a Graph-Theoretic Approach to Map Conceptual Designs to XML Schemas

M. Franceschet, **D. Gubiani**, A. Montanari, C. Piazza

17 December 2010

Introduction

- The most common applications of XML involve the storage and exchange of data
- An XML database allows to store data in XML format based on a specific XML schema
- The design is a crucial phase in the development of database

Related Work

Integration of XML with relational databases:

- **the mapping ER conceptual schemas into some XML schema language**
- the translation of relational logical schemas into some XML schema language
- the development of conceptual models for XML databases

Our Goal

- We propose **a mapping from ER to XML Schema**
- We give **a graph-theoretic interpretation of the structure nesting problem**
- We implement the devised translation and embed it into ChronoGeoGraph, **a software framework for the conceptual and logical design of spatio-temporal XML and relational databases**

The Mapping from ER to XML Schema

We propose a mapping from ER to XML Schema with the following properties:

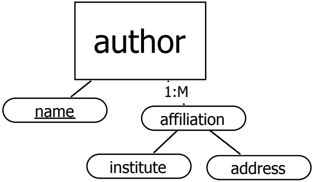
- information and integrity constraints are preserved (an extension to the standard XML Schema has been implemented to capture the constraints missed in the translation)
- no redundance is introduced
- different hierarchical views of the conceptual information are permitted
- the resulting structure is highly connected and highly nested
- the design is reversible

XML Schema Notation

- We embed ER schemas into a more succinct *XML schema notation* (XSN) whose expressive power lies in between DTD and XML Schema
- XSN allows one to specify sequences and choices of elements as in DTD
- XSN extends DTD with the following three constructs:
  - *occurrence constraints*: `item[x,y]`
  - *key constraints*: `KEY(A.KA)` or `KEY(A.K1, A.K2)`
  - *foreign key constraints*: `KEYREF(B.FKA --> A.KA)`
- The mapping of XSN into XML Schema is straightforward

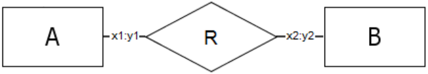
Entities and Attributes

- Each entity is mapped into an element with the same name
- Entity attributes are mapped into child elements:
  - composed attributes are translated by embedding the sub-attribute elements into the composed attribute element
  - multi-valued attributes are encoded using suitable occurrence constraints



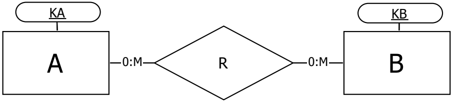
author(name,affiliation+)
  affiliation(institute,address)
KEY(author.name)

Binary Relationships

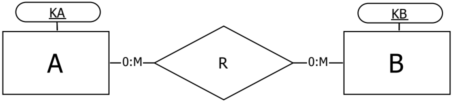


We analyzed all $2^4 = 16$ cases
comparing flat and nesting translation

Relationships with cardinality (0,N)-(0,N)



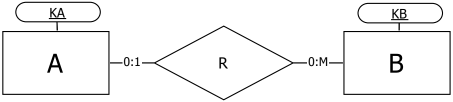
Relationships with cardinality (0,N)-(0,N)



A(KA, R*)
  R(KB)
B(KB)
KEY(A.KA), KEY(B.KB)
KEYREF(R.KB --> B.KB)

B(KB, R*)
  R(KA)
A(KA)
KEY(B.KB), KEY(A.KA)
KEYREF(R.KA --> A.KA)

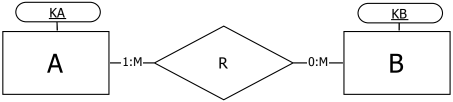
Relationships with cardinality (0,1)-(0,N)



A(KA, R?)
  R(KB)
B(KB)
KEY(A.KA), KEY(B.KB)
KEYREF(R.KB --> B.KB)

B(KB, R*)
  R(KA)
A(KA)
KEY(B.KB), KEY(A.KA)
KEYREF(R.KA --> A.KA)
KEY(R.KA)

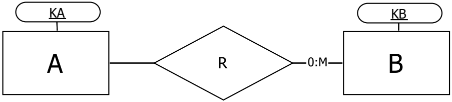
Relationships with cardinality (1,N)-(0,N)



A(KA, R+)
  R(KB)
B(KB)
KEY(A.KA), KEY(B.KB)
KEYREF(R.KB --> B.KB)

B(KB, R*)
  R(KA)
A(KA)
KEY(B.KB), KEY(A.KA)
KEYREF(R.KA --> A.KA)
CHECK("left min")

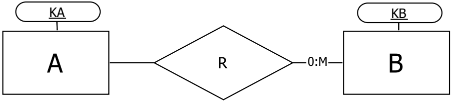
Relationships with cardinality (1,1)-(0,N) - 1



A(KA, R)
  R(KB)
B(KB)
KEY(A.KA), KEY(B.KB)
KEYREF(R.KB --> B.KB)

B(KB, R*)
  R(A,KA)
A(KA)
KEY(B.KB), KEY(A.KA)
KEYREF(R.KA --> A.KA)
KEY(R.KA)
CHECK("left min")

Relationships with cardinality (1,1)-(0,N) - 2

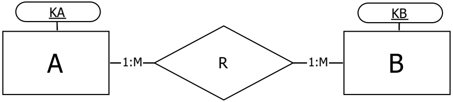


A(KA, R)
  R(KB)
B(KB)
KEY(A.KA), KEY(B.KB)
KEYREF(R.KB --> B.KB)

B(KB, R*)
  R(A)
    A(KA)
KEY(B.KB), KEY(A.KA)

- The nesting of entities that participate to relationships with cardinality $(1,1)$ minimizes the number of constraints

Relationships with cardinality (1,N)-(1,N)



A(KA, R+)	B(KB, R+)
R(KB)	R(KA)
B(KB)	A(KA)
KEY(A.KA), KEY(B.KB)	KEY(B.KB), KEY(A.KA)
KEYREF(R.KB --> B.KB)	KEYREF(R.KA --> A.KA)
CHECK("right min")	CHECK("left min")

- The case $A \overset{(1,N)}{\longleftrightarrow} R \overset{(1,N)}{\longleftrightarrow} B$ is the only one in the mapping of relationships in which we must use external constraints

Translation Pattern for Binary Relationships - 1

- The translation pattern for binary relationships can be summarized as follows:
  - the cardinality constraint associated with the entity whose corresponding element includes the element for the relationship can be forced by `occurs` constraints
  - the cardinality constraint associated with the other entity is imposed depends on its specific form

Translation Pattern for Binary Relationships - 2

- $(1, 1)$ constraint, that characterized functional relationships, can be entirely checked although the nesting structure
- For other constraints, all entities included into the relationship element require the addition of a `keyref` constraint ($(0, N)$, $(0, 1)$, $(1, N)$)
- In addition, the cardinality constraints:
  - $(0, 1)$ also needs a `key` constraint
  - $(1, N)$ also needs an external constraint

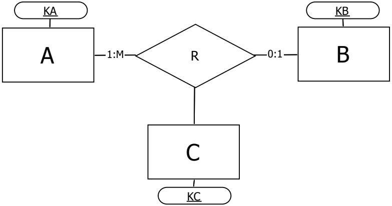
Translation Pattern for Binary Relationships - 3

- To minimize the number of constraints:
  - the outermost element corresponds to an entity that participates in R with cardinality constraint $(1, N)$
  - if there is not such an entity, we choose an entity that participates with cardinality constraint $(0, 1)$
  - if there is not such an entity as well, we choose one that participates with cardinality constraint $(0, N)$
  - if all two entities participate with cardinality constraint $(1, 1)$, we will choose one of them
  - then, the element corresponding to R is nested in the outermost element and it includes the element, or the reference to the element, corresponding to the other entity

Relationships of Higher Degree

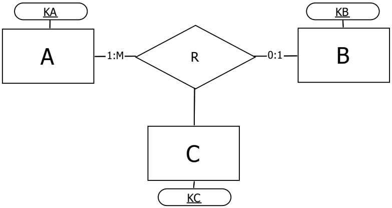
- The rules to translate binary relationships can be generalized to relationships of higher degree:
  - the outermost element corresponds to an entity that participates in R with cardinality constraint $(1, N)$
  - if there is not such an entity, we choose an entity that participates with cardinality constraint $(0, 1)$
  - if there is not such an entity as well, we choose one that participates with cardinality constraint $(0, N)$
  - if all entities participate with cardinality constraint $(1, 1)$, we will choose one of them
  - then, the element corresponding to R is nested in the outermost element and it includes the elements, or the references to the elements, corresponding to all the other entities

Example of Relationship of Higher Degree



A(KA,R+)
  R(KB,C)
    C(KC)
B(KB)
KEY(A.KA)
KEY(B.KB)
KEY(C.KC)
KEY(R.KB)
KEYREF(R.KB-->B.KB)

Example of Relationship of Higher Degree

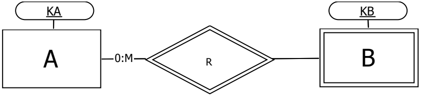


A(KA,R+)
  R(KB,C)
    C(KC)
B(KB)
KEY(A.KA)
KEY(B.KB)
KEY(C.KC)
KEY(R.KB)
KEYREF(R.KB-->B.KB)

- Alternative solution: we can preliminarily apply reification to replace every relationship of higher degree by a corresponding entity related to each participating entity by a suitable binary relationship

Weak Entities and Identifying Relationships

- A weak entity always participates in the identifying relationship with cardinality constraint (1,1)
- The key of the element for the weak entity is obtained by composing the partial key with the key of the owner entity

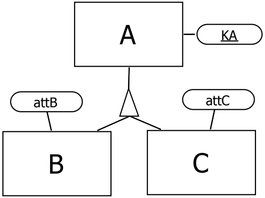


A(KA,R*)
R(B)
B(KB, KA)
KEY(A.KA)
KEY(B.KB, B.KA)
CHECK(B.KA=A.KA)

- It is not possible to remove the owner key `KA` from the element for the weak entity *B* because the key constraint `KEY(B.KB, A.KA)` cannot be expressed in XML Schema

Specializations

- The mapping of specialization can fully exploit the hierarchical nature of the XML data model



Disjoint Specializations

- Partial:

A(KA, (B|C)?)	A(KA)
B(attB)	B(KA,attB)
C(attC)	C(KA,attC)
KEY(A.KA)	KEY(A.KA),
	KEY(B.KA | C.KA)
	REFKEY(B.KA-->A.KA)
	REFKEY(C.KA-->A.KA)

Disjoint Specializations

- Partial:

A(KA, (B|C)?)	A(KA)
B(attB)	B(KA,attB)
C(attC)	C(KA,attC)
KEY(A.KA)	KEY(A.KA),
	KEY(B.KA | C.KA)
	REFKEY(B.KA-->A.KA)
	REFKEY(C.KA-->A.KA)

- Total:

A(KA, (B|C))	B(KA,attB)
B(attB)	C(KA,attC)
C(attC)	KEY(B.KA | C.KA)
KEY(A.KA)	

Overlapping Specializations

- Partial:

A(KA, B?, C?)	A(KA)
B(attB)	B(KA,attB)
C(attC)	C(KA,attC)
KEY(A.KA)	KEY(A.KA), KEY(B.KA), KEY(C.KA)
	KEY(B.KA),KEYREF(B.KA-->A.KA)
	KEY(C.KA),KEYREF(C.KA-->A.KA)

Overlapping Specializations

- Partial:

A(KA, B?, C?)	A(KA)
B(attB)	B(KA,attB)
C(attC)	C(KA,attC)
KEY(A.KA)	KEY(A.KA), KEY(B.KA), KEY(C.KA)
	KEY(B.KA),KEYREF(B.KA-->A.KA)
	KEY(C.KA),KEYREF(C.KA-->A.KA)

- Total:

A(KA, ((B,C?) | C))	B(KA,attB)
B(attB)	C(KA,attC)
C(attC)	KEY(B.KA)
KEY(A.KA)	KEY(C.KA)

Specializations Observations

- The generalization to specializations involving $n > 2$ child entities is immediate in all cases except for the total-overlapping case

$$\rho(a_1, ..., a_n) = \begin{cases} a_1 & \text{if } n = 1 \\ (a_1, a_2?, ..., a_n?) | \rho(a_2, ..., a_n) & \text{if } n > 1 \end{cases}$$

Specializations Observations

- The generalization to specializations involving $n > 2$ child entities is immediate in all cases except for the total-overlapping case

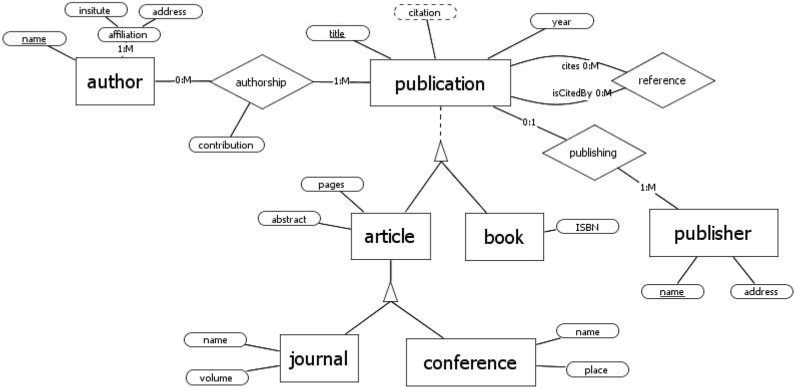
$$\rho(a_1, \ldots, a_n) = \begin{cases} a_1 & \text{if } n = 1 \\ (a_1, a_2?, \ldots, a_n?) | \rho(a_2, \ldots, a_n) & \text{if } n > 1 \end{cases}$$

- Multiple specializations break nesting strategy
  - a child entity may have more than one parent entity
  - the resulting schema is a directed acyclic graph, which cannot be directly dealt with such a data model
  - to encode multiple specializations, we can use a flat encoding similar to the relational mapping

XML VS Relational Model

- Thanks to its hierarchical nature, the XML logical model allows one to capture a larger number of constraints specified at conceptual level than the relational one
  - for all cardinality constraints of the form $(1, N)$ of a relationship there is no way to preserve the minimum cardinality constraint 1 in the mapping of ER schemas into relational ones
  - the same happens with specializations

Example: Citation-Enhanced Bibliographic DataBase - 1



Example: Citation-Enhanced Bibliographic DataBase - 2

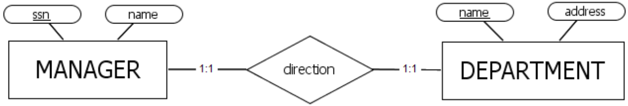
```
publication(title, year, citations, reference*, authorship+, (article | book)?)
  reference(title)
  authorship(name, contribution)
  article(pages, abstract, (journal | conference))
    journal(name, volume)
    conference(name, place)
  book(ISBN)
publisher(name, address, publishing+)
  publishing(title)
author(name, affiliation+)
  affiliation(institute, address)

KEY(publication.title), KEY(publisher.name)
KEY(author.name), KEY(publishing.title)
KEYREF(reference.title --> publication.title)
KEYREF(authorship.name --> author.name)
KEYREF(publishing.title --> publication.title)
```

Nesting the Structure

- Nesting the XML structure has two advantages:
  - the **reduction of the number of constraints** inserted in the mapped schema and hence of the validation overhead
  - the **decrease of the (expensive) join operations** needed to reconstruct the information at query time

Decrease of the Join Operations



manager(ssn, name, direction)
  direction(name)
department(name, address)
KEY(manager.ssn)
KEY(department.name)
KEY(direction.name)
KEYREF(department.name --> direction.name)
KEYREF(direction.name --> department.name)

manager(ssn, name, direction)
  direction(department)
    department(name, address)
KEY(manager.ssn)
KEY(department.name)

To retrieve the addressof the departmentdirected by William Strunk:

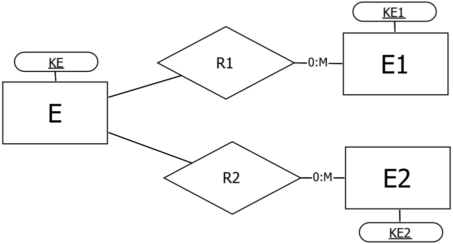
/department[name =
  /manager[name = "William Strunk"]/direction/name]/address

/manager[name = "William Strunk"]/direction/department/address

Single Constructs VS ER Schemas

- Translation rules described previously are applied to the single elements of an ER schema
- We do not translate ER constructs in isolation, but an ER schema including a number of related constructs
- For each ER construct, the choice of the specific translation rule to apply depends on the way in which the construct occurs in the schema

An Example



- The element for E cannot be included both in the element for R_1 and in that for R_2

Preferred Rule VS Alternative Rule

```
E1(KE1, R1?)
   R(E)
      E(KE)
KEY(E1.KE1)
KEY(E.KE)
```

Preferred Rule VS Alternative Rule

```
E1(KE1, R1?)
   R(E)
      E(KE)
KEY(E1.KE1)
KEY(E.KE)

E2(KE2, R2?)
   R(KE)
KEY(E2.KE2)
REFKEY(R2.KE2-->E2.KE2)
KEY(R.KE)
CHECK("right min")
```

Preferred Rule VS Alternative Rule

```
E1(KE1, R1?)
   R(E)
      E(KE)
KEY(E1.KE1)
KEY(E.KE)

E2(KE2, R2?)
   R(KE)
KEY(E2.KE2)
REFKEY(R2.KE2-->E2.KE2)
KEY(R.KE)
CHECK("right min")
```

```
E1(KE1, R1?)
   R(E)
      E(KE, R2)
        R2(KE2)
E2(KE2)
KEY(E1.KE1)
KEY(E.KE)
KEY(E2.KE2)
REFKEY(R2.KE2-->E2.KE2)
```

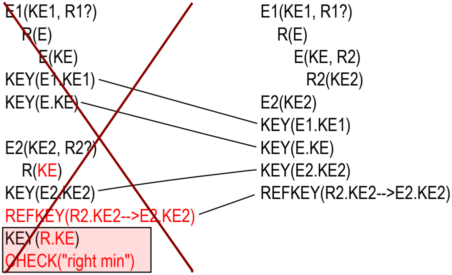
Preferred Rule VS Alternative Rule

```
E1(KE1, R1?)
   R(E)
      E(KE)
KEY(E1.KE1)
KEY(E.KE)

E2(KE2, R2?)
   R(KE)
KEY(E2.KE2)
REFKEY(R2.KE2-->E2.KE2)
KEY(R.KE)
CHECK("right min")
```

```
E1(KE1, R1?)
   R(E)
      E(KE, R2)
        R2(KE2)
E2(KE2)
KEY(E1.KE1)
KEY(E.KE)
KEY(E2.KE2)
REFKEY(R2.KE2-->E2.KE2)
```

Preferred Rule VS Alternative Rule



- The preferred translation rule can be applied to one of the relationships only, while for the other relationship we must resort to the alternative translation rule

The Nesting Problem

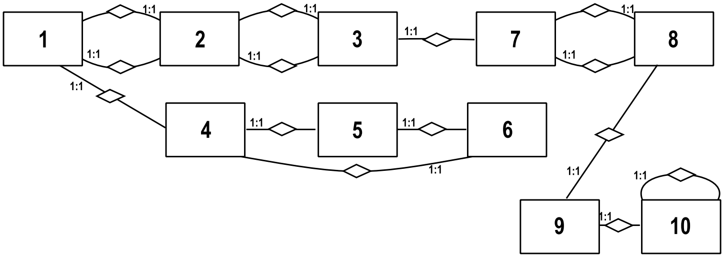
- To keep the algorithm as simple as possible, we preliminarily restructure the ER schema by removing higher-order relationships and specializations
- As shown before, the nesting structure induced by total functional relationships is not always uniquely determined:
  - some entity can be nested in more than one other entity (*nesting confluence*)
  - *nesting loops* can occur
- How do we find the "*best*" nesting structure?

The Nesting Problem in Graph Theory

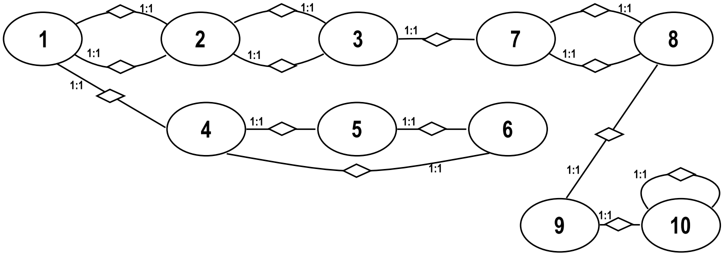
- Let S be an ER schema and the corresponding *nesting graph* $G = (V, E)$ be a directed graph such that:
  - the nodes in V are the entities of S that participate in some total functional relationship and
  - $(A, B) \in E$ whenever there is a total functional relationship R relating A and B

    The direction of the edges indicates the entity nesting structure
- A spanning forest is a subgraph G' of G such that: (i) G' and G share the same node set; (ii) each node in G' has at most one predecessor; (iii) G' has no cycles

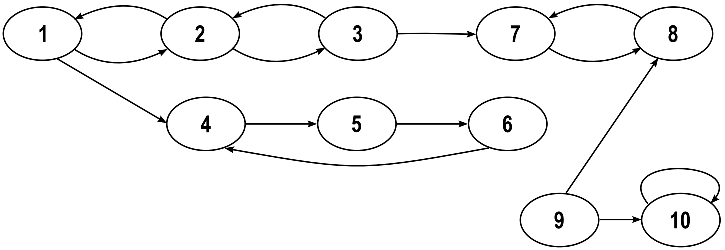
The Nesting Problem: an Example



The Nesting Problem: an Example



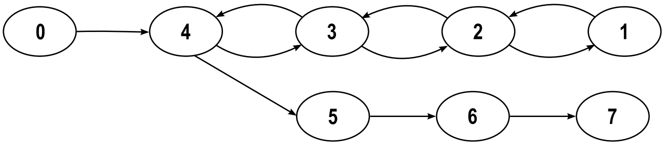
The Nesting Problem: an Example



Two Nesting Problems

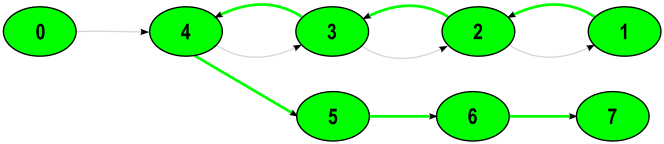
- **The Maximum Depth Nesting Problem**
  Given a nesting graph G for an ER schema, find a Maximum Depth Spanning Forest, that is a spanning forest with the maximum sum of node depths
- **The Maximum Density Nesting Problem**
  Given a nesting graph G for an ER schema, find a Maximum Density Spanning Forest, that is a spanning forest with the maximum number of edges, or, equivalently, with the minimum number of trees

Two Different Nesting Problems



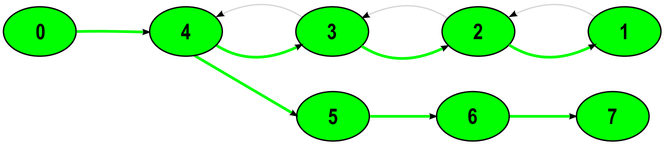
- A *Maximum Density Spanning Forest* is obtained by removing edges (1,2), (2,3), and (3,4): it is composed of one tree, 7 edges, and the sum of node depths is 19
- A *Maximum Depth Spanning Forest* is the simple path from node 1 to node 7 plus the node 0: it comprises 2 trees, 6 edges, and the sum of node depths is 21

Two Different Nesting Problems



- **A Maximum Depth Spanning Forest is the simple path from node 1 to node 7 plus the node 0: it comprises 2 trees, 6 edges, and the sum of node depths is 21**
- A *Maximum Density Spanning Forest* is obtained by removing edges (1,2), (2,3), and (3,4): it is composed of one tree, 7 edges, and the sum of node depths is 19

Two Different Nesting Problems



- A *Maximum Depth Spanning Forest* is the simple path from node 1 to node 7 plus the node 0: it comprises 2 trees, 6 edges, and the sum of node depths is 21
- **A Maximum Density Spanning Forest is obtained by removing edges (1,2), (2,3), and (3,4): it is composed of one tree, 7 edges, and the sum of node depths is 19**

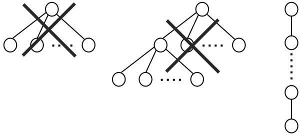
The Maximum Depth Nesting Problem - 1

Theorem (Complexity)

Let G be a digraph. The maximum depth nesting problem for G is NP-complete.

PROOF by reducing the Hamiltonian path problem to the MDNP

IDEA: to maximize the depth of a generic forest the nodes has to be pushed as deep as possible, leading to a chain.



maximum depth nesting problem with depth $S_F = (|V| \cdot (|V| - 1))/2$. We proved that a graph G = (V,E) has an Hamiltonian path if and only if G has a spanning forest of depth $(|V| \cdot (|V| - 1))/2$.

The Maximum Depth Nesting Problem - 2

Lemma

Let $G = (V, E)$ be a strongly connected digraph such that $(u, v) \in E$ if and only if $(v, u) \in E$. It holds that if F is a maximum depth spanning forest for G, then F is a tree.

Theorem (Approximability)

Unless $P = NP$, there is no constant ratio approximation algorithm for the maximum depth problem.

The Maximum Depth Nesting Problem for DAG

Theorem

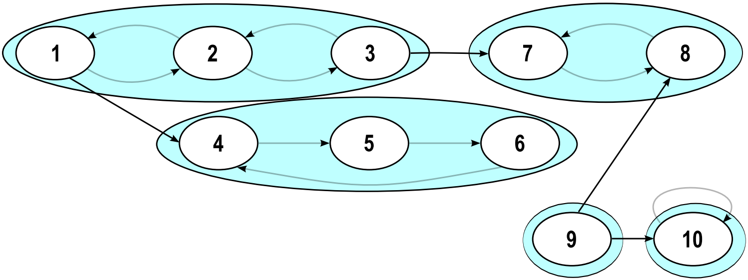
Let $G = (V, E)$ be a DAG and let F be a maximum depth spanning forest for it. Then, F is a maximum density spanning forest for G.

The Algorithm Maximum_Density

1. compute the graph H of the strongly connected components of G (let $C = \{C_1, \ldots, C_n\}$ be the set of nodes of H)
2. compute a maximum density spanning forest $K = (C, E_K)$ for H
3. compute a set of edges E' as follows: for each edge $(C_j, C_i) \in E_k$, pick an edge (u, v) such that $(u, v) \in E$, $u \in C_j$ and $v \in C_i$ and add (u, v) to E'
4. for each strongly connected component C_i of G:
    - A) if there is an edge (u, v) in E' with v in C_i, then compute a tree $T_i = (C_i, E_i)$ rooted at v and spanning C_i
    - B) else pick a node v in C_i and compute a tree $T_i = (C_i, E_i)$ rooted at v and spanning C_i
5. output the forest $F = (V, E' \cup E_1 \cup E_2 \cup \cdots \cup E_n)$

MAXIMUM_DENSITY - STEP 1

Compute the graph H of the strongly connected components of G (let $C = \{C_1, \ldots, C_n\}$ be the set of nodes of H)

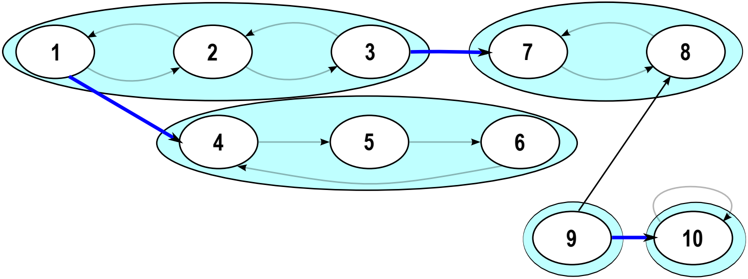


MAXIMUM_DENSITY - STEP 2

Compute a maximum density spanning forest $K = (C, E_K)$ for H as follows:

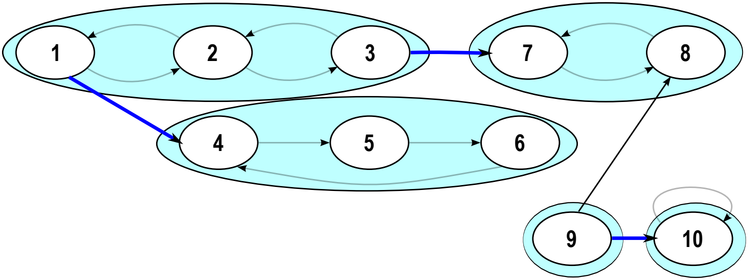
A) compute H^{-1} and, for each node C_i, the rank $rank_{H^{-1}}(C_i)$

B) for each node C_i in H, if C_i is not a root node in H, then pick a node C_j such that (C_j, C_i) is in H and $rank_{H^{-1}}(C_j) = rank_{H^{-1}}(C_i) - 1$ and add the edge (C_j, C_i) to E_K



Maximum_Density - step 3

Compute a set of edges E' as follows: for each edge $(C_j, C_i) \in E_k$, pick an edge (u, v) such that $(u, v) \in E$, $u \in C_j$ and $v \in C_i$ and add (u, v) to E'

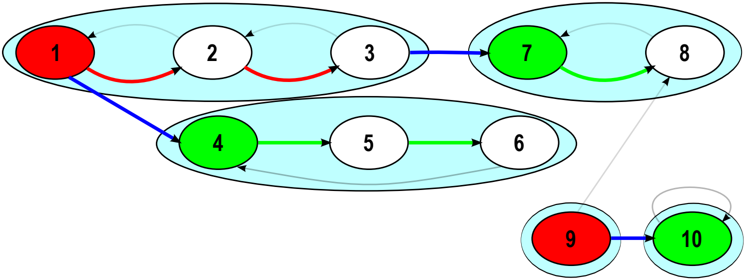


MAXIMUM_DENSITY - STEP 4

For each strongly connected component C_i of G:

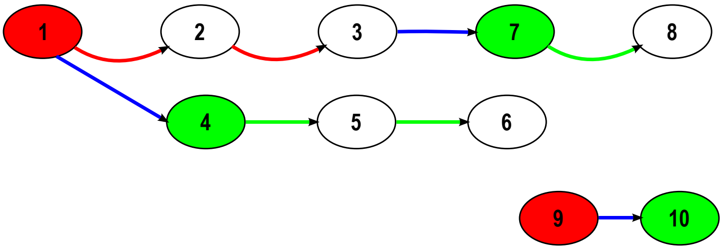
A) if there is an edge (u, v) in E' with v in C_i, then compute a tree $T_i = (C_i, E_i)$ rooted at v and spanning C_i

B) else pick a node v in C_i and compute a tree $T_i = (C_i, E_i)$ rooted at v and spanning C_i



MAXIMUM_DENSITY - STEP 5

Output the forest $F = (V, E' \cup E_1 \cup E_2 \cup \cdots \cup E_n)$



The Maximum Density Nesting Problem

Lemma

The spanning forest K generated by step 3 of the algorithm Maximum_Density is a maximum depth spanning forest for H.

Theorem (Correctness and Complexity)

Let G be a digraph. The algorithm Maximum_Density computes a maximum density spanning forest for G in linear time.

The Maximum Density Nesting Problem for DAG

Theorem

Let G be a DAG. The algorithm Maximum_Density computes a maximum depth spanning forest for G in linear time.

The Constrained Nesting Problem - 1

- To make the translation algorithm more flexible, we introduce a constrained variant of the considered problems that gives the designer the possibility to impose the application of the preferred translation rule to some relationships
  - this amounts to force the maintenance of some edges of the original digraph

The Constrained Nesting Problem - 1

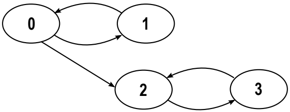
- To make the translation algorithm more flexible, we introduce a constrained variant of the considered problems that gives the designer the possibility to impose the application of the preferred translation rule to some relationships
  - this amounts to force the maintenance of some edges of the original digraph

PROBLEM:

Given a digraph G and a set of its edges C, find a spanning forest, containing all edges in C, with the maximum number of edges (**constrained maximum density problem**) or with the maximum sum of node depths (**constrained maximum depth problem**)

The Constrained Nesting Problem - 2

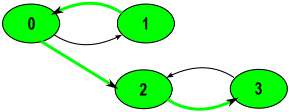
- The solution of the constrained version does not necessarily coincide with that of the original problem



  - different solutions to the maximum density problem exist each one consisting of 1 tree with 3 edges and none of them contains the edge (3,2)
  - a maximum density spanning forest containing the edge (3,2) necessarily consists of 2 trees with 1 edge each

The Constrained Nesting Problem - 2

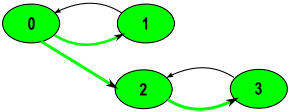
- The solution of the constrained version does not necessarily coincide with that of the original problem



  - **different solutions to the maximum density problem exist each one consisting of 1 tree with 3 edges and none of them contains the edge (3,2)**
  - a maximum density spanning forest containing the edge (3,2) necessarily consists of 2 trees with 1 edge each

The Constrained Nesting Problem - 2

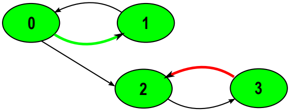
- The solution of the constrained version does not necessarily coincide with that of the original problem



  - **different solutions to the maximum density problem exist each one consisting of 1 tree with 3 edges and none of them contains the edge (3,2)**
  - a maximum density spanning forest containing the edge (3,2) necessarily consists of 2 trees with 1 edge each

The Constrained Nesting Problem - 2

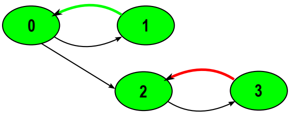
- The solution of the constrained version does not necessarily coincide with that of the original problem



  - different solutions to the maximum density problem exist each one consisting of 1 tree with 3 edges and none of them contains the edge (3,2)
  - **a maximum density spanning forest containing the edge (3,2) necessarily consists of 2 trees with 1 edge each**

The Constrained Nesting Problem - 2

- The solution of the constrained version does not necessarily coincide with that of the original problem



  - different solutions to the maximum density problem exist each one consisting of 1 tree with 3 edges and none of them contains the edge (3,2)
  - **a maximum density spanning forest containing the edge (3,2) necessarily consists of 2 trees with 1 edge each**

The Constrained Nesting Problem - 3

- The constrained versions of the problems may lack a solution

Lemma (Existence of a solution)

Let $G = (V, E)$ be a digraph and $C \subseteq E$. The constrained maximum density (resp., depth) problem has a solution if and only if neither loops nor confluences occur in C.

The Constrained Depth Nesting Problem

Theorem

Let $G = (V, E)$ be a digraph and $C \subseteq E$. The constrained maximum depth problem for G and C is NP-complete. Moreover, unless $P = NP$, there is no a constant ratio approximation algorithm for it.

The Constrained Density Nesting Problem - 1

1. check that C contains neither loops nor confluences; otherwise, stop with failure (it has no solution)
2. compute the set of target nodes $T = \{v \mid \exists(u, v) \in C\}$ in C
3. compute the graph $\overline{G} = (V, \overline{E})$ such that $(u, v) \in \overline{E}$ iff $(u, v) \in C \vee (v \notin T \wedge (u, v) \in E)$
4. apply Maximum_Density to $\overline{G}$ (let $\overline{F}$ be the output it produces)
5. for each edge $(u, v) \in C$, if $(u, v) \notin \overline{F}$, then let (r, s) be an edge on the path from v to u in $\overline{F}$ such that $(r, s) \notin C$. Replace (r, s) by (u, v) in $\overline{F}$
6. output the forest $\overline{F}$

The Constrained Density Nesting Problem - 2

Theorem

Let $G = (V, E)$ be a digraph and let $C \subseteq E$. Constrained_Maximum_Density solves the constrained maximum density problem for G and C in linear time.

The Constrained Density Nesting Problem for DAG

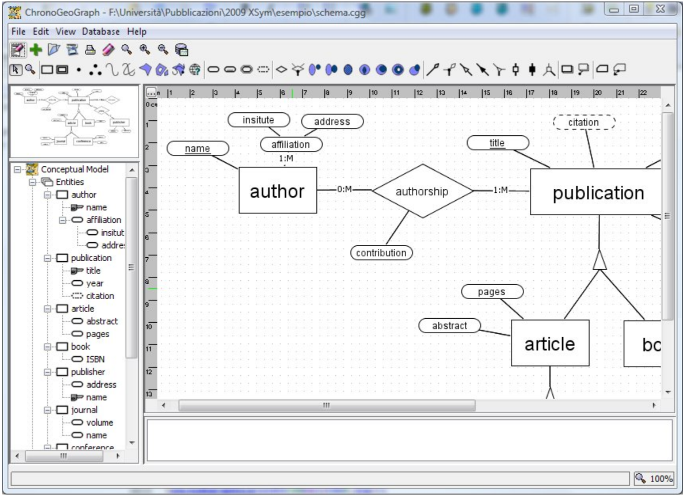
Lemma

Let $G = (V, E)$ be a DAG and $C \subseteq E$ which does not contain confluences. Let $T = \{v \mid \exists(u, v) \in C\}$ and $\overline{G} = (V, \overline{E})$ be such that $(u, v) \in \overline{E}$ iff $(u, v) \in C \vee (v \notin T \wedge (u, v) \in E)$. If $\overline{F}$ is a solution of the maximum depth problem for $\overline{G}$, then $\overline{F}$ is also a solution of both the constrained maximum depth problem and the constraint maximum density problem for G and C.

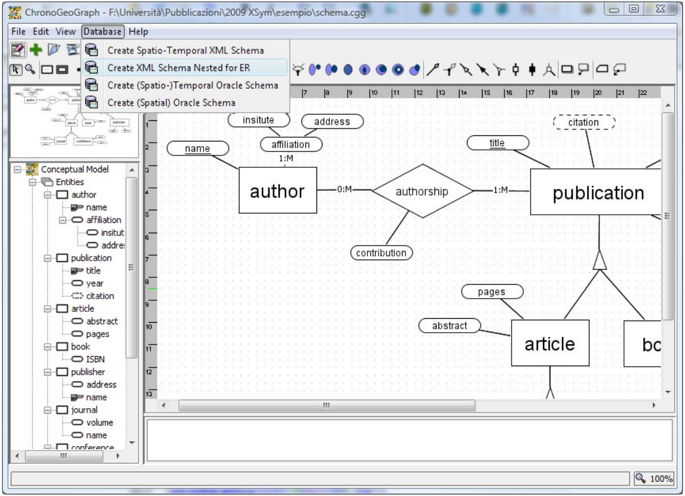
Theorem

Let $G = (V, E)$ be a DAG and let $C \subseteq E$. Constrained_Maximum_Density solves the constrained maximum depth problem for G and C in linear time.

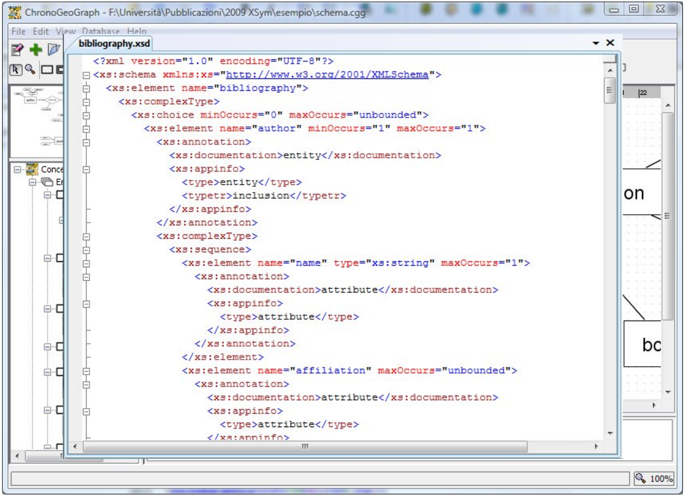
ChronoGeoGraph: the Mapping in Action



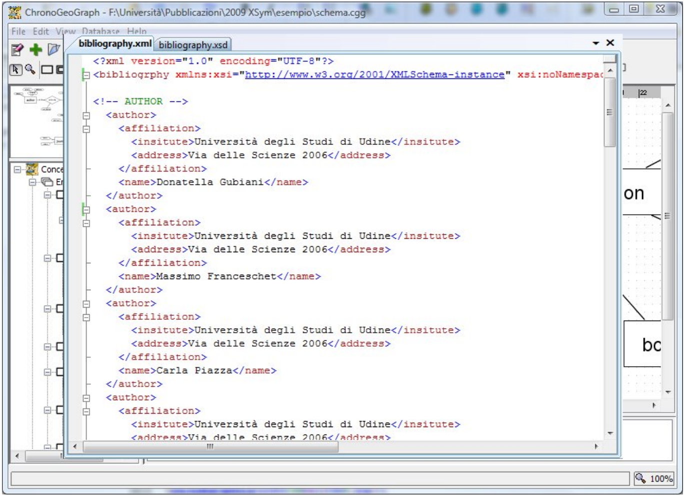
ChronoGeoGraph: the Mapping in Action



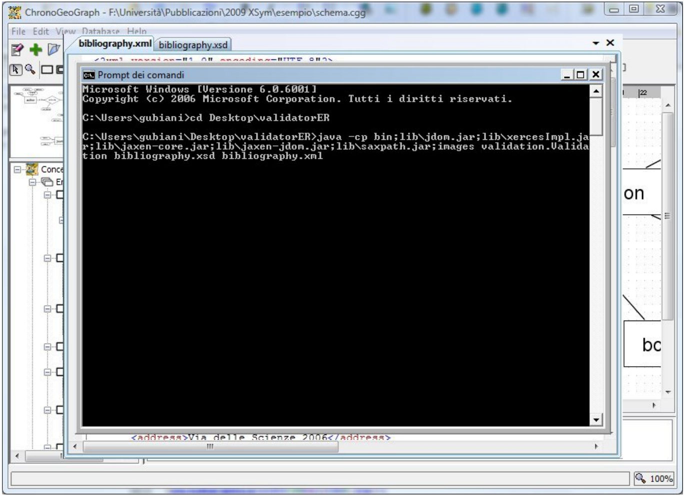
ChronoGeoGraph: the Mapping in Action



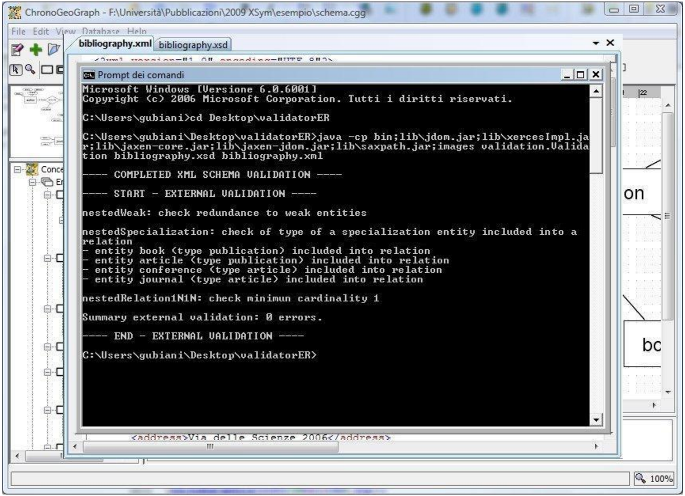
ChronoGeoGraph: the Mapping in Action



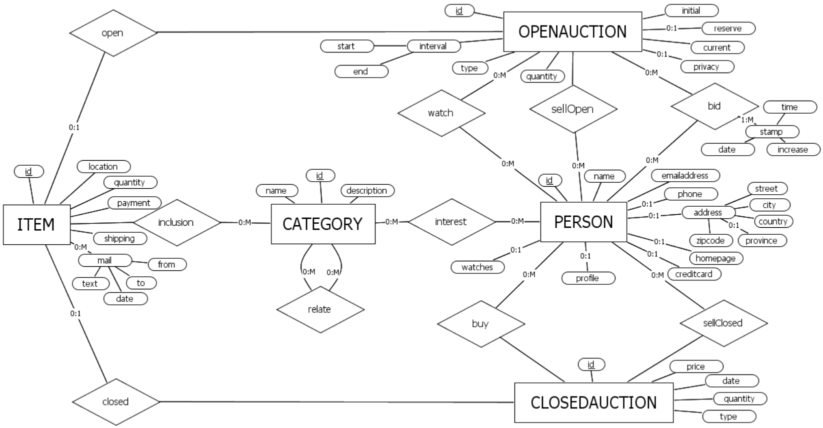
ChronoGeoGraph: the Mapping in Action



ChronoGeoGraph: the Mapping in Action



Experimental Evaluation: XMark Conceptual Design



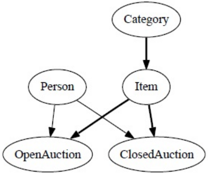
XMark Mapping - Flat

```
// element definitions
site((Category | Person)*)
Category(id, inclusion*, relate*)
  relate(categoryref)
  inclusion(Item)
    Item(id, open?, closed?)
      open(OpenAuction)
        OpenAuction(id, sellOpen, bid*)
          sellOpen(personref)
          bid(personref, stamp)
            stamp(date, time, increase)
      closed(ClosedAuction)
        ClosedAuction(id, buy, sellClosed)
          buy(personref)
          sellClosed(personref)
Person(id, interest*, watch*)
  interest(categoryref)
  watch(openauctionref)
```

```
// key constraints
KEY(Category.id)
KEY(Item.id)
KEY(OpenAuction.id)
KEY(ClosedAuction.id)
KEY(Person.id)

// foreign key constraints
KEYREF(sellOpen.personref --> Person.id)
KEYREF(bid.personref --> Person.id)
KEYREF(buy.personref --> Person.id)
KEYREF(sellClosed.personref --> Person.id)
KEYREF(interest.categoryref --> Category.id)
KEYREF(watch.openauctionref --> OpenAuction.id)
KEYREF(relate.categoryref --> Category.id)
```

XMark Mapping - Nest



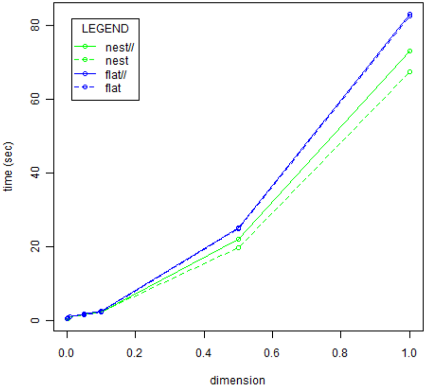

```
// element definitions
site((Category|Item|Person|OpenAuction|ClosedAuction)*)
OpenAuction(id, open, sell, bid*)
 open(itemref)
 sellOpen(personref)
 bid(personref, stamp)
   stamp(date, time, increase)
ClosedAuction(id, closed, buy, sell)
 closed(itemref)
 buy(personref)
 sellClosed(personref)
Item(id, inclusion)
 inclusion(categoryref)
Category(id, relate*)
  relate(categoryref)
Person(id, interest*, watch*)
  interest(categoryref)
  watch(openauctionref)
// key constraints
KEY(OpenAuction.id)
KEY(ClosedAuction.id)
KEY(open.itemref)
KEY(closed.itemref)
KEY(Item.id)
KEY(Category.id)
KEY(Person.id)
// foreign key constraints
KEYREF(open.itemref --> Item.id)
KEYREF(closed.itemref --> Item.id)
KEYREF(inclusion.categoryref --> Category.id)
KEYREF(relate.categoryref --> Category.id)
KEYREF(interest.categoryref --> Category.id)
```

XMark Instance

- The XMark benchmark includes a scalable data generator that produces well-formed, meaningful XML documents that are valid with respect the XMark schema
- We mapped these XML instances into corresponding instances for the nested and flat designs, using Java classes that we coded

Validation Performance

scale	flat//	nest//	flat	nest
0.001	0.41	0.36	0.39	0.38
0.005	0.65	0.57	0.63	0.69
0.010	0.96	0.90	0.86	0.90
0.050	1.64	1.61	1.59	1.45
0.100	2.53	2.15	2.31	2.27
0.500	25.01	21.99	24.77	19.66
1.000	83.09	73.22	82.62	67.46



Query Evaluation

- We testes four different queries on three open-source XML query engines:
  - **BaseX** (version 6): a native XML database
  - **Saxon** (release B 9.1.0.8 for Java): a native processor for XSLT and XQuery
  - **MonetDB/XQuery** (release 4): a XML-enabled database which maps XML into the relational data model
- We ran all experiments on a 2.53 GHz machine with 2.9 GB of main memory running Ubuntu 9.10 operating system

Query 1

Categories and the items they contain.

FLAT

```
let $doc := doc("xmark.xml")
for $category in $doc/site/Category
for $item in $doc/site/Item
where $item/inclusion/categoryref = $category/id
return
<result>
 {$category/id}
 {$item/id}
</result>
```

NEST

```
let $doc := doc("xmark.xml")
for $category in $doc/site/Category
for $item in $category/inclusion/Item
return
<result>
 {$category/id}
 {$item/id}
</result>
```

Query 2

Categories and the open auctions bidding items belonging to these categories.

FLAT

```
let $doc := doc("xmark.xml")
for $category in $doc/site/Category
let $item := for $i in $doc/site/Item
        where $i/inclusion/categoryref=$category/id
        return $i
for $auction in $doc/site/OpenAuction
where $auction/open/itemref = $item/id
return
<result>
 {$category/id}
 {$auction/id}
</result>
```

NEST

```
let $doc := doc("xmark.xml")
for $category in $doc/site/Category
for $auction in
   $category/inclusion/Item/open/OpenAuction
return
<result>
 {$category/id}
 {$auction/id}
</result>
```

Query 3

The open and corresponding closed auctions.

FLAT

```
let $doc := doc("xmark.xml")
for $open in $doc/site/OpenAuction
for $closed in $doc/site/ClosedAuction
where $closed/closed/itemref = $open/open/itemref
return
<result>
 {$open/id}
 {$closed/id}
</result>
```

NEST

```
let $doc := doc("xmark.xml")
for $open in $doc//OpenAuction
for $closed in $open/ancestor::Item//ClosedAuction
return
<result>
 {$open/id}
 {$closed/id}
</result>
```

Query 4

People and the closed auctions bidding items bought by these people.

FLAT

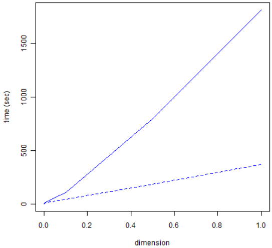
```
let $doc := doc("xmark.xml")
for $people in $doc/site/Person
for $auction in $doc/site/ClosedAuction
where $auction/buy/personref = $people/id
return
<result>
 {$people/id}
 {$auction/id}
</result>
```

NEST

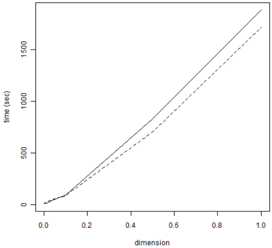
```
let $doc := doc("xmark.xml")
for $people in $doc//Person
for $auction in $doc//ClosedAuction
where $auction/buy/personref = $people/id
return
<result>
 {$people/id}
 {$auction/id}
</result>
```

Query Evaluation: BaseX

BaseX	Q1		Q2		Q3		Q4	
scale	nest	flat	nest	flat	nest	flat	nest	flat
0.001	0.00	0.01	0.00	0.01	0.00	0.00	0.02	0.01
0.005	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.01
0.010	0.01	0.01	0.01	0.02	0.01	0.01	0.02	0.02
0.050	0.04	0.04	0.03	0.06	0.01	0.02	0.06	0.05
0.100	0.07	0.06	0.05	0.11	0.02	0.02	0.09	0.10
0.500	0.28	0.29	0.19	0.80	0.08	0.22	0.70	0.83
1.000	0.55	0.58	0.37	1.81	0.15	0.56	1.71	1.88



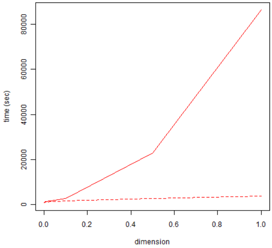
Q2



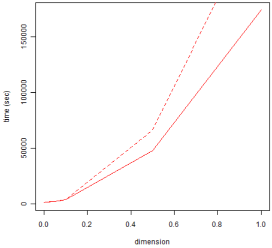
Q4

Query Evaluation: Saxon

Saxon	Q1		Q2		Q3		Q4	
scale	nest	flat	nest	flat	nest	flat	nest	flat
0.001	0.89	0.91	0.88	0.89	0.87	0.86	0.91	0.86
0.005	1.03	1.05	1.06	1.10	1.00	1.16	1.18	1.22
0.010	1.15	1.32	1.14	1.41	1.15	1.49	1.59	1.56
0.050	1.61	1.90	1.63	2.08	1.54	2.07	2.30	2.29
0.100	1.64	2.21	1.87	2.58	1.81	2.69	3.75	3.59
0.500	2.54	7.07	2.88	14.52	2.80	22.86	65.85	47.70
1.000	3.50	19.99	4.10	46.23	3.90	86.15	264.34	173.95



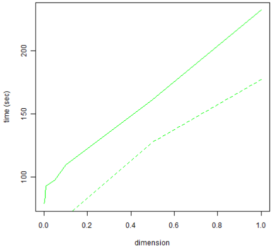
Q3



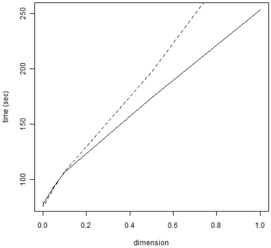
Q4

Query Evaluation: MoneDB/XQuery

MDB/XQ	Q1		Q2		Q3		Q4	
scale	nest	flat	nest	flat	nest	flat	nest	flat
0.001	0.05	0.08	0.06	0.12	0.04	0.08	0.03	0.03
0.005	0.04	0.08	0.06	0.12	0.04	0.08	0.03	0.03
0.010	0.04	0.08	0.06	0.12	0.06	0.09	0.04	0.04
0.050	0.04	0.09	0.06	0.13	0.06	0.10	0.04	0.05
0.100	0.05	0.09	0.07	0.14	0.07	0.11	0.05	0.05
0.500	0.05	0.18	0.10	0.22	0.13	0.16	0.10	0.09
1.000	0.05	0.25	0.15	0.34	0.18	0.23	0.16	0.15



Q3



Q4