3 Search on Linked Data

The Linked Data abstract model, as detailed on Chapter 2, is based on the use of *subject-predicate-object* tuples to represent data. This model is very well suited for a graph representation, in which subjects and objects are represented as nodes and predicates are edges of the graph. Once represented as a graph, data can be displayed for human consumption. This format has the advantages of being visually simple to understand, using a somewhat natural way to represent relationships between data, and the fact that it can be improved to provide additional cues such as node grouping. The tuple format representation is exemplified in Table 1. The graph representation of the same data is illustrated in Figure 6.

subject	predicate	object
mov1	name	The Avengers
mov2	name	Hitchcock
mov1	year	2012
mov2	year	2012
mov1	type	Movie
mov2	type	Movie
mov1	cast	act1
mov2	cast	act1
dir1	directed	mov1
dir2	directed	mov2
dir1	type	Director
dir2	type	Director
dir3	type	Director
dir3	name	Hitchcock
act1	type	Actor
act1	name	Scarlett Johansson

Table 1 - Movies RDF Example SPO Tuples

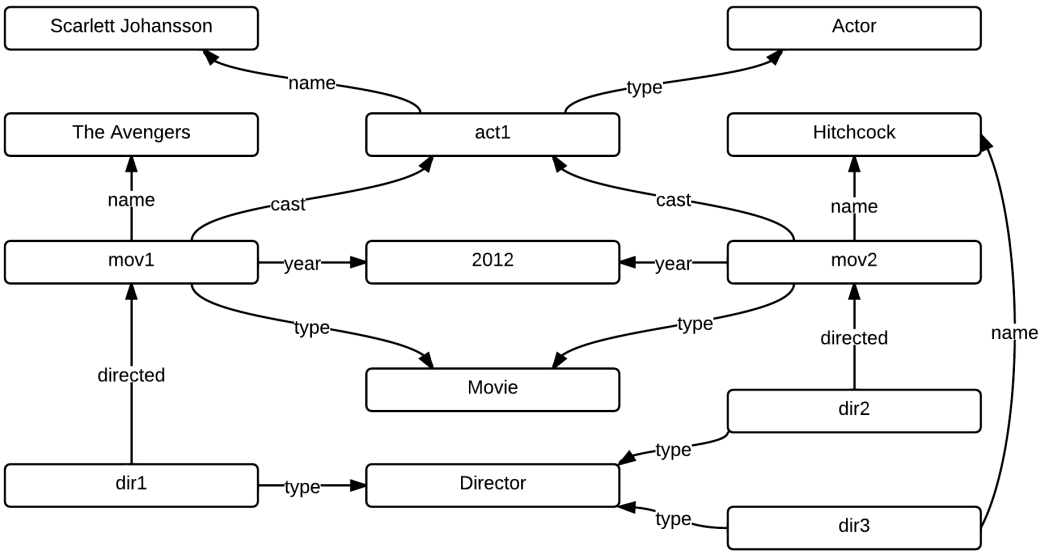

Figure 6 - Movies RDF Example Graph Representation

The ability to search, as well as query, are expected of Linked Datasets. For human consumption, a very natural approach is to use keyword search. This way of searching is very well established for documents on the World Wide Web, where indexing and searching tools such as Google and Bing are extensively used as the starting point for Web data consumption.

Keyword search over Linked Data, however, presents a few challenges. Simply extracting tuples, containing the searched keywords, will leave out of the mass of data other related data that could be relevant to human consumption of the returned result, as can be seen on Table 2.

subject	predicate	object
mov2	name	Hitchcock
dir3	name	Hitchcock

Table 2 - RDF Search example over tuples for the keyword "Hitchcock"

This can be improved if some of the related data concerning selected tuples was also retrieved, which leads to another application of the graph representation. So, instead of selecting tuples, the keyword search should, in addition to returning the selected nodes, also return the neighbourhood of these nodes that are relevant to the search in question. Figure 7 illustrates this scenario. A ranking algorithm can then be used to determine which nodes and how deep the related edges and nodes will be navigated and retrieved along the result set.

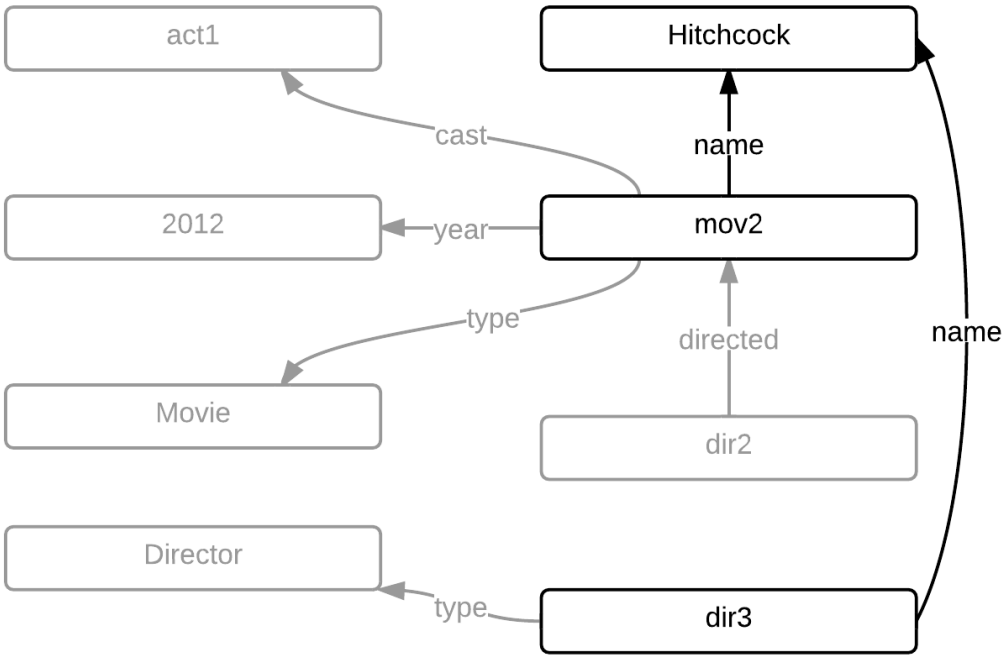

Figure 7 - Example search for keyword "Hitchcock" result as RDF Graph and immediate neighbours

That general search approach involves two steps after the graph representation is available. First, it should retrieve from the graph the parts that match the keywords, including all graph elements that are connected to those matches. Then, it should rank those combined graph segments to present the top ranking results. In this approach, the graph search is a crucial step. This is typically done with pre-computed indexing systems. Those systems build an index of nodes with some relevant metadata about it (e.g., cardinality, related edges and nodes keywords etc.). The task of locating nodes relevant to keywords in an index like this is a well-explored theme and can be efficiently executed.

However, traversing the graph to determine the connections between graph segments is complex and time-consuming. If expected to be solved on-the-fly at query processing time, it will result in an unresponsive system. To minimize that impact, current systems do not perform a full search. Also, index schemes do not provide a way to directly determine the connection between distant nodes, and to retrieve a result set with more or all related data to those selected nodes, a subsequent step of navigating the graph or a pre-defined heuristics must be introduced.

3.1.Tensor-based search

De Virgilio proposes [1] an approach to graph search based on principles of linear algebra. The central concept is to capture all associations on the RDF graph before query processing and indexing those on tensors. Because tensors can be represented as matrices, this approach can be scaled using the extensive applied techniques already developed on tensor calculus and computational algebra, and it can be distributed with recent cloud computing techniques, which is the motivation of this dissertation. Furthermore, the tensor-based approach doesn't need to take into consideration any vocabularies or expected structure of the dataset being indexed, using the abstract RDF model of tuples as described earlier. This suits well the semi-structured nature of many datasets that don't have a well-defined schema.

For a full description of the proposal, the reader should refer to the original paper [1]. Here the mathematic principles and some intermediate definitions are skipped to retain focus on main definitions, outputs, and the necessary process steps of the approach to produce those outputs. The details about the specific algorithms to generate the resulting indexes and the sparse matrix are detailed on Chapter 4, where they are analysed to extract their requirements for the distributed approach proposed in this dissertation.

3.2.Full-paths and templates

The tensor-based approach defines some initial concepts. First, it describes the graph as:

$G = (V,L,E)$, where:

G is a labelled directed graph
V is the set of nodes
L is the set of labels
E is the set of edges, on the form $e(v,u)$, where v,u belong to V and e belongs to L.

A node v_{source} is a source if there are no edges directed to it, i.e., v_{source} belongs to V, but there is no $e(v,u)$ where $u = v_{source}$. Similarly, a node v_{sink} is a

sink if there are no edges directing from it, i.e., v_{sink} belongs to V, but there is no $e(v,u)$ where $v = v_{sink}$. All other nodes are intermediate nodes. From a graph perspective, any node on the graph can be reached from at least one path originating from a source node. This leads to the definition of a full-path:

$p_t = v_1\text{-}e_1\text{-}v_2\text{-}e_2\text{-}...\text{-}e_f\text{-}v_f$, where:

v_i belongs to V

e_j belongs to L

v_1 is a source

v_f is a sink

$v_i\text{-}e_j$ is a token in p_t

f is the length of the full-path p_t, i.e., the number of its nodes.

Full-paths are similar when they have the same sequence of edge labels composing it. They can be expected to convey homogeneous information. To represent that, given a full-path, its sequence of edge labels, i.e., e_j, is its template. Similar full-paths, i.e., full-paths with the same sequence of edge labels, will share the same template. All nodes, full-paths, and templates should be indexed on id-value stores, represented as tables in the examples seen on Table 3, used as reference on the next step of the process.

id	node
1	mov1
2	The Avengers
3	mov2
4	Hitchcock
5	2012
6	Movie
7	act1
8	dir1
9	dir2
10	Director
11	dir3
12	Actor
13	Scarlett Johansson

id	template
1	*-type-*
2	*-directed-*-year-*
3	*-directed-*-name-*
4	*-directed-*-cast-*-type-*
5	*-directed-*-cast-*-name-*
6	*-directed-*-type-*
7	*-name-*

id	full-path
1	dir1-type-Director
2	dir1-directed-mov1-year-2012
3	dir1-directed-mov1-name-The Avengers
4	dir1-directed-mov1-cast-act1-type-Actor
5	dir1-directed-mov1-cast-act1-name-Scarlett Johansson
6	dir1-directed-mov1-type-Movie
7	dir2-directed-mov2-year-2012
8	dir2-directed-mov2-cast-act1-type-Actor
9	dir2-directed-mov2-cast-act1-name-Scarlett Johansson
10	dir2-directed-mov2-name-Hitchcock
11	dir2-directed-mov2-type-Movie
12	dir2-type-Director
13	dir3-name-Hitchcock
14	dir3-type-Director

Table 3 - Example of tables for nodes, full-paths and templates indexes

3.3.Sparse matrix representation

The tensorial representation of the RDF graph can be better understood by its implementation as a sparse matrix of tuples, supported by three sets of indexes. On the sparse matrix, the relation between a given node index, a full-path, and the set of properties of that relation can be described as:

$M_{i,j} = (o,l,t)$, where:

i denotes the index of the given node

j denotes the index of the given full-path

o is the position of the node with index i on the full-path with index j

l is the length of the full-path with index j

t is the index of the template of the full-path with index j

To further simplify implementation, the multidimensional matrix can be flattened to a bi-dimensional matrix of tuples $s = (o,l,t)$. And to avoid unnecessary usage of memory, in the cases where the node with index i does not belong to the path with index j, the tuple can be omitted. As a result, the tensorial representation can be implemented as a sparse matrix of tuples.

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}	n_{13}
p_1	-	-	-	-	-	-	-	(1,2,1)	-	(2,2,1)	-	-	-
p_2	(2,3,2)	-	-	-	(3,3,2)	-	-	(1,3,2)	-	-	-	-	-
p_3	(2,3,3)	(3,3,3)	-	-	-	-	-	(1,3,3)	-	-	-	-	-
p_4	(2,4,4)	-	-	-	-	-	(3,4,4)	(1,4,4)	-	-	-	(4,4,4)	-
p_5	(2,4,5)	-	-	-	-	-	(3,4,5)	(1,4,5)	-	-	-	-	(4,4,5)
p_6	(2,3,6)	-	-	-	-	(3,3,6)	-	(1,3,6)	-	-	-	-	-
p_7	-	-	(2,3,2)	-	(3,3,2)	-	-	-	(1,3,2)	-	-	-	-
p_8	-	-	(2,4,4)	-	-	-	(3,4,4)	-	(1,4,4)	-	-	(4,4,4)	-
p_9	-	-	(2,4,5)	-	-	-	(3,4,5)	-	(1,4,5)	-	-	-	(4,4,5)
p_{10}	-	-	(2,3,3)	(3,3,3)	-	-	-	-	(1,3,3)	-	-	-	-
p_{11}	-	-	(2,3,6)	-	-	(3,3,6)	-	-	(1,3,6)	-	-	-	-
p_{12}	-	-	-	-	-	-	-	-	(1,2,1)	(2,2,1)	-	-	-
p_{13}	-	-	-	(2,2,7)	-	-	-	-	-	-	(1,2,7)	-	-
p_{14}	-	-	-	-	-	-	-	-	-	(2,2,1)	(1,2,1)	-	-

Figure 8 - Sparse matrix as a bi-dimensional matrix of tuples

3.4.Retrieval and maintenance queries

The tensor-based approach also defines a set of six processing queries for information retrieval over the stored sparse matrix and six maintenance queries for updating and changing the graph when the dataset is updated. The retrieval queries are:

- Node Query - takes as input a node *n* to find all full-paths containing the given node;
- Final Node Query - a specialization of the Node Query, it also takes as input a node *n* to find all full-paths that end at the given node;
- Path Query - takes as input a full-path *p* to find all nodes belonging to the given full-path;
- Path Intersection Query - takes as input two full-paths p_1 and p_2 to verify if an intersection exists between them;
- Path Intersection Retrieval Query - takes as input a full-path *p* to find all full-paths that have an intersection with the given full-path;
- Path Cutting Query - takes as input a full-path *p* and a node *n* to find the part of the given full-path that either starts or ends on the given node.

And the maintenance queries are:

- Node Deletion - removes a given node *n* from the graph;
- Node Insertion - adds a given node *n* to the graph;
- Edge Deletion - removes a given edge *e* between two existing nodes from the graph;
- Edge Insertion - adds a given edge *e* between two existing nodes to the graph;
- Node Update - updates name or value of the given node;
- Edge Update - updates the name of an existing edge.

3.5.Summary

In this chapter, we provided an overview of the challenges of Linked Data search. We discussed the problems of searching and extracting the correct amount of information from an RDF graph and the challenges to retrieve data on large

scale datasets. We then discussed the tensor-based approach proposed by de Virgilio and hinted at a simplified set of definitions for the approach. These simplifications keep the original retrieval and maintenance sets and flatten the multi-dimensional matrix to a bi-dimensional one composed of tuples. In the next chapter, we elaborate on this idea, by demonstrating how the proposed simplification - working with a bi-dimensional matrix - can be used in a distributed solution, allowing a scale up parallel system to store the RDF graph dataset.