Outline

- Problem Statement and Motivation
  - Generalized Path Queries
  - Algebraic Problem Interpretation
- Background
  - Algebraic Path Problem Solving
- Approach
  - Integrating of graph pattern matching with algebraic path problem solving
- Evaluation
- Conclusion

Computer Science NC STATE UNIVERSITY

Motivating Example

Query: Find relationships between passengers on any flights to Washington DC between June 30 and July 6 who purchased one-way tickets by cash, and countries on the CIA watchlist where there is at least one financial link through any bank.

Pattern p1:
Passengers on flights to Washington DC between June 30 and July 6 who purchased one-way tickets by cash

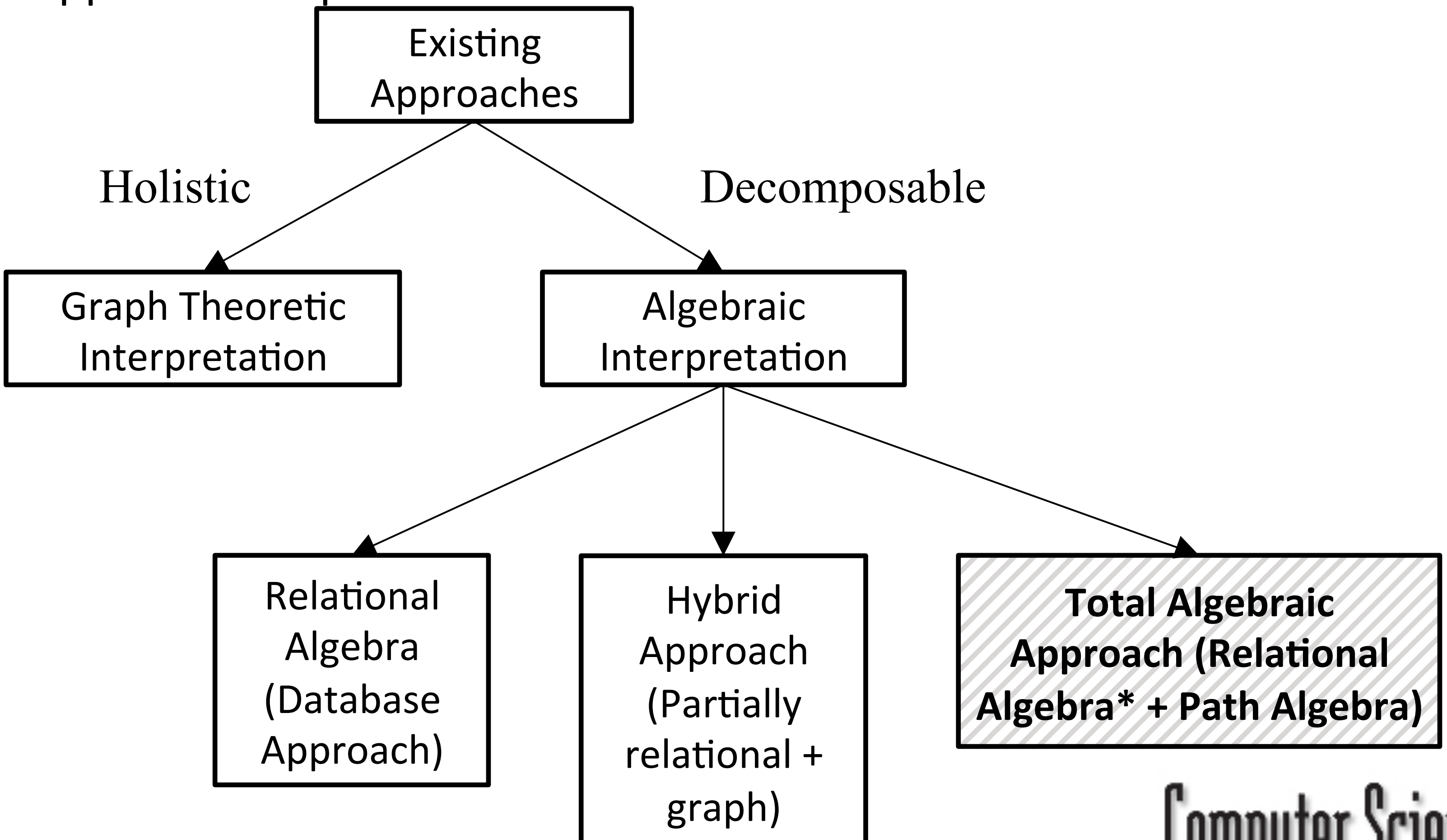
?

Pattern p2:
Countries on the CIA watchlist

1. **Relationships are paths that must be extracted not merely matched!!!**
   - e.g. different from path expression/property path queries
2. **Participating entities are not explicitly stated**

Existing Problem Interpretation Approaches

- For graph data models like RDF there are multiple ways to approach the problem



Comparison of Approaches

Graph Theoretic

- Different problems translate to different graph problems i.e. different algorithms
  - Ex. is sub-graph homeomorphism problem (NP hard).
  - Others: Subgraph Isomorphism, Shortest Paths, Disjoint Paths
- Often difficult to parallelize or optimize
- Cant deal with declarative specifications

Algebraic

- Decompose into smaller operators
  - Reuse is possible
  - Adapt optimization and composition of smaller operators
- Relational algebra not ideal
  - Arbitrary path length requires iteration and fixpoint semantics
- Hybrid approaches exist
  - Pattern matching (algebraic), traversal (graph theoretic)
    - Emphasis on shortest paths for traversals

Computer Science
NC STATE UNIVERSITY

Our Proposal: A Total Algebraic Approach

Query: Find relationships between passengers on any flights to Washington DC between June 30 and July 6 who purchased one-way tickets by cash, and countries on the CIA watchlist where there is at least one financial link through any bank.

Problem interpretation is as a **Generalized Path Query – gpqs** with

1. Two entity sets declaratively specified as patterns – P1, P2
2. A connection set linking p1, p2 instances
3. A connection constraint on the connection set (edge type membership)

1 : Solvable with algebraic graph pattern matching

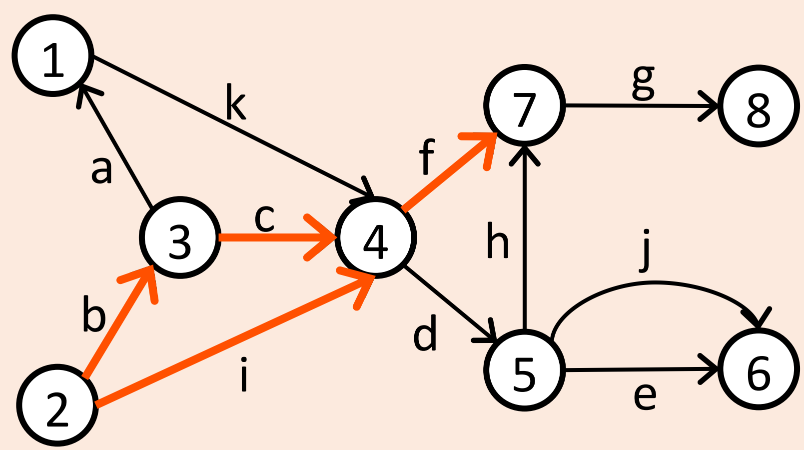
2, 3 : Solve with algebraic path finding (plus extension)

Algebraic Path Problem Solving Framework

- An efficient algebraic path problem solving approach introduced by Tarjan[23, 24].
- **Basic Definitions:**
  - An edge *e* in a directed labeled graph $G = (V,E)$ is denoted as $e = (v_1, v_2)$ with label $\lambda(e) = l_e$ where $v_1, v_2 \in V$ and $e \in E$.
  - A path *p* in such a graph $G = (V,E)$ is an alternating sequence of nodes and labeled edges terminating in a node *p*
    - $= \{v_1, l_{e1}, v_2, l_{e2}, \ldots, v_n, l_{en}, v_{n+1}\}$ where $v_1, v_2, \ldots, v_n, v_{n+1} \in V$ and $e_1, e_2, \ldots, e_n \in E$.

Path Encoding as Path Expressions

- A **Path Expression** [23, 24] of type ***(s, d)***, ***PE(s, d)***, is a 3-tuple ⟨*s*, *d*, *R*⟩, where
  - *R* is a regular expression over the set of edges defined using union(∪), concatenation(•) and closure (*).
  - The language *L(R)* of *R* represents paths from *s* to *d* where the graph contains nodes *s* and *d*.

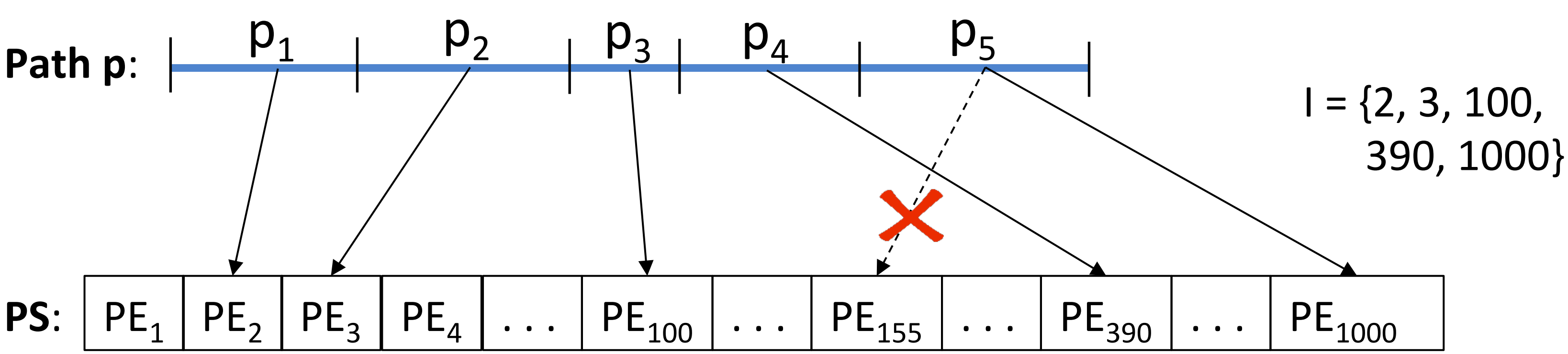


$PE(2, 7) = \langle 2, 7, (\boxed{(b \bullet c \bullet f)} \cup \boxed{(i \bullet f)})\rangle$

$PE(1, 4) = \langle 1, 4, (k)\rangle$

Graph Path Information Representation:
A Sequence of Path Expressions

- A **Path Sequence (PS)** [23,25] is a unique ordering of path expressions that represent all path information in a graph, such that for any path *p*,
  - there is a unique partition of *p* into non empty subpaths, and
  - a unique sequences of indices *I of PS,* such that
  - the *ith* subpath of *p* is represented by the path expression in PS at the *ith* index in *I*.



- The path sequence representation of a graph allows for path problems to be solved by single forward scan.

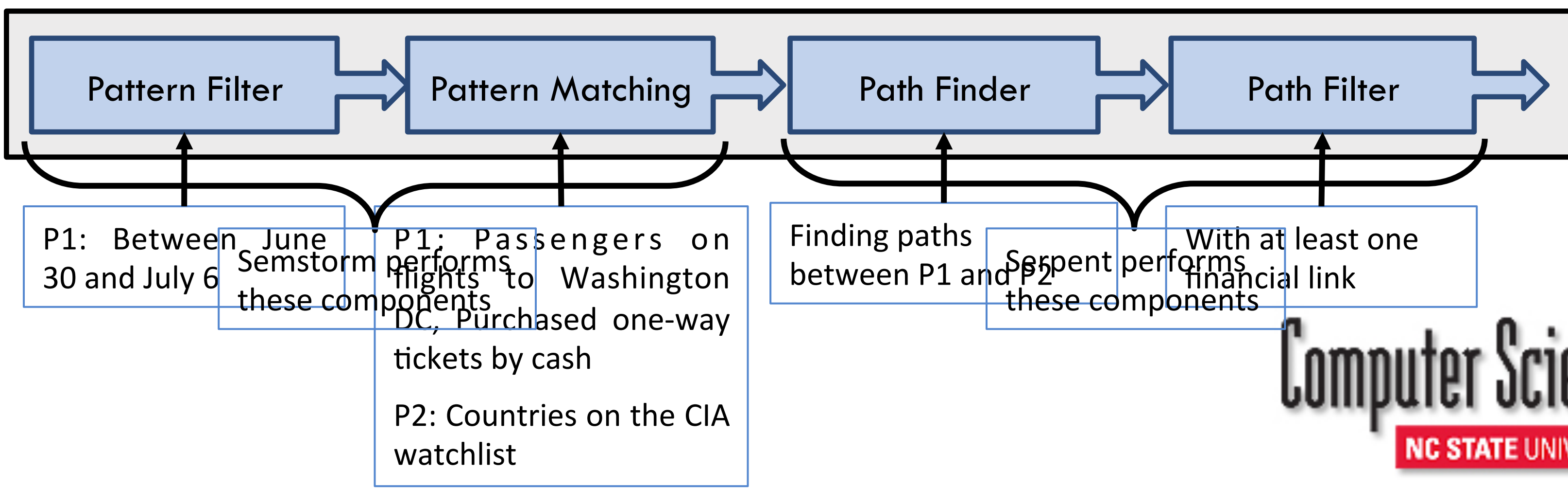
Path Computation using Path Sequence

- A simple propagation SOLVE algorithm [23,25] can be used to solve path problems.
- The SOLVE algorithm assembles path information while scanning the path-sequence from left to right.
- At every iteration of the SOLVE algorithm the following step is performed

$$PE(s,w\downarrow i)\cup(PE(s,v\downarrow i)\ \boxed{?}\ PE(v\downarrow i,w\downarrow i))\rightarrow SA[w\downarrow i]$$

Overview of Approach (1)

- We built a prototype by integrating
  - **SemStorm** [31]:
    - a Hadoop-based file organization storage system
    - supports efficient graph pattern matching query execution using an algebraic query evaluation technique
    - uses Apache Tez as the execution environment.
  - **Serpent** [25,27]:
    - platform for finding all paths between a set of sources and destinations.
    - builds on the path algebraic technique using path-sequences.



Overview of Approach (2)

1. **Query Expression**
   - *Goal:* Minimize disruption to existing infrastructure, e.g. parser.
   - *Solution:* Use syntactic sugar to represent path operator.
2. **Query Planning**
   - *Goal:* Integrate graph pattern matching with path problem solving
   - *Solution:* Modify graph pattern matching query plan by adding algebraic path querying operators to produce a gpqs query plan.
3. **Query Execution**
   - *Goal:* Execute gpqs.
   - *Solution:* Translate gpq logical query plan into physical query plan by introducing required physical query operators.

Introduction of Syntactic Sugar – ?pathVar as a special property variable

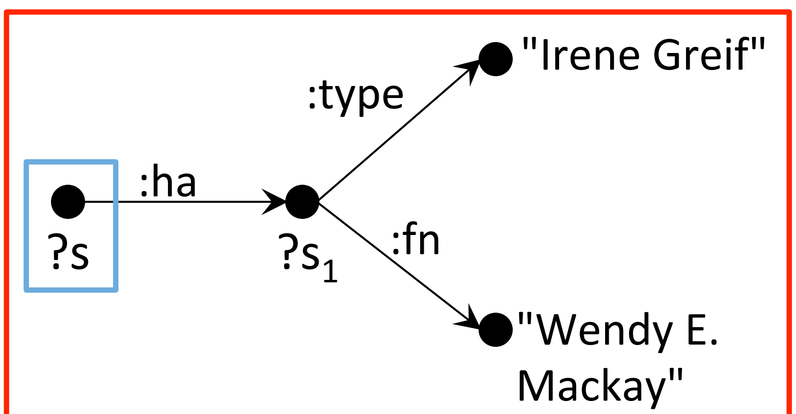
- Avoids change to SPARQL's query syntax.
- For triple pattern **⟨?s ?pathVar ?d⟩**,
  - ?pathVar is actually a path variable
  - removed and handled specially while rest of query is compiled normally as graph pattern query.

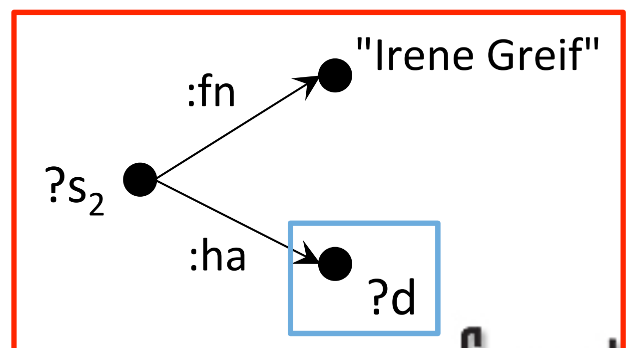
```
SELECT * WHERE {
  ?s1 rdf:type akt:Affiliated-Person .
  ?s1 akt:full-name "Wendy E. Mackay" .
  ?s akt:has-author ?s1 .
  ?s2 akt:full-name "Irene Greif" .
  ?s2 akt:has-affiliation ?d .
  ?s ?pathVar ?d .
}
```

Introduction of Syntactic sugar – ?pathVar as a Property Variable

```
SELECT * WHERE {
  ?s1 rdf:type akt:Affiliated-Person .
  ?s1 akt:full-name "Wendy E. Mackay" .
  ?s akt:has-author ?s1 .
  ?s2 akt:full-name "Irene Greif" .
  ?s2 akt:has-affiliation ?d .
  ?s ?pathVar ?d .
}
```

- **Challenge:** Need to track the path source and destination variables in the graph pattern
- **Solution:** Use SemStorm's datastructures for tracking variables.





Overview of Approach

1. **Query Expression**
   - *Goal:* Minimize disruption to existing infrastructure, e.g. parser.
   - *Solution:* Use syntactic sugar to represent path operator.
2. **Query Planning**
   - *Goal:* Integrate graph pattern matching with path problem solving
   - *Solution:* Modify graph pattern matching query plan by adding algebraic path querying operators to produce a gpqs query plan.
3. **Query Execution**
   - *Goal:* Execute gpqs.
   - *Solution:* Translate gpq logical query plan into physical query plan by introducing required physical query operators.

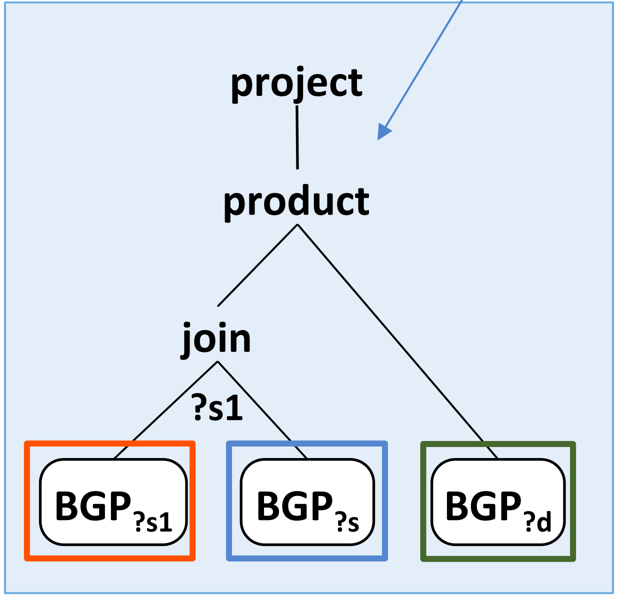
Logical Plan Example as a tree

Disconnected sub-graph patterns leads to cross-product

Apache Jena Query Plan (Sparql Syntax Expression – SSE)

```
(prefix ((akt:http://www.aktors.org/ontology/portal#)
        (rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#))
 (project (?s1 ?s ?d)
  (product
   (join
    (BGP
     [triple ?s1 rdf:type akt:Affiliated-Person]
     [triple ?s1 akt:full-name "Wendy E. Mackay"]
    )
    (BGP
     [triple ?s akt:has-author ?s1]
    ))
   (BGP
    [triple ?s2 akt:full-name "Irene Greif"]
    [triple ?s2 akt:has-affiliation ?d]
   )))
```

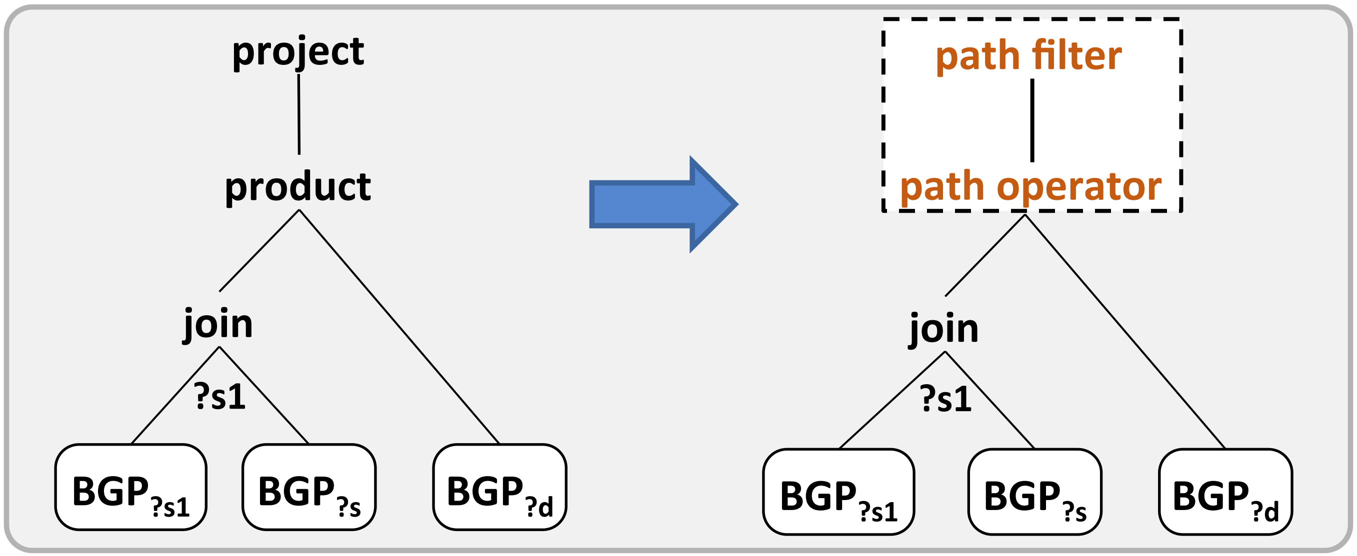
Logical Query Plan as a tree



https://jena.apache.org/

Logical Plan Transformation Example

- Remove cross-product and projection operators and introduce path query operators.



Computer Science
NC STATE UNIVERSITY

Overview of Approach

1. **Query Expression**
   - *Goal:* Minimize disruption to existing infrastructure, e.g. parser.
   - *Solution:* Use syntactic sugar to represent path operator.
2. **Query Planning**
   - *Goal:* Integrate graph pattern matching with path problem solving
   - *Solution:* Modify graph pattern matching query plan by adding algebraic path querying operators to produce a gpqs query plan.
3. **Query Execution**
   - *Goal:* Execute gpqs.
   - *Solution:* Translate gpq logical query plan into physical query plan by introducing required physical query operators.

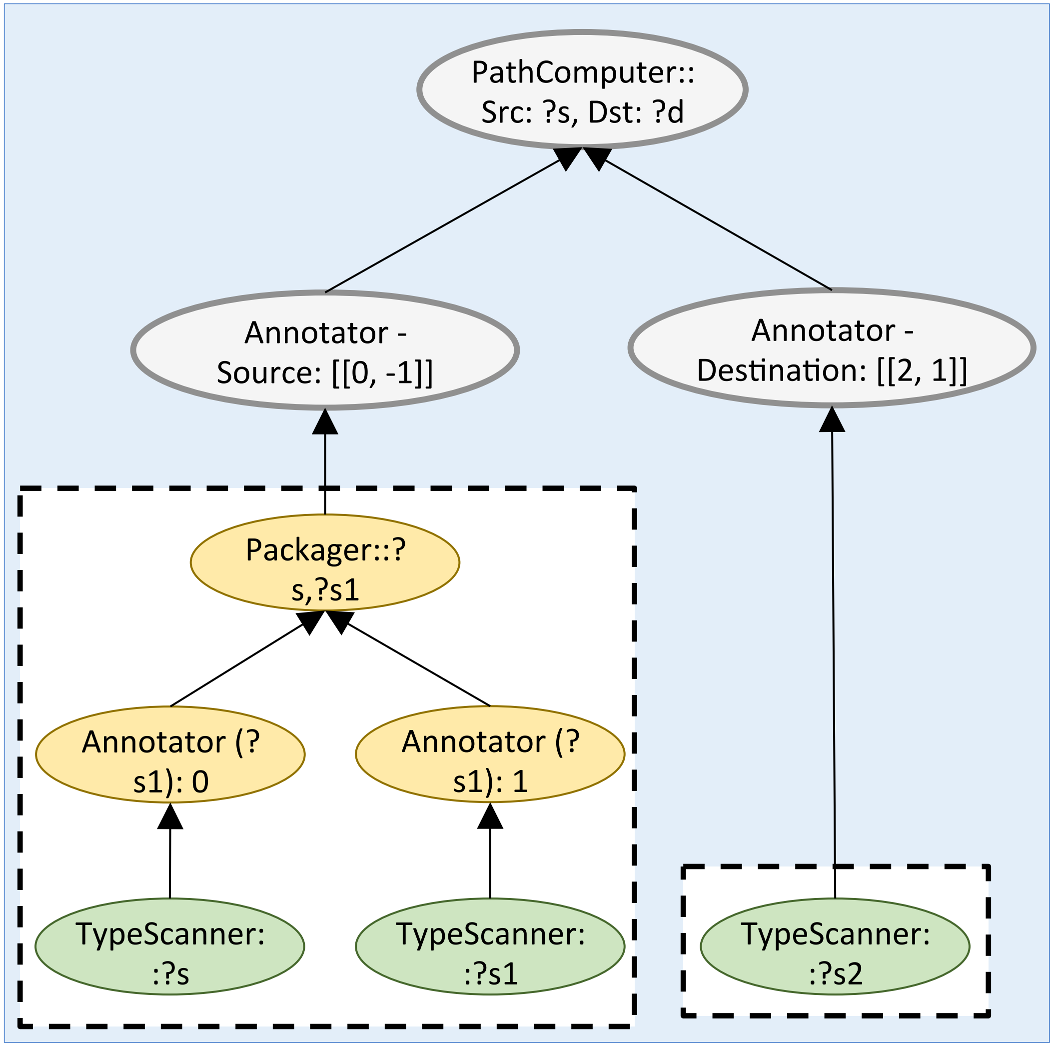
Computer Science
NC STATE UNIVERSITY

Implementation Strategy

- Physical query operators for our platform are Tez vertices
- Physical query plan is represented by the Tez DAG.
- The following physical query operators were introduced
  - **Annotator Vertex for source, destination and constraint variables** identify the source, destination or constraint variables, allowing only the bindings for that variable to pass through.
  - **PathComputer Vertex** is the path operator which performs the final path computation.

Example Query & DAG

Physical query plan



Example gpq

```
PREFIX akt: <http://www.aktors.org/ontology/portal#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT * WHERE
{
  ?s1 rdf:type akt:Affiliated-Person .
  ?s1 akt:full-name "Wendy E. Mackay" .
  ?s akt:has-author ?s1 .

  ?s2 akt:full-name "Irene Greif" .
  ?s2 akt:has-affiliation ?d .
  ?s ?pathVar ?d .
}
```

Computer Science
NC STATE UNIVERSITY

Evaluation

Test Setup:

- We compared our integrated system with an existing platform on
  1. Expressiveness.
  2. Query compilation time comparison with and without path operator.
  3. Performance.
  4. Completeness of results.

- Evaluation was conducted on single node server
  - running HDFS
  - with Xeon octa core x86 64 CPU (2.33 GHz),
  - 40GB RAM,
  - two HDDs (3.6TB and 445GB).

Computer Science
NC STATE UNIVERSITY

Issues with Existing Platforms

Neo4j:

- The fast BFS algorithm is only for **finding shortest path**.
- For finding all paths the **slower exhaustive DFS** is used.
- Gpqs could not be run on Neo4j as it was running out of resources and crashing.

Stardog:

- Uses algebraic operators but **applies path filter first** and then joins with graph pattern.
- This is efficient only if the path filter is highly restrictive.
- Limited support for path constraints.
- Hence, could not compare constrained queries.

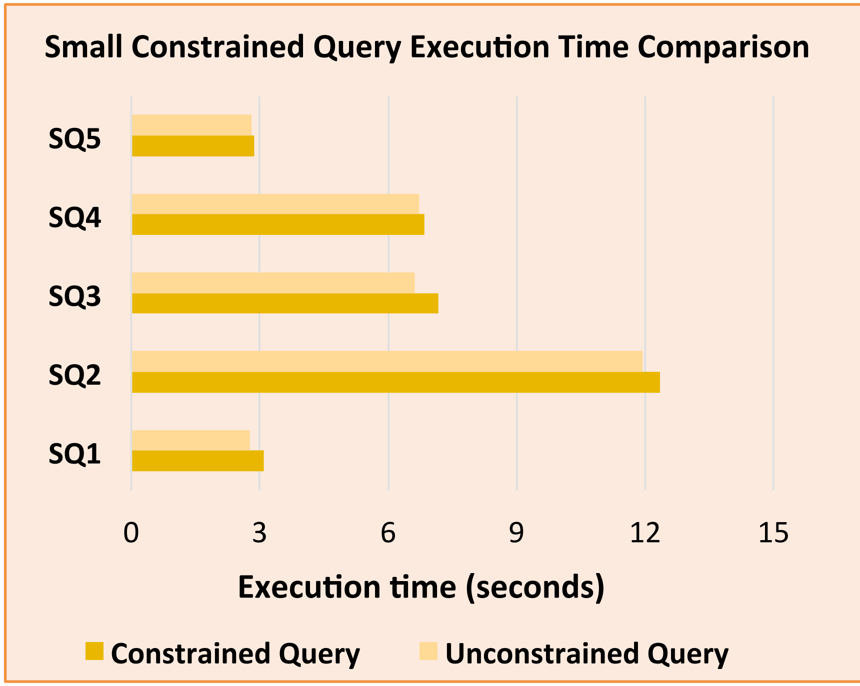
Dataset and Queries

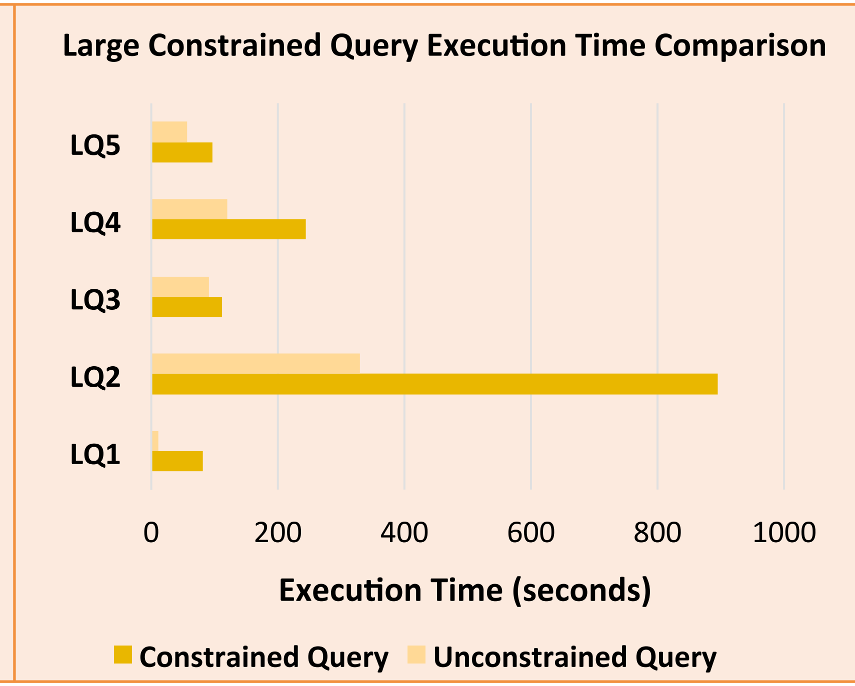
- Our queries were evaluated on the BTC500M dataset (size 0.5GB, 2.5 million triples).
- The queries were formulated to find paths that are at least three hops long.

Number of sources and Destinations					
Queries	**Sources**	**Destinations**	**Queries**	**Sources**	**Destinations**
$SmallQuery_1$	25	2	$LargeQuery_1$	13641	907
$SmallQuery_2$	4	6	$LargeQuery_2$	29974	32583
$SmallQuery_3$	4	3	$LargeQuery_3$	11793	6
$SmallQuery_4$	29	7	$LargeQuery_4$	29974	2290
$SmallQuery_5$	26	31	$LargeQuery_5$	2290	32582

- The queries vary from small set of sources and destinations to very large set of sources and destinations.

Constrained vs. Unconstrained GPQs



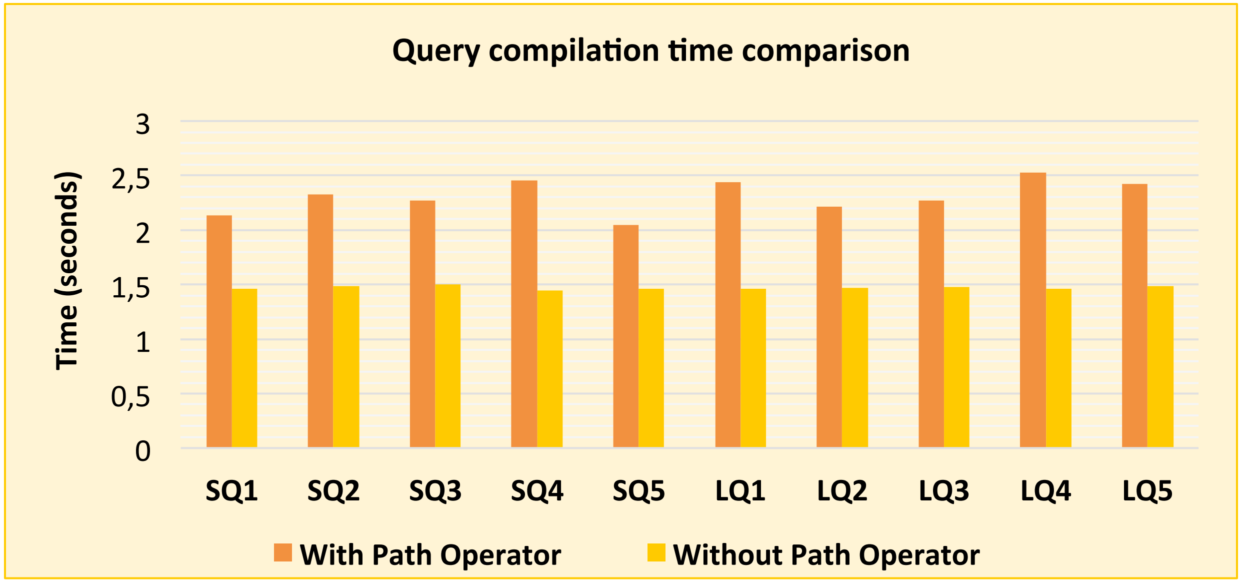


- Stardog has limited support for path constraints.
- Neo4j has predicate functions (all, any, exists, none, single) similar to constraints.
- The constrained queries took longer time to complete since these include an extra filtering step.

Computer Science
NC STATE UNIVERSITY

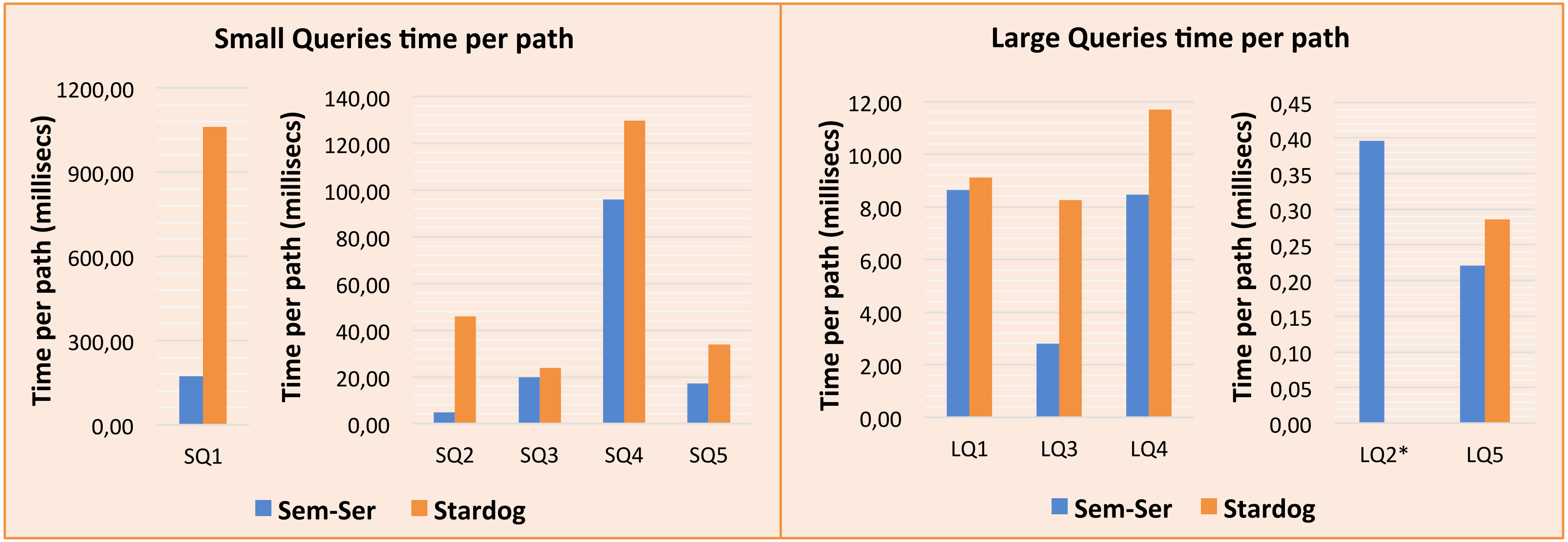
Query Compilation Time Comparison

- The path operator does not have much effect on the query compilation time.
- In most cases the compilation time increased by less than one second.



Computer Science
NC STATE UNIVERSITY

Performance Evaluation

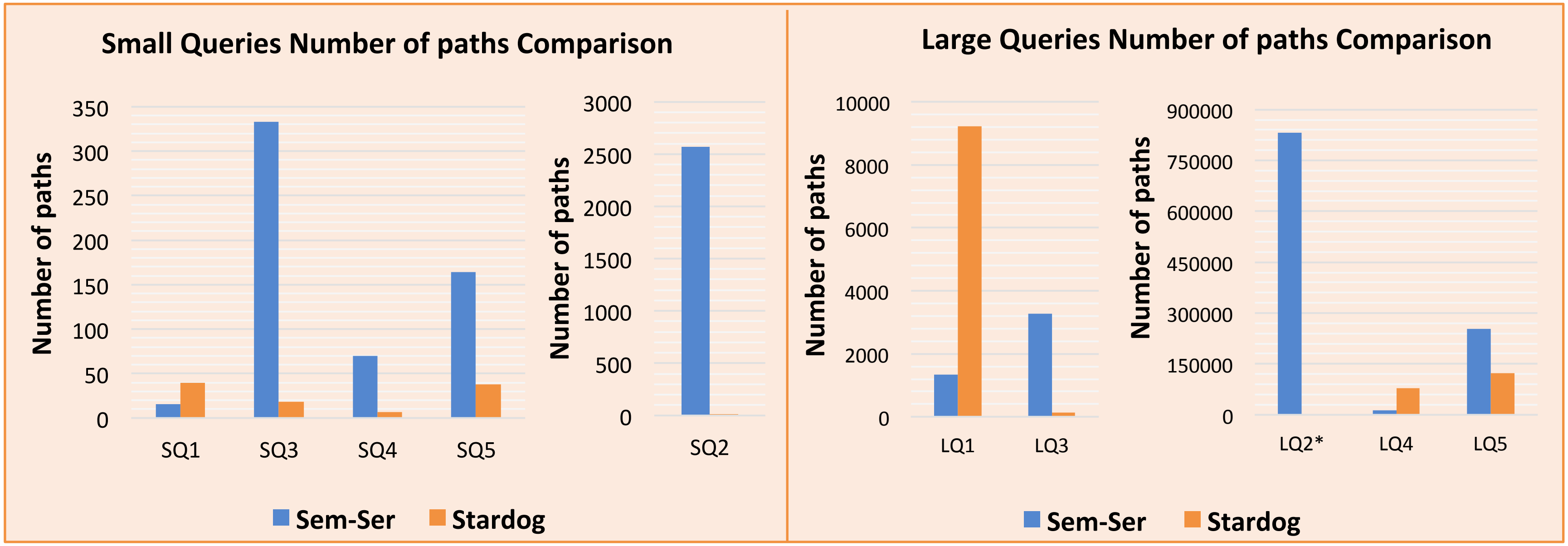


- Stardog performs better in terms of absolute time taken.
- However, for most queries the number of paths found by Stardog is much less.
- Hence, we plotted the time taken per path identified.

Algebraic path evaluation is also more MQO amenable

Computer Science
NC STATE UNIVERSITY

Completeness of Results



- Stardog produced **incomplete results**.
- BTC has a lot of self-loops: triples like
  ⟨acm:58567 akt:has-publication-reference acm:58567⟩.
- Stardog does not consider these triples in its paths.
- Stardog results also contain **duplicate paths**.

Computer Science
NC STATE UNIVERSITY

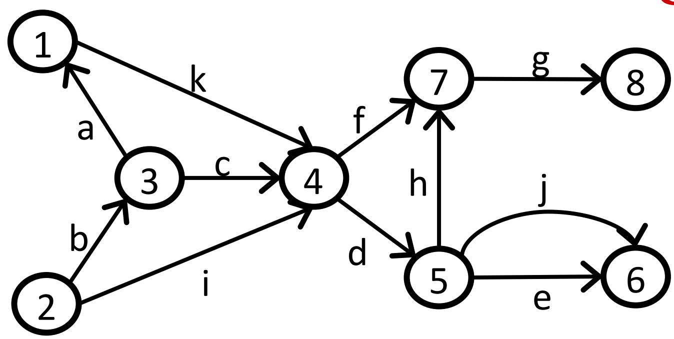
Conclusion

- An algebraic query evaluation strategy for generalized path queries with declaratively defined sources, destinations and constraints.
- A general framework to integrate any graph pattern matching platform with a path computation platform.
- An example implementation of an integrated platform.
- Performance comparison of integrated platform with a popular existing platform.
- The work presented here is partially funded by NSF grant IIS-1218277 and CNS-1526113.

Computer Science
NC STATE UNIVERSITY

Thank You!

Example Path Sequence and Solve Algorithm



1:	(1, 4, *k*)	**2:**	(2, 3, *b*)
3:	(2, 4, *i*)	**4:**	(3, 4, *a* • *k* ∪ *c*)
5:	(4, 5, *d*)	**6:**	(4, 7, *f*)
7:	(5, 6, *e* ∪ *j*)	**8:**	(5, 7, *h*)
9:	(7, 8, *g*)	**10:**	(3, 1, *a*)

Solving(s =1, d): Initialize: PE(s, s) = λ → SA[s], PE(s, d) = ∅ for d ≠ $\mathcal{S}$ → SA[d]

Step i (iteration i): PE(s, w_i) U (PE(s,v_i) • PE(v_i,w_i)) → SA[w_i]

Step 1 (s=1, v_1=1, w_1=4): PE(1, 4) U (PE(1,1) • PE(1, 4))

= SA[4] U (SA[1] • PE(1,4)) = ∅ U (λ • k) = k → SA[4]

. . . . . .

Step 5 (s=1, v_5=4, w_5=5): PE(1, 5) U (PE(1,4) • PE(4, 5))

= SA[5] U (SA[4] • PE(4,5)) = ∅ U (k • d) = k • d → SA[5]

. . . . . .

Step 7 (s=1, v_7=4, w_7=5): PE(1, 6) U (PE(1,5) • PE(5, 6)) = SA[6] U (SA[5] • PE(5, 6))

= ∅ U ((k • d) • (e U j)

= (k • d • e) U (k • d • j) → SA[6]

Computer Science NC STATE UNIVERSITY