Distance based learning on relational algebra representations

Artificial Intelligence Group
Department of Computer Science
University of Geneva

Introduction

The presentation consists of the following parts:

- Give the representation formalism
- Show how we can define distances over relational structures using:
  - Classical distances defined over tuples and sets of tuples
  - Kernels on tuples and sets of tuples
- Present some results
- Conclude

Why relational algebra?

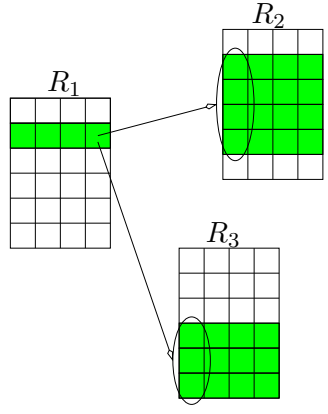
- There is a huge amount of information stored in relational databases
- No need for representation conversion which is not always a trivial task
- Very natural to model sets of objects
- Provides a robust data modeling tool with well understood semantics from a big audience (database practitioners)
- Can increase the acceptance of relational learning by the database community

Relational algebra notation (I)

- *Relational database schema*: a set of relations $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$
- A relation R_i is a set of tuples, more specifically a subset of some Cartesian product $D_1 \times \ldots \times D_{z_i}$ where D_l is the domain of attribute A_l, (i.e., a set of values-data type).
- *Tuple* R_{i_j} of a relation R_i is a relationship between a set of values: $R_{i_j} = (v_{j1}, v_{j2}, \ldots, v_{jz_i})$ and v_{jl} the value of the A_l attribute in the R_{i_j} tuple.
- *Potential key*: an attribute A_k of a relation R_i that assumes a unique value for each tuple in the relation.
- *Foreign key*: an attribute A_l of a relation R_j that references a potential key A_k of relation R_i.

Relational algebra notation (II)

- *One to many relations*: The association between A_k and A_l models one-to-many relation, i.e. one element of R_i can be associated with a *set* of elements of R_j.
- A *link* is defined on the basis of a foreign key relation and is a quadruple of the form $l(R_i, A_k, R_j, A_l)$ where either A_l is a foreign key of R_j referencing a potential key A_k of R_i or vice versa. $L(R_i)$ is the set of links in which R_i participates.

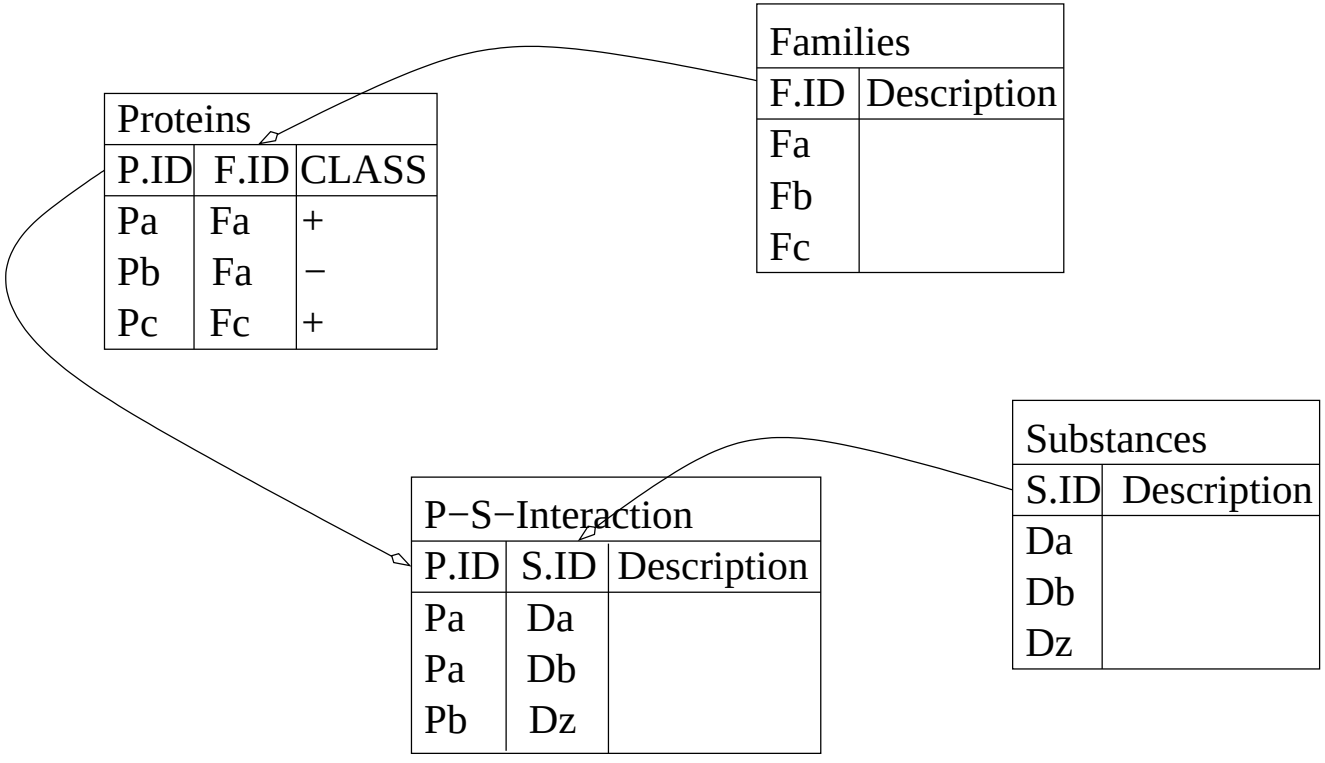


- *Standard attributes*: Attributes of a relation R_i that are not keys (i.e. referenced keys, foreign keys or attributes defined as keys but not referenced) The set of standard attributes will be denoted as $\mathcal{I}_{A,R_i}$.
- *Set attributes*: Each link of $L(R_i)$ defines an attribute of type set. Its values are sets of tuples from a given relation. The set of attributes of type set will be denoted as $\mathcal{I}_{L(R_i),R_i}$.

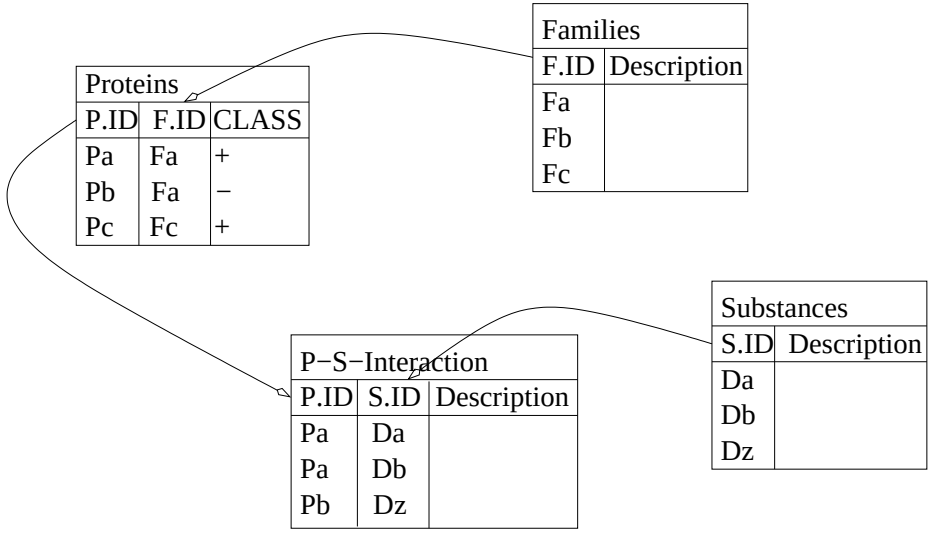
Relational instances

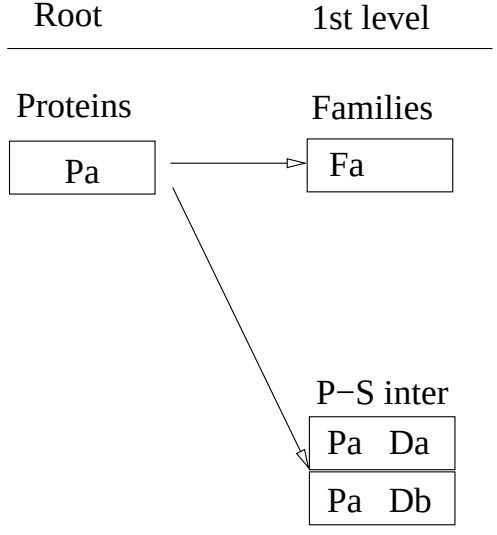
- A relational instance could span through all the relations of a given relational database.
- We assume that it exists a main relation, M, where the class of each relational instance is given.
- To retrieve its description we should follow all the links in which it participates.
- Following the links might result in loops and thus replication of information.
- The resulting representation is a tree like structure where nodes contain tuples from different relations.

An example

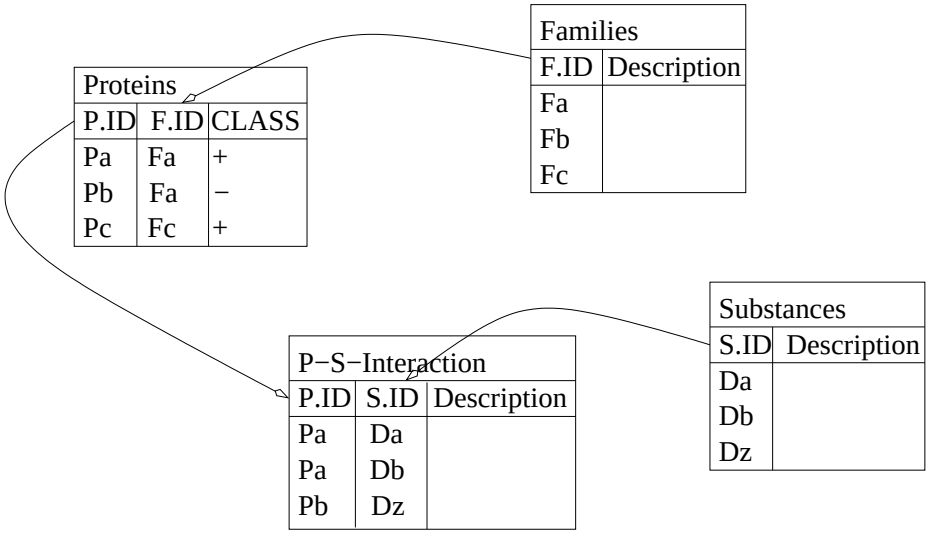

Families
F.ID
Description
Fa
Fb
Fc
Proteins
P.ID
F.ID
CLASS
Pa
Fa
+
Pb
Fa
−
Pc
Fc
+
Substances
S.ID
Description
Da
Db
Dz
P−S−Interaction
P.ID
S.ID
Description
Pa
Da
Pa
Db
Pb
Dz

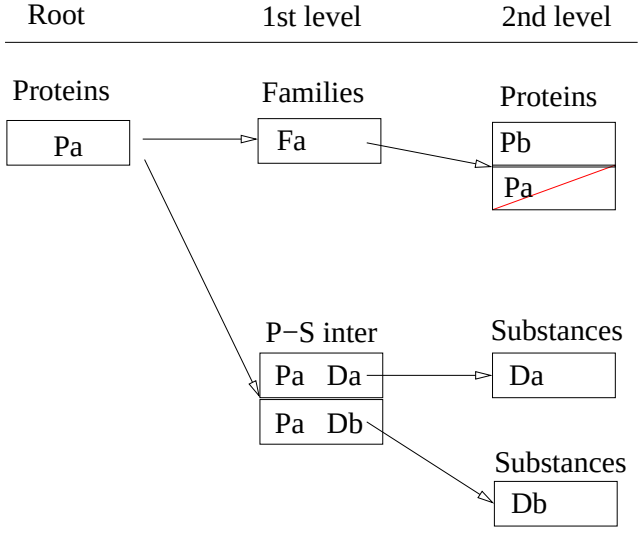
An example (I)


Families
F.ID Description
Fa
Fb
Fc
Proteins
P.ID F.ID CLASS
Pa Fa +
Pb Fa −
Pc Fc +
Substances
S.ID Description
Da
Db
Dz
P−S−Interaction
P.ID S.ID Description
Pa Da
Pa Db
Pb Dz

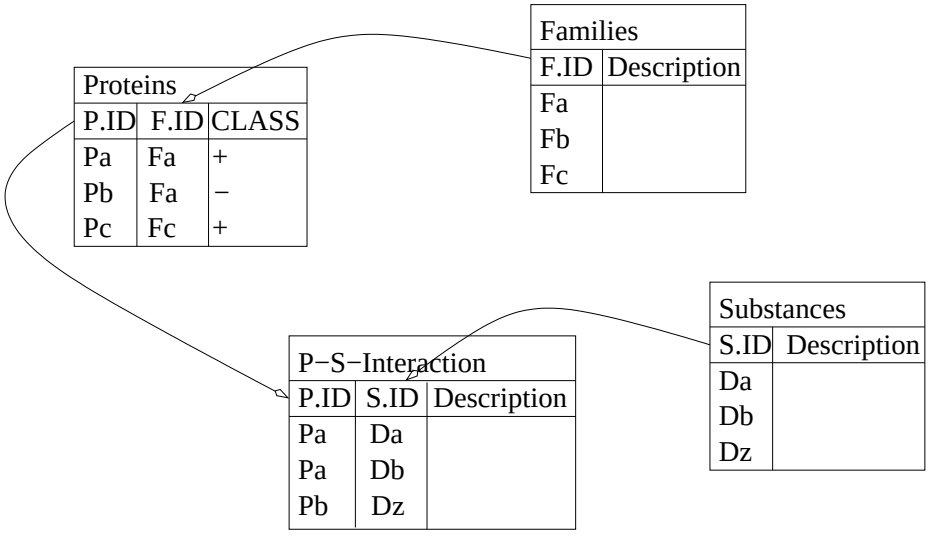

Root
1st level
Proteins
Families
Pa
Fa
P−S inter
Pa Da
Pa Db

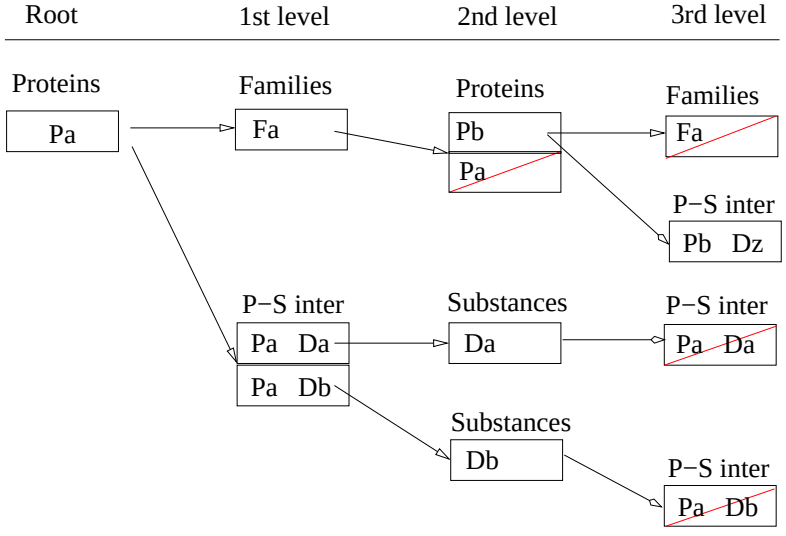
An example (II)



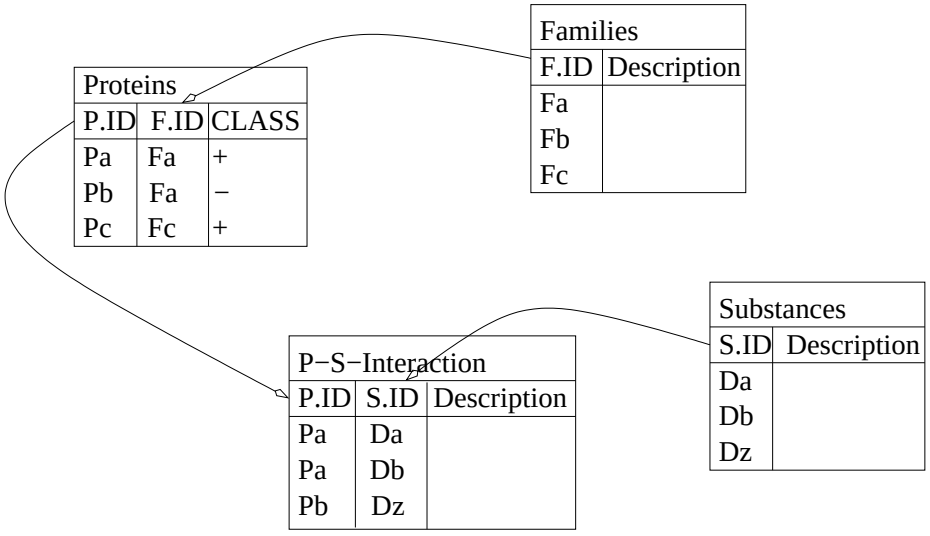


An example (III)





An example (IV)



Families

F.ID	Description
Fa	
Fb	
Fc	

Proteins

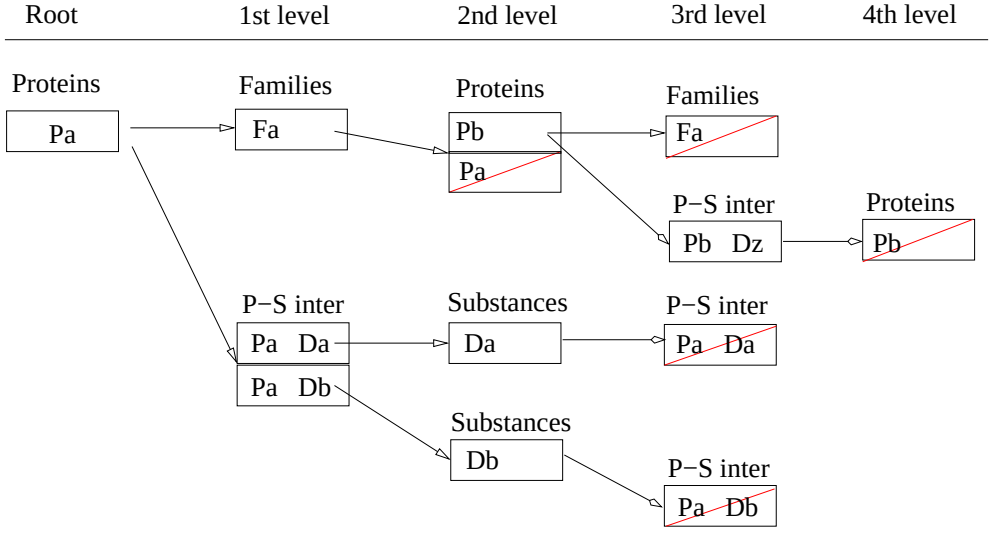
P.ID	F.ID	CLASS
Pa	Fa	+
Pb	Fa	−
Pc	Fc	+

Substances

S.ID	Description
Da	
Db	
Dz	

P−S−Interaction

P.ID	S.ID	Description
Pa	Da	
Pa	Db	
Pb	Dz	



Operators on relational instances

- Our building blocks are tuples of relations and sets of tuples.
- We have to define operators on both of them.
- These operators will be combined in order to define an operator on the full relational instance.

In the following we will see how we can define distance operators using classical euclidean distances and kernels.

The term distance is used somehow abusively, the final operator, depending on the properties of its constituents, can be a dissimilarity, a distance, a pseudo-metric or a metric.

Classical distances on relational algebra structures

- Remember here that the attributes of a tuple of a relation R_i consist of two parts:
  - the standard attributes, $\mathcal{I}_{A,R_i}$,
  - the attributes of type set, $\mathcal{I}_{L(R_i),R_i}$.
- We need:
  - A distance, $D(R_{i_a}, R_{i_b})$, between tuples R_{i_a}, R_{i_b}, of relation R_i,
  - A distance between sets of tuples of R_i.

Distance over tuples of a relation R_i

The distance, $D(R_{i_a}, R_{i_b})$, between tuples R_{i_a}, R_{i_b} of a relation R_i is based on the euclidean distance:

$$\begin{aligned} D(R_{i_a}, R_{i_b}) &= \sqrt{\frac{\Sigma_{k=1}^{N} d^2(v_{ak}, v_{bk})}{N}} \\ &= \sqrt{\frac{1}{N}} \sqrt{\sum_{k=1}^{|\mathcal{I}_{A,R_i}|} d^2(v_{ak}, v_{bk}) + \sum_{k=1}^{|\mathcal{I}_{L(R_i),R_i}|} d^2(v_{ak}, v_{bk})} \end{aligned}$$

- v_{ak}, v_{bk}: the values of the R_{i_a}, R_{i_b}, for attribute A_k.
- The sum runs over all standard and set attributes of R_i, thus $N = |\mathcal{I}_{A,R_i}| + |\mathcal{I}_{L(R_i),R_i}|$.
- The functionality of $d(.,.)$ depends on the type of its arguments.

Distances over sets of tuples of a relation R_i

- Consider two sets:
  - $v_{ak} = \{v_x\} \subseteq R_i$,
  - $v_{bk} = \{v_y\} \subseteq R_i$ then:
- Let $d(.,.)$ be a distance measure defined on R_i.
- The set distance measure D defined on 2^{R_i} as:

$$D \ : \ 2^{R_i} \times 2^{R_i} \rightarrow R_0^+, \ D(v_{ak}, v_{bk}) = f(\{d(v_x, v_y) | (v_x, v_y) \in v_{ak} \times v_{bk}\})$$

  is some function of the pairwise distances, $d(v_x, v_y)$, of the set of all pairs $(v_x, v_y) \in v_{ak} \times v_{bk}$.
- v_{ak} and v_{bk} should be non-empty and finite sets.

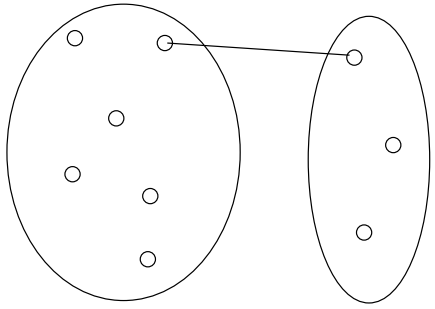
Distances over sets of tuples of a relation R_i (cont.)

- A number of possibilities with different semantics and formal properties.

Set Distance	Reflexive	Symmetric	Strict	Triangle inequality	type
Single Linkage	+	+	-	-	distance
Complete Linkage	-	+	-	-	-
Average Linkage	-	+	-	-	-
Sum of minimum distances	+	+	-	-	distance
RIBL	+	-	-	-	dissimilarity
Hausdorff	+	+	+	+	metric
Tanimoto	+	+	-	+	pseudo-metric
Surjections	+	+	+	-	pseudo-metric
Linkings	+	+	+	-	pseudo-metric
Fair Surjections	+	+	+	-	pseudo-metric
Matchings	+	+	+	+	metric

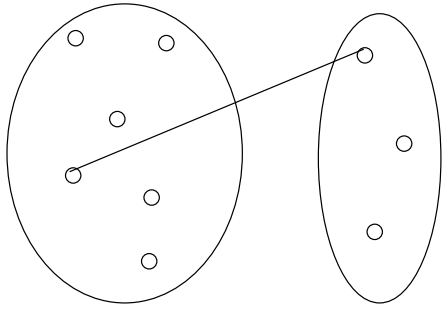
- The central idea is the definition of a mapping of the elements of one set to elements of the other set.
- The choice of a set distance for a given relation should reflect domain knowledge.

Examples of simple mappings between sets



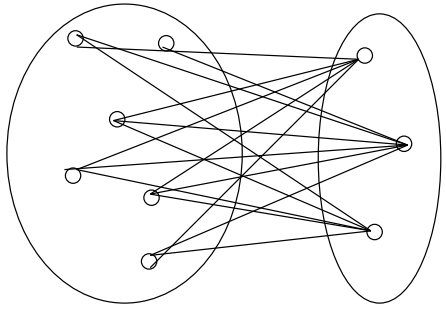
Single Linkage:

$$d(v_{ak}, v_{bk}) = \min_{(v_x, v_y) \in v_{ak} \times v_{bk}} D(v_x, v_y)$$



Complete Linkage:

$$d(v_{ak}, v_{bk}) = \max_{(v_x, v_y) \in v_{ak} \times v_{bk}} D(v_x, v_y)$$



Average Linkage:

$$d(v_{ak}, v_{bk}) = \frac{1}{|v_{ak}||v_{bk}|} \sum_{(v_x, v_y) \in v_{ak} \times v_{bk}} D(v_x, v_y)$$

Examples of distances based on families of mappings between sets

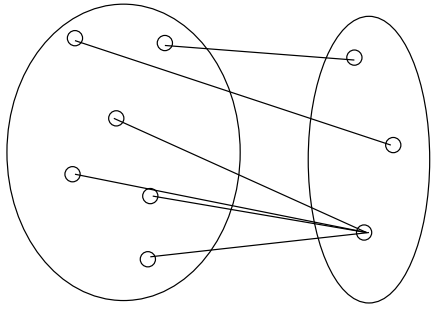
- Each of the distances bellow works on a specific family of mappings $M \subseteq v_{ak} \times v_{bk}$.

- The final distance is given by:

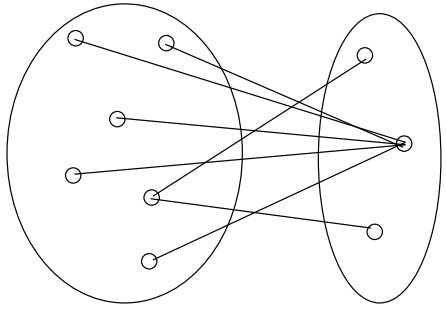
$$D(A, B) = \min_{M_i \in M} \sum_{(v_x, v_y) \in M_i} d(v_x, v_y)$$

  i.e. the mapping that minimizes the sum of distances.

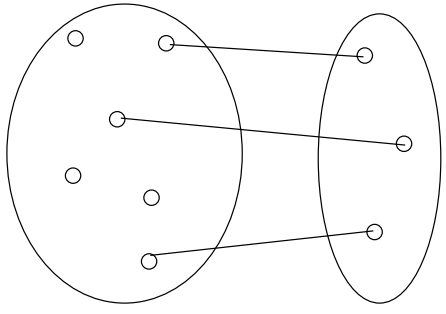
Examples of distances based on families of mappings between sets



Surjections, M is the set of all the possible surjections of the larger to the smaller set.



Linkings, M is the set of all possible linkings. A linking is a mapping of one set to the other where each element of a set participates in at least one pair of the mapping.



Matchings, M is the set of all possible matchings. In a matching each element of the two sets is associated with at *most* one element of the other set. There is a penalty for the elements that are not matched.

Distance over relational instances

- Let a, b, be two relational instances defined in the main relation M then:

$$D(M_a, M_b) = \sqrt{\frac{1}{N}} \sqrt{\sum_{k=1}^{|\mathcal{I}_{A,M}|} d^2(v_{ak}, v_{bk}) + \sum_{k=1}^{|\mathcal{I}_{L(M),M}|} d^2(v_{ak}, v_{bk})}$$

- Obviously v_{ak} and v_{bk} can be sets of tuples from a relation R_i with which M is related:
  - To compute all the pairwise distances between elements of v_{ak} and v_{bk}, we should use again $D(.,.)$ on the tuples of R_i.
  - Computation is recursive with alternating applications of $D(.,.)$ and $d(.,.)$
- Formal properties of $D(.,.)$ are determined by the formal properties of the set distances.
- As we move from the main relation information is given less importance (good or bad?)

A fast glance at kernels

- Kernel: a symmetric function $K : X \times X \rightarrow \mathbb{R}^+$
  - X does not have to be a vector space, can be any space
  - $\forall x, y \in X,\ K(x, y) =< \phi(x), \phi(y) >$
  - ϕ a mapping from X to some feature space endowed with an inner product
- The beauty of kernels is that they compute directly the inner product in the feature space without computing the mapping.
- any kernel induces a pseudo-metric in the feature space via:

$$d(\phi(x), \phi(y)) = \sqrt{K(x, x) - 2K(x, y) + K(y, y)}$$

Kernels on relational algebra structures

Exactly the same ideas as in distances. We need:

- A kernel, $K(R_{i_a}, R_{i_b})$ between tuples R_{i_a}, R_{i_b} of relation R_i.
- A kernel between sets of tuples of R_i.

Kernels over tuples of a relation R_i

We will introduce some extra notation, let

$$v_{a\mathcal{I}_{A,R_i}} = (v_{a1}, v_{a2}, \ldots, v_{a|\mathcal{I}_{A,R_i}|})$$

be the vector of standard attributes of tuple R_{i_a}. Then we have two alternatives for $K(.,.)$:

- The direct sum kernel:

$$K_{\sum}(R_{i_a}, R_{i_b}) = \frac{1}{1 + |\mathcal{I}_{L(R_i),R_i}|}(k(v_{a\mathcal{I}_{A,R_i}}, v_{b\mathcal{I}_{A,R_i}}) + \sum_{k=1}^{|\mathcal{I}_{L(R_i),R_i}|} k(v_{ak}, v_{bk}))$$

- The $\mathcal{R}$ convolution kernel:

$$K_{\mathcal{R}}(R_{i_a}, R_{i_b}) = k(v_{a\mathcal{I}_{A,R_i}}, v_{b\mathcal{I}_{A,R_i}}) \prod_{k=1}^{|\mathcal{I}_{L(R_i),R_i}|} k(v_{ak}, v_{bk})$$

  and normalized in feature space:

$$K_{\mathcal{R}}(R_{i_a}, R_{i_b}) = \frac{K_{\mathcal{R}}(R_{i_a}, R_{i_b})}{\sqrt{K_{\mathcal{R}}(R_{i_a}, R_{i_a}), K_{\mathcal{R}}(R_{i_b}, R_{i_b})}}$$

Again functionality of $k(.,.)$ depends on the type of its arguments.

A kernel over sets of tuples of a relation R_i

- Similar to set distances a kernel on sets is defined on the basis of all the pairwise kernels of the tuples of the two sets.
- Unlike set distances we have a single valid choice, if $v_{ak} = \{v_x\} \subseteq R_i, v_{bk} = \{v_y\} \subseteq R_i$

$$k(v_{ak}, v_{bk}) = \sum_{v_x, v_y} K_{\mathcal{R} \vee \Sigma}(v_x, v_y)$$

- Again the issue of normalization rises, two possibilities:
  - Averaging:

$$k(v_{ak}, v_{bk}) = \frac{k(v_{ak}, v_{bk})}{|v_{ak}||v_{bk}|}$$

  - Feature space normalization:

$$k(v_{ak}, v_{bk}) = \frac{k(v_{ak}, v_{bk})}{\sqrt{k(v_{ak}, v_{ak}) k(v_{bk}, v_{bk})}}$$

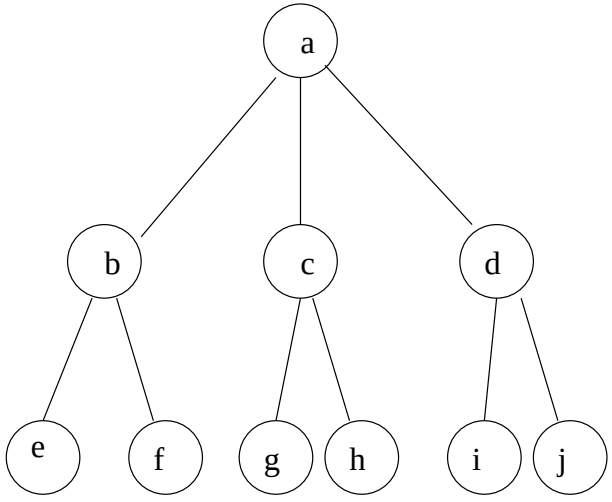
Kernels over relational instances

In complete analogy with the distances the kernel $K(a, b)$ will be computed recursively by alternating computations of $k(.,.)$ and $K(.,.)$.

What kind of structures can be modelled?

- The system can model almost everything as long as you are able to fit it in a relational representation.
- Obviously the most easy to model are problems where instances are sets.
- Although not described here we have extended the system so that it can model lists.
- Trees can be modeled, either ordered or unordered, with not to much effort.
- Graphs can be also modelled with a little bit more effort.

How to model unordered trees?



Trees		
parent.id	node.id	other attrs describing node
?	a	...
a	b	...
b	e	...
b	f	...
a	c	...
c	g	...
c	h	...
a	d	...
d	i	...
d	j	...

- *node.id* is the primary key of the table Trees
- *parent.id* is a foreign key pointing to the *node.id* key.
- Important observation: In the standard algorithm the importance of the information found lower in the recursion is smaller. This might not be appropriate when you are modeling trees. One solution is to drop the normalization by the cardinalities.

Now you can also model sets of trees

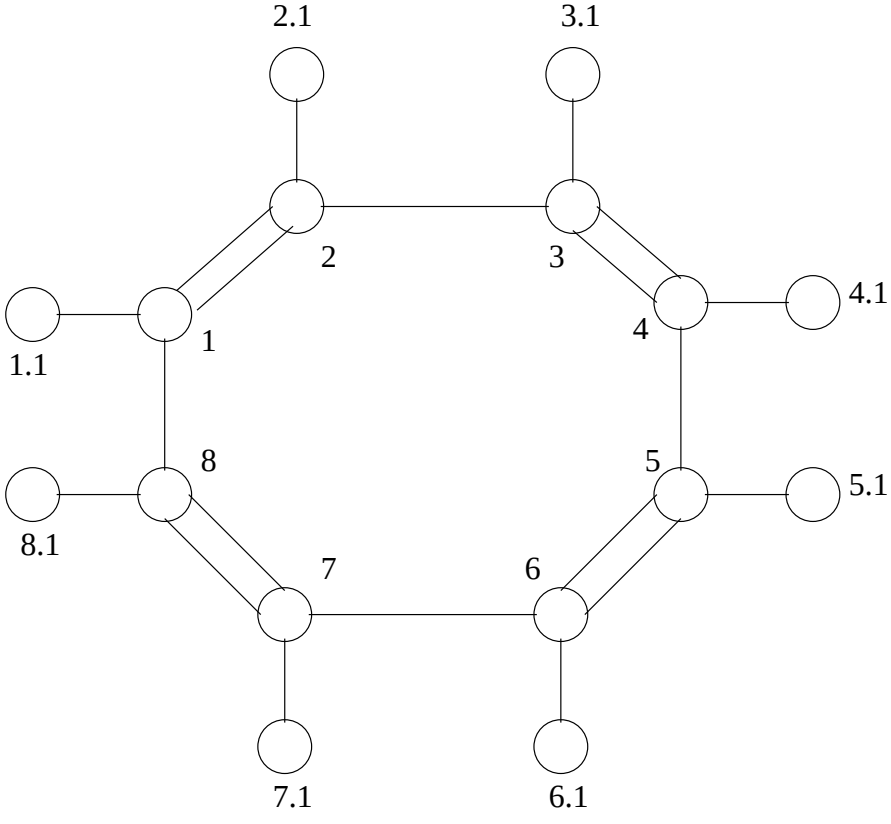
Trees		
parent.id	node.id	other attrs describing node
?	a	...
a	b	...
b	e	...
b	f	...
a	c	...
c	g	...
c	h	...
a	d	...
d	i	...
d	j	...

instances		
instance.id	instance description	tree.id
instance.1	...	root.a
instance.1	...	root.b
...	...	...
instance.n	...	root.z

- *instance.id* is the primary key of the instances table
- *tree.id* is a foreign key pointing to the Trees table at the root node (*node.id*) of the corresponding tree.

Modelling Graphs

Typically problems that appear in biology and chemistry where a molecule is a graph.



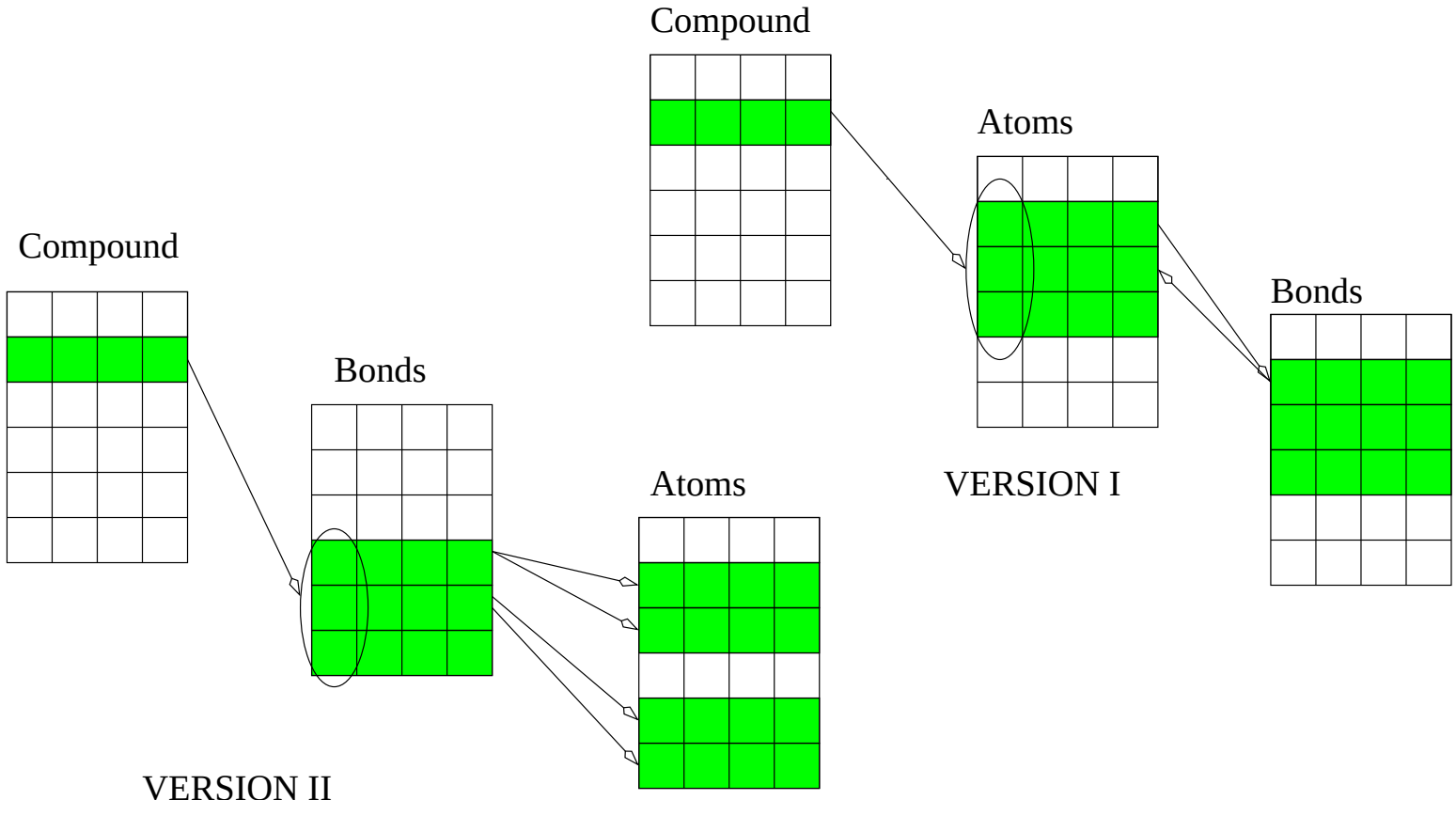
Modelling Graphs

Many options, model the graph as:

- a set of atoms, put an extra table describing the bonds between atoms.
- a set of bonds, model a bond as a set of atoms (actually two atoms).
- a set of trees, each tree starts at one of the atoms of the graph.
- a set of paths, infeasible with standard distances there are exponentially many possible paths.

Which is the best? This is an issue of active research, and probably something to take a look at your project.

Mutagenicity dataset



Comments on results

- Kernels seem to have a disadvantage compared to distances, two possible explanations:
  - primitive handling of sets,
  - grouping of standard attributes unlike in the distances
- Kernels are less easier to use than standard distances:
  - many different possible kernels on the vectors of standard attributes
  - they can have a number of parameters
  - different normalization possibilities

  nevertheless the kernels on the relational structures seem to be tolerant to different normalization methods and parameter settings.
- Results similar or better than state of the art systems

Relation to the ILP framework

In terms of data representation there is a direct mapping of relational algebra concepts to Logic Programming used extensively in Inductive Logic Programming (Dzeroski and Lavrac, Relational Data Mining, 2001).

- a relation name R_i is a predicate symbol R_i,
- a relation R_i, i.e. a set of tuples, corresponds to a predicate R_i defined extensionally as a collection of ground facts,
- an attribute A_l of the relation R_i is an argument of the predicate R_i,
- a tuple R_{i_j} corresponds to the j^{th} ground fact built on the predicate R_i,

BUT:

- LP can not handle relations between objects, ILP copes with that using typed LP.
- LP cannot handle sets.
- In ILP one has to specify how variables should be instantiated (modes declaration).

The above require significant effort and care when doing a change of representation from a relational database to a representation that is used in a typical ILP system. Within relational algebra this change is simply not necessary.

Conclusions

- Novel and general framework based on relational algebra for learning over relational schemata
- Introducing the concept of attributes of type set based on the notion of foreign keys
- Tree-like structure representation of learning examples
- Brings multi-relational learning closer to the database community
- Modeling of relational data straightforward
- Natural approach to set modeling fundamental in relational algebra
- Possible to exploit domain knowledge by explicitly defining how sets of tuples of a given relation should be treated, i.e. which set distance should be used.