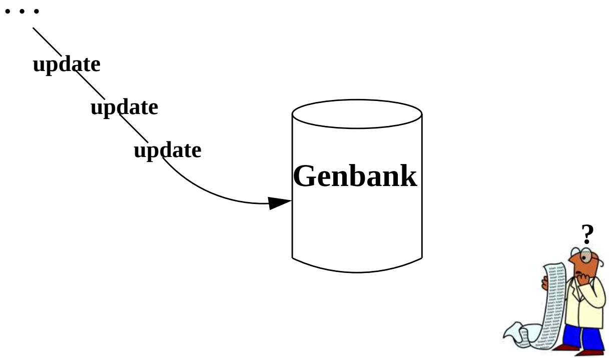
An extensible light-weight XML-based monitoring system for sequence databases

Dieter Van de Craen[1] Frank Neven[1] Kerstin Koch[1]

[1]Hasselt University and Transnational University of Limburg

22 July 2006

Motivation



Question: Is there a gene with high similarity to my sequence ?

Motivation

Existing Solutions

- Alerting Systems
  - BioMail, Jade, Science Direct: literature
  - PubCrawler: PubMed, Genbank
- XML filtering systems
  - XFilter, YFilter, XMLTK: no full XPath

Monitoring System

Goals

- light-weight: locally installed
- extensible: XML/XPath-based
- user-friendly: web-interface
- efficient

Outline

1. Motivation
2. Introduction to XML and XPath
3. System Overview
4. Evaluation
    - Brute Force
    - XML Streaming
    - Query Containment
5. Experiments
6. Incremental Maintenance
7. Conclusion

XML and XPath

- eXtensible Markup Language:
  - standard for data exchange on the Web,
  - XML formats for biological data: BSML, GO-XML,. . .

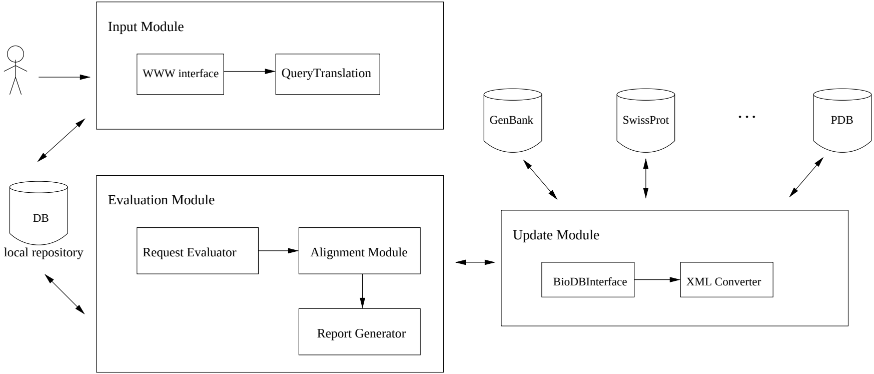
```
<Feature-table title="Features">
   ...
  <Qualifier value="genomic DNA" value-type="mol_type"/>
  ...
</Feature-table>
```

- XPath:
  - XML pattern language for locating information in XML documents
  - Examples:

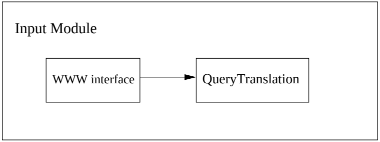
```
   //Qualifier[@value-type="mol_type"]/@value
```

```
 boolean(//Qualifier[@value-type="mol_type" and
                  contains(@value,"DNA"])
```

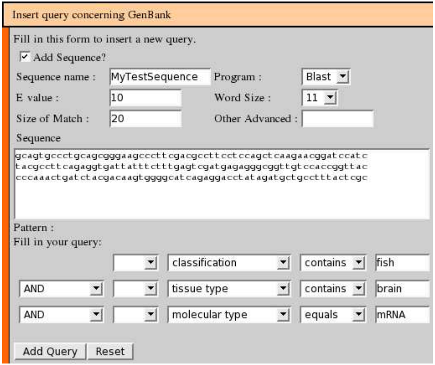
System Overview



System Overview



- WWW Interface
- Query Translator



Insert query concerning GenBank

Fill in this form to insert a new query.

☑ Add Sequence?

Sequence name : MyTestSequence Program : Blast

E value : 10 Word Size : 11

Size of Match : 20 Other Advanced :

Sequence

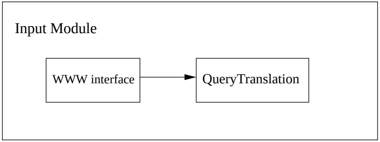
```
gcagtgccctgcagcgggaagcccttcgacgccttcctccagctcaagaacggatccatc
tacgccttcagaggtgattatttctttgagtcgatgagagggcggttgtccaccggttac
cccaaactgatctacgacaagtggggcatcagaggacctatagatgctgcctttactcgc
```

Pattern :

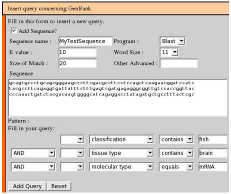
Fill in your query:

		classification	contains	fish
AND		tissue type	contains	brain
AND		molecular type	equals	mRNA

System Overview



- WWW Interface
- Query Translator



Insert query concerning GenBank

Fill in this form to insert a new query.

☑ Add Sequence?

Sequence name : MyTestSequence Program : Blast

E value : 10 Word Size : 11

Size of Match : 20 Other Advanced :

Sequence

gcagtgccctgcagcgggaagcccttcgacgccttcctccagctcaagaacggatccatc
tacgccttcagaggtgattatttctttgagtcgatgagagggcggttgtccaccggttac
cccaaactgatctacgacaagtggggcatcagaggacctatagatgctgcctttactcgc

Pattern :

Fill in your query:

		classification	contains	fish
AND		tissue type	contains	brain
AND		molecular type	equals	mRNA

⟹

Blast

ID	sequence	Evalue	wordsize	MatchSize
...				
51	gcagtgcc...	10	11	20
...				

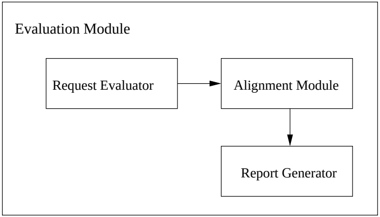
Mapping

ID	variable	querytype	keyword	value
...				
51	v_51_1	contains	classification	fish
51	v_51_2	contains	tissue_type	brain
51	v_51_3	equals	molecular_type	mRNA
...				

Query

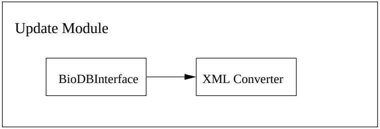
ID	userID	database	formula
...			
51	8	genbank	v_51_1 & v_51_2 & v_51_3
...			

System Overview



- Request Evaluator: evaluates metadata constraints of the requests on the update
- Alignment Module: alings every selected sequence with the corresponding source sequences
- Report Generator

System Overview



- BioDBInterface: checks availability of updates
- XML Converter

Outline

1. Motivation
2. Introduction to XML and XPath
3. System Overview
4. Evaluation
   - Brute Force
   - XML Streaming
   - Query Containment
5. Experiments
6. Incremental Maintenance
7. Conclusion

Evaluation

Input: set of monitoring requests, set of records in the update

Evaluation

1. Compute which requests match which records in the update
2. For these matches align their sequences
3. Build a report if the alignment is satisfying

Evaluation

Input: set of monitoring requests, set of records in the update

Evaluation

1. Compute which requests match which records in the update
2. For these matches align their sequences
3. Build a report if the alignment is satisfying

Bottleneck: Step 1

Evaluation

Evaluation Strategies:

1. Brute force
2. XML Streaming
3. Query Containment

Brute Force

- Test every metadata constraint for every entry in the update
- Evaluation of the XPath expressions: Xalan

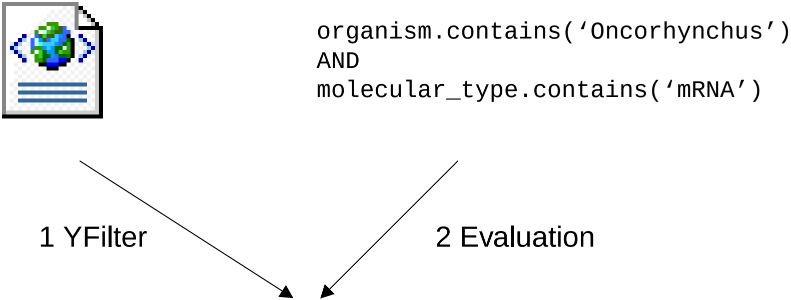
XML Streaming

XML stream query processing systems offer efficient XPath evaluation, but support a limited fragment

Idea: proceed in two steps:

1. retreive all the values for search fields for a record in the update using YFilter
   $\Longrightarrow$ complex value representation
2. evaluate the metadata constraints on this complex value representation

XML Streaming



```
organism.contains('Oncorhynchus')
AND
molecular_type.contains('mRNA')
```

f	values
organism	{"Oncorhynchus mykiss"}
accession	{ "AM181351" }
keyword	{ "vitronectin protein 1", "vtn1 gene" }
molecular type	{"mRNA"}
...	...

Query Containment

Idea: related topics of research will lead to related metadata constraints

$\Longrightarrow$ query containment: a constraint $p \subseteq$ a constraint p' if a record r satisfies p, r will also satisfy p'

Example:

organism.equals('Oncorhynchus mykiss') AND molecular_type.contains('mRNA')

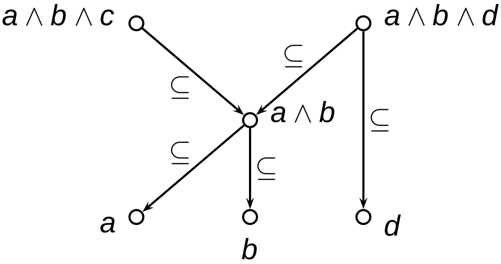
$\subseteq$

organism.contains('Oncorhynchus')

Query containment reduces to unsatisfiability of propositional logical formulas: coNP-complete

$\Longrightarrow$ Limmat

Containment DAG



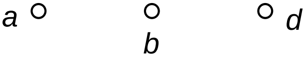
- node: set of equivalent constraints
- edge $n \xrightarrow[\subseteq]{} n'$ if every contraint in n is contained in every constraint in n'

Containment DAG

metadata constraints: a, b, d, $a \wedge b$, $a \wedge b \wedge c$ and $a \wedge b \wedge d$

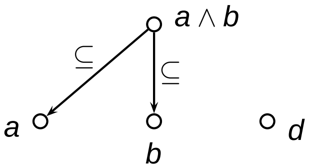
Containment DAG

metadata constraints: a, b, d, $a \wedge b$, $a \wedge b \wedge c$ and $a \wedge b \wedge d$



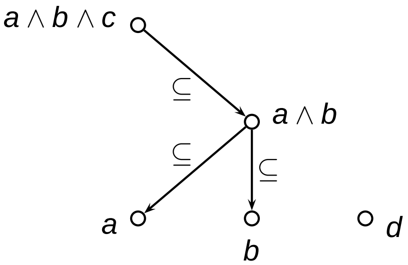
Containment DAG

metadata constraints: a, b, d, $a \wedge b$, $a \wedge b \wedge c$ and $a \wedge b \wedge d$



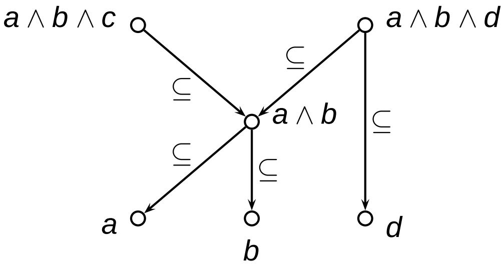
Containment DAG

metadata constraints: a, b, d, $a \wedge b$, $a \wedge b \wedge c$ and $a \wedge b \wedge d$



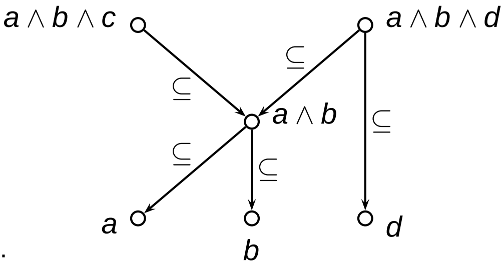
Containment DAG

metadata constraints: a, b, d, $a \wedge b$, $a \wedge b \wedge c$ and $a \wedge b \wedge d$



Containment DAG

metadata constraints: a, b, d, $a \wedge b$, $a \wedge b \wedge c$ and $a \wedge b \wedge d$

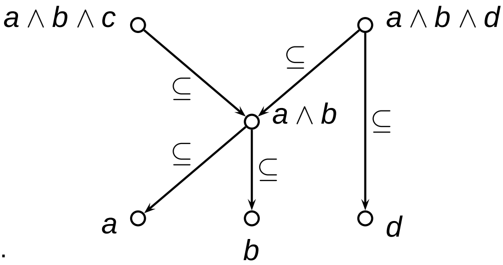


Observations:

1. only one constraint is evaluated for a set of equivalent constraints
2. if a record r matches a constraint in node n then all constraints in descendant nodes of n match r
3. if a record r does not match a constraint in node n then all constraints in ancestor nodes of n do not match r

Containment DAG

metadata constraints: a, b, d, $a \wedge b$, $a \wedge b \wedge c$ and $a \wedge b \wedge d$

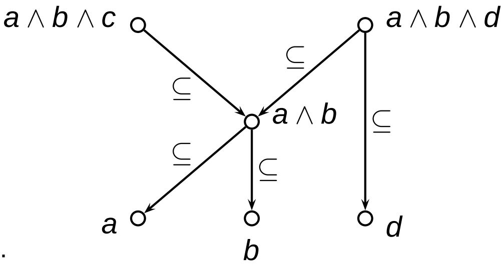


Observations:

1. only one constraint is evaluated for a set of equivalent constraints
2. if a record r matches a constraint in node n then all constraints in descendant nodes of n match r
3. if a record r does not match a constraint in node n then all constraints in ancestor nodes of n do not match r

Containment DAG

metadata constraints: a, b, d, $a \wedge b$, $a \wedge b \wedge c$ and $a \wedge b \wedge d$

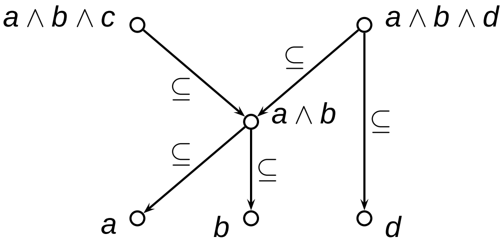


Observations:

1. only one constraint is evaluated for a set of equivalent constraints
2. if a record r matches a constraint in node n then all constraints in descendant nodes of n match r
3. if a record r does not match a constraint in node n then all constraints in ancestor nodes of n do not match r

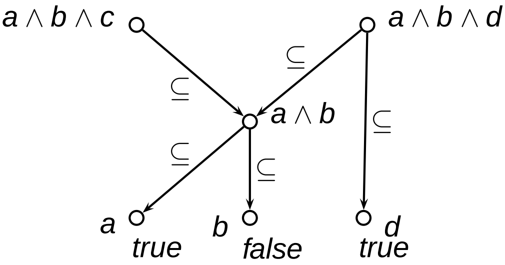
Optimizations

1. false propagation: start at the sinks, if a node does not match a record propagate this to its ancestors
2. true propagation: start at the sources, if a node matches a record propagate this to its descendants



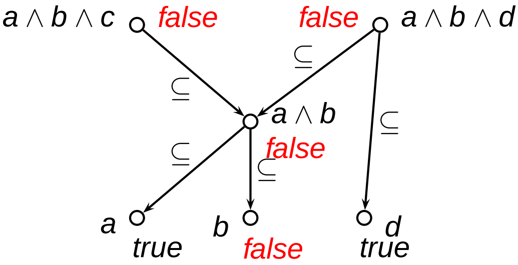
Optimizations

1. false propagation: start at the sinks, if a node does not match a record propagate this to its ancestors
2. true propagation: start at the sources, if a node matches a record propagate this to its descendants



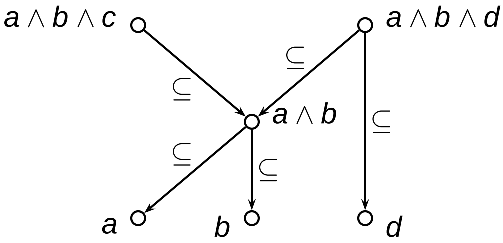
Optimizations

1. false propagation: start at the sinks, if a node does not match a record propagate this to its ancestors
2. true propagation: start at the sources, if a node matches a record propagate this to its descendants



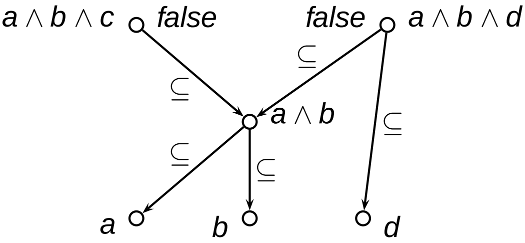
Optimizations

1. false propagation: start at the sinks, if a node does not match a record propagate this to its ancestors
2. true propagation: start at the sources, if a node matches a record propagate this to its descendants



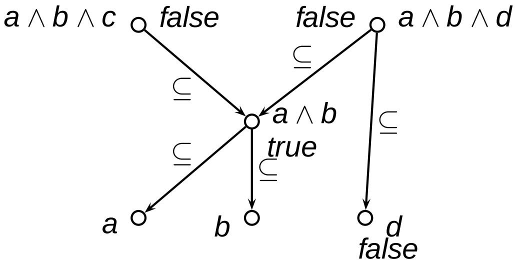
Optimizations

1. false propagation: start at the sinks, if a node does not match a record propagate this to its ancestors
2. true propagation: start at the sources, if a node matches a record propagate this to its descendants



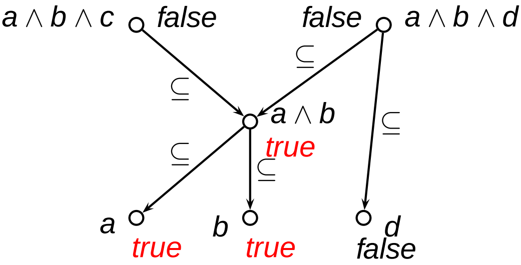
Optimizations

1. false propagation: start at the sinks, if a node does not match a record propagate this to its ancestors
2. true propagation: start at the sources, if a node matches a record propagate this to its descendants



Optimizations

1. false propagation: start at the sinks, if a node does not match a record propagate this to its ancestors
2. true propagation: start at the sources, if a node matches a record propagate this to its descendants

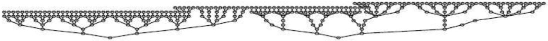


Outline

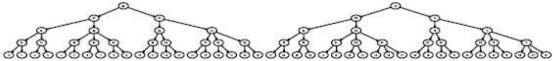
1. Motivation
2. Introduction to XML and XPath
3. System Overview
4. Evaluation
   - Brute Force
   - XML Streaming
   - Query Containment
5. Experiments
6. Incremental Maintenance
7. Conclusion

Experiments

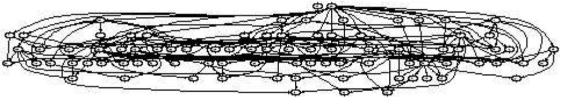
- 3 types of containment DAGs:
  1. T1: for false propagation, sinks are the most general, upwards subsequently refined, only AND



  2. T2: for true propagation, sources are the most restricted, downwards subsequently relaxed, only AND



  3. R: created by random constraints, AND, OR, NOT, ( )



- constraints created by extracting values out of updates
- number of requests: 1000 till 5000

Experimental Results: 5000 monitoring requests, update of 10^5 records

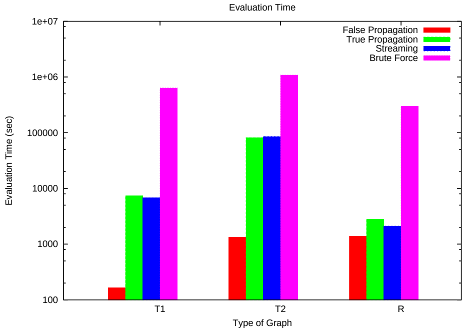


Figure: Average evaluation time in seconds

	T1	T2	R
False propagation	0.5%	52%	38%
True propagation	99%	99%	93%
Streaming	100%	100%	100%
Brute force	100%	100%	100%

Figure: Average percentage of evaluated nodes in the DAG

Comparison with a relational database approach

Idea: parse the XML into a relational database and use the database mechanisms for querying

- Queries: 1000 random queries in CNF
  - # clauses: 1 to 5
  - # literals per clause: 1 to 3
  - constraints created by extracting values out of updates
- experiments done with MySQL

Comparison with a relational database approach

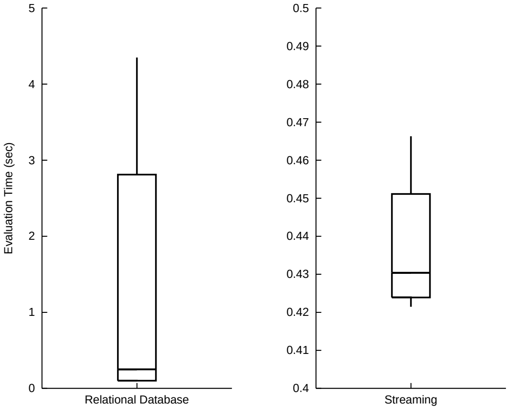


Figure: Boxplot of the evaluation time per query for 10^5 records

Comparison with a relational database approach

- Relational database
  - Loading the update into the DB: 655 sec
  - Evaluation: 1803 sec
  - Total: 2461 sec
- Streaming
  - Constructing complex values of the update: 230 sec
  - Evaluation: 438 sec
  - Total: 673 sec
- Advantages Streaming:
  - ± constant evaluation time per query
  - less overhead

Outline

1. Motivation
2. Introduction to XML and XPath
3. System Overview
4. Evaluation
   - Brute Force
   - XML Streaming
   - Query Containment
5. Experiments
6. Incremental Maintenance
7. Conclusion

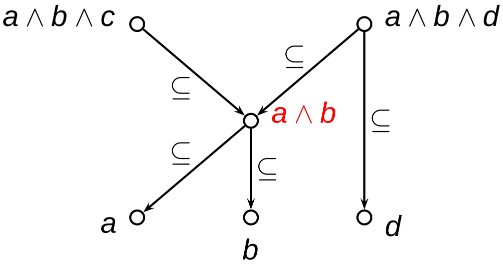
Incremental Maintenance

- containment DAG is independent of the updates
- maintained incrementally: deletion and insertion of monitor requests

Incremental Maintenance

1. Deletion
    1. Locate the corresponding node
    2. Remove the constraint, if the node is empty add edges from the parents to the children

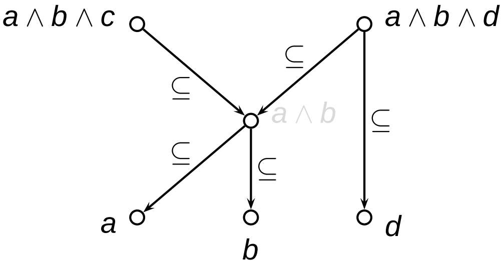
Deletion of $a \wedge b$



Incremental Maintenance

1. Deletion
   1. Locate the corresponding node
   2. Remove the constraint, if the node is empty add edges from the parents to the children

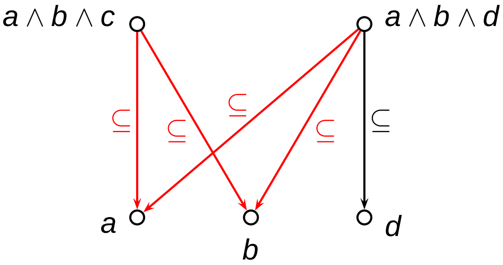
Deletion of $a \wedge b$



Incremental Maintenance

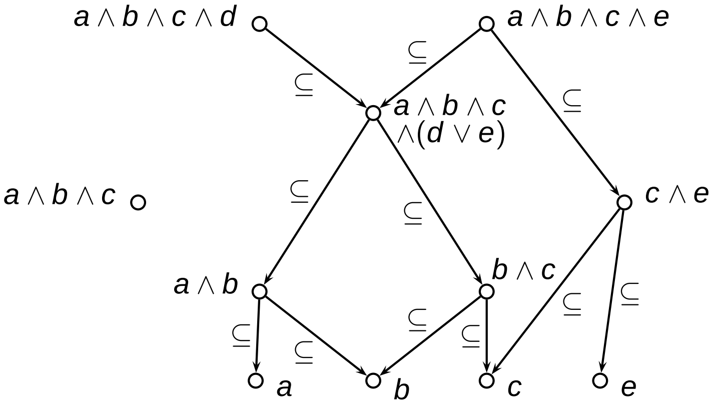
1. Deletion
   1. Locate the corresponding node
   2. Remove the constraint, if the node is empty add edges from the parents to the children

Deletion of $a \wedge b$



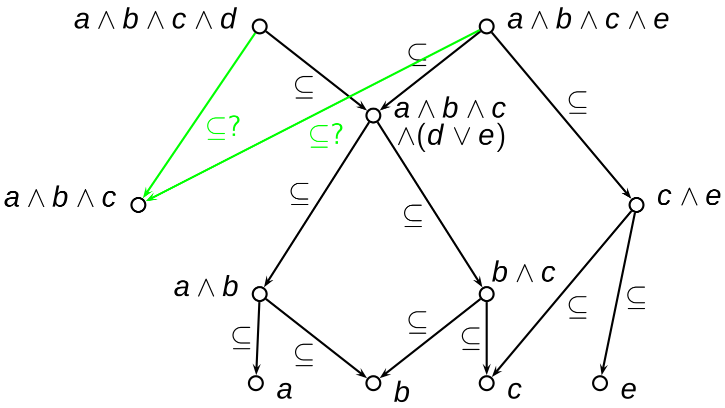
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



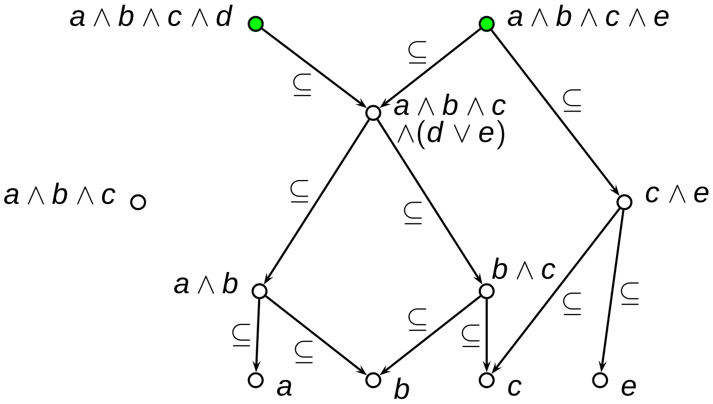
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



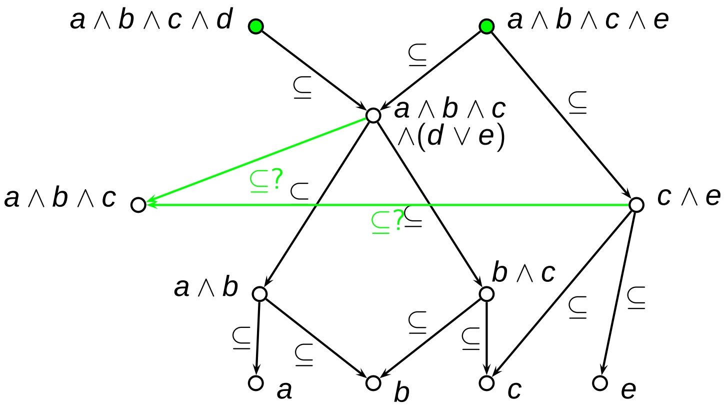
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



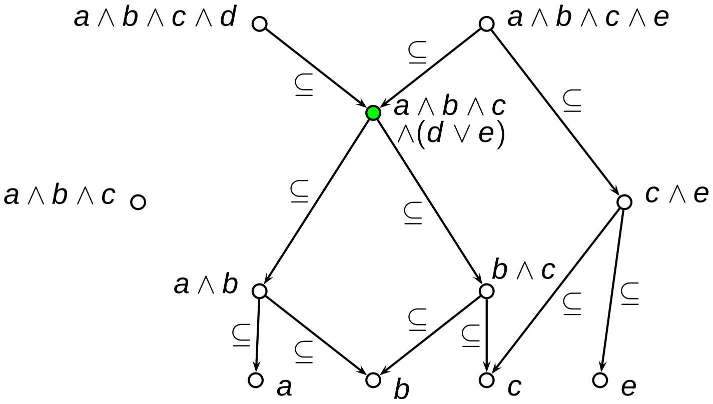
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



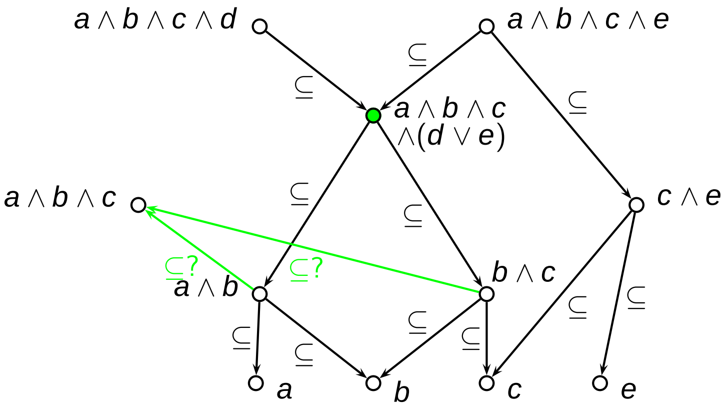
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



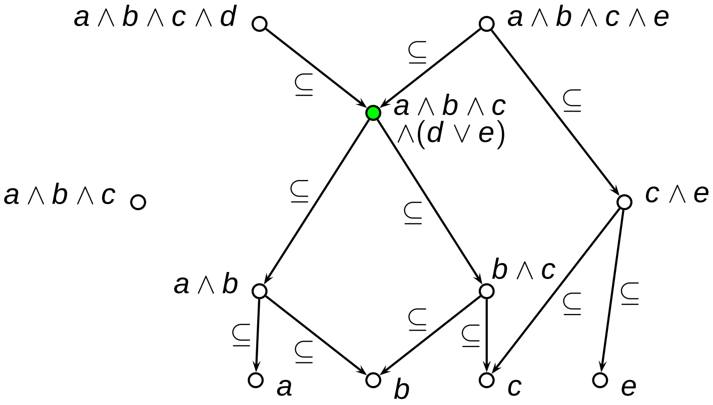
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



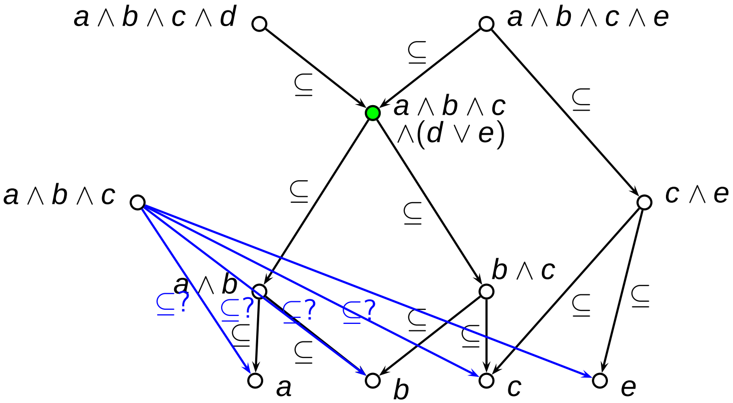
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



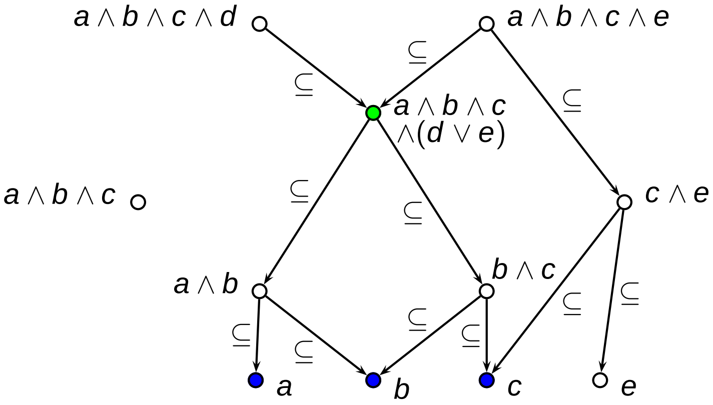
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



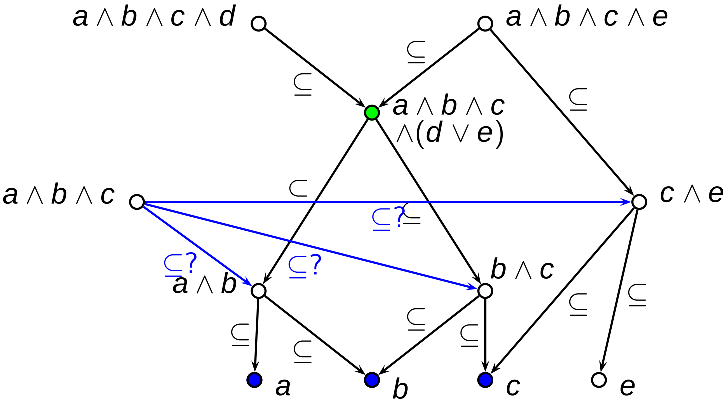
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



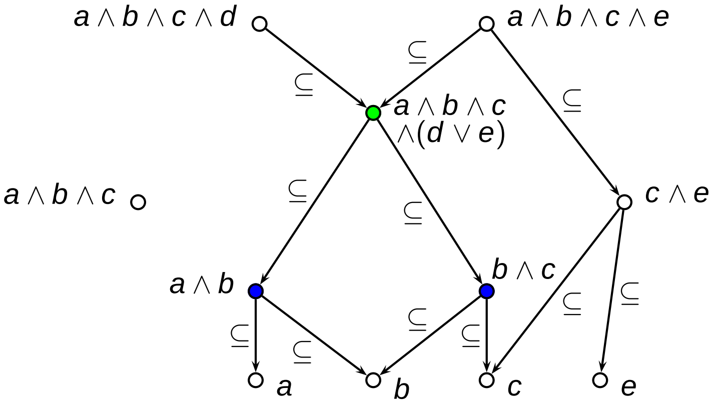
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



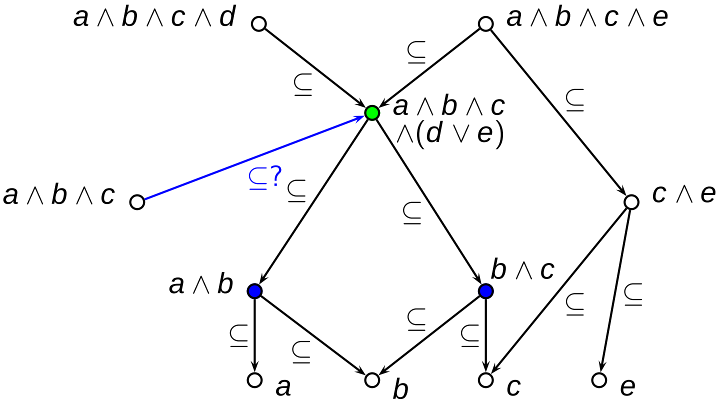
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



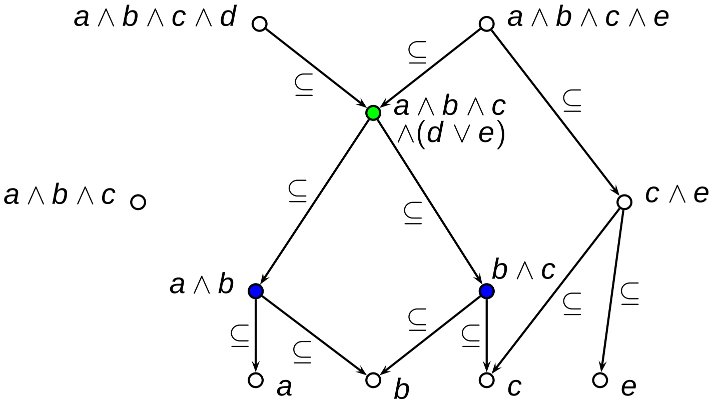
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



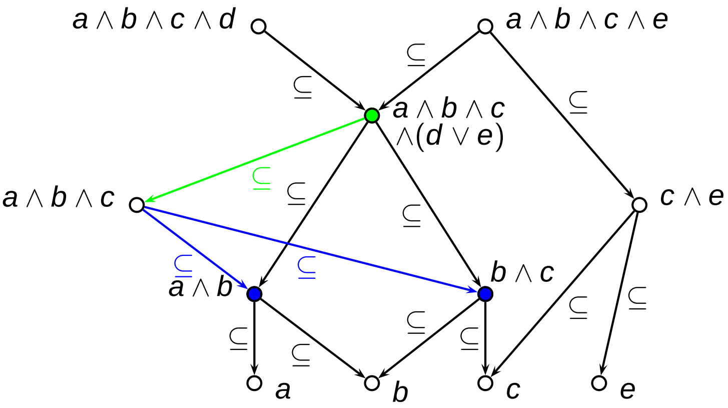
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them


$a \wedge b \wedge c \wedge d$
$a \wedge b \wedge c \wedge e$
$a \wedge b \wedge c \wedge (d \vee e)$
$a \wedge b \wedge c$
$c \wedge e$
$a \wedge b$
$b \wedge c$
a
b
c
e
$\subseteq$

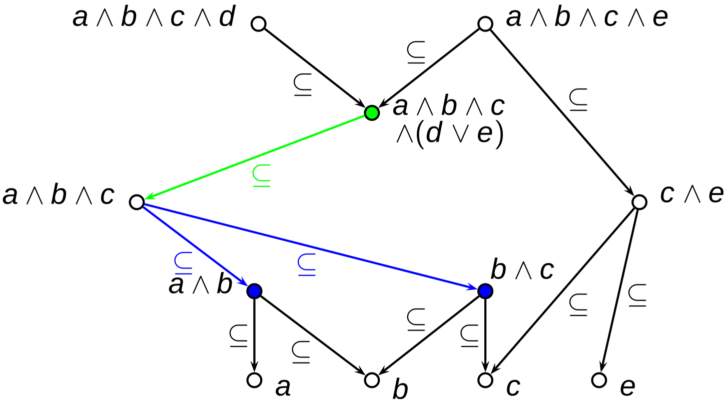
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



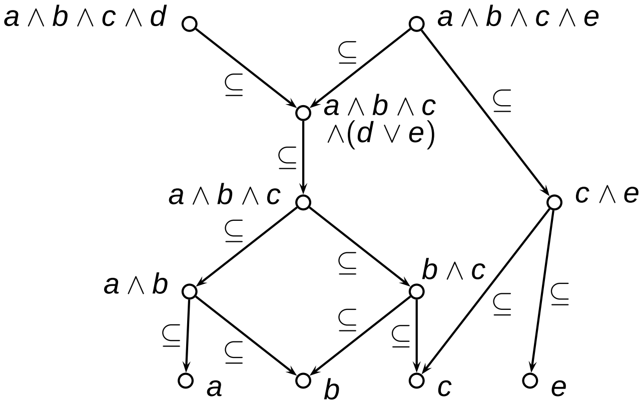
Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them


$a \wedge b \wedge c \wedge d$
$a \wedge b \wedge c \wedge e$
$\subseteq$
$a \wedge b \wedge c \wedge (d \vee e)$
$a \wedge b \wedge c$
$c \wedge e$
$a \wedge b$
$b \wedge c$
a
b
c
e

Incremental Maintenance

2. Insertion of a constraint $a \wedge b \wedge c$
   Idea: compute initial upper and lower borders, U and L, and gradually refine them



Incremental Maintenance

2. Insertion
   - expensive: add a constraint to a containment DAG containing
     1250 nodes: 15 sec
     2500 nodes: 30 sec
     5000 nodes: 60 sec
   - bottleneck: complexity of the containment check
   - solution: transform constraints into DNF
     $\Rightarrow$ 25% increase in size
     construction containment DAG with 5000 nodes: 60 sec

Conclusion

- Main Results
  - developed XSeqM, an extensible, light-weight monitoring system for biological databases
  - experimentally valited our evaluation method
- Future Work
  - Deploy the system
  - Test the system on real world data
  - Further optimize the false propagation method on containment DAGs of type R: OBDD