

# 0. Query Merging for XML Streams

Overview of techniques for multi-query optimization in the SPEX framework

Referent [Tim Furche {tim@furche.net}](mailto:tim@furche.net)

# Query Merging

---

# 0. Query Merging for XML Streams

## Content

### 1. Why?

1. Multiple Queries
2. Continuous XML Streams

### 2. How?

1. Problem Formulation
2. Query Plan for SPEX
3. Query Compaction
4. Query Merging

### 3. When?

1. Current Status
2. Remaining Schedule

# 1. Why ... ... streams

- Application areas
  - Internet
  - live feeds, e.g., monitor or sensor networks, live broadcasting
  - pipelines for data processing
- Streamed vs. stored data processing
  - DBMS concerned about
    - secondary storage access
    - size of intermediary results
  - stream system
    - each element processed **exactly** once
      - at least once ← no skipping ahead
      - at most once → no intermediary results
    - **minimize operations per element**
      - near real-time processing

# 1. Why ...

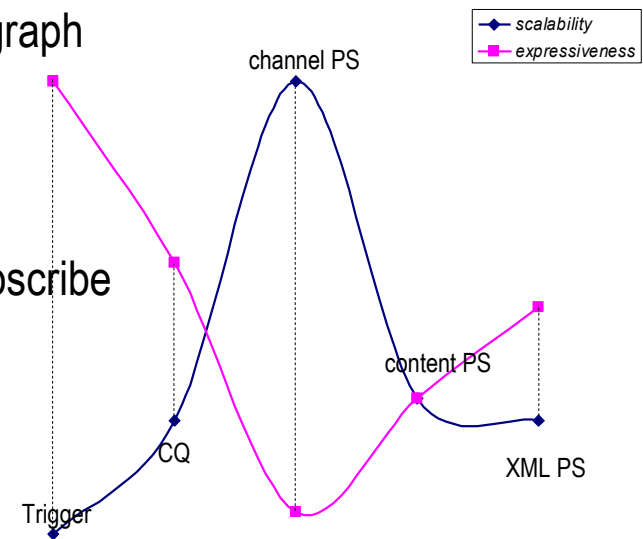
## ... multiple queries

- Multi-query optimization:
    - **single query plan** for multiple queries
  - Traditional DBMS:
    - multi-query optimization no core area
    - recently some attention in context of OLAP
    - focus: containment and reordering of operations, *not* sharing
  - Hot for stream systems
    - operator sharing reduces number of operations
    - huge gain for large number of queries on small information space
- ⇒ Several proposals for multi-query systems on streams
- tuple streams: Telegraph, STREAM, Aurora
  - XML streams: publish-subscribe or event notification systems  
XFilter, YFilter, XTrie, XMLTK, MatchMaker

# 1. Why ...

## ... multiple queries – related work

- Scalable trigger systems      TriggerMan
- Continuous query systems      STREAM, Aurora, Telegraph
- Publish-subscribe systems
  - channel-based: NNTP, PointCast
  - content-based: READY, Gryphon, Siena, Rebeca, Le Subscribe
  - XML-based: XFilter, YFilter, XTrie, MatchMaker
- Scalability vs. Expressiveness
  - low scalability      → unacceptable delay
  - low expressiveness      → impreciseness for specifying intent



⇒ Challenge: **increase expressiveness without sacrificing scalability**

⇒ How much scalability is feasible?

- limited by communication capabilities

# 1. Why ...

## ... continuous streams I

- XML-based publish-subscribe systems (e.g., XFilter, XTrie)
  - stream of small documents → allows intermediary data structures linear in document
  - selection postponement
- continuous streams
  - no assumption on the structure → unbounded document and element size
  - arbitrary recursive structures
  - ⇒ no intermediary data structures feasible
  - ⇒ inline selection
  - ⇒ limit on size of potential result by approximation for blocking operators
- continuous streams for
  - media broadcasting and filtering, e.g., using MPEG-7
  - syndication, e.g., Google News
  - monitor aggregation, i.e., aggregation of several monitor networks

# 1. Why ...

## ... continuous streams II

- Can a query engine for continuous streams still be competitive on streams of small documents?
- SPEXs network of DPDTs is competitive for single queries
  - no better single query processor
  - XML-based publish-subscribe systems
    - SPEX has more powerful query language
    - restricted SPEX competitive

Approach		Focus	Space–Complexity–Time	
XSM	FSM with buffers	joins, construction	$n \times L$	$n^L$
$\chi\alpha\omicron\varsigma$	specific “matchstructure”	parent, ancestor	$n^L$	$n^L$
XSQ	HDPDT	simple predicates, aggregation	$n + 2^L$	$n \times L$
SPEX	network of DPDTs	RPQ: generic predicates, reverse axes	$(n + L) \times d$	$n \times d \times L$

# 1. Why ... ... therefore

- publish-subscribe systems for continuous streams require
  - more expressive query language, yet
  - prohibit intermediary data structures
- SPEX enables querying of continuous streams
  - no intermediary data structures required (e.g., by predicate inlining)
  - query language clearly more expressive

⇒ **extending SPEX to support multiple queries**

## 2. How?

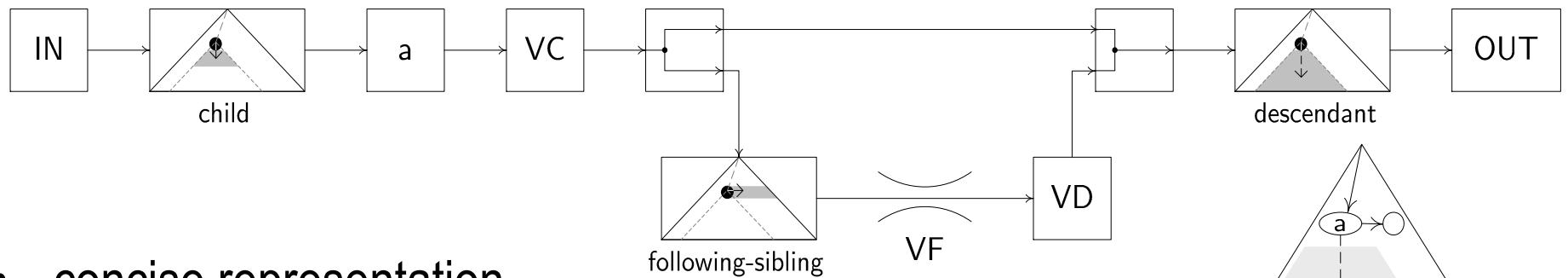
### Minimum Common Super Plan

- MINIMUM COMMON SUPER PLAN (MCSP)
  - optimization problem
  - find the common query plan with minimal cost for a set of queries
  - cost defined by some cost function, e.g., number of transducers (nodes)
- common query plan  $p$  for queries  $q$  and  $q'$ 
  - restriction to result variables in  $q$  ( $q'$ ) is a query plan for  $q$  ( $q'$ )
- top-down
  - concatenate all queries into a new query  $q$
  - compute an optimal query plan for  $q$
- bottom-up
  - compute query plans for all queries
  - merge the query plans optimally

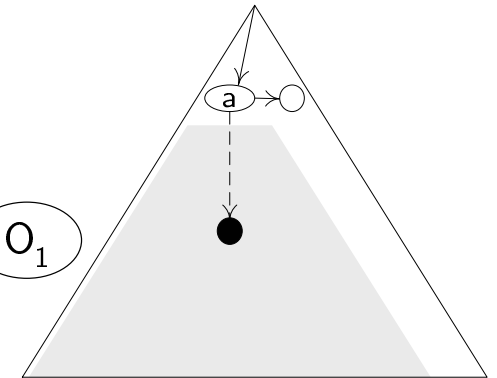
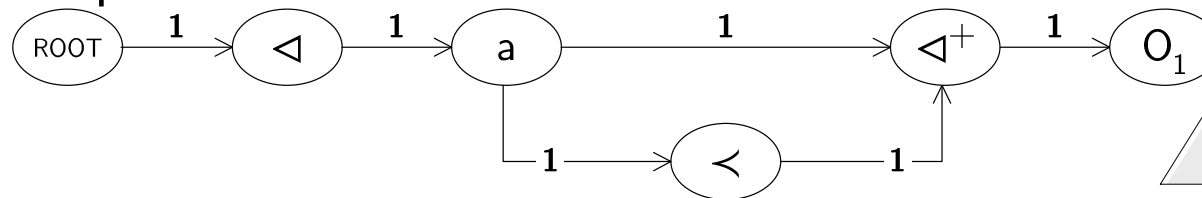
## 2. How?

### Query Networks in SPEX

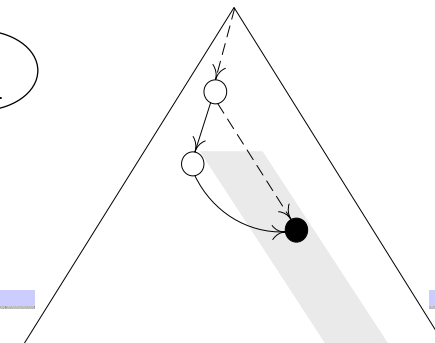
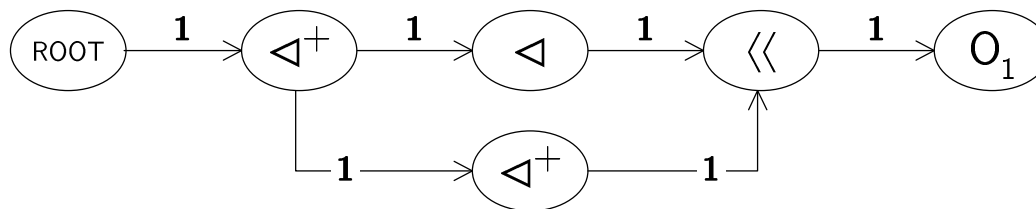
- SPEX network: directed-acyclic graph of transducers



- concise representation



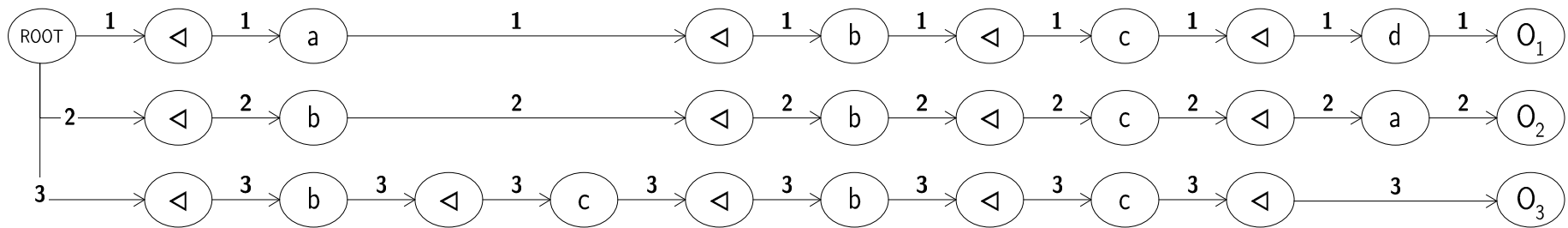
- inherently graph-structured, arbitrary degree, non-planar, connected



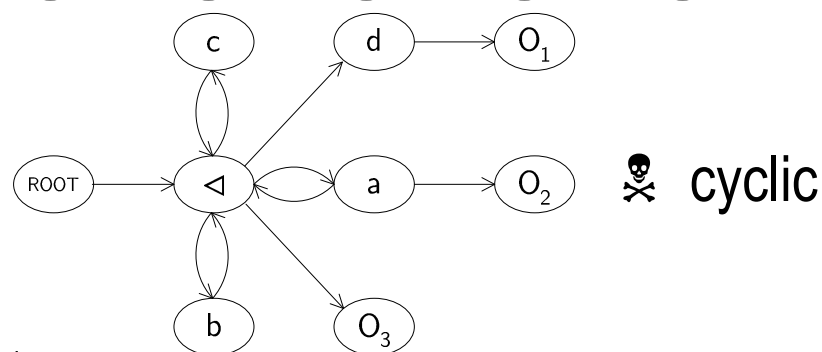
## 2. How?

### Minimum Common Supergraph

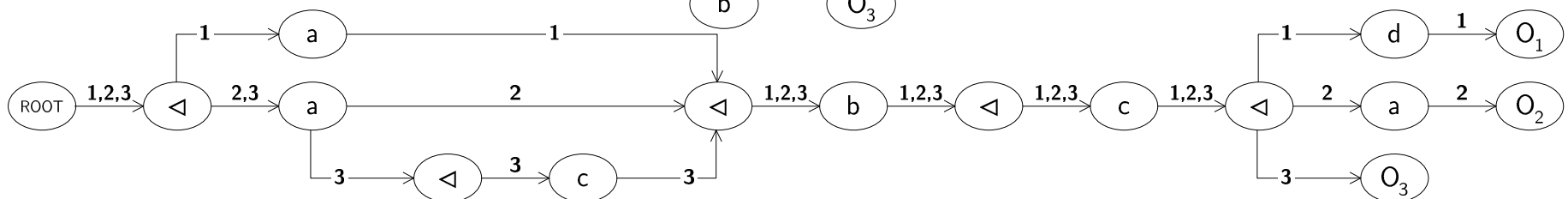
- MCSP for SPEX: minimum common supergraph
  - property for SPEX: acyclic



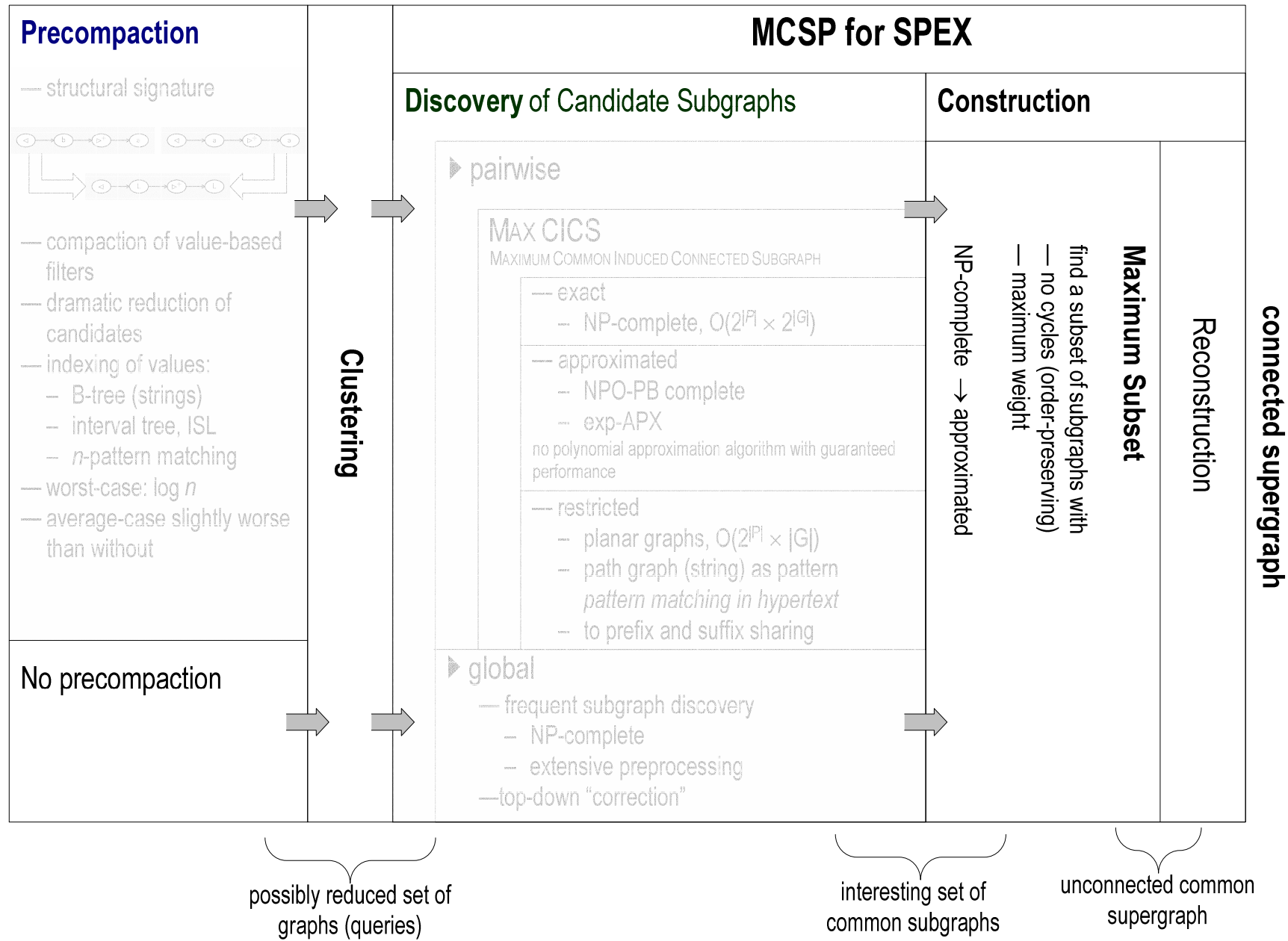
node-induced MCS:



acyclic MCS:



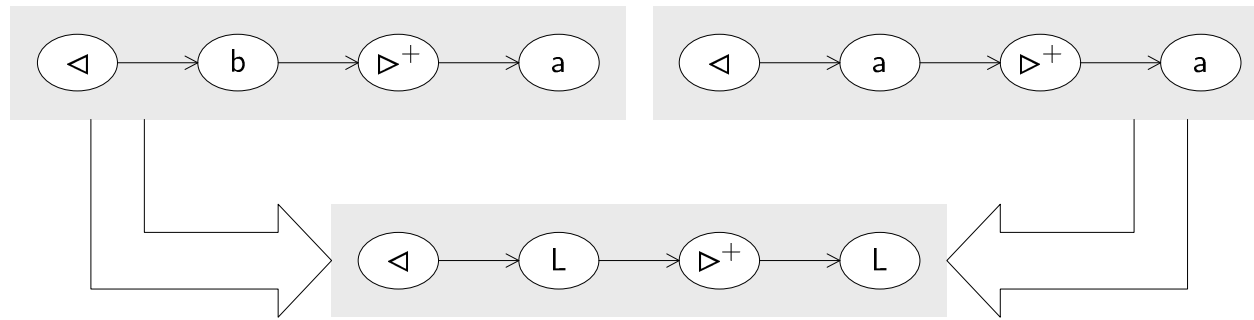




## 2. How?

### Query Compaction

- structural signature → dramatic **reduction of candidates**



- compaction of value-based filters using, e.g.,
  - strings
    - exact-match: combination of B-tree and hashing
    - regular expressions:  $n$ -pattern matching algorithm for regular expressions
  - numeric data: interval tree, segment tree, ISL
- average case *might* be slightly worse
  - no structural pruning
- guaranteed worst-case complexity

## 2. How?

### Candidate Subgraph Discovery

- pairwise, i.e., MAXIMUM COMMON INDUCED CONNECTED SUBGRAPH problem
  - exact: NP-complete,  $O(2^{|P|} \times 2^{|G|})$
  - approximated
    - NPO-PB complete, exp-APX, i.e.,
    - no polynomial approximation algorithm with guaranteed performance
  - restricted
    - planar graphs,  $O(2^{|P|} \times |G|)$  ☠
    - path graph as pattern → pattern matching in hypertext
    - sharing of prefixes and suffixes only
- global
  - frequent subgraph/substructure discovery
    - NP-complete
    - very expensive preprocessing of the queries necessary
  - top-down “correction”

