q_1	q_2	q_3	q_1	q_2	q_3	q_1	q_2	q_3
q_1	0	0	0.5	0	0.5	0.5	0	0.5	1
q_2	0	0	0.5	0.5	0	0	0.5	0	1
q_3	0.5	0.5	0	0.5	0	0	1	1	0

Using equal weights $\overline{W} = \langle \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \rangle$ we compute the distance between each query:

$$D(q_1, q_2) = 0.33$$

$$D(q_1, q_3) = 0.66$$

$$D(q_2, q_3) = 0.50$$

According to these weights q_1 and q_2 are most similar. If we consider a different set of weights $\overline{W} = \langle \frac{1}{2}, 0, \frac{1}{2} \rangle$ we get a different distance comparison:

$$D(q_1, q_2) = 0.25$$

$$D(q_1, q_3) = 0.75$$

$$D(q_2, q_3) = 0.75$$

Under this weighting scheme the similarity between q_1 and q_2 becomes even more obvious. \square

4.1 Visualizing Query Similarity

In this section we utilize the distance measurement framework described in the previous section to generate visualizations of a collection of database queries in order to get an overall feel for the similarity between queries. For data we have created a utility to create random queries. For purposes of maintaining continuity with the previous section we will use the same dissimilarity measures and both weighting schemes.

We utilize a matrix layout for representing the similarity showing the pairwise comparisons for all queries. The matrix allows us to simultaneously use two weighting schemes. A simple color scale is used to communicate similarity:

Red - 75-100% similar

Yellow - 50-75% similar

Green - 25-50% similar

Blue - 1-25% similar

Black - No similarity

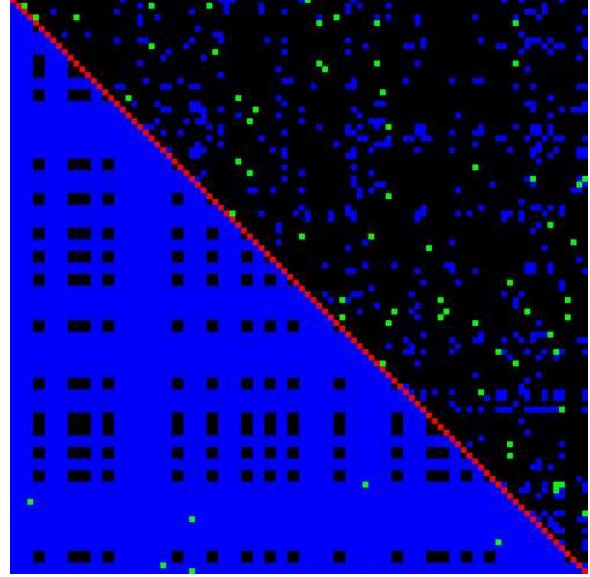


Figure 4. The matrix representation for pairwise comparisons of structural similarity for 100 randomly generated queries.

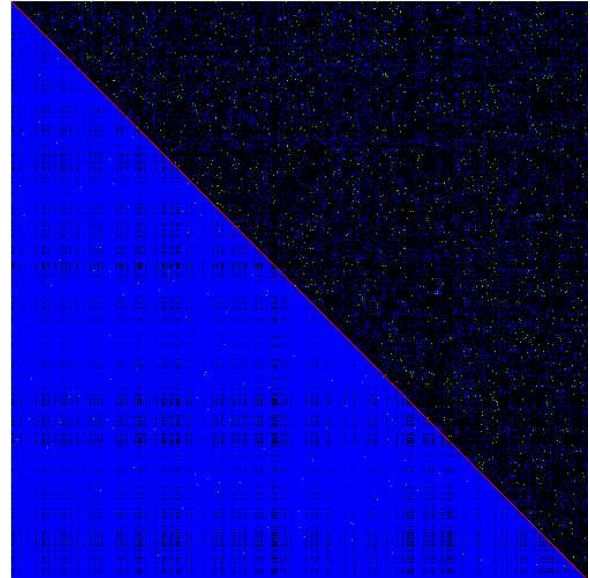


Figure 5. Visualization for 1000 random queries.

Figure 4 shows the resulting visualization for the same 100 queries using the graph technique. The lower left portion of the matrix uses equal weights, while the upper right portion uses the weighting scheme that ignores the attributes projected.

Certain patterns are worthy of further explanation. First, the equal weighting scheme does not discriminate well be-

tween the queries, as evidenced by the lower left portion of the display. The other weighting scheme shows a number of queries that have high similarity to other queries. The random nature of the data yields many queries that have average similarity when the number of relations is modest - in this case 50. When the number of queries is increased to 1000 (e.g. more diverse searches) the sparsity yields better discrimination between the queries, even for the simple weighting scheme, as shown in Figure 5.

We show a magnification of the data in Figure 6. As can be seen in this figure, there are numerous examples of similar queries.

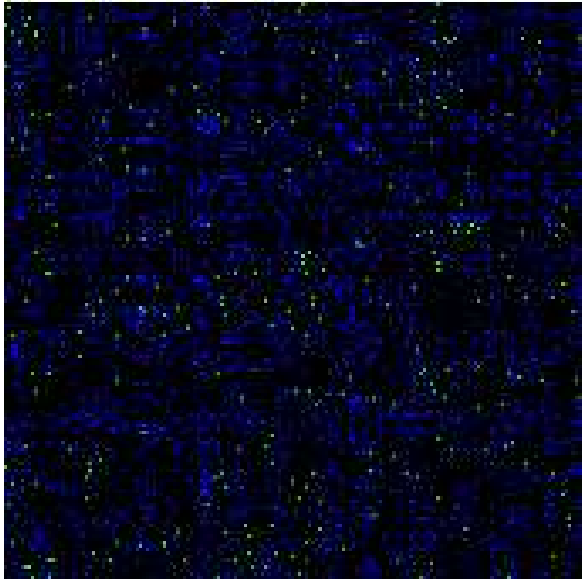


Figure 6. Upper left section of the previous visualization.

The results shown in this section indicate that a coarser similarity measure, if chosen correctly, does better at separating the datapoints than a measure that considers more comparisons. This is positive, in the sense that simple comparisons may yield an easier understanding of the phenomenon for end users.

5 Conclusion and Future Work

This paper describes an approach to similarity-based visualizations of database queries. A formalism for identifying the structural components of queries was presented. A combined distance measure framework was described that supported all pairwise comparisons of queries. Through this approach we gain insight into our data that is otherwise unavailable.

We envision several applications in which these techniques might be useful. First, as a user interface for information retrieval or data browsing activities, in which a user could get visual feedback and guidance on similar queries posed by other

users. Secondly, database administrators can use the technique for identifying frequently-used queries and then construct views to support more effective queries. Thirdly, for database designers a view of similar queries might suggest constraints or index methods to improve performance.

Acknowledgements

I would like to thank the Database Research Group at Indiana University for their feedback on early stages of this work.

References

- [1] M. Angelaccio, T. Catarci, and G. Santucci. QBD*: A fully visual query system. *Journal on Visual Languages and Computing*, 1(2):255–273, 1990.
- [2] M. Angelaccio, T. Catarci, and G. Santucci. QBD*: A graphical query language with recursion. *IEEE Transactions on Software Engineering*, 16(10):1150–1163, 1990.
- [3] C. Bachman. Data structure diagrams. *Data Base*, 1(2), March 1969.
- [4] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.
- [5] P. P. S. Chen. The entity-relationship model — toward a unified view of data. *Proceedings of the 11th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Kerr(ed), pp.173*, 1975.
- [6] E. F. Codd. A Relational Model for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.
- [7] I. F. Cruz. DOODLE: a visual language for object-oriented databases. In *SIGMOD 1992, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 71–80, 1992.
- [8] D. P. Groth. Visualizing query structure. In *Proceedings of the 18th ISCA International Conference on Computers and Their Applications CATA*, March 2003.
- [9] D. P. Groth and E. L. Robertson. An entropy-based approach for visualizing database structure. In *Proceedings of the Sixth IFIP Conference on Visual Database Systems (VDB6)*, pages 157–170, May 2002.
- [10] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, 1997.
- [11] J. D. Leeuw and G. Michailidis. Graph layout techniques and multidimensional data analysis, 2000.
- [12] J. Paredaens, J. V. den Bussche, M. Andries, M. Gemis, M. Gyssens, I. Thyssens, D. V. Gucht, V. Sarathy, and L. V. Saxton. An overview of GOOD. *SIGMOD Record*, 21(1):25–31, 1992.