# S+EPPs: Construct and Explore Bisimulation Summaries, plus Optimize Navigational Queries; all on Existing SPARQL Systems

Mariano P. Consens
University of Toronto
consens@cs.toronto.edu

Valeria Fionda
University of Calabria, Rende(CS), Italy
fionda@mat.unical.it

Shahan Khatchadourian
University of Toronto
shahan@cs.toronto.edu

Giuseppe Pirrò
ICAR-CNR, Rende(CS), Italy
pirro@icar.cnr.it

## ABSTRACT

We demonstrate S+EPPs, a system that provides fast construction of bisimulation summaries using graph analytics platforms, and then enhances existing SPARQL engines to support summary-based exploration and navigational query optimization. The construction component adds a novel optimization to a parallel bisimulation algorithm implemented on a multi-core graph processing framework. We show that for several large, disk resident, real world graphs, full summary construction can be completed in roughly the same time as the data load. The query translation component supports Extended Property Paths (EPPs), an enhancement of SPARQL 1.1 property paths that can express a significantly larger class of navigational queries. EPPs are implemented via rewritings into a widely used SPARQL subset. The optimization component can (transparently to users) translate EPPs defined on instance graphs into EPPs that take advantage of bisimulation summaries. S+EPPs combines the query and optimization translations to enable summary-based optimization of graph traversal queries on top of off-the-shelf SPARQL processors. The demonstration showcases the construction of bisimulation summaries of graphs (ranging from millions to billions of edges), together with the exploration benefits and the navigational query speedups obtained by leveraging summaries stored alongside the original datasets.

## 1. INTRODUCTION

Querying and exploring big graphs is of crucial importance in several contexts, including social networks, Linked Open Data, and exploratory search. In this context, techniques to "reduce" the size of graphs in order to provide answers to *navigational queries* in a timely manner, and without compromising the quality of results, become a must. One way of achieving this goal is to construct *bisimulation summaries* [10], that is, graph-based indexes that group nodes with equivalent structure [3, 5]. These summaries provide valuable insight into describing and exploring the unique semi-structure of data collections [6].

Recent work [11] shows that summary-based optimizations can improve query performance, at the expense of modifying the internals of the target SPARQL system. Other proposals construct external, stand-alone, indexes based on summaries, but do not integrate them into existing SPARQL query processors [13].

The `S+EPPs` system supports a class of *navigational queries* defined by the Extended Property Paths (EPPs) language [2]. EPPs extend the navigational core of the SPARQL language, that is, property paths, by allowing to write more expressive navigational queries in a succinct way. Our approach to optimize EPPs *on top of existing SPARQL processors* consists of three components. First, an algorithm that translates EPPs into standard SPARQL queries. Second, a mechanism to construct and store summaries alongside the original graph in an RDF store. Third, a rewriting optimization, which converts graph traversal queries into equivalent SPARQL queries to be executed over the RDF graphs of both the original data and the summary.

The literature describes parallel construction of bisimulation summaries [1] using message-passing along with their adaptation to MapReduce environments [9, 12]. The fixpoint nature of bisimulation is well-suited to iterative graph processing; efficient and scalable summary construction is challenging because existing MapReduce solutions do not significantly decrease per-iteration times as the computation progresses.

The `S+EPPs` system constructs bisimulation summaries in roughly the same amount of time that it takes to input the data into a processing framework. We introduce a summary construction implementation [7] in GraphChi [8], a multi-core graph parallel processing framework. We present a novel and very effective *singleton optimization* that allows us to achieve the goal of drastically reducing per-iteration times after only a few iterations. We experimentally validate that our GraphChi-based implementation achieves the goal of constructing summaries in an amount of time similar to the time required to load the dataset and then write the summary for several large, disk resident, real world graphs.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th, 2015, Kohala Coast, Hawaii.
*Proceedings of the VLDB Endowment,* Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.

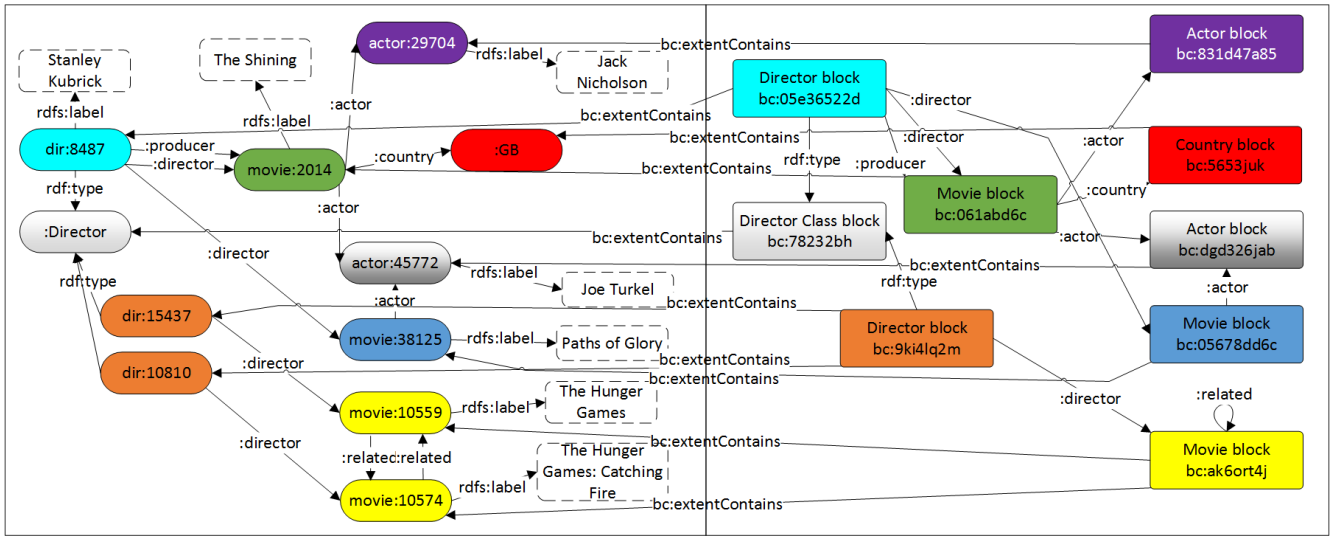


**Figure 1: A fragment of data from LinkedMDB - dataset graph (left) and summary graph (right)**

We also compare our multi-core approach with Hadoop and give evidence to support our GraphChi-based summary construction on a laptop as well as on large multi-core systems.

The `S+EPPs` approach to store summaries alongside the corresponding instance data enables the use of RDF tools (e.g., visualizations) to explore both summaries and instances. In the demonstration we show how exploration tools facilitate the writing of navigational queries.

**Contributions.** To the best of our knowledge, `S+EPPs` is the first system that combines fast building of bisimulation summaries with navigational query translators that leverage existing SPARQL query processors to take advantage of summary-based optimizations without modifying the internals of SPARQL engines. In addition, the availability of summaries as graphs alongside the original instance graphs provides additional support for data exploration activities.

The rest of this paper is organized as follows. In Section 2, we elaborate on the construction of summaries. In Section 3, we outline the language of EPPs, the translations, and how they can be used to implement summary-based optimization techniques. In Section 4, we describe the demonstration scenarios.

## 2. SUMMARY CONSTRUCTION

**Preliminaries.** We consider an RDF *dataset* graph as a finite set of labeled, directed edges between distinctly labeled nodes. We construct structural summaries that group nodes of a dataset graph using bisimilarity as the notion of equivalence. Figure 1 (a) (right) shows a dual bisimulation summary materialized as an RDF graph, which connects to the original data fragment in Figure 1 (a) (left). Each block node in the summary is identified by a URI (minted from a hash function) that represents a non-overlapping subset of the fragment's nodes using `bc:extentContains` edges to each of the dataset graph nodes belonging to the block (referred to as the block's *extent*). Blocks are also connected by edges that summarize the corresponding semantic relationships between dataset nodes across blocks (e.g., `:actor`, `:director`, `:country`, etc.). Specifically, a summary edge means that each node in the source block's extent has an edge to some node in the target block's extent, and dually, each node in the target block's extent has an incoming edge from some node in the source block's extent. This specification warranties that graph traversals in the summary graph will match exactly those nodes in the instance that satisfy those same traversals.

We demonstrate a GraphChi-based algorithm to construct summaries in around the same amount of time as it takes to load the instance dataset and then write the summary. GraphChi supports an iterative Bulk Synchronous Parallel (BSP) model using multi-core parallelism to target scalability on "just a laptop". BSP is a node-centric processing model by which nodes in the current iteration execute an *Update* function that depends on values from the previous iteration. Our summary construction algorithm, described in [7], uses the BSP processing model with the parallel, hash-based approach defined in Blom and Orzan [1], with the important addition of a novel *singleton optimization*. A block with exactly one dataset graph node in its extent is called a *singleton*, and our optimization skips processing singletons. We highlight that the singleton optimization is not captured by the stable/unstable block optimization described in [1]. Also, the system of [9] focuses on optimizing processing of (the few) large blocks, while we target the opposite spectrum of block sizes because we can skip processing (the many) singleton blocks.

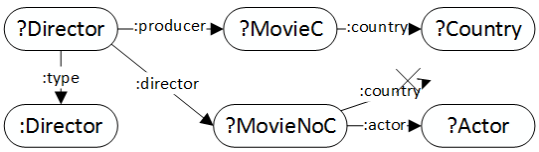


**Figure 2: Example query graph**

```
epp ::= '^' epp | epp '+' | epp '?' | epp'*' | epp '/' epp |
        epp '|' epp |'(' epp ')' | [pos]^1 test [pos]^2 |
        epp '&' epp  |  epp '~' epp
test ::='!' test | test '&&' test | test '||' test |
        '(' test ')' | base
base ::= uri | 'TP('pos ',' epp ')' | 'T('EExp')'
EBInCall ::=BuiltInCall |  pos
pos ::='_s' | '_p' | '_o'
```

**Table 1: Syntax of EPPs**

## 3. EPPS TO SPARQL TRANSLATIONS

Our demonstration focuses on *navigational* queries expressed in the EPPs language [2]. The EPPs syntax shown in Table 1 allows expressing navigational queries as *traversal* operations over edge labels in an RDF graph[1]. EPPs navigational patterns can be executed either on the instance graph (denoted as EPPsI) or the summary graph (denoted as EPPsS). EPPs can be automatically translated into a SPARQL query.

In what follows, we describe the S+EPPs translations using one of the example EPPs used in the demonstration, showing its translation to SPARQL. The navigational query answers *find* ***actors*** *who appear in* ***movies without country*** *information; additionally, the* ***director*** *of those movies should also be the* ***producer*** *of a movie* ***with country*** *information.* We can express this query as an EPPsI from the *:Director* node to all the reachable *?Actor* nodes.

```
:Director   _o(rdf:type && TP(_s,:producer/:country))_s/
(:director && !TP(_o,:country))/:actor  ?Actor
```

The above EPPsI is evaluated on the instance graph in Figure 1 (a) (left) as follows. First, there is a backward traversal over `rdf:type` edges to the node `dir:8487` from where it is necessary to check the existence of a traversal across `:producer` edges followed by `:country` edges. Then, there is also a traversal from those nodes across `:director` edges to movies; then, movies that support a `:country` edge traversal (i.e., `movie:38125`) have to be kept. Finally, a traversal from `movie:38125` across `:actor` edges returns actor nodes, i.e., `actor:45772`. The EPPstoSPARQL translation automatically generates the following SPARQL query over the instance graph.

```
SELECT DISTINCT ?Actor WHERE { GRAPH <http://instance> {
  ?Director rdf:type :Director .
  ?Director :director ?MovieNoC .
  FILTER EXISTS { ?Director :producer ?MovieC .
    ?MovieC :country ?Country . }
  ?MovieNoC :actor ?Actor .
  FILTER NOT EXISTS { ?MovieNoC :country ?someCountry }}}
```

This query corresponds to the graph pattern shown in Figure 2, which can be executed on existing SPARQL processors. Note that movies with country information are discovered via the pattern (`?MovieC`,`:country`,`?Country`) while its counterpart, the edge with a cross through it, searches for movies without country information. Finally, note that the translation of the EPPsI is a SELECT query that, in this case, returns distinct `?Actor` nodes.

Summary-based optimization occurs when a navigational query on the instance is translated into a navigational query on the summary. In the translated query, edge traversals occur between blocks instead of between instance nodes, and a last step is added to traverse `bc:extentContains` edges to navigate from summary blocks back to instance nodes. The EPPsItoEPPsS translation implements this optimization.

[1]note that [pos]$^1$ defaults to _s and [pos]$^2$ defaults to _o

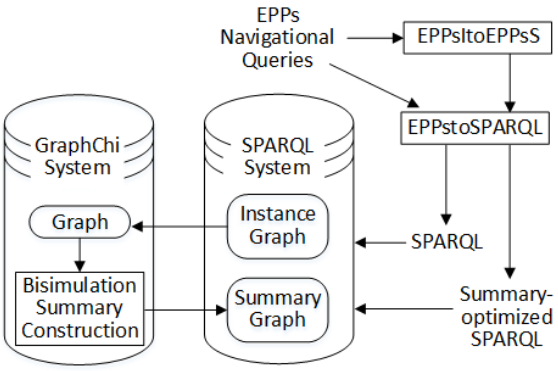


**Figure 3: Architecture of S+EPPs system**

Returning to our example EPPsI from the *:Director* node to all the reachable *?Actors* nodes, the EPPsItoEPPsS translation produces the following EPPsS.

```
:Director _o bc:extentContains _s/_o(rdf:type &&
TP(_s,:producer/:country))_s/(:director && !TP(_o,:country))
/:actor/bc:extentContains  ?Actor
```

The EPPstoSPARQL translation of the example EPPsS gives the following (summary-optimized) SPARQL query.

```
SELECT ?Actor WHERE {
  GRAPH <http://summary/blockedges> {
    ?Director rdf:type ?DirectorTypeBlock .
    ?Director :director ?MovieNoC .
    FILTER EXISTS { ?Director :producer ?MovieC .
      ?MovieC :country ?Country .}
    ?MovieNoC :actor ?ActorBlock .
    FILTER NOT EXISTS { ?MovieNoC :country ?someCountry .}}
 GRAPH <http://summary/extentedges> {
    ?DirectorTypeBlock bc:extentContains :Director.
    ?ActorBlock bc:extentContains ?Actor .} }
```

## 4. DEMONSTRATION

Figure 3 summarizes the S+EPPs approach. The main goal of the demonstration is to inform attendees about the features of S+EPPs. We demonstrate scenarios that include efficient summary construction using a novel singleton optimization, query optimization, and summary-based exploration. Our demonstration includes LinkedMDB [4] and DBpedia, two datasets from the Linked Open Data project. LinkedMDB's dataset graph has around 1.3M nodes and 6M edges, and DBpedia's dataset graph has over 48M nodes and 317M edges.

**Summary Construction.** We demonstrate to attendees that multi-core architecture is a viable way to construct summaries of large graphs. We process LinkedMDB on our demonstration laptop. We process real-world datasets on a remote multi-core machine that has 8 Xeon X6550 2GHz 8-core CPUs, and which can process 64 nodes in parallel. The machine allocates 80GB of main-memory to the Java 8 JVM, though our implementation executes on unmodified GraphChi 0.2 using less than half that amount. Our results show that our implementation computes DBpedia's summary over 50% faster than the time to load the dataset graph and write the RDF summary; We further demonstrate

the effectiveness of our summary construction with a comparison to a Hadoop-based implementation [12] that shows that our implementation is up to 8x faster.

**Query Optimization.** We now describe our demonstration's main scenario - navigational query optimization on graphs with hundreds of millions of edges. Attendees can choose from EPPs that induce neighborhoods with a variety of path lengths, cycles, forward traversals, and backward traversals, and that return zero to millions of results. We also provide attendees with the opportunity to learn to write their own EPPs. Our demonstration will validate that our summaries are effective for navigational query optimization across a variety of unmodified SPARQL processors, as well as a range of summaries.

We begin demonstrating this scenario by showing how to answer a navigational query on LinkedMDB. Using a visualization tool, we show an RDF fragment from the full LinkedMDB dataset that describes relationships between actors, movies and directors using properties including `:actor`, `:year`, and `:country`. We point to *Joe Turkel* as an actor of interest that we would like to query for; then, we show an EPPsI (which we call $Q_a^i$) that returns our actor of interest constructed by using EPPs features like concatenation, constraints, and negation. We execute $Q_a^i$ on an unmodified SPARQL processor and show its results.

We describe a challenge with querying a dataset that has millions of nodes and edges and visualize an RDF fragment of such a dataset's summary. We then show the EPPsS $Q_a^s$ that results from the EPPsItoEPPsS translation of the EPPsI $Q_a^i$, highlighting the `bc:extentContains` predicates linking summary blocks and the instances in their extent. We execute $Q_a^s$ on the same unmodified SPARQL processor as above, where both the LinkedMDB dataset and summary are stored, and show that the result of $Q_a^s$ matches that of $Q_a^i$; we also compare the execution times of $Q_a^i$ and $Q_a^s$.

We show to attendees how EPPs support answering queries using summaries on unmodified SPARQL processors via the EPPstoSPARQL and EPPsItoEPPsS translations. Next, we show the EPPstoSPARQL translation of $Q_a^i$ and how it returns *Joe Turkel*. We then show the EPPstoSPARQL translation of $Q_a^s$ which includes SPARQL graph patterns linking summary blocks with extent instances; then, we underline how this is automatically handled via the EPPstoSPARQL translation. We give evidence of the versatility of our approach on several popular unmodified SPARQL processors including Jena/Fuseki, Sesame, and Virtuoso. In comparing query performance across SPARQL systems, our experimental results show that Sesame benefits less from summaries than Fuseki. Moreover, although using summaries with Fuseki sometimes improves query performance more than Virtuoso, the latter provides the best improvement ratio on average across all three systems with over 7x improvement.

In the last part of the demonstration, we compare query performance of a full structural summary with query performance of *selective summaries*. A selective summary is one that considers a subset of predicates to summarize. The motivation for using fewer predicates is that, since a structural summary aims to improve navigational query performance, then a summary that focuses on specific traversals can potentially provide equal or better query performance; we construct selective summaries using scenarios such as long-running queries and queries from real-world query logs. We again invite attendees to pick from a variety of navigational queries for DBpedia or to write their own EPPs. Suppose an attendee writes the EPPs expression $Q_j$. We execute $Q_j$ on a Virtuoso system that has the full summary loaded alongside the dataset graph and show that the full summary can improve query performance over 10x in comparison to answering queries using the dataset graph. We also execute $Q_j$ on two other Virtuoso systems, each of which loads the full DBpedia instance and one of the d30 or d13 selective summaries (d30 picks the 30 distinct predicates present in at least one user query from real-world query logs, while d13 picks the 13 predicates present in the longest running queries and is a subset of d30). Our results show that selective summaries can improve query performance over 15x.

## Acknowledgments

The authors generously acknowledge the support of NSERC. G. Pirrò's work was partially supported by the Cyber Security Technological District, financed by the Italian Ministry of Education, University and Research.

## 5. REFERENCES

[1] S. Blom and S. Orzan. A Distributed Algorithm for Strong Bisimulation Reduction of State Spaces. *Electr. Notes Theor. Comput. Sci.*, 68(4):523–538, 2002.

[2] V. Fionda, G. Pirrò, and M. P. Consens. Extended Property Paths: Writing More SPARQL Queries in a Succinct Way. In *AAAI*, 2015.

[3] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *VLDB*, 1997.

[4] O. Hassanzadeh and M. P. Consens. Linked Movie Data Base. In *LDOW*, 2009.

[5] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing Simulations on Finite and Infinite Graphs. In *FOCS*, 1995.

[6] S. Khatchadourian and M. P. Consens. ExpLOD: Summary-based Exploration of Interlinking and RDF Usage in the Linked Open Data Cloud. In *ESWC*, 2010.

[7] S. Khatchadourian and M. P. Consens. Constructing Bisimulation Summaries on a Multi-Core Graph Processing Framework Graphs. In *GRADES*, 2015.

[8] A. Kyrola, G. Blelloch, and C. Guestrin. GraphChi: Large-scale Graph Computation on Just a PC. In *OSDI*, 2012.

[9] Y. Luo, Y. Lange, G. H.L. Fletcher, P. Bra, J. Hidders, and Y. Wu. Bisimulation Reduction of Big Graphs on MapReduce. In *BNCOD*, 2013.

[10] T. Milo and D. Suciu. Index Structures for Path Expressions. In *ICDT*, 1999.

[11] F. Picalausa, Y. Luo, G. H. L. Fletcher, J. Hidders, and S. Vansummeren. A Structural approach to Indexing Triples. In *ESWC*, 2012.

[12] A. Schätzle, A. Neu, G. Lausen, and M. Przyjaciel-Zablocki. Large-scale Bisimulation of RDF graphs. In *SWIM*, 2013.

[13] O. Udrea, A. Pugliese, and V. S. Subrahmanian. GRIN: A graph based RDF Index. In *AAAI*, 2007.