Sideway Value Algebra for Object-Relational Databases

Özsoyoğlu[#], G, Al-Hamdani[#], A, Altıngövde[+], I.S, Özel, S.A, Ulusoy[+], Ö, Özsoyoğlu[#], Z.M

[#]EECS Dept, Case Western Reserve University, Cleveland, Ohio 44106
[+]Comp. Eng. Dept, Bilkent University, Ankara, Turkey
(tekin, abd, ozsoy)@eecs.cwru.edu (ismaila, selma, oulusoy)@cs.bilkent.edu.tr

Abstract

Using functions in various forms, recent database publications have assigned "scores", "preference values", and "probabilistic values" to object-relational database tuples. We generalize these functions and their evaluations as *sideway functions* and *sideway values*, respectively. Sideway values represent the advices (recommendations) of data creators or preferences of users, and are employed for the purposes of ranking query outputs and limiting output sizes during query evaluation as well as for application-dependent querying.

This paper introduces SQL extensions and a sideway value algebra (SVA) for object-relational databases. SVA operators modify and propagate sideway values of base relations in automated and generic ways. We define the SVA join, and a recursive SVA closure operator, called TClosure. Output tuples of the SVA join operator are assigned sideway values on the basis of the sideway values and similarities of joined tuples, and the operator returns the highest ranking tuples. TClosure operator recursively expands a given set of objects (as tuples) according to a given regular expression of relationship types, and derives sideway values for the set of newly reached objects.

We present evaluation algorithms for SVA join and TClosure operators, and report experimental results on the performance of the operators using the DBLP Bibliography data and synthetic data.

1 Introduction

Recent database applications on the web have necessitated the attachment of (a) functions to relations of object-relational databases, and (b) function evaluations to tuples of object-relations. Using functions in various forms, recent publications have assigned "scores" [Coh98], "preference values" [AW00, HKP01], and "probabilistic values" [BMP92] to object-relational database tuples. We refer to these values and the functions that generate them as *sideway values* and *sideway value functions*, respectively. Sideway functions and sideway values represent the recommendations of data creators. We illustrate with a web querying example.

**Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002**

Example 1.1 Consider a web resource that is modeled (using metadata extracted from it by data mining techniques) in terms of *topics,* relationships among topics (*metalinks*), and topic occurrences (i.e., *topic sources*) within information resources [A+01]. Topics have *names* (a keyword or a phrase) as well as *types* and *domains* and other attributes. Arbitrarily specified words/phrases are allowed for topic names. Topics, topic sources, and metalinks are assigned real-valued importance values (i.e., sideway values) in the range of [0, 1], which the users employ in specifying their queries. Different sets of topics, metalinks, and topic sources, together with their importance values, constitute the advices of different "experts". *Topics* and *Metalinks* relations of each expert contain the topics and metalinks defined by the expert, respectively. A certain web document (or, a part of it) is designated as a topic source for a topic. Metalinks represent relationships among topics (not sources); i.e., metalinks are "meta" relationships. E.g., Given two topic names, "query optimization" and "sort-merge join", the *Prerequisite* metalink instance "query optimization $\rightarrow^{Pre}$ sort-merge join, with importance value 0.8" states that "prerequisite to (viewing, learning, etc. sources on) query optimization is (viewing, learning, etc. sources on) sort-merge join", and this metalink instance is deemed to have the importance (sideway) value of 0.8.

Sideway functions and sideway values are selectively employed by users for two purposes:

(a) *User-guided query output ranking and size control.* Users choose in their queries which sideway values to consider and in which manner, for ranking and limiting query output sizes. And, unlike the previous approaches, the sideway values can be used to provide not only final query output size controls, but also *intermediate query output size controls*.

(b) *Querying.* Users query sideway values as if they were attribute values. E.g., for web querying, users compare in their queries the importance values of two topics.

Sideway values are not necessarily maintained as a column of base relations; sometimes, they can be defined by functions (e.g., "preference functions" [HKP01]) and attached to the base relations of the database in different forms: (a) *Open form* [Coh98, AW00] where, for each tuple, a value is specified, (i.e., sideway values are stored in a column of the base relation), (b) *Closed form* [HKP01] where each tuple's sideway value is derived from a closed function. (E.g., for a relation R with attributes X and Y, we may have f (X,Y) = a.X + b.Y where a and b are constants), (c) *Semi-closed form* where the function specifies a value for a set of tuples identified

through regular expressions. As an example, for the web resources of the National Institute for Health (US), an SVA function f() for topics is f(<TName = "*kidney complications*, TType = *, TDomain=*>) =0.9 which states that any tuple with a topic name containing the string "kidney complications", regardless of type and domain, is at the importance level of 0.9.

SVA queries specify the propagation and modifications of sideway values of input relations to query output relations in automated ways. Once the desired sideway values are selected, the query output sizes can be controlled in three ways: (i) for the final output size control, a *ranking threshold* k (i.e., output only the top-ranking k tuples [CK97, CK98, CG99, CH02]), or (ii) for intermediate and final output size controls, a *sideway value threshold* V_t (i.e., output all the tuples with sideway values above the threshold V_t), or (iii) both the ranking threshold k and the sideway value threshold V_t are specified.

Following recent works (e.g., [Coh98]), we integrate approximate (similarity) comparisons into our querying framework. In this paper, we concentrate on the similarity of textual phrases; and, to compare them, we use the TF-IDF model [Salt89] from the IR domain. We illustrate with an example.

Example 1.2 Consider the web resources DBLP Bibliography [DBLP01] and the ACM SIGMOD Anthology. Assume that information about papers (e.g., paper titles, index terms, author names, etc.) in the DBLP Bibliography and ACM SIGMOD Anthology are collected as topics, and stored into the Topics relation, as illustrated in Table 1.1(a). As an example, the tuple with tuple id T08 is the 1980 paper of E.F. Codd [Codd80]. And, the importance of the tuple with tuple id T01 is 0.9. Assume that the user asks for all database researchers with a name similar to "E. Codd", and the similarity between " Edward Codd" and "E. Codd" is judged to be 0.7. Then the tuple T01 is returned to the user with the revised importance value of 0.9*0.7=0.63.

TId	TName	TType	Tdomain	TImp
T01	E. F. Codd	researcher	database	0.9
T08	data models database management	research paper	database	0.8

(a) Topics relation

Mid	MType	AntecedentId	ConsequentId
M01	ResearchPaperOf	T01	T08

(b) Metalinks relation

Table 1.1 Topics and Metalinks relations of a database

This paper defines SQL extensions and a *Sideway Value Algebra* (SVA) for the object-relational model, which (a) allow users to selectively modify and propagate sideway values of base relations in automated and generic ways, and (b) employ them for efficient query processing. We focus on three new SVA operators here and present evaluations of two of them (see [O+02] for details):

- *SVA join with text similarity* modifies sideway values on the basis of the textual similarity of tuples. We discuss SVA join evaluation algorithms that terminate their evaluations early, by using sideway values.
- The recursive *SVA closure operator*. We describe this operator within the context of web querying, and illustrate it for querying the DBLP Bibliography and the ACM SIGMOD Anthology. The operator, called *Topic Closure*, starts with a set X of topics, a regular expression of metalink types, and a relation M representing metalinks M involving topics, expands X using the regular expression and metalink axioms, and terminates the closure computations selectively when "derived" sideway values of newly "reached" topics either get sufficiently small or are not in the top-k output tuples. That is, the derived topic importance values get smaller than a threshold V_t or are guaranteed not to produce top-k-ranking output tuples.

In section 2, we discuss SQL extensions, logical query trees, and the SVA selection, join, and topic closure operators with examples (using the queries of SQL-TC, "topic-centric" SQL [A+01]). Section 3 discusses nested-loops-based SVA join algorithms. In section 4, we discuss TClosure algorithms where the regular expression is reduced to a single metalink type. Section 5 is the experimental results.

2 SQL Extensions and Sideway Value Algebra

We illustrate SQL extensions and the SVA with examples.

Example 2.1 Consider the query **"**Based on the advice at www.expert.com/advice, find the topic ids of 20 highest topic-importance-ranked papers having index terms with similarity above 0.9 to "join algorithms". Employ a product-based importance propagation function that uses only topic importance values".

select M.ConsequentId
using advice at www.expert.com/advice **as database** DB
from DB.Topics T, DB.Metalinks M
where T.TType="Index Term" **and**
M.MType="*IndexedBy"* **and**
T.TId **is-in** M.AntecedentId **and**
T.TName $\cong_{(0.9)}$ "join algorithms"
propagate importance as product **function of** T
stop after 20 most important

where DB denotes the topic advice database at www.expert.com/advice, Topics is the topics relation in DB, with attributes TId, TName, and TType; Metalinks is the metalinks relation in DB, with attributes MType, AntecedentId, ConsequentId; and *IndexedBy* metalink type has the signature *IndexedBy:* SetOf IndexTermId → PaperId (i.e., for this metalink type, AntecedentId is IndexTermId, and ConsequentId is PaperId). The last atomic formula, $\cong_{(0.9)}$, of the where clause states that the topic (index term) name of T is similar to "join algorithms" with similarity above 0.9. We assume that the similarity between an index term name and the phrase "join algorithms" is evaluatable by information retrieval techniques, e.g., by using the vector space model and the TF-IDF weighting scheme [Salt89] to represent the topic names (see section 3.1). The "propagate importance" clause specifies the sideway value function for output tuples. In this example, the clause states that

the importance values of output tuples are computed from the importance values of the base relation T using a "product" function revised with similarities. Table 2.1 lists possible propagate-importance clauses and the corresponding SVA output function specifications. Other output sideway value functions such as numeric maximum, numeric median, etc., are possible. We assume that the possible choices for output sideway value functions are small and known a priori. Note that this query uses only the topic importance values, but not metalink importance values, as specified by the "propagate importance" clause. And, in this query and others, there is no direct information resource access (to simplify the presentation and examples); i.e., here, only topic (paper) ids, but not the paper sources, are output.

SQL clause	Effect
product	$fl_{in} * fr_{in} * sim()$
numeric average	$Ave(fl_{in}, fr_{in}) * sim()$
geometric average	$Sqrt(fl_{in} * fr_{in}) * sim()$

Table 2.1 Examples of output sideway value function f_{out} for a join with (i) two input relations R and S having the sideway value functions fl_{in} and fr_{in} , respectively, and (ii) a text similarity join condition between R and S with similarity function sim().

The sideway value function of base relations is denoted by f_{in}, and has the normalized range [0, 1]. During SVA operations, we materialize the output sideway function of an operator, i.e., convert it into the open-form from the other two forms, and keep it as a (new) column while processing queries.

2.1 SQL Extensions

We define a generic, user-guided, and system-enforced mechanism for users to control query output sizes, by using sideway values. In Example 2.1, we presented three clauses for SQL, namely,

(a) The clause ***using advice at*** *www.xx* ***as database*** which specifies the database,

(b) The clause ***propagate importance as*** *xx* ***function of*** which specifies a generic formula for propagating sideway values of output tuples, and

(c) the clause ***stop after k*** which specifies the ranking threshold, or ***stop with threshold*** V_t which specifies the sideway value threshold, or ***stop after k and with threshold*** V_t, which specifies both types of stopping conditions.

The clause in (a) is self explanatory.

A. Propagating Sideway Values. The clause in (b) is used to propagate importance values of input relations to the output tuples. The general form is:

Propagate importance as <type> **function of** <arg list>

where type is one the specific function types (e.g., min, max, product, arithmetic average (avg), geometric average (gavg), etc.), and argument list is a sublist of relations listed in the *From* clause of the SQL query.

As an example, in the SQL query:

Select … From R, S, T, V …
Where …Propagate importance as Product function of (R, S)

the *propagate-importance* clause states that, when propagating sideway values of relations R and S, the system will use a product function, and the tuple sideway values of T and V are suppressed (not used). That is, consider an intermediate relation e_i, obtained during the execution of the above query, which is an output of a relational algebra expression E_i. Then, e_i will have tuple sideway values iff E_i has at least one of R or S as its input arguments. Next, we define the execution semantics of the *propagate-importance* clause for a binary operator (such as join or Cartesian product—we omit unary operators to save space) with operands E_1 and E_2, where E_1, E_2 denote intermediate relations (resulting from algebra expressions with zero or more operators):

(i) E_1 and E_2 are either R or S, respectively, or each has at least one of R or S as an argument: If E_i is R (or S) then the tuple sideway values of E_i are the same as R; otherwise they are computed recursively by considering the operators in E_1 and E_2. Then, the tuple sideway values for the output of the binary operator are computed as the product of the tuple sideway values for E_1 and E_2.

(ii) Neither E_1 nor E_2 is R (or S), and neither has at least one of R or S as an argument: In this case, neither of the operands E_1 and E_2 have tuple sideway values (i.e., they are suppressed). Hence, output tuples do not have tuple sideway values either.

(iii) Only one of E_1 or E_2 is R or S, or has at least one of R or S as an operand: Let E_1 be the operand involving R or S. Then E_1 has tuple sideway values, and E_2 doesn't. Then, the tuple sideway value of a tuple t of E_1 is simply passed to the output tuples of the binary operator that t contributes.

Thus, we have presented unambiguously a single-clause-based (i.e., *propagate-importance)* sideway value propagation technique, and its effects on the construction of the logical query trees. Having propagated sideway values to the nodes of logical query trees in turn allows us to introduce size-controls at the intermediate nodes of logical trees, and perform query optimization.

B. Propagating Stopping Conditions. We extend SQL with stopping conditions, whose utility is to significantly lower the query processing times. Ideally, we want stopping conditions to be propagatable to the intermediate nodes of the logical query tree (i.e., to the algebra operators), in which case the query processing times of all SVA operators are drastically reduced. Unfortunately, the clause *stop after k*, which specifies the size of the final query Q output (i.e., the top-k query), is difficult to propagate to intermediate logical query tree nodes of Q during query processing [CK97, CK98]. In contrast, the clause *stop with threshold* V_t (i.e., output all tuples with sideway values above V_t) *always propagates to all logical query tree nodes of query Q* (as a single operator stopping condition) when the sideway value propagation function returns a value less than its input sideway values. Note that the product function used in this paper satisfies this

property. As an example, in the logical query tree of example 2.4, the stopping condition with threshold V_t=0.95 is propagated into the two selection and one join in the logical tree, as shown in figure 2.3.

Another approach to specifying stopping conditions that propagate to logical query tree nodes naturally is to attach stopping conditions to the where clause predicates of SQL. As an example, the users may request that (a) the number of top-sideway-valued tuples (objects) that satisfy a certain predicate p (e.g., PersonCity = "Cleveland" or PersonAddress = "*Cleveland*") is k, or (b) all of the tuples that satisfy a given predicate p have sideway values above a threshold V_t. In such cases, the predicate-attached stopping conditions naturally propagate to the logical query tree nodes where the predicates appear.

Once the stopping conditions are propagated to the logical query tree nodes (i.e., individual SVA operators), what remains is the evaluation of individual SVA operators, which we discuss in sections 3 and 4. Next, we illustrate SQL extensions, logical query trees, and the SVA operators with examples.

In the logical query tree examples discussed next, we use the following notation: Operators with superscript * are SVA operators. Operators without superscript * are normal relational algebra (RA) operators. A unary RA operator without * in its superscript simply carries (into its output tuples) the sideway values of its only operand relation. A binary operator RA without a superscript * may carry (into its output tuples) sideway values of either its left (hand side) relation or its right (hand side) relation, indicated by superscript L or R, respectively.

For each RA operator, there is an SVA counterpart extended with an output sideway value function f_{out} and the output threshold β, which is either the integer-valued ranking threshold, or the real-valued sideway value threshold V_t in the range [0, 1].

2.2 SVA Selection: $\sigma^*_{C, fout, \beta}$ (R)

The selection operator σ^* takes as input a relation R (with a sideway value function f_{in} in open, closed, or semiclosed form), a selection condition C, an output sideway value propagation function f_{out}, and the output threshold β where β is either a positive integer k as the ranking threshold, or the real-valued sideway value threshold V_t in the range [0, 1], or the two-tuple (k, V_t). The operator σ^* returns, in decreasing order of output sideway values, either (i) top k f_{out}-ranking output tuples that satisfy the selection condition C (when β is k), or (ii) all tuples of R with an f_{out}-sideway value greater than V_t and satisfy the selection condition C (when β is V_t), or (iii) top k f_{out}-ranking output tuples that satisfy the selection condition C and with an f_{out}-sideway value greater than V_t (when β is the two-tuple (k, V_t)).

Example 2.2 The logical query tree of example 2.1 is shown in figure 2.1. The notation $\cong_{(t)}$ denotes text similarity-based equality formula with the similarity acceptance threshold of t; i.e., $\cong_{(t)}$ returns True/False. The function sim() computes the text similarity of two strings, and returns a value in the range [0, 1]. Here, sim() is used to modify the importance values of output tuples according to their Tname similarity to "join algorithms", as illustrated in Table 2.1. Note that, as specified in the propagate importance clause, this query does not use metalink importance values. In this example, the SVA selection operator has the β value of 0; therefore this operator returns all qualified output tuples.

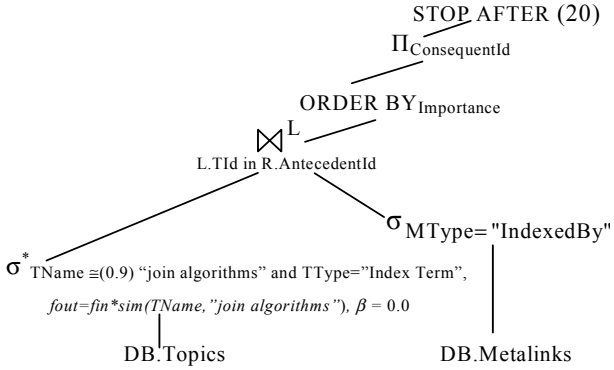

Figure 2.1 Logical Query Tree of Example 2.1.

2.3 SVA Join: (L) $\overset{*}{\bowtie}_{A\ \theta\ B,\ fout,\ \beta}$ (R)

The join operator takes as input two relations L and R with sideway value functions fl_{in} and fr_{in} respectively, a join condition θ on attributes A and B of relations L and R, respectively, a sideway value propagation function f_{out} for the output tuples, and an output threshold β. The join operator then produces joined tuples of L and R with sideway values of output tuples computed as specified by f_{out}, and satisfying the output threshold β.

We illustrate the SVA join operator with two examples.

Example 2.3 Using the advice at www.expert.com/advice, find five researchers who (a) published papers with index terms having similarity to "join algorithms" above 0.9, and (b) have the highest importance values computed using the geometric averages of all involved importance values.

select M.ConsequentId
using advice at www.expert.com/advice **as database** DB
from DB.Topics T, DB.Metalinks M
where TType="Index Term" **and**
M.MType=**"***ResearchTopicOf"***and**
T.Td **is-in** M.AntecedentId **and**
T.TName $\cong_{(0.9)}$ "join algorithms"
propagate importance as gmtrc-average **function of** T, M
stop after 5 most important

The logical query tree of example 2.3 is shown in figure 2.2. ConsID and AntId are consequent and antecedent attributes of the Metalinks relation. We assume that *ResearchTopicOf* is a metalink type that specifies the relationship between index terms of papers and the authors of these papers (obtained by mining the ACM Anthology digital library). The signature of the metalink type is *ResearchTopicOf: SetOf IndexTermId* → *ResearcherId*. Due to the clause "propagate importance", this query chooses researchers on the basis of a geometric average of the importance values of researchers (topics) and their

ResearchTopicOf type metalinks. The SVA join in this case is exact (i.e., no similarity computations are involved).

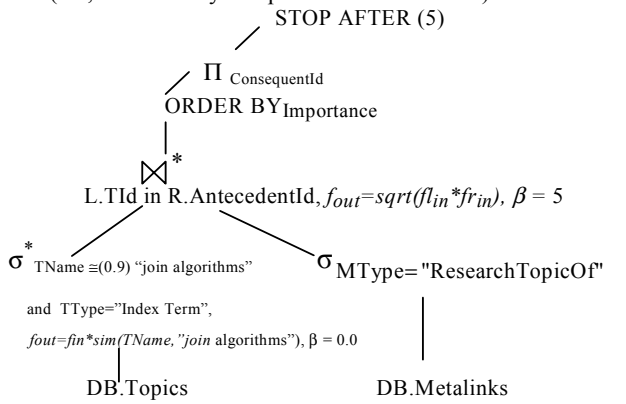

Figure 2.2 Logical Query Tree of Example 2.3.

Example 2.4. Using the advices at www.expert1.com/advice (DB1) and at www.expert2.com/advice (DB2), find the titles and URLs of pairs of papers advised by DB1 and DB2 such that (a) the derived importance value of the paper pair (as defined by the max function) is of importance above 0.95 and in the top-5-rank, and (b) the topic name of the paper from DB1 has a similarity of 0.98 or above to the topic name of the paper from DB2. Employ a max-based importance propagation function that uses all of the involved importance values.

select T1.TName, T1.URL, T2.TName, T2.URL
using advice at www.expert1.com/advice **as database** DB1**,** www.expert2.com/advice **as database** DB2
from DB1.Topics T1, DB2.Topics T2
where T1.TType="*Paper"* **and** T2.TType="*Paper"* **and** T1.TName $\cong_{(0.98)}$T2.TName
propagate importance as max **function of** T1, T2
stop after 5 most important and with threshold 0.95

The logical query tree of example 2.4 is shown in figure 2.3. Note that the SVA join is similarity-based, and the sideway value threshold of 0.95 is propagated to all of the three operators, namely, the two SVA selections and one SVA join.

2.4 SVA Topic Closure

Next we define a recursive operator that takes into account the sideway values of its input tuples. This operator, called the "topic closure" operator, is motivated by web querying; thus, we describe this operator, within the context of topics and metalinks, not generically.

Notation: TClosure$^{*}_{\mathcal{R},\ Metalinks,\ FPath,\ FPathMerge,\ \beta}$(X)

The *topic closure operator* computes the topic closure X^+ of a set X of topics with respect to a regular expression $\mathcal{R}$ of metalink types (and, thus, wrpt the set of axioms characterizing the metalink types in $\mathcal{R}$), a metalink instance relation Metalinks (containing all metalink instances). More specifically, the topic closure operator takes as input (1) two relations, namely, the relation X of topics with a sideway value function f_X and the relation Metalinks with a sideway value function f_M, and (2) four parameters: (a) the regular expression $\mathcal{R}$, (b) a path-based importance value propagation function *FPath* that specifies how to compute the importance values of newly reached topics with respect to a single path, (c) the function *FPathMerge* that specifies how to merge the importance values of a given topic obtained through different paths, and (d) the output threshold β. It then computes the closure X^+ of X with respect to < $\mathcal{R}$, Metalinks, f_X, f_M, *FPath*, *FPathMerge,* β > where each new topic in the closure is represented as an output tuple, and has a *derived* importance value satisfying the output threshold β.

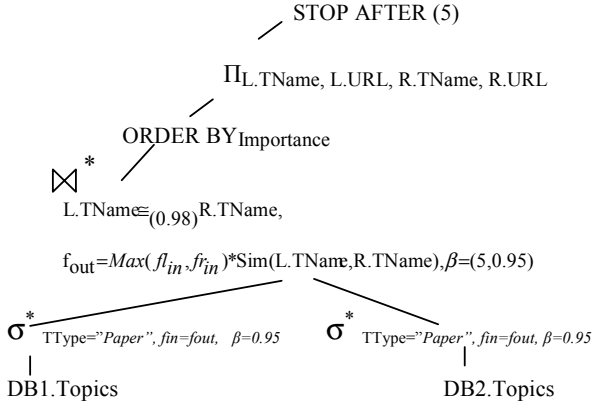

Figure 2.3 Logical Query Tree of Example 2.4.

$\mathcal{R}$ is a regular expression of metalink types. E.g., the regular expression *PrerequisitePapers*IndexedTerms* finds the index terms in *all* the prerequisite papers (of a given paper topic). Next we illustrate the notion of paths that satisfy $\mathcal{R}$ with an example.

Example 2.5 Let A, B, C, D, and T denote single topics. The metalinks A$\rightarrow^{RelatedTo}$ B, B$\rightarrow^{RelatedTo}$ C and C$\rightarrow^{RelatedTo}$ T constitute a path P = {A, M_1, B, M_2, C, M_3, T} where all nodes are single topics and all metalinks M_1, M_2, and M_3 have the type *RelatedTo* (i.e. $\mathcal{R}$ = *RelatedTo)*. As another example, metalinks AB $\rightarrow^{Pre}$ C, C $\rightarrow^{Pre}$ DE, and DE$\rightarrow^{Pre}$ T form a path P={AB, M_1, C, M_2, DE, M_3, T} that starts with a set of topics AB, followed by a single topic C, then a set of topics DE, and ends with a single topic T. The path P satisfies $\mathcal{R}$ = *Prerequisite* since all of its metalinks M_1, M_2, and M_3 are of type *Prerequisite*.

FPath is the derived importance value propagation function with respect to a single path. In this paper, we use the product function as *FPath*. As an example, assume that the topic t is reached from a topic x in X using a path P = <x m1 a m2 t> where a is a topic with importance value v_a, m1 and m2 are metalinks with importance values v_{m1} and v_{m2}, and the metalink types of m1 and m2 satisfy the regular expression $\mathcal{R}$. Assume *FPath* is Product. Then, the derived importance value of t with respect to P, denoted by $Imp_d(t, P, \mathcal{R})$, is computed as the product of all the importance values in P that satisfies $\mathcal{R}$, i.e., $v_x*v_{m1}*v_a*v_{m2}*v_t$, where v_a and v_t are the importance values of x and t, respectively.

The intuition for the semantics of derived topic importance values is as follows: assume topic t is reached through path P. The derived importance value of t in the closure should be a function of the length and the type of path P, and less than or equal to the importance value of t.

As the length of P increases, the derived importance value of t should decrease because t is farther away from (and is *less* related to) the topics in X, the original set of topics listed by the user. Thus, $Imp_d(t, P, \mathcal{R})$ with respect to path P should be a monotonically decreasing function of the length of path P (i.e., *path-monotone*).

FPathMerge is one of Product, NumAve, Min, Max, etc., specifying how to compute the derived importance value $Imp_d(t, \mathcal{R})$ of topic t in X^+ in terms of the $Imp_d(t, P, \mathcal{R})$ values obtained with respect to each path P.

We now specify the execution semantics of TClosure* procedurally as follows:

(a) Locate metalink paths P from a topic in X to a topic t not in X, where P "satisfies" the regular expression $\mathcal{R}$, and compute $Imp_d(t, P, \mathcal{R})$ values.

(b) Compute the derived importance value of t as $sv=Imp_d(t, \mathcal{R})$, and add the new topic t to the closure of X if sv satisfies the sideway value threshold β. That is, If β is a positive integer k as the ranking threshold, then sv satisfies β if sv is among the top-k output sideway values. If β is the real-valued sideway value threshold V_t in [0, 1], then sv satisfies β if $sv > V_t$.

Importance-value driven topic closure operator has similarities to the recently introduced focused crawling techniques on the web [DLGG89] in that the search space of topic sources are crawled in a focused manner.

Example 2.6. Using the advice at www.expert.com/advice, find the titles and URLs of five highest importance-valued papers such that the selected papers (a) are either papers with titles similar to "Advances in Spatial Databases" with a similarity above 0.85, or their prerequisites (recursively), and (b) have the highest importance values computed using a product function as *FPath,* and min function as *FPathMerge*.

select T2.TName, T2.URL
using advice as www.expert.com/advice **as database** DB
from DB.Topics T1, DB.Topics T2, DB.Metalinks M
where T1.TName$\cong_{(0.85)}$"Advances in Spatial databases"
and M.MType="*PrerequisitePapers"* **and**
T2.TId =any(*PrerequisitePapers*,Product, Min,* T1.TId, M)
propagate importance as product **function of** T1, T2, M
stop after 5 most important

The logical query tree for example 2.6 is given in Figure 2.4.

2.4.1 Metalink Axioms

Some metalink types have associated axioms; for example, *RelatedTo* is both transitive and reflexive. *IsIn* is transitive, but not reflexive; *SubTopicOf* is transitive. Therefore, when a user asks for topics that are *RelatedTo* topics in X, we need the topic closure X^+ of X with respect to the metalink type *RelatedTo,* which is formed of all topics that are logically implied by the set X.

Computing topic closures requires a sound and complete set of axioms for metalink types, and a polynomial-time algorithm that computes the topic closure using the axioms. Consider the *Pre(requisite)* metalink type. In [O+01], we gave the following axiomatization for the *Pre* metalink type.

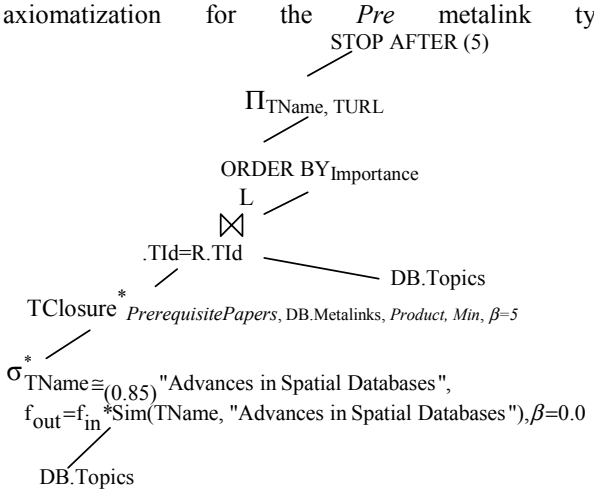

Figure 2.4 Logical Query Tree of Example 2.6.

Case 1. Prerequisite metalinks are not left-hand-side (LHS) decomposable (that is, A, $B\rightarrow^{Pre}$ C is not equivalent to the metalink $A\rightarrow^{Pre}$ C and the metalink $B\rightarrow^{Pre}$ C), and are allowed to be cyclic. *Axioms*: Let X, Y, and Z denote sets of topics.

- Subset-Reflexivity. If $Y \subseteq X$ then $X\rightarrow^{Pre} Y$.
- Augmentation. If $X\rightarrow^{Pre} Y$ then $XZ \rightarrow^{Pre} YZ$ for any Z.
- Transitivity. If $X\rightarrow^{Pre} Y$ and $Y\rightarrow^{Pre} Z$ then $X\rightarrow^{Pre} Z$

These are the so-called Armstrong's axioms [RG00].

Case 2. Prerequisite metalinks are not LHS-decomposable and are acyclic.

Axioms: Let X, Y, Z and W denote sets of topics.

- Pseudo-transitivity. If $X\rightarrow^{Pre} Y$ and $WY\rightarrow^{Pre} Z$ then $WX\rightarrow^{Pre} Z$.
- Split/join. If $X\rightarrow^{Pre} YZ$ then $X\rightarrow^{Pre} Y$ and $X\rightarrow^{Pre} Z$, and vice-versa.

[O+01] proves that above axioms are sound and complete.

Case 3. Prerequisite metalinks are LHS-decomposable. We first decompose the LHS of all metalinks so that all metalinks have a single topic in the left and the right hand sides. And, then the only axiom is

- Transitivity. If $A\rightarrow^{Pre} B$ and $B\rightarrow^{Pre} C$ then $A\rightarrow^{Pre} C$ where A, B, C are topics.

In all three cases, the topic closure X^+ of a set X of topics wrpt the type *Prerequisite* can be found by using an O(n.l) topic closure algorithm where n is the number of prerequisite metalinks, and l is the length of the encoding for a prerequisite metalink [O+01]. For all metalink types, we assume the existence of sound and complete axioms.

3 Evaluating the SVA Join

3.1 Text Similarity Metrics

For those functions that require the similarity comparison ≅, we assume that a vector space based similarity model is employed [Coh98]. The vector space model first creates a vocabulary (W) of all words (i.e.,

terms) included in the document collections, and then represents each document with a vector v of |W| terms. The vector entries are real numbers representing term weights. Let v^t denote the vector v element for term t. We use the weighting scheme TF-IDF, which assigns a zero weight for those terms that do not appear in the document, and computes the weights of the other terms using the formula $v^t = (\log (TF_{v,t}) + 1). \text{Log}(IDF_t)$, where $TF_{v,t}$ (term frequency) is the number of occurrences of term t in the document represented by v, and IDF_t is the inverse document frequency that is defined as the ratio of the number of all documents to the number of documents including t. We focus on attributes with short phrases such as topic names. The TF-IDF values are normalized and the similarity of two documents represented with vectors v and u is the cosine of the angle between them, which is defined as Cosine $(u, v) = \sum_{t \text{ in } W} v^t * u^t$

We assume that term vectors that correspond to string-based attributes of tuples, as well as the vocabulary, are computed a priori. In this section, we assume that vocabulary is small enough to fit in the main memory, whereas all other input and output relations may be arbitrarily large.

Since pipelining is preferable for threshold-based query processing algorithms [RG00], and the nested-loop join algorithm does not disrupt pipelining [Graef93], next, we discuss nested loops-based SVA join algorithms. Moreover, the nested-loop join is appropriate with arbitrary join conditions.

3.2 Nested-Loops-Based Sideway-Value-Threshold Join algorithms

We now discuss SVA join algorithms that return joined tuples with derived values above a specified sideway value threshold. We sketch two algorithms for join conditions specifying (i) an arbitrary (user-defined) predicate θ over the join attributes, or (ii) an approximate match in terms of the similarity of the join attributes.

Definition. *Monotone f_{out}.* Let sv_t denote the sideway value of tuple t. Given relations R and S with tuples r and s respectively, let $f_{out}(r, s)$ denote the sideway value of the joined output tuple r.s. Then, $\forall r_1, r_2 \in R$ and $\forall s_1, s_2 \in S$, if $f_{out}(r_1, s_1) \le f_{out}(r_2, s_2)$ whenever $sv_{r1} \le sv_{r2}$ and $sv_{s1} \le sv_{s2}$, the function f_{out} is said to be *monotone* with respect to input sideway values of R and S.

Functions product, numeric average and geometric average are monotone with respect to their input sideway values.

Algorithm NLoop_{SVT}

Input : Sorted Relations R and S wrpt sideway values; $f_{out}()$ function; join condition r.A θ s.B; sideway value threshold V_t

Output: {r.s | r∈R **and** s∈S **and** $f_{out}(r, s) \ge V_t$ **and** r.A θ s.B}

```
{i := 1;
    while (f_out(r_i, s_1) ≥ V_t and i ≤ |R|)
    { j := 1;
      while (f_out(r_i, s_j) ≥ V_t and j ≤ |S|)
        { if r_i.A θ s_j.B then add r_i.s_j into the output;
          j++ }   i++ } }
```

Figure 3.1 NLoop_{SVT} algorithm

Given a query involving a join with a monotone f_{out} function, we improve the nested-loop join algorithm by enforcing new stopping conditions while processing the inner and outer loops, as shown in the NLoop_{SVT} algorithm in Figure 3.1.

In the NLoop_{SVT} algorithm, the inner loop exits whenever the $f_{out}()$ value of the output tuple r.s is below the threshold V_t, where r is in R and s is in S. Similarly, the outer loop exits at the i^{th} iteration whenever the $f_{out}()$ value of the output tuple $r_i.s_1$ is below the threshold V_t, where r_i is in R and s_1 is the first tuple in S.

In an ordinary block-nested loops (BNL) join [RG00], assuming that the size of R is M pages with p tuples per page, the size of S is N pages with q tuples per page, and the memory has B+2 buffer pages, we can read B pages of the outer relation R, and scan the inner relation S by using one of the remaining two buffer pages, leaving the last page to collect the output tuples. In this case, the disk access cost of the BNL algorithm is M + (M*N/B) [RG00]. In the worst case, the disk access cost of the NLoop_{SVT} algorithm is the same with the disk access cost of the BNL algorithm. However, in the expected case, the disk access cost of the NLoop_{SVT} algorithm will be reduced depending on how large V_t is. Assume that we revise the allocation of buffer pages as B/2 pages each to the relations R and S; the sideway values in R and S are uniformly distributed; and $f_{out}()$ is the product function, which is monotone. Thus, the tuples in the first B/2 blocks of R have sideway values in the range of [(1 − B/2M), 1]. Similarly, the tuples in the first B/2 blocks of S have sideway values in the range of [(1 − B/2N),1]. During the first outer loop iteration, the inner loop will terminate in the j^{th} iteration when the lowest expected sideway value of a join tuple in the buffer is equal to (or ε less than) the sideway value threshold V_t. That is,

$$(1 - B/2M) * (1 - j*B/2N) = V_t$$

Rearranging the above equality, we have

$$j = \frac{N}{B/2} * (1 - V_t) - \left(\frac{2N - B}{2M}\right)$$

Assuming N>>B and M ≈ N, the above equality reduces to $j=(N/(B/2))*(1-V_t)$. That is, in the expected case, for V_t=0.9, the inner loop terminates with 10% of the disk block accesses from S. Since R sideway values are sorted and decreasing in value, for any outerloop tuple of R, S will always be accessed at most for the first $b_S=(N/(B/2))*(1-V_t)$ blocks. And, since the above computations are symmetric for R and S, in the expected case, NLoops_{SVT} algorithm will terminate with $b_R=(M/(B/2))*(1-V_t)$ disk block accesses from R as well. Thus, the expected number E of disk accesses is

$E = (B/2)* b_S + (B/2)(b_S - (B/2)) + (B/2)(b_S - 2(B/2))+ \ldots + (B/2)(b_S - (b_R - 1)* (B/2))$

Assuming $b_S = b_R = b$, we have

$$E = (B/2)*b^2 - (B/2)^2*((b^2 - b)/2)$$

This, as shown in the experimental results section, is significantly less than the cost of the BNL algorithm.

When the join condition specifies an approximate matching (based on the similarity of the text-valued join attributes being above a given threshold t_{sim}), we cannot directly make use of the similarity function sim(r, s), as it is not monotone, and thus makes f_{out} non-monotone. However, we can still use the $NLoops_{SVT}$ algorithm of Figure 3.1 with provisions: (a) the functions f_{out} (r_i, s_1) and f_{out} (r_i, s_j) in the outer and the inner while loop conditions are replaced by $sv_{ri} * sv_{s1}$ and $sv_{ri} * sv_{sj}$, respectively, where sv_{ri}, sv_{s1} and sv_{sj} are the sideway values of tuples r_i, s_1 and s_j. (b) In the inner while loop, we check if f_{out} (r_i, s_j)= $sv_{ri}*sv_{sj}$.sim(r_i.A, s_j .B) $\geq V_t$ and sim(r_i.A, s_j .B) $\geq t_{sim}$ where A in R and B is S are the join attributes. If so, the tuple $r_i.s_j$ is output.

Note that, so far, the join algorithm has not employed the similarity function in improving its running time. We now summarize an algorithm that uses the vector-space model and the similarity function in improving the efficiency of the join algorithm.

Remark 1. Let $\mathbf{u_r} = <u_1\ u_2\ \ldots\ u_x>$ be the term vector corresponding to the join attribute A of tuple r of R, where u_i represents the weight of the term i in A. Assume that the *filter vector* $\mathbf{f_S} = <w_1\ ..\ w_x>$ is created such that each value w_i is the max weight of the corresponding term i among all vectors of S. Then, if Cosine ($\mathbf{u_r}$, $\mathbf{f_S}$) $< V_t$ then r can not be similar to any tuple s in S with similarity above V_t.

In this paper, the value Cosine ($\mathbf{u_r}$, $\mathbf{f_S}$) is called as the *maximal similarity* of a record r in R to any other record s in S. The maximum value of a term for a given relation is determined while creating the vectors for the tuples, and the filter vector for each relation may be formed as a one-time cost. In figure 3.2, we summarize the $NLoop_{Sim\text{-}SVT}$ algorithm which makes use of the sorted order of relations R and S by $sv_r *$ Cosine ($\mathbf{u_r}$, $\mathbf{f_S}$), and sv_s, respectively (also one-time costs). Note that, with both while loop conditions, *false drops* are possible; that is, a tuple r in R and a tuple s in S may satisfy the while loop conditions, only to be eliminated from the output in the if statement within the inner while loop (the if condition tests the values of the actual f_{out}() and sim() functions). On the other hand, while loop conditions do not allow *false dismissals*; that is, a join tuple that is in the output will be added into the output.

Algorithm $NLoop_{Sim\text{-}SVT}$
Input : Relations R and S;
text-valued join attributes r.A and s.B; Buffers B_S and B_R;
sim function sim()=Cosine(); sim threshold t_{sim}

Output: {r.s | r∈R **and** s∈S **and** $f_{out}(r, s) \geq V_t$ **and** Cosine($\mathbf{u_r}$, $\mathbf{u_S}$) > t_{sim} }

1. Sort R by $sv_r *$ Cosine($\mathbf{u_r}$, $\mathbf{f_S}$); Sort S by sv_s;
2. Read tuples from the top of R into a block B_R where, for each r_i in B_R, $sv_{ri} * sv_{s1} *$ Cosine($\mathbf{u_{ri}}$, $\mathbf{f_S}$) $\geq V_t$;
3. Repetitively, read tuples from the top of S into a block B_S, where, for each s_j in B_S, $sv_{r1} * sv_{sj} *$ Cosine($\mathbf{u_{r1}}$, $\mathbf{f_S}$) $\geq V_t$, and compare and join tuples in B_R and B_S:
   for each r ∈ B_R **do for each** s ∈ B_S **do**
   if ($sv_r * sv_s *$ Cosine ($\mathbf{u_r}$, $\mathbf{u_s}$)≥ V_t **and** Cosine ($\mathbf{u_r}$, $\mathbf{u_s}$)≥ t_{sim})
   then add r.s into the output;
4. Repeat 2-3 until $sv_{ri} * sv_{s1} *$ Cosine($\mathbf{u_{ri}}$, $\mathbf{f_S}$) $< V_t$

Figure 3.2 $NLoop_{Sim\text{-}SVT}$ Algorithm

3.3 Nested-Loops-Based Ranking-Threshold (Top-K) Join Algorithms

It is easy to give an SVA join algorithm with top-k output sideway values. Assume that (i) input relations are sorted with respect to sideway values, and (ii) the f_{out}() function is monotone. The algorithm $NLoops_{Top\text{-}k}$ begins in a nested loop like manner, and computes the first k (but not top k yet) joined output tuples, referred to as the "Top-k-Set". And, the sideway value of the k^{th} joined tuple becomes the lower bound (minSV); i.e., no tuple with a sideway value below this lower bound can be in the top-k output. The algorithm proceeds in a nested-loops manner, and updates the lower bound and the current Top-k-Set whenever it computes a join output with a new sideway value larger than the minimum sideway value of Top-k-Set.

Similar to the algorithm $NLoop_{Sim\text{-}SVT,}$ the algorithm $NLoop_{Top\text{-}k}$ can be revised for a ranking-threshold algorithm $NLoop_{Sim\text{-}Top\text{-}k}$ with approximate matching conditions, which, to save space, is not presented here.

4 Evaluating the Topic Closure

We now summarize TClosure algorithms to compute the topic closure X^+ for the simplest case where the regular expression $\mathcal{R}$ is a single metalink type M (For full algorithms, see [O+02]). Each metalink V $\rightarrow^M$ Tid is represented by a tuple in the Metalinks table, where V is a set of topic identifiers and Tid is a topic identifier. If a metalink type is LHS-decomposable then each metalink with V in the left-hand-side is decomposed into multiple metalinks with a single topic in the left-hand-side.

4.1 Sideway-Value-Threshold-based Topic Closure

We create an index MIndex for all metalink instances; and the TClosure algorithm uses only MIndex to find the closure of a given set of topics. We assume that all metalinks are right-hand-side decomposed.

The index MIndex has five attributes: MType, Tid1, Imp(Tid1), ParentList, and ChildList. The MType attribute specifies a metalink type. The Tid1 attribute contains the topic identifier of the topic from which the metalink originates, and the attribute Imp(Tid1) is the importance value of the topic Tid1. The attribute ParentList is a list of topic identifiers of topics from which there are metalinks of type MType to the topic Tid1. The attribute ChildList is a list of triplets <Tid2, Imp(Tid2),Imp(Mid)> where the triplet <Tid2, Imp(Tid2), Imp(Mid)> represents a metalink that has Mid as its metalink identifier, the topic with Tid1 as its antecedent

node, the topic with Tid2 as its consequent node, the type MType as its metalink type, Imp(Tid2) as the importance value of the topic with Tid2, and Imp(Mid) as the importance value of the metalink.

The key for MIndex is the two attributes metalink type MType and topic identifier Tid1. Therefore, the MIndex entries with the key (MType, Tid1) contains all metalinks of type MType that have the topic with Tid1 as its antecedent. The entries of MIndex are sorted by (MType, Tid1) so that the metalink of the same type are together within the index. Table 4.1 shows the initial index MIndex for the Metalinks relation graphically illustrated in Figure 4.1.

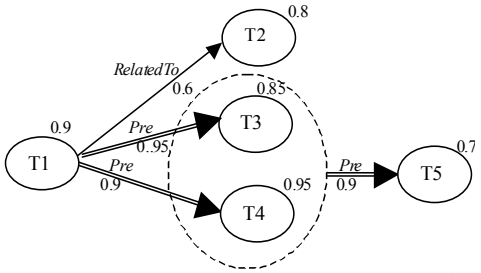

Figure 4.1. A graphical representation for the metalinks in Example 4.1.

While creating MIndex, if there are metalinks which are not LHS-decomposable then we create a second index H(yper)Index, to maintain all nodes that are not decomposable; and the topic closure algorithm uses HIndex to compute the closure of a given set of topics. The HIndex table has two attributes Tid and NodeList. The attribute Tid is the topic identifier of a topic t within the nondecomposable node. The NodeList attribute is a list of pairs <TidSet, Hid> where the pair <TidSet, Hid> represents the Tid's of the nondecomposable (hyper) node (which contains Tid), and Hid is a new topic identifier for the node. Table 4.2 illustrates HIndex for the nondecomposable node {T3, T4} in Example 4.1. We generate a new entry in MIndex for each nondecomposable node with the identifier Hid as its Tid1 value, and with a set of topic ids that it contains as its "ParentList". For example, in Table 4.1, the entry with Tid1 value of H1 and the ParentList value of {T3, T4} represents the nondecomposable (hyper) node H1 in the HIndex table.

In this section, to simplify the presentation, we assume that the metalink type M has only the transitivity axiom, but may or may not be LHS-decomposable. And, the product function is used to compute *FPath*=$\text{Imp}_d(t, P, \mathcal{R})$.

The topic closure of a set X of topics wrpt $\mathcal{R}$ = M and a sideway value threshold V_t is computed as follows. For each topic t in the topic closure X^+, we create a triplet of the form <t.Tid, $\text{Imp}_d(t, \mathcal{R} = M)$, {p | p is a path of type M from a topic or topics in X to t}>. We use a set-valued variable DiscoveredTids to contain the topics already in the closure, but not yet checked for paths emanating from them. We construct X^+ by repetitively computing $X^{(0)}$, $X^{(1)}$, …, $X^{(i)}$ where $1 \le i$. In the first iteration, for each topic t in X, a triplet <t.Tid, $\text{Imp}_d(t, \mathcal{R})$, {t}> is created in $X^{(1)}$ and the topic identifier Tid of t is added into the DiscoveredTids variable.

MType	Tid1	Imp(Tid1)	ParentList	ChildList <Tid2, Imp(Tid2), Imp(Mid)> triplets
Pre	T1	0.9	{}	<T3,0.85,0.95>, <T4,0.95,0.9>
Pre	T3	0.85	{T1}	-
Pre	H1	Avg(0.9, 0.95)= 0.925	{T3, T4}	<T5,0.7,0.9>
RelatedTo	T1	0.9	{T2, T3, T4}	<T2,0.8,0.6>,<T3,0.85, 0.95>, <T4,0.95,0.9>

Table 4.1 MIndex Table

Tid	NodeList <TidList, Hid>
T3	<{T3,T4}, H1>
T4	<{T3,T4}, H1>

Table 4.2. HIndex Table

In each iteration of the closure algorithm, a topic t1 is removed from the DiscoveredTids, and all metalinks that emanate from topic t1 are visited. A triplet <t2,$\text{Imp}_d(t2,\mathcal{R})$, t2.paths> for the consequent topic t2 of each visited metalink is added into the currently computed topic closure $X^{(i)}$, if the triplet does not exist in $X^{(i)}$. If the triplet exists then new paths into t2.paths are added and $\text{Imp}_d(t,\mathcal{R})$ is recomputed. The topic t2 is then added into DiscoveredTids. If the metalink type MType, for which the topic closure is to be computed, is not LHS-decomposable then the algorithm checks if topic t1 is in the LHS of a metalink of type M. The algorithm uses the HIndex table to find all HIndex entries that contain topic t1 as a member of their LHS set of topics. For each such HIndex, if all of its LHS topics are in the currently computed topic closure $X^{(i)}$ then new hyperpaths are created and new derived importance values are computed for every metalink that emanates from the HIndex. When the DiscoveredTids is empty, the algorithm stops, and $X^+=X^{(i)}$. We refer to this algorithm as the ThresholdTClosure algorithm.

Example 4.1 *(Topic Closure Computation for a LHS-Decomposable Metalink Type).* We use the MIndex instance in Table 4.1. Assume that we have the axiom: If $A \rightarrow^{Pre} B$ then $A \rightarrow^{RelatedTo} B$ where A and B are topics. Also, assume that we want to compute the topic closure for the set X={T1} with SV threshold V_t=0.4 using the metalink type M=*RelatedTo*. Also, assume that the average function is used for *FPathMerge.* Since X={T1}, $X^{(1)}$={<T1, 0.9, {T1}>} and DiscoveredTids={T1}. Note that the *RelatedTo* metalink type is LHS decomposable. In the first iteration, topic T1 is removed from DiscoveredTids. Topic T2 has a path T1.T2, obtained using the metalink $T1(0.9) \rightarrow^{RT(0.6)} T2\ (0.8)$, and its derived importance value is $\text{Imp}_d(T2,RelatedTo)$= 0.9 * 0.6 * 0.8 = 0.43. Therefore, the triplet <T2, 0.43, {T1.T2}> is added into $X^{(1)}$. After the first iteration, $X^{(2)}$ ={<T1, 0.9,{T1}>,<T2, 0.43, {T1.T2}>} and DiscoveredTids = {T2}. Next, the algorithm terminates since there is no *RelatedTo* metalink emanating from topic T2, therefore, DiscoveredTids becomes empty, and the output of the closure operator is {<T1,0.9>, <T2, 0.43>}.

Example 4.2 *(Hyperpath)* In Figure 4.1, {(T1$\rightarrow^{Pre}$ T3),(T1$\rightarrow^{Pre}$ T4)}$\rightarrow^{Pre}$ T5 forms a hyperpath of type *Pre* from topic T1 to topic T5. In order to compute the topic closure of type *Pre* for topic T1, the topic T5 should be considered if both topics T3 and T4 are added to the topic closure.

Example 4.3 *(Topic Closure Computation for a Non-Left-Hand-Side Decomposable Metalink).* Compute the topic closure for a set of topics X={T1} with sideway value threshold V_t=0.7 using the metalink type M=*Pre*. Assume that (a) *FPathMerge* is max, and (b) the geometric average is used to compute the derived importance value of a hypernode. Again, we use the MIndex instance in Table 4.1 to compute X^+ as {<T1, 0.76>, <T3, 0.727>, <T4, 0.729>}.

During closure computations, a metalink instance (i.e., a tuple in MIndex) can be visited more than once if there are multiple paths to the left-hand-side topic node of the metalink. To avoid visiting the same metalink more than once, we use the parent-child relationship between topics. A topic node with Tid1 is in the parent list of another topic node with Tid2 in the metalink M if there is a metalink Tid1 $\rightarrow^M$ Tid2. In the ThresholdTClosure algorithm, we use a set-valued variable PostponedTids to add the restriction that a topic node can not be "processed" until all nodes in its parent list is processed.

The algorithm ThresholdTClosure needs to maintain *all paths* from the set of input topics X to a given topic instance *a* in order to compute the derived importance value of *a* using a generic function. However, some functions, such as max, need to maintain only a single path to compute the derived importance value of a given topic. One can give an algorithm ThresholdTClosureMax that does not maintain the path information for any topic, and computes the derived importance value of a topic *x* by comparing its "current" derived importance value with respect to that of the "currently visited" path P. Clearly, ThresholdTClosureMax is much more efficient than ThresholdTClosure.

4.2 Ranking-based Topic Closure

The RankingTClosureMax algorithm is used to compute the top k-ranked topic closure using the maximum function. The algorithm finds the topics with the k highest derived importance values in the topic closure of a set X of input topics. It first computes the initial candidate top k ranked topics from the input topics X. Then, in each iteration i, it extracts the i^{th} top-ranked topic from the current k-i+1 candidate top-ranked topics and updates the current candidate topics by processing all emanating metalinks from the i^{th} topic. Therefore, the algorithm needs k iterations in order to compute the top-k-ranked topic closure of a set X of input topics.

The RankingTClosureMax algorithm maintains two lists X^+ and CandidateTopics of size at most k.The algorithm requires at most $\Omega(k * |X|)$ time to compute the initial CandidateTopics list, where |X| is the size of the input topic set X. Then, the algorithm iterates k times in order to compute the top-k-ranked topic closure, and, in each iteration, it finds the next top k topics and updates the CandidateTopics list by appling the metalinks that emanate from a given top-k topic. Therefore, the expected running time of this algorithm is very fast.

5 Experimental Results

5.1 SVA Join Operator

To evaluate the four SVA-Join algorithms discussed in Sections 3.2 and 3.3, we first extracted the titles of journal and conf. papers from the DBLP [DBLP01] data set into two different files, R and S. File R contains more than 91000 journal paper titles (12 Mbytes in size), and file S contains more than 132000 conference paper titles (18 Mbytes in size). Next, we eliminated the stopwords (i.e., removed words like "the", "a", "of", etc.) from the words in each title, stemmed them and created the word list (vocabulary) for the whole collection (including about 43000 words). The word list was kept in the main memory. Then, we created the vectors for each record of R and S, which were added to paper title records in files R and S. The SVA values for R and S records were generated randomly.

Below, we provide the experimental evaluation of the SVA-join algorithms in terms of the number of tuple-comparisons for a given query. The number of comparisons gives an idea about both the number of tuples read from each relation and the in-memory optimizations we apply (as in the case of $NLoop_{Sim\text{-}SVT}$ and $NLoop_{Sim\text{-}Top\text{-}k}$). Results involving disk-accesses and execution times are clearly symmetric with the number of comparisons made, and not reported here due to the lack of space.

All experiments have been performed on a dual-processor Pentium IIII 800 PC with 1-GB main memory running WindowsNT 4.0. The input and output buffer sizes were simulated to hold 10,000 tuples.

A. Evaluating $NLoop_{SVT}$ and $NLoop_{Top\text{-}k}$: The algorithms $NLoop_{SVT}$ and $NLoop_{Top\text{-}k}$ join tuples of R and S on the basis of an arbitrary join condition (predicate) θ, and return the joined tuples that are over a given threshold V_t or ranked in the top-k results. For the following experiments, $f_{out}()$ is specified as the product of the sideway values of joined tuples. We assume that join condition θ is a user-defined predicate, which states that a conf. paper tuple is to be joined with a journal paper if they have at least one author in common and the conf. paper is published at most 2 years before the journal paper. Clearly, this predicate can be specified as a user defined function (UDF) (syntax omitted to save space).

To evaluate a join with such an arbitrary predicate θ, an ordinary block-nested loops (BNL) algorithm compares each and every tuple, computes the sideway values for those tuples satisfying the user defined predicate, and finally retrieves the ones that are above the specified threshold or in the specified top-k set. On the other hand, $NLoop_{SVT}$ and $NLoop_{Top\text{-}k}$ evaluate the arbitrary predicate

only for those tuples with a derived sideway value that satisfies the query constraints. In Figures 5.1 and 5.2, we demonstrate the performance of these algorithms, compared against the "blind" BNL approach. Note that, the savings of the proposed algorithms increase, as the SV threshold value increases or, inversely, as the *k* value decreases. For instance, when the SV threshold value is 0.9, the number of tuple comparisons performed by $NLoop_{SVT}$ is approximately 300 millions, 1/40 of the BNL approach which makes 12 billion comparisons. For this case, $NLoop_{SVT}$ reads only 15% of R and S from the disk, whereas BNL reads all tuples of the relations. The saving in terms of execution time is in the order of magnitudes (i.e., seconds vs. hours). Furthermore, the savings increase as the complexity of user defined predicate increases.

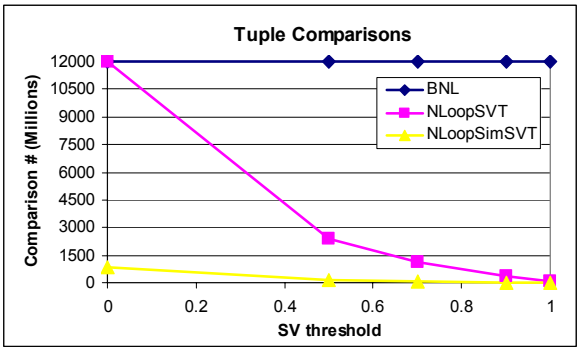

Figure 5.1 Performance values of BNL, $NLoop_{SVT}$ and $NLoop_{Sim\text{-}SVT}$ and algorithms.

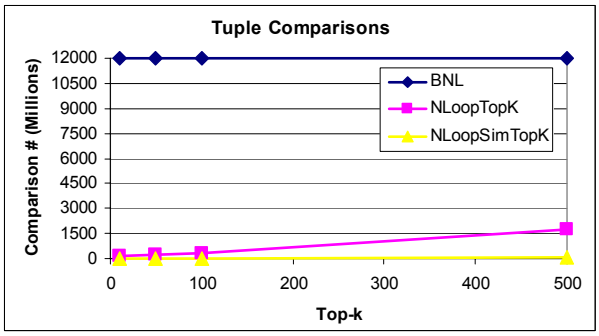

Figure 5.2 Performance values of BNL, $NLoop_{Top\text{-}k}$ and $NLoop_{Sim\text{-}Top\text{-}k}$ algorithms.

B. Evaluating $NLoop_{Sim\text{-}SVT}$ and $NLoop_{Sim\text{-}Top\text{-}k}$: The algorithms $NLoop_{Sim\text{-}SVT}$ and $NLoop_{Sim\text{-}Top\text{-}k}$ perform similarity-based (approximate) joins. In the following experiments, the tuples of R and S are joined if they have titles similar to each other with a similarity value greater than a specified threshold (90%). In this case, $f_{out}( )$ is specified as the product of the sideway values of joined tuples and this derived value is further multiplied with the similarity value of tuples, obtained using the cosine similarity measure.

Figures 5.1 and 5.2 illustrate the performance superiority of $NLoop_{Sim\text{-}SVT}$ and $NLoop_{Sim\text{-}Top\text{-}k}$ with respect to the BNL. For instance, to retrieve tuple pairs with titles that are 90% similar and have a derived sideway value greater than 0.9, BNL achieves a total of 12 billion comparisons, whereas $NLoop_{Sim\text{-}SVT}$ makes only 23 million comparisons. This improvement is due to the fact that similarity based algorithms are tailored to exploit the vector-space model to its greatest extent. Max-similarity filters reduce the number of tuples to be compared. Finally, we create an in-memory inverted index [Salt89] for the tuples of outer relation on the fly, and compare tuples that only have common words in their titles.

To summarize, for arbitrary predicates and monotone SV functions, algorithms $NLoop_{SVT}$ and $NLoop_{Top\text{-}k}$ improve the performance of BNL considerably. For the special case of similarity-based joins, the algorithms are further optimized and more gains are obtained.

5.2 SVA Topic Closure Operator

For the TClosure algorithm, we synthetically generated the data for the Topics file, Metalink file, and the set X of input topics. A disk-based MIndex file (see section 4.1) is generated from the topic and metalinks files. In order to efficiently retrieve a tuple from the MIndex file, we used in-memory LRU buffer and a sparse index table. In implementations of the topic closure algorithms, we used max as the *FPathMerge* function.

We generated a topic file with N tuples. Each tuple is 100Bytes, and has a random importance value in [MinT, 1.0], where MinT is the minimum topic importance value. In the Metalinks file, we generated N* R decomposable acyclic metalinks (i.e., without hypernodes), where R is the ratio of metalinks to topics. We divided the topics of the Topics file into groups, each of random size between 50 to 100 topics. For each group G of topics, we generated |G| * R randomly distributed metalinks with random metalink importance values in [minM, 1.0], where MinM is the minimum metalink importance value in the Metalinks file. In a given group G of topics, 80% of the metalinks are from group G to the next group G+1 and the remaining 20% are from group G to the other groups. Also, we selected a set X of input topics randomly from the Topics file, and generated random importance values in [MinX, 1.0] for each topic, where MinX is the min. derived importance value of the set X of input topics.

To evaluate the algorithms, we used a topic file with N, $1000 \leq N \leq 100{,}000$, topics (i.e., file size is between 100KB and 10GB) and topic importance values are in [0.4, 1.0]. The Metalinks file has metalinks/topics ratio of 3 (i.e., R=3), and metalink importance values are in [0.4, 1.0]. The size of the set X of input topics is 100 topics, and their topic importance values are in [0.5, 1.0]. Also, we used an in-memory buffer of size 100KB and a sparse index table of size 1000 tuples.

The experimental results for *the number of disk accesses* for threshold-based topic closure algorithm and ranking-based topic closure algorithm are illustrated in figure 5.4 and figure 5.5, respectively. In both figures, the difference between the disk accesses of two different N values

increases when the sideway threshold value is decreased or the value of k is increased.

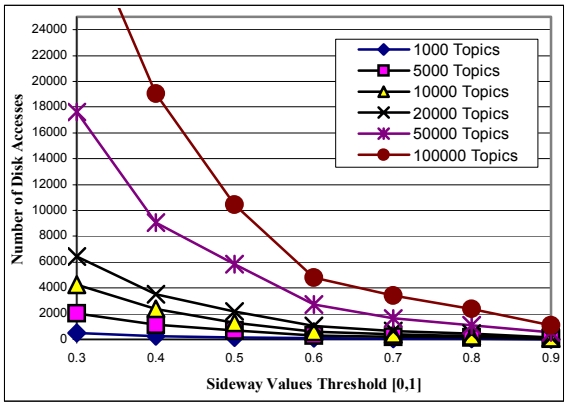

Figure 5.4 Number of Disk Accesses for Sideway-Value-Threshold-Based Topic Closure

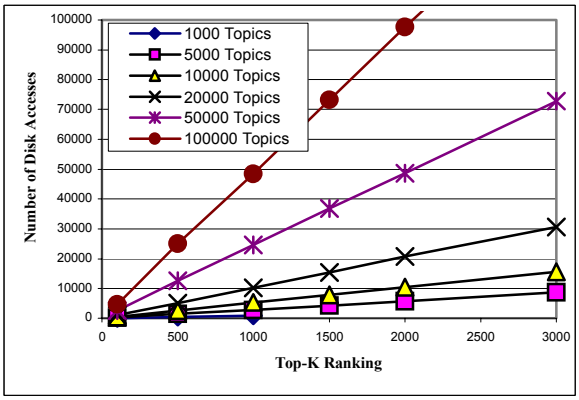

Figure 5.5 Number of Disk Accesses: Ranking-based TClosure

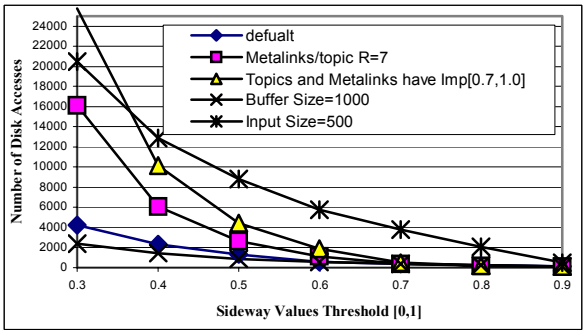

Figure 5.6 Threshold-Based Topic Closure: Different parameters

To evaluate with different parameters, we changed one parameter at a time, and compared the results with those in figures 5.4 and 5.5. Figures 5.6 and 5.7 illustrate the experimental results with N=10,000 topics (i.e., 1GB file). In figure 5.6, the number of disk accesses increase sharply when the ratio R of metalinks to topics is changed from R=3 to 7, the importance values are increased, the size of the buffer is decreased, or the size of the input topics X are increased. Figure 5.7 shows that the ranking-based algorithm is less sensitive to the changes of these parameters; there is only small change in the disk accesses when one of the parameters is changed.

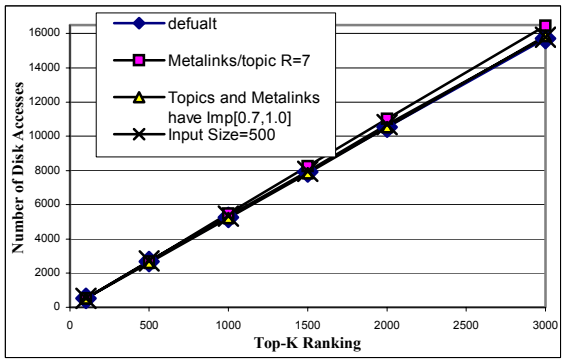

Figure 5.7 Ranking-based TClosure: different parameters

Research Acknowledgement

This research is supported by a joint grant from TUBITAK (*grant no. 100U024*) of Turkey and the National Science Foundation (*grant no. INT-9912229*) of the USA. A. Al-Hamdani's research is supported by a graduate scholarship from Sultan Qaboos University, Oman.

6 References

[A+01] Altingovde, I.S. et al, "Topic-Centric Querying of Web Information Resources", DEXA'01.

[AW00] Agrawal, R., Wimmers, E.L., "A Framework for Expressing and Combining Preferences", SIGMOD'00.

[BMP92] Barbará, D., Garcia-Molina, H., Porter, D., "The Management of Probabilistic Data", IEEE TKDE, 1992.

[CH02] Chan, K., C-C, Hwang, S-W., "Minimal Probing: Supporting Expensive Predicates for Top-k Queries", SIGMOD'02.

[CK97] Carey, M.J., Kossmann, D., "On Saying "Enough Already!" in SQL", SIGMOD'97.

[CK98] Carey, M.J., Kossmann, D., "Reducing the Braking Distance of an SQL Query Engine", VLDB'98.

[CG99] Chaudhuri, S, Gravano, L., "Evaluating Top-*k* Selection Queries", VLDB'99.

[Codd80] E. F. Codd, "Data Models in Database Management", Data Abstraction, Databases workshop, 1980.

[Coh98] Cohen, W. W., "Integration of Heterogeneous Databases Based on Textual Similarity", SIGMOD'98.

[DBLP 01] DBLP Bibliography, by Michael Ley, 2001.

[DLGG89] M. Diligenti, F. Coetzee, S. Lawrence, C. Giles, M. Gori, Focused Crawling Using Context Graphs, VLDB'00.

[Graef93] Graefe, G., "Query Evaluation Techniques for Large Databases", *ACM Computing Surveys,* 1993.

[KCPA01] G. Karvounarakis, V. Christophides, D. Plexousakis, S. Alexaki, *Querying RDF Descriptions for Community Web Portals*, 17imes Journees Bases de Donnees Avancees (BDA'01), pp. 133-144, Agadir, Maroc, 2001.

[HKP01] Hristidis, V. et al, "PREFER: A System for Multi-parametric Ranked Queries", SIGMOD'01.

[O+01] Ozsoyoglu et al, "Electronic Books in Digital Libraries", IEEE *Advances in Digital Libraries Conf.,* 2000.

[O+02] Ozsoyoglu, G. et al, "Sideway values, SQL Extensions, SVA", at http://art.cwru.edu/vldb/p640

[RG00] Ramakrishnan, R., Gehrke, J., *Database Management Systems*, McGraw-Hill, 2000.

[Salt89] Salton, G., *Automatic Text Processing*, Addison-Wesley, 1989.