Leiden University

Efficient Frequent Query Discovery in FARMER

Siegfried Nijssen and Joost N. Kok

ECML/PKDD-2003, Cavtat

Introduction

- Frequent structure mining: given a set of complex structures (molecules, access logs, graphs, (free) trees, ...), find substructures that occur frequently
- Frequent structure mining approaches:
  - **Specialized:** efficient algorithms for sequences, trees (Freqt, uFreqT) and graphs (gSpan, FSG)
  - **General:** ILP algorithms (Warmr), biased graph mining algorithms (B-AGM)

Introduction

- [Yan, SIGKDD'2003]
  Comparison between gSpan and WARMR on confirmed active Aids molecules:

6400s	WARMR
2s	gSpan

- Our goal: to build an *efficient* WARMR-like algorithm

Overview

- Problem description
- Optimizations:
  - Use a bias for tight problem specifications
  - Perform a depth-first search
  - Use efficient data structures in a new complete enumeration strategy which combines pruning with candidate generation
  - Speed-up evaluation by storing intermediate evaluation results, construct low-cost queries
- Experiments & conclusions

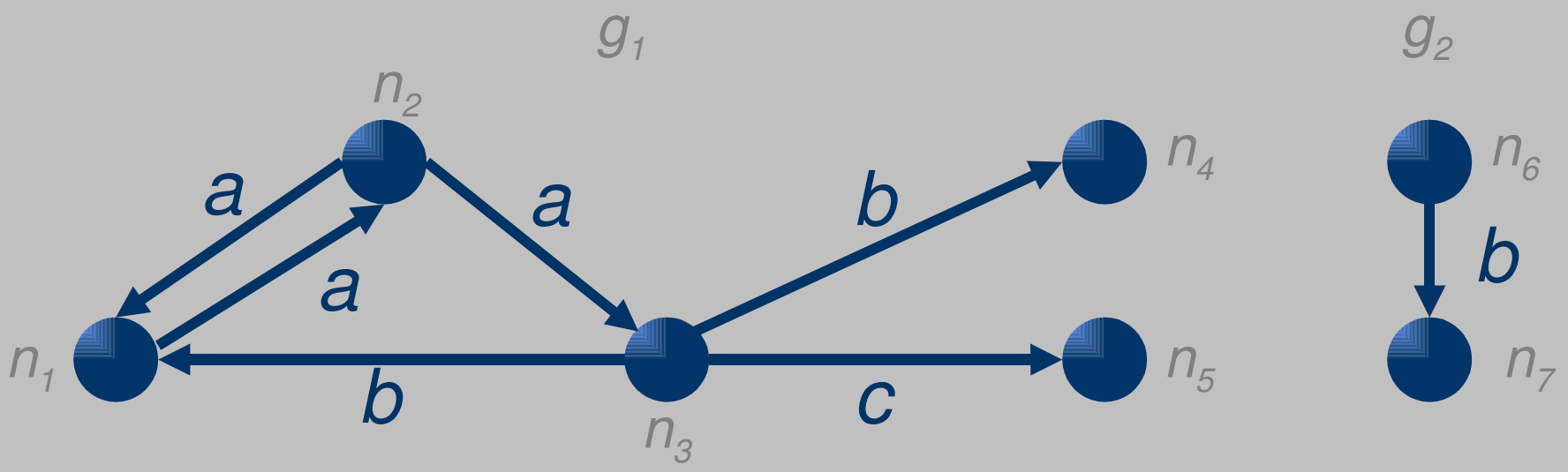
Problem description

- The task of the algorithm is:

Given 1 a database of Datalog facts

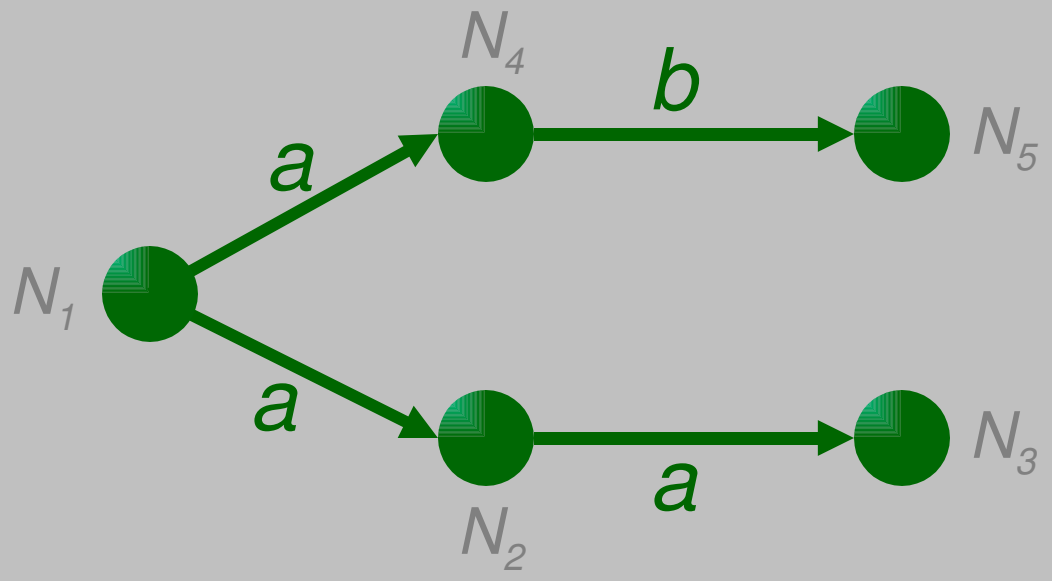
Find 2 a set of queries that 3 occurs 4 frequently

Database of Facts



- $\{e(g_1,n_1,n_2,a),e(g_1,n_2,n_1,a),e(g_1,n_2,n_3,a),$
  $e(g_1,n_3,n_1,b),e(g_1,n_3,n_4,b),e(g_1,n_3,n_5,c),$
  $e(g_2,n_6,n_7,b)\}$

Queries



- $k(G) \leftarrow e(G,N_1,N_2,a),e(G,N_2,N_3,a),$
  $e(G,N_1,N_4,a),e(G,N_4,N_5,b)$

Queries - Bias

- For a fixed set of predicates many kinds of queries possible:
  - $k(G) \leftarrow e(G,N_1,N_2,a),e(G,N_2,N_3,a),$ $e(G,N_1,N_4,a),e(G,N_4,N_5,b)$
  - $k(G) \leftarrow e(G,N_1,N_2,L),e(G,N_2,N_3,L),$ $e(G,N_1,N_4,L),e(G,N_4,N_5,L)$
- Our algorithm requires the user to specify a *mode bias with types, primary keys, atom variable constraints, ...*

Occurrence of Queries

- Database D: $\theta=\{G/g_1,N_1/n_2,N_2/n_1,N_3/n_2,N_4/n_3,N_5/n_1\}$

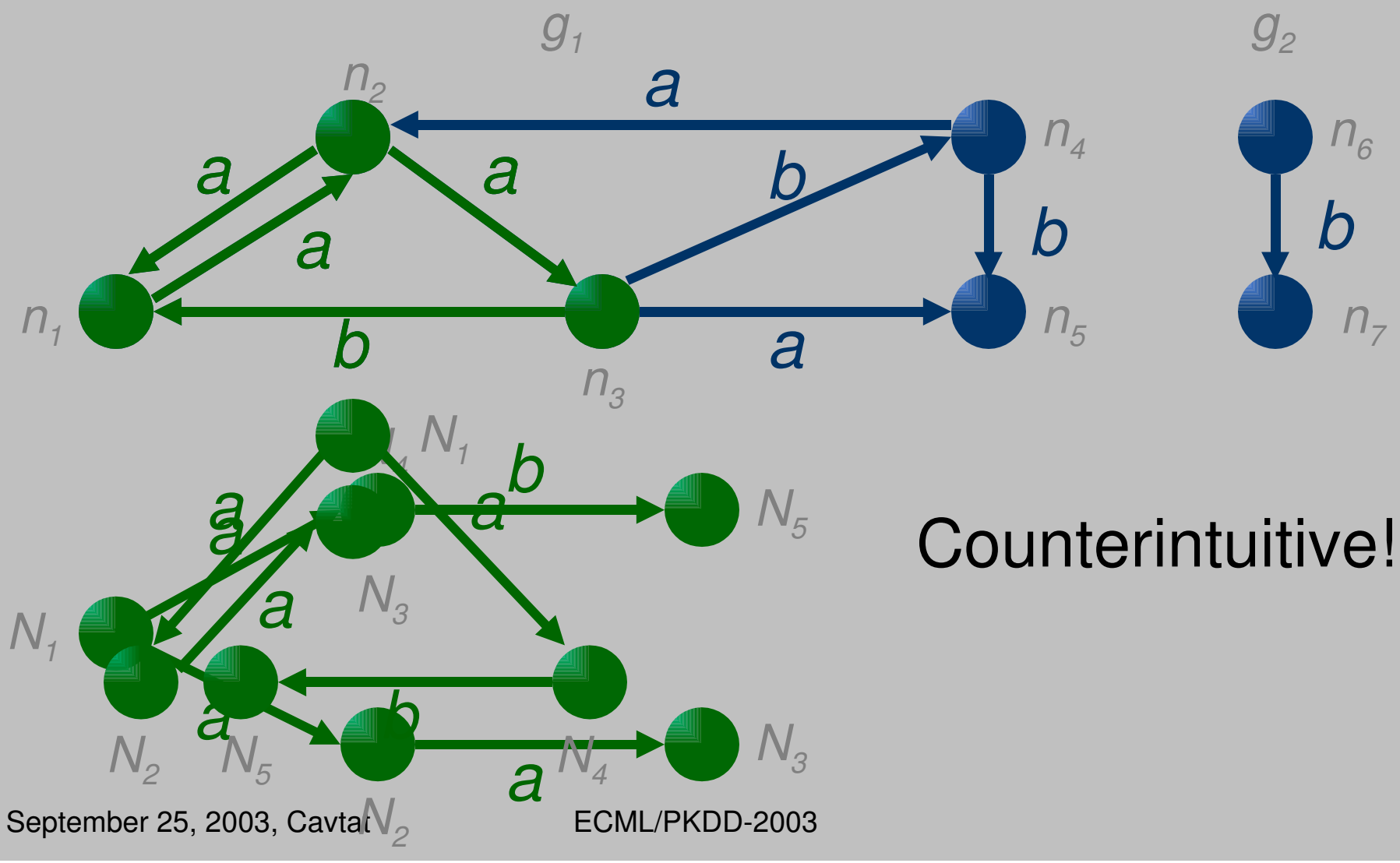
  $\{e(g_1,n_1,n_2,a),e(g_1,n_2,n_1,a),e(g_1,n_2,n_3,a),$
  $e(g_1,n_3,n_1,b),e(g_1,n_3,n_4,b),e(g_1,n_3,n_5,a),$
  $e(g_1,n_4,n_2,a),e(g_1,n_4,n_5,b),e(g_2,n_6,n_7,b)\}$

- Query Q:

  $k(G) \leftarrow e(G,N_1,N_2,a),e(G,N_2,N_3,a),$
  $e(G,N_1,N_4,a),e(G,N_4,N_5,b)$

- (WARMR) θ-subsumption: $D \models Q$ iff there is a substitution θ, $(Q\theta) \subseteq D$

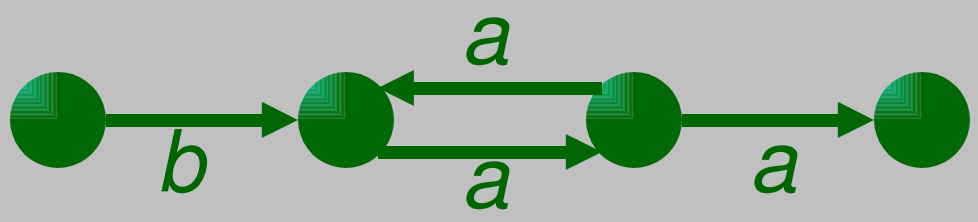
Occurrence of Queries



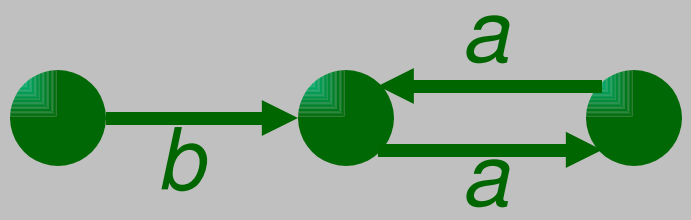
Counterintuitive!

Occurrence of Queries

Equivalent:



$k(G) \leftarrow$
$e(G,N_1,N_2,b),e(G,N_2,N_3,a),$
$e(G,N_3,N_2,a),e(G,N_3,N_4,a)$



$k(G) \leftarrow$
$e(G,N_1,N_2,b),e(G,N_2,N_3,a),$
$e(G,N_3,N_2,a)$

Counterintuitive!

Occurrence of Queries

- (FARMER here) OI-subsumption: $D \models Q$ iff there is a substitution θ, $(Q\theta) \subseteq D$ and*:*
  - θ is injective
  - θ does not map to constants in Q
- Advantages over OI-subsumption:
  - in many situations (eg. graphs) more intuitive
  - if queries are equivalent, they are alphabetic variants; mode refinement is easier (proper)
- Disadvantages?

Frequency

- Database *D*:

  $\{e(g_1,n_1,n_2,a), e(g_1,n_2,n_1,a), e(g_1,n_2,n_3,a),$
  $e(g_1,n_3,n_1,b), e(g_1,n_3,n_4,b), e(g_1,n_3,n_5,a),$
  $e(g_1,n_4,n_2,a), e(g_1,n_4,n_5,b), e(g_2,n_6,n_7,b)\}$

- Query *Q*:

  $k(G) \leftarrow e(G,N_1,N_2,a)$

- Frequency *freq(Q)*: the number of different values for *G* for which the body is subsumed by the database.

Monotonicity

- Frequently: frequency ≥ *minsup*, for predefined threshold value *minsup*
- Monotonicity: if Q_2 OI-subsumes Q_1,

$$freq(Q_1) \geq freq(Q_2)$$

$\Rightarrow$ if a query is infrequent, it should not be refined

$\Rightarrow$ if a query is subsumed by an infrequent query, it should not be considered

FARMER

FARMER(Query *Q*)::

1. determine refinements of *Q*

2. compute frequency of refinements

3. sort refinements

for each frequent refinement *Q'* **do**

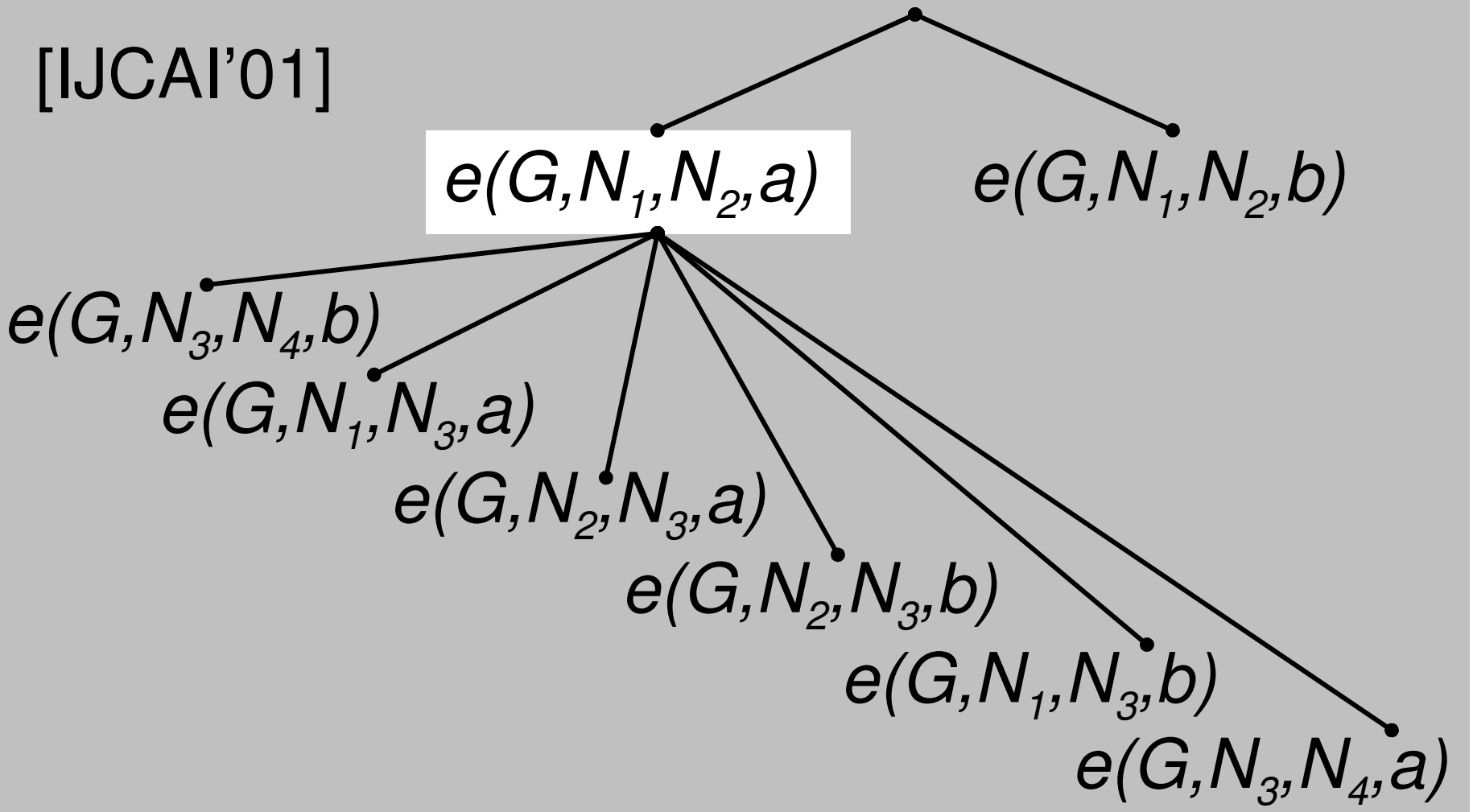
FARMER*(Q')*

Determine Refinements

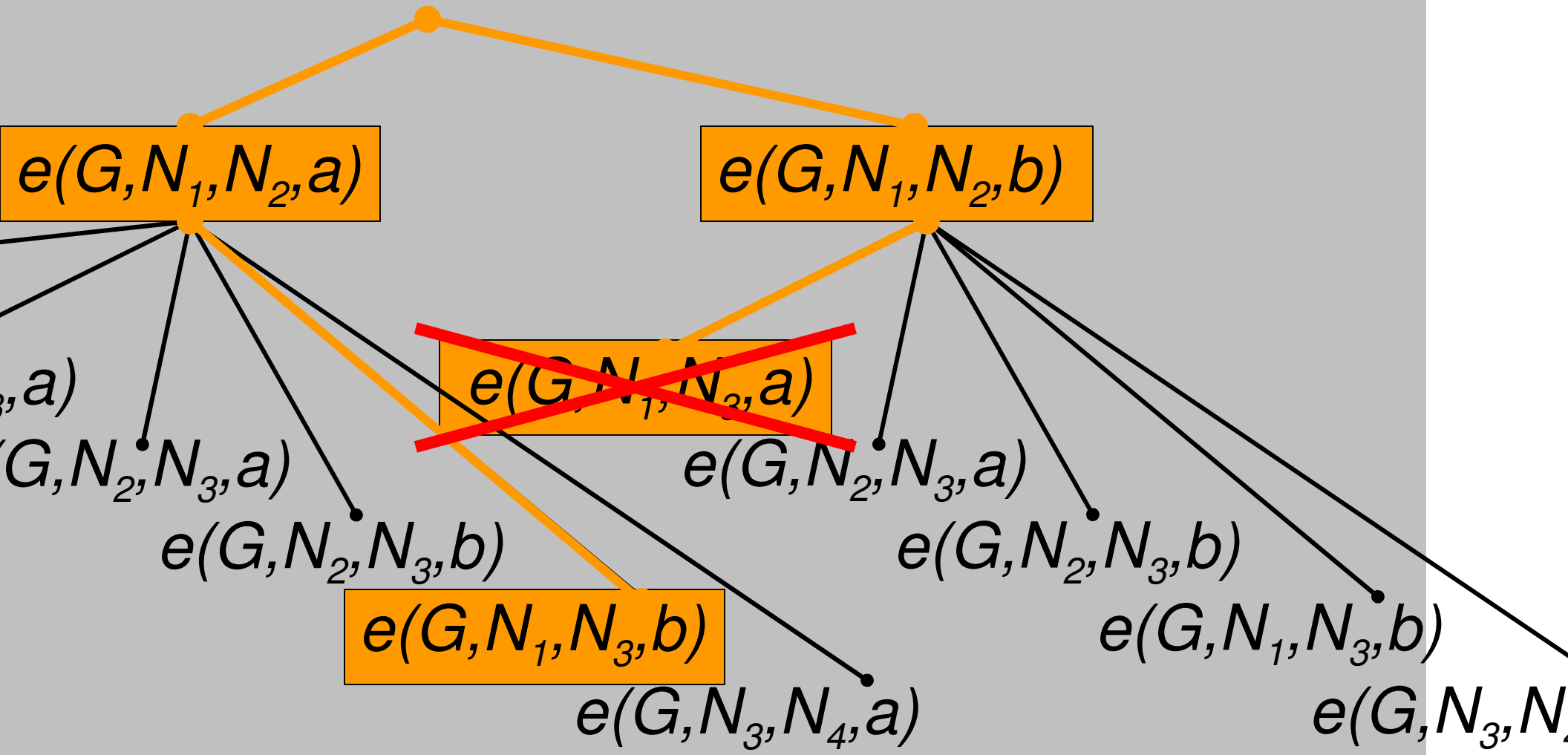
- Only one variant of each query should be counted and outputted
- Main problem: query equivalency under OI has graph isomorphism complexity
- Our approach:
  - use *ordered* tree-based heuristics
  - use efficient data structures to determine equivalency
  - perform also other pruning during exponential search

Determine Refinements

- [IJCAI'01]



Determine Refinements



Determine Refinements

- (In the paper) we prove that
  - Refinement with this strategy is *complete*: of every frequent query defined by the bias, at least one variant is found
  - The order of siblings does not matter for completeness (but they must have some order)

Determine Refinements

- Incrementally generate variants
- Search for the variant (under construction) in the existing part of the query tree
- To optimize this search, siblings are stored in a tree-like hash structure
- If a query is found that is infrequent $\Rightarrow$
  query Q is pruned
  (monotonicity constraint!)

Frequency Computation

- Main problem: the complexity of finding an OI substitution is the same as subgraph isomorphism, and is therefore NP complete

- Our approach: try to avoid as much as possible that the same (exponential) computation is performed twice

Frequency Computation

- $D = \{e(g_1,n_1,n_2,a), e(g_1,n_2,n_1,a), e(g_1,n_2,n_3,a), e(g_1,n_3,n_1,b), e(g_1,n_3,n_4,b), e(g_1,n_3,n_5,a), e(g_1,n_4,n_2,a), e(g_1,n_4,n_5,b), e(g_2,n_6,n_7,b)\}$
- $Q = k(G) \leftarrow e(G,N_1,N_2,b)$

- For each value of *G* for which the database subsumes the query, the `first' substitution is stored

Frequency Computation

- Once a query is refined, for each refinement the first subsuming substitution has to be determined
- This computation is performed in one backtracking procedure for all refinements together (like query packs)
- This search starts from the subsitution of the original query

Frequency Computation

- $D = \{e(g_1,n_1,n_2,a), e(g_1,n_2,n_1,a), e(g_1,n_2,n_3,a), e(g_1,n_3,n_1,b), e(g_1,n_3,n_4,b), e(g_1,n_3,n_5,a), e(g_1,n_4,n_2,a), e(g_1,n_4,n_5,b), e(g_2,n_6,n_7,b)\}$

- $Q = k(G) \leftarrow e(G,N_1,N_2,b)$ ⟨
  - $e(G,N_2,N_3,a)$ ←
  - $e(G,N_2,N_3,b)$ ←
  - $e(G,N_1,N_3,b)$
  - $e(G,N_3,N_4,b)$

Sorting Order

- $D = \{e(g_1,n_1,n_2,a), e(g_1,n_2,n_1,a), e(g_1,n_2,n_3,a), e(g_1,n_3,n_1,b), e(g_1,n_3,n_4,b), e(g_1,n_3,n_5,a), e(g_1,n_4,n_2,a), e(g_1,n_4,n_5,b), e(g_2,n_6,n_7,b)\}$
- $Q1 =$

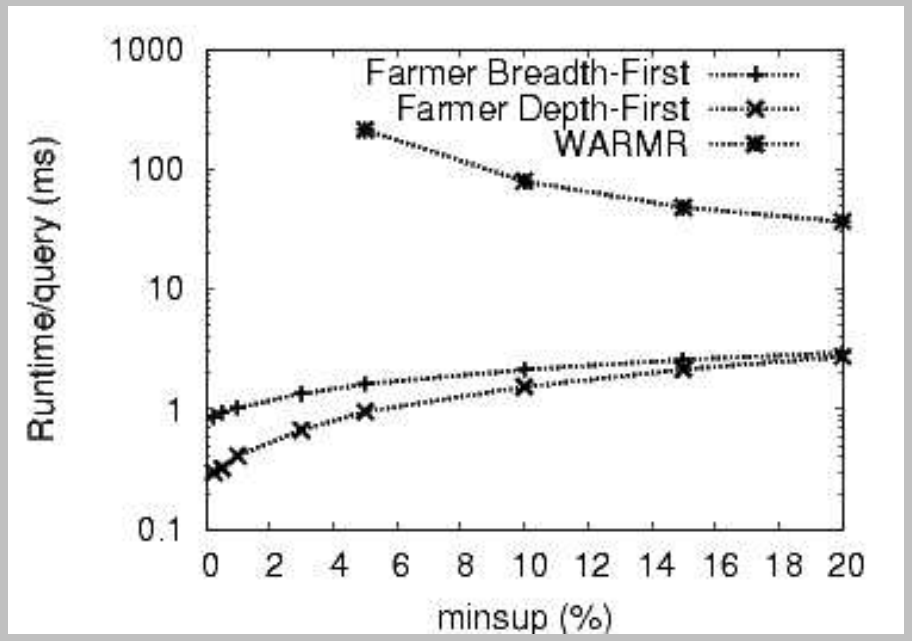
  $k(G) \Leftarrow e(G,N_1,N_2,b), e(G,N_2,N_3,a), e(G,N_2,N_3,b)$

- $Q2 =$

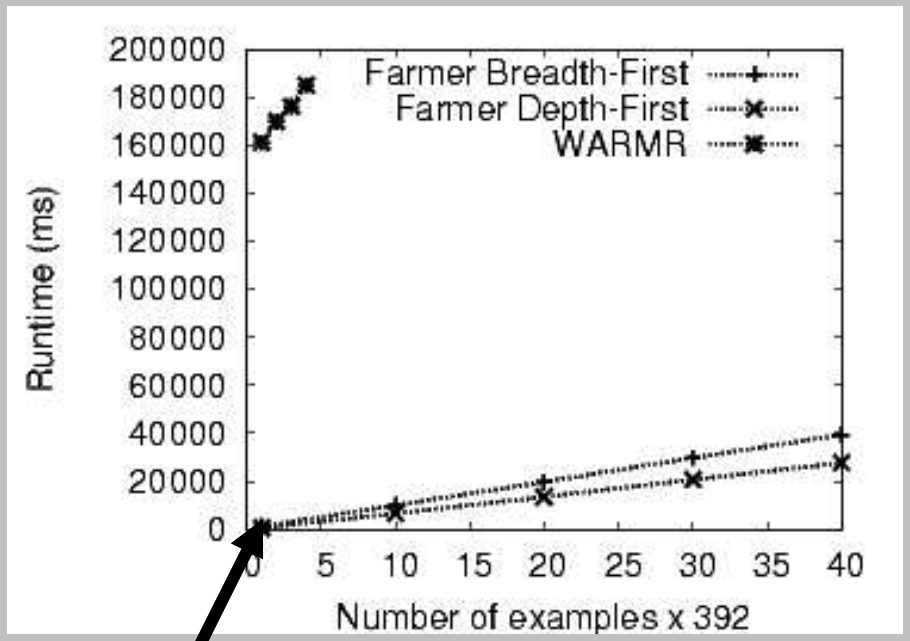
  $k(G) \Leftarrow e(G,N_1,N_2,b), e(G,N_2,N_3,b), e(G,N_2,N_3,a)$

Experimental Results

392 examples



minsup=5%



- Bongard dataset
- Warmr emulates OI

1s

- 192MB 350Mhz

Experimental Results

Machine	Algorithm	6%	7%
Pentium III 500Mhz 448MB	gSpan	5s	
Dual Athlon MP1800+ 2GB	FSG IP	11s	7s
Athlon XP1600+ 256MB	Farmer	72s	48s
Pentium II 350Mhz 192MB	Farmer	224s	148s
Pentium III 500Mhz 448MB	FSG	248s	
Dual Athlon MP1800+ 2GB	FSG II	675s	23s
Pentium III 350Mhz 192MB	Warmr>1h	>1h	

- Predictive Toxicology dataset

Conclusions

- We decreased the performance gap between specialized algorithms and ILP algorithms significantly
- We did so by:
  - using (weak) object identity
  - using a new complete enumeration strategy
  - choosing query evaluation strategies with low costs (much memory however required!)
- Future: provide better comparisons