LarKC

The Large Knowledge Collider:

a platform for large scale integrated reasoning and Web-search

FP7 – 215535

D2.4.2 Approximate Activation Spreading

Coordinator: Ivan Peikov, Onto

With contributions from: Maurice Grinberg, Vladimir Haltakov, Hristo Stefanov, Atanas Kiryakov, Damyan Ognyanoff, Ruslan Velkov

Document Identifier:	LarKC/2008/D2.4.2 /v2
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	9.0
Date:	25.03.2010
State:	Final
Distribution:	Public

Executive Summary

This deliverable describes two new approaches to the task of selecting ('priming') relevant nodes with respect to a query. Such a reduction of the queried graph can be viewed as a query optimization technique, especially useful in the context of sufficiently large datasets. Alternatively, 'priming' provides means for selection as needed by a LarKC selection plug-in. The new approaches can complement or substitute the standard Spreading Activation (SA) method DualRDF, presented in D2.4.1 [1]. DualRDF is relatively slow because it is based on the multiplication of a high rank matrix with a vector (the rank is equal to the number of nodes in the dataset and can be of the order of millions). In this document two approximate SA methods are proposed, featuring high computational efficiency. Both methods perform extraction of a numerical connectivity matrix for the nodes in the dataset, based on the statements in the dataset, application of optimized storage format and operations on it.

The first of them is called Cluster based SA (CbSA) and is based on the assumption that meaningful clusterings (i.e. sets of clusters) of the nodes in the dataset can be found. The current deliverable presents a formal framework for this approach, which will facilitate future analysis, implementation, and tests.

The second approach presented, Node Selection based SA (NSbSA), takes advantage of the sparsity of the nodes' connectivity matrix and existing formats for compact representation of sparse matrices.

In the course of work, several computational explorations were carried out. Various programming languages and hardware were compared which involved combinations of Matlab, Java, C, and GPU-based (NVIDIA CUDA, [3]) computations, as well as different sparse matrix storage formats. The Linked Data Semantic Repository (LDSR, [4]), which contains about 100 million nodes and about 850 million statements, was used as a testbed for the NSbSA approach. The numerical evaluation indicates that NSbSA is quite promising for real-time applications, when used with a CUDA device and optimized data storage. When applied to the LDSR dataset, NSbAS activated $65*10^6$ nodes in 15 iterations for about 5 seconds (in comparison, in similar settings DualRDF selected about 75 thousand entities in 34 seconds). Possible combinations of DualRDF, implementing exact SA, and approximate, but more efficient, CbAS and NSbAS methods are also discussed.

DOCUMENT INFORMATION

IST Project Number	FP7 - 215535	**Acronym**	LarKC
Full Title	The Large Knowledge Collider: a platform for large scale integrated reasoning and Web-search		
Project URL	http://www.larkc.eu/		
Document URL			
EU Project Officer	Stefano Bertolo		

Deliverable	**Number**	D2.4.2	**Title**	Approximate Activation Spreading
Work Package	**Number**	WP2	**Title**	

Date of Delivery	**Contractual**		**Actual**	
Status	version 9.0		final ☒ draft □	
Nature	prototype ☒ report □ dissemination □ other □			
Dissemination level	public ☒ consortium □			

Authors (Partner)				
Responsible Author	**Name**	Ivan Peikov	**E-mail**	ivan.peikov@ontotext.com
	Partner	Onto	**Phone**	

Abstract (for dissemination)	This deliverable describes two new approaches to the implementation of spreading activation components and the first steps in their analysis and testing. Numerical evaluations over LDSR are provided.
Keywords	spreading activation, node clustering, node selection

Version Log			

Project Consortium Information

Participant's name	Partner	Contact
Semantic Technology Institute Innsbruck, Universitaet Innsbruck		Prof. Dr. Dieter Fensel, Semantic Technology Institute (STI), universitaet Innsbruck, Innsbruck, Austria, E-mail: dieter.fensel@sti-innsbruck.at
AstraZeneca AB		Bosse Andersson AstraZeneca Lund, Sweden Email: bo.h.andersson@astrazeneca.com
CEFRIEL - SOCIETA CONSORTILE A RESPONSABILITA LIMITATA		Emanuele Della Valle, CEFRIEL - SOCIETA CONSORTILE A RESPONSABILITA LIMITATA, Milano, Italy, Email: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O.		Michael Witbrock, CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia, Email: witbrock@cyc.com
Höchstleistungsrechenzentrum, Universitaet Stuttgart		Georgina Gallizo, Höchstleistungsrechenzentrum, Universitaet Stuttgart, Stuttgart, Germany, Email: gallizo@hlrs.de
MAX-PLANCK GESELLSCHAFT ZUR FOERDERUNG DER WISSENSCHAFTEN E.V.		Dr. Lael Schooler Max-Planck-Institut für Bildungsforschung Berlin, Germany Email: schooler@mpib-berlin.mpg.de
Ontotext Lab, Sirma Group Corp		Atanas Kiryakov, Ontotext Lab, Sofia, Bulgaria Email: atanas.kiryakov@sirma.bg
SALTLUX INC.		Kono Kim, SALTLUX INC, Seoul, Korea, Email: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT		Dr. Volker Tresp, SIEMENS AKTIENGESELLSCHAFT, Muenchen, Germany, E-mail: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD		Prof. Dr. Hamish Cunningham, THE UNIVERSITY OF SHEFFIELD Sheffield, UK, Email: h.cunningham@dcs.shef.ac.uk

VRIJE UNIVERSITEIT AMSTERDAM		Prof. Dr. Frank van Harmelen, VRIJE UNIVERSITEIT AMSTERDAM, Amsterdam, Netherlands, Email: Frank.van.Harmelen@cs.vu.nl
THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY		Prof. Dr. Ning Zhong, THE INTERNATIONAL WIC INSTITUTE, Mabeshi, Japan, Email: zhong@maebashi-it.ac.jp
INTERNATIONAL AGENCY FOR RESEARCH ON CANCER	International Agency World Health Organization	Dr. Paul Brennan, INTERNATIONAL AGENCY FOR RESEARCH ON CANCER, Lyon, France, Email: brennan@iarc.fr
INFORMATION RETRIEVAL FACILITY		Dr. John Tait INFORMATION RETRIEVAL FACILITY Vienna, Austria Email: john.tait@ir-facility.org

TABLE OF CONTENTS

List of Acronyms

SA	Spreading Activation
RDF	Resource Description Framework
LDSR	Linked Data Semantic Repository
LOD	Linked Open Data

List of Tables

List of Figures

1. Introduction

Spreading Activation (SA) methods described in this deliverable are intended to extend and complement the already existing components (DualRDF and PageRankRDF) proposed in D2.4.1 [2]. Their design and analysis is a further step in the exploration of cognitively-inspired methods for selection[1] and more generally for reasoning, with emphasis on performance. The focus of the approaches presented here is the *computational efficiency* in processing very large datasets. While based on conventional SA, the methods presented are approximations, based on the cognitively-inspired factors broadly defined as *clustering* and *relevance*, and are expected to lead to meaningful selections in less time.

The first approach, described here, called Cluster based SA (CbSA), is based on the assumption that the nodes of the dataset can be clustered in a meaningful way. Clustering can be quite general, e.g. based on the degree of connectivity of the nodes, type of nodes, taxonomical organization, etc. The basic idea of this approach is to substitute the $n_{nodes} \times n_{nodes}$ dimensional connectivity matrix between the nodes with a $n_{nodes} \times n_{largest\ cluster}$ dimensional connectivity matrix between the nodes and the clusters to which they belong. In this deliverable, the notation and the basic formalisms for this approach are presented together with suggestions for further implementation.

The second approach, called Node Selection based SA (NSbSA), takes advantage of the *sparsity* of the connectivity matrix and explores the efficiency of substituting standard SA, based on matrix-vector multiplication, with path finding (starting from the query nodes) in a graph, based on finding non-zero elements and predefined node selection rules. The intention behind this approach is to follow relevant types of connections (predicates) and make a list of the nodes connected to the query nodes, counting the number of times they have been encountered without using the same connection twice. The number of independent selections can be used as a measure of relevance of the context and the query at hand. This approach is evaluated numerically and compared to DualRDF.

For the numerical tests presented below, the Linked Data Semantic Repository (LDSR, [4]), has been used. LDSR contains about 100 million nodes and about 850 million statements. SA is performed using a connectivity matrix, extracted from the dataset and processed separately by the SA components. One of the methodological results reported here is the comparison of various programming languages, technologies and storage formats with respect to computational efficiency. This analysis allowed for considerable memory usage optimizations and implementation of computationally efficient algorithms, including parallel computations on NVIDIA CUDA (Compute Unified Device Architecture) cards [3].

This deliverable is structured as follows:

- In Section 2, CbSA is described and a formal framework, enabling further analysis and implementation, is introduced. The possibilities for gain in efficiency within this approach, compared to DualRDF, are discussed;

[1] In LarKC, “selection” is a phase in a pipelined reasoning process during which part of the data (an RDF dataset), relevant to the reasoning task, is selected for further processing.

- In Section 0, NSbSA is introduced, discussed, and compared to the standard SA mechanisms implemented in DualRDF;
- In Section 3, NSbSA is evaluated with respect to computational and storage efficiency, and compared to DualRDF. Comparisons between various implementations (using Matlab, C, Java, and NVIDIA CUDA card) and storage formats are also given.
- In Section 5, a summary of the deliverable is given and possible directions for future work are outlined.

2. Clustering Based Spreading Activation (CbSA)

CbSA is based on the assumption that the nodes in the dataset can be grouped in clusters. Such clusters can exist in the nodes' connectivity matrix and could be made explicit by rearranging the matrix rows and columns. On the other hand, clusters can be identified by additional analysis of the dataset, taking into account the classes of nodes, types of predicates, nodes' degree of connectivity, and/or some explicit or derived hierarchy (e.g., see [2] and [5]). The basic idea of this approach is to substitute the original node connectivity matrix, which is $n_n \times n_n$ dimensional (where n_n is the number of nodes in the dataset), with a matrix whose dimension is $n_n \times \max(n(C_1), n(C_2), \ldots, n(C_{n_C}))$ (where n_C is the number of clusters and $n(C_m)$ is the number of nodes in the cluster C_m). This latter matrix is expected to be of a considerably lower dimension than the original connectivity matrix, as $n_C << n_n$. If such a clustering is identified, the connectivity matrix can be reduced to sub-matrices (e.g. see [6]). In the case that not only the nodes can be grouped in clusters but the clusters themselves can grouped in clusters (clusters of clusters), additional connectivity matrices will be needed. For instance, chairs, tables, sofas, shelves, etc. can belong to different subclasses of furniture, e.g. kitchen furniture, restaurant furniture, cinema furniture, etc. Each of these specific pieces of furniture will belong to its cluster (class) but the clusters belong to a larger cluster (class), namely 'furniture'. On the other hand, a cluster of nodes, e.g. related to furniture, can be semantically related to another cluster (class in this particular case) of objects. For instance, a kitchen table can be strongly related to knife and fork. In order to have a chance to activate the node 'knife' if 'kitchen table' is activated, the clusters they belong to (two distinct classes in this example) must be connected.

If a node from a cluster is a source of activation (e.g. is part of the query), it can activate all the other nodes in the cluster. If the connections are the same, e.g. 1's, all the members of the cluster will get activated, which will affect the selectivity and the purpose of SA. Therefore, the usefulness of this approach is strongly related to the identification of the appropriate distance metric. Distance should allow for differentiation between the nodes in a cluster, attributing larger weights to the most typical representatives of a cluster. For instance, if the cluster is defined by a node with high degree of connectivity (a hub), one possible choice for a distance metric could be the weighted inverse shortest distance between a node and the hub node.

In the next section, we describe the basic notation and formalisms of this approach, followed by suggestions for further implementation.

2.1. Basic Terms and Notation

The following notation is introduced to simplify the description of the CbSA approach:

- $\mathbf{K}(id, \mathbf{O}_{id}, n_n)$ – a knowledge base (dataset), identified by its id, built on top of a set of ontologies $\mathbf{O}_{id} \subseteq \{\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_{n_O}\}$; n_n is the number of nodes in the dataset and n_{n_O} the number of available ontologies.

- $k_l(id)$ – defines a specific node $l \in \{1,\dots,n_{id}\}$ in $\mathbf{K}(id,\mathbf{O}_{id},n_n)$.
- $C^{(m)}(id,n_{C^{(m)}})$ – the m^{th} clustering (set of clusters) of order (number of clusters) $n_{C_m} \in \{1,\dots,n_n\}$, where $n_{C^{(m)}} = 1$ stands for $\mathbf{K}(id,\mathbf{O}_{id},n_n)$ and $n_{C^{(m)}} = n_n$ stands for the set of all $k_l(id)$.

The notations concerning the clusters are the following:

- $C_j^{(m)}(id,n_{C_j^{(m)}})$ – is the j^{th} cluster in clustering m for the knowledge base id and $j \in \{1,\dots,n_{C^{(m)}}\}$; the dimension of $C_j^{(m)}(id,n_{C^{(m)}})$, i.e. the number of nodes in this cluster is $n_{C_j^{(m)}}$.
- $\mathbf{W}^{(l)}(id,C^{(m)})$ – connectivity matrix between the nodes in the dataset id for the clustering $C^{(m)}(id,n_{C^{(m)}})$; the index l refers to the metric used to define the weights.
- $\mathbf{D}^{(l)}(id)$ – the distance matrix for dataset id based on the metric l.

In the rest of section 2, for simplicity and without loss of generality, the indexes, referring to the dataset and ontologies will be dropped. All the equations, given below, can be expressed by using more than one dataset and ways of clustering. However, the purpose of this section is not to derive rigorously a complete formalism for the general case, but rather to explain the main idea in some detail and demonstrate the benefits of the approach.

2.2. *Distance and Connectivity Matrices*

When clustering is used for activation optimization, the connection between any query node and the set of clusters has to be specified. In order to activate the most relevant nodes with respect to a set of clusters (e.g. taxonomy based), the 'distance' between each node in the dataset and the clusters must be evaluated. In principle, when distance between nodes is calculated with respect to a specific metric, the clusters can be derived using standard clustering methods like k-means, QT, fuzzy c-means, etc. [8].

At least two options are possible:

- To split the whole dataset into clusters and to enforce that each node belongs to exactly one cluster (e.g., to the one it is closest to).
- To define fuzzy clustering which allows for each node to 'belong' to several clusters, as measured by the distance between the node and these clusters.

The distance matrix $\mathbf{D}^{(l)}$ can be based on:

- $\mathbf{D}^{(l)}(n_n \times n_C)$ – the distance matrix between any node in n and the clusters in C clustering based on the metric l; the corresponding connectivity matrix will be denoted $\mathbf{W}_{nC}^{(l)}(n_n \times n_C)$.
- $\mathbf{D}^{(l)}(n_C \times n_C)$ – the distance matrix between the clusters of one clustering, based on the metric l; the corresponding connectivity matrix will be denoted $\mathbf{W}_{CC}^{(l)}(n_C \times n_C)$..

- $\mathbf{D}^{(l)}(n_{C^{(m)}} \times n_{C^{(j)}})$ – the distance matrix between clusters of two different clusterings m and j based on the metric l; the corresponding connectivity matrix will be denoted $\mathbf{W}^{(l)}_{C^{(m)}C^{(j)}}(n_{C^{(m)}} \times n_{C^{(j)}})$.

2.3. Spreading Activation Using Clustering

To introduce cluster based spreading activation, it is useful to introduce the basic notation and rules in the standard case. Spreading activation requires a vector with initial activities (input vector) and a connectivity matrix (weight matrix) in which each connection is a predicate from the dataset and is equal to 1. If we assume that the activities of the input elements (target) are 1's and that they are incorporated in the dataset, spreading activation can be implemented as follows:

- $\mathbf{t}(n_n) = \mathbf{I}_{n_n}[k_{t_1}, k_{t_2}, \ldots, k_{t_n}]$, is the query (target) represented as a vector, where n is the number of nodes in the query, which are identified within the dataset and $\mathbf{I}_{n_n}[l_1, l_2, \ldots, l_n]$ is a n_n-dimensional vector with n elements equal to 1, with indexes listed within the square brackets, and zeros elsewhere;
- $\mathbf{k}(n_n)$ is a vector of all n_n nodes' activations for the dataset $\mathbf{K}(id, \mathbf{O}_{id}, n_n)$;
- $\mathbf{W}^{(0)}(n_n \times n_n)$ is the original nodes' connectivity matrix for all nodes in the dataset, with 1's as weights, standing for existing predicates defined as the metric '0'.

Spreading activation is schematically given by the iteration process (assuming identity activation function and no decay):

$$\begin{aligned}
&\mathbf{t}(n_n) = \mathbf{I}_{n_n}[k_{t_1}, k_{t_2}, \ldots, k_{t_n}] \\
&\mathbf{k}^{(0)}(n_n) = \mathbf{t}(n_n) \\
&\mathbf{k}^{(1)}(n_n) = \mathbf{W}^{(0)}(n_n \times n_n)\mathbf{k}^{(0)}(n_n) \\
&\mathbf{k}^{(2)}(n_n) = \mathbf{W}^{(0)}(n_n \times n_n)\mathbf{k}^{(1)}(n_n) \\
&\\
&\mathbf{k}^{(it)}(n_n) = \mathbf{W}^{(0)}(n_n \times n_n)\mathbf{k}^{(it-1)}(n_n) = [\mathbf{W}^{(0)}(n_n \times n_n)]^{it}\mathbf{k}^{(0)}(n_n)
\end{aligned} \tag{1}$$

where $\mathbf{t}(n_n)$ is the initial query (considered to be the target for the retrieval); and $\mathbf{W}^{(0)}$ is the original nodes' connectivity matrix.

The SA process given by eq. (1), is the one implemented in DualRDF [2] and involves matrix-vector multiplication which is very resource demanding for large matrices. Therefore, approximating the matrix with a block-diagonal matrix can improve substantially the performance. This can be achieved, for instance, when nodes are grouped into clusters.

Let us assume that a meaningful clustering of the nodes has been found, based on some distance metric, and each node belongs to only one cluster (which can be a strong approximation if the nodes are interrelated). Then, for each node, the distance between the node and the center of the cluster is known, as well as the distance between the centers of the clusters, [8]. As discussed above, the distances can be converted to connection weights, e.g. by taking the inverse of the distance, i.e. smaller distances will correspond to larger weight and vice versa. These weights define a connectivity matrix between the nodes and the clusters (each node is connected to the cluster to which it belongs) and a connection matrix for the clusters. Using these two connectivity matrices, the standard SA, based on the original nodes' connectivity matrix, can either be decomposed into SA from the nodes to the clusters and then from the clusters to the nodes or, if the connections between the clusters are taken into account, SA can be done in three stages - from the nodes to the clusters, among the clusters, and then from the clusters to the nodes. In the former case the original connectivity matrix is transformed into a block-diagonal matrix as nodes belong to a single cluster and cannot activate nodes from another cluster. For this case, the connectivity matrix reads:

$$\mathbf{W}^{(l)}(n_n \times n_n) = \mathbf{W}_{nC}^{(l)}(n_n \times n_C) \times \mathbf{W}_{nC}^{(l)T}(n_n \times n_C) \tag{2}$$

The matrix $\mathbf{W}_{nC}^{(l)}(n_n \times n_C)$ is the matrix of connections between the nodes and the clusters. The process of spreading activation with the connectivity matrix defined by eq. (2) is schematically shown in Figure 1. This process is similar to activation propagation between the layers of an artificial neural network with linear units in which the hidden layer is composed of clusters, connected to specific input and output nodes.

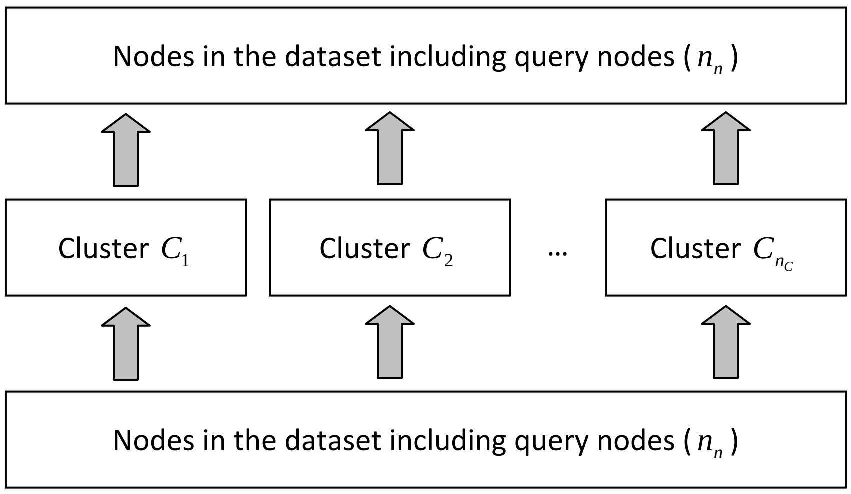
Nodes in the dataset including query nodes (n_n)
Cluster C_1
Cluster C_2
...
Cluster C_{n_C}
Nodes in the dataset including query nodes (n_n)

Figure 1. Spreading activation according to eq. (2)

Assuming that a node can be in just one cluster, the connectivity matrix defined in eq. (2) will have the structure:

$$\mathbf{W}^{(l)}(n_n \times n_n) = \begin{vmatrix} \mathbf{W}(n(C_1^{(l)}) \times n(C_1^{(l)})) & 0 & \cdots & 0 \\ 0 & \mathbf{W}(n(C_2^{(l)}) \times n(C_2^{(l)})) & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & \mathbf{W}(n(C_{n_{C^{(l)}}}^{(l)}) \times n(C_{n_{C^{(l)}}}^{(l)})) \end{vmatrix} \qquad (3)$$

where $\mathbf{W}(n(C_i^{(l)}) \times n(C_i^{(l)}))$ is the weight matrix (based on some distance matrix) between the i nodes belonging to cluster $C_i^{(k)}$.

This block-diagonal structure of the connectivity matrix, expressed via clusters, allows for higher efficiency of computations of SA based on matrix-vector multiplication. Moreover, it allows for parallelization based on the SA within each cluster separately. Despite these advantages, the usefulness of CbSA depends on the choice of clustering and distance metrics which should maximize the selectivity of the method as well as preserve its computational benefits. As mentioned above, if the weights among the clusters and the nodes are 1's, after one iteration of spreading activation all the nodes from a cluster will have the same activation, when a member of this cluster is the source of activation. This compromises the advantages of dimension reduction and of SA.

We can further assume that the clusters themselves are connected as discussed above. If the connectivity matrix for the clusters is accounted for, it will enable activation of a cluster by another cluster and SA will explore a larger set of nodes. In that case, we use the following, more general definition (compared to the one, given in eq. (2); taking into account not only for the connections within a cluster but also among clusters) of the connectivity matrix:

$$\mathbf{W}^{(l)}(n_n \times n_n) = \mathbf{W}_{nC}^{(l)}(n_n \times n_C) \times \mathbf{W}_{CC}^{(l)}(n_C \times n_C) \times \mathbf{W}_{nC}^{(l)T}(n_n \times n_C) \qquad (4)$$

where the notations are the same as the ones for eq. (2), and the matrix $\mathbf{W}_{CC}^{(l)}(n_C \times n_C)$ is the matrix of connections between the clusters.

This case of connectivity, with mutual activation of the clusters, is schematically presented in Figure 2. Using again the multi-layered neural network metaphor, the cluster layer (the hidden layer) will have interconnections among its nodes (see Figure 2).

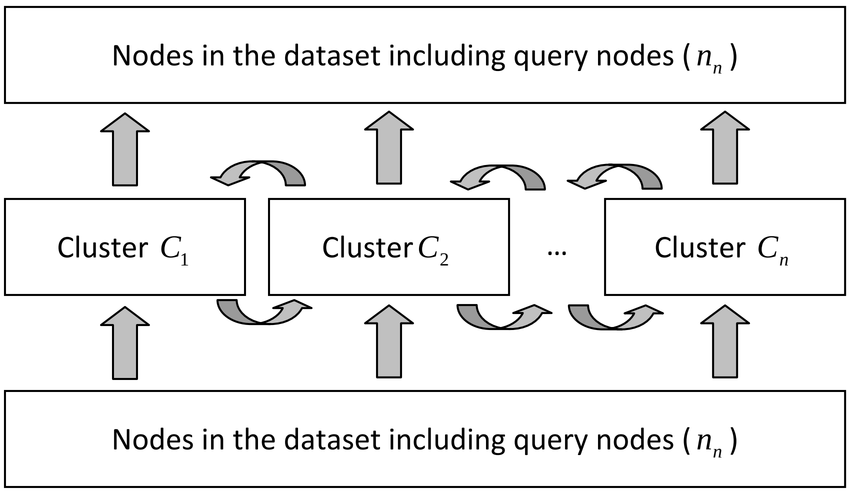
Nodes in the dataset including query nodes (n_n)
Cluster C_1
Cluster C_2
…
Cluster C_n
Nodes in the dataset including query nodes (n_n)

Figure 2. Spreading activation according to eq. (4)

The generalization of the expression for the connectivity (weight) matrix in the case of having more than one clustering is straightforward.

The usefulness of node clustering and the expected gain in efficiency, due to the block-diagonal form of the connectivity matrix and dimensionality reduction, and the possibilities for parallelization can be explored following the approaches suggested in [2], [5], [7] and [9].

Node Selection Based Spreading Activation (NSbSA)

In this section, we describe a fundamentally different approach to SA, compared to the one described in the preceding section. NSbSA is based on the original nodes' connectivity matrix $\mathbf{W}^{(0)}(n_n \times n_n)$, which contains only 1's as weights. This matrix can be derived off-line using the statements from the dataset.

2.4. Method

Instead of matrix-vector multiplication, used in SA approaches like DualRDF or CbSA, NSbSA implements SA by searching for non-zero elements in a sparse vector. The process consists of:

- following the connections of the nodes from the query (the seeds),
- finding the nodes they are connected to,
- iteratively repeating this procedure with the newly selected nodes.

The efficiency of this method, apart from the possibilities for efficient implementation and storage (see the discussion in Section 3), is related to the use of each node which is a source of activation *only once*. Once a node has been used for SA, its connections are never used again. This is not the case in standard SA methods, in which all the connections of active nodes are used over and over again. This is of course extremely resource consuming from computational point of view. In NSbSA, the process of node selection can encounter a node which has been selected in a previous iteration. In this case, a counter attached to each node is incremented. The larger the value of the counter is, the more relevant the node is considered to be. The number of times a node is selected is a measure of the independent paths from the seeds of activation in the query to this node. NSbSA will closely correspond to DualRDF if in the latter no threshold and decay are used [2].

For NSbSA, the role of a threshold and decay can be played by an additional selection rule, which reflects the relevance of each node, apart from its specific connection in the nodes' connectivity matrix. For instance, such a rule can be specified to select the nodes with a PageRank higher than a threshold. Another rule can be based on a node's degree of connectivity, chosen within some range, which discards isolated or trivial non-informative connections. More sophisticated selection rules could involve the specific context and query at hand but will be probably much more computationally and memory expensive. When such additional selection rules are applied, some of the nodes which otherwise would have been selected will be skipped. One option for further exploration will be to return to these skipped nodes and effectively only delay their processing while retrieval of the previously selected nodes is already taking place.

2.5. Spreading Activation as Non-Zero Elements Search

The method, outlined briefly in subsection 2.4, can be presented in a structured algorithmic form as follows. SA is typically an iterative process, starting with nodes which are active. In the present case, it starts with the nodes from the query which are initially the sources of activation (seeds). The query can be formalized as a vector with dimension equal to the number of nodes and 1's for the nodes in the query and zeros otherwise. The nodes' connectivity matrix contains only 0's and 1's. The process starts with the nodes of the query. For each of them the corresponding rows in the connectivity matrix is searched for non-zero elements. The non-zero elements point to other nodes. These nodes are taken as seeds and the process is repeated until the preset number of iterations is reached. At each iteration step, the new nodes found are compared to the previously selected. Only the new nodes are used to proceed further. If a node has been already selected, a counter for this node is incremented so that the node is not used in subsequent iterations, as its connections have been already followed. This process can be expressed by an iterative process similar to the one expressed in Eq. (1):

$$
\begin{aligned}
&\mathbf{t}(n_n) = \mathbf{I}_{n_n}[k_{t_1}, k_{t_2}, \ldots, k_{t_n}] \\
&\mathbf{f}^{(0)}(n_n) = \mathbf{t}(n_n) \\
&\mathbf{k}^{(0)}(n_n) = \mathbf{t}(n_n) \\
&\mathbf{f}^{(1)}(n_n) = \mathbf{f}^{(0)}(n_n) + \sum_{c=1}^{n_n} \mathrm{W}^{(0)T}(\mathbf{k}^{(0)}(n_n), c) \\
&\mathbf{k}^{(1)}(n_n) = \Theta\left(\sum_{j=1}^{n_n} \mathrm{W}^{(0)T}(\mathbf{k}^{(0)}(n_n), j) \right) - \mathbf{k}^{(0)}(n_n) \\
&\qquad \ldots \\
&\mathbf{f}^{(it)}(n_n) = \mathbf{f}^{(it-1)}(n_n) + \sum_{j=1}^{n_n} \mathrm{W}^{(0)T}(\mathbf{k}^{(it-1)}(n_n), j) \\
&\mathbf{k}^{(it)}(n_n) = \Theta\left(\sum_{j=1}^{n_n} \mathbf{W}^{(0)T}(\mathbf{k}^{(it-1)}(n_n), j) \right) - \sum_{i=0}^{it-1} \mathbf{k}^{(i)}(n_n)
\end{aligned}
\tag{5}
$$

or

$$
\mathbf{k}^{(it)}(n_n) = \Theta\left(\sum_{j=1}^{n_n} \mathbf{W}^{(0)T}(\mathbf{k}^{(it-1)}(n_n), j) \right) - \Theta\left(\mathbf{f}^{(it-1)}(n_n)\right)
$$

where $\mathbf{t}(n_n)$ is the initial query (considered to be the target for the retrieval); $\mathbf{W}^{(0)}$ is the original nodes' connectivity matrix; the notation $\mathbf{W}^{(0)T}(\mathbf{k}^{(it-1)}(n_n), j)$ means taking the row indexes corresponding to the non-zero elements of $\mathbf{k}^{(it-1)}(n_n)$ and the summation over *j* is summation over the columns of the nodes' connectivity matrix $\mathbf{W}^{(0)}$; Θ is a Heavyside function producing a vector (matrix) with the same dimension as the argument vector (matrix) but with elements equal to 1 for any element of the argument larger than 0, and 0 otherwise; $\mathbf{f}^{(it)}(n_n)$ is accumulating the nodes that have been activated and the number of times it happened via independent connections (each connection is used only once).

3. Implementation and Benchmarking

The tests, presented in this section are based on the approach defined in Section 0. The dataset used for the experiments is extracted from the LDSR. All explicit statements, together with the statements inferred using light-weight reasoning, are extracted as a list of connections. For each statement, a connection is created between the subject and the object node. The size of the extracted dataset is 101 005 678 nodes and 854 235 512 statements.

3.1. Running a Benchmark with Various Technologies

The benchmark algorithm consists of finding the average value of pairs of integers. The benchmark data file contains approximately 470 million integer pairs. This test application is really simple, but allows for the algorithm to be easily and efficiently implemented in various programming languages. The purpose of the test is to give some idea about the speed of different configurations and does not aim to be very precise.
Table 1 below summarizes the results obtained:

- the column '*Loading*' describes the module that is used to load the data for the benchmark and trigger the calculation process;
- the column '*Calculation*' describes the module used to perform the calculation;
- the column '*Loading time*' shows the time needed to load the benchmark data file into memory;
- the column '*Transfer time*' shows the time consumed in transferring data from the loading to the calculation module ;
- the column '*Calculation time*' shows the time for calculation;
- the column '*Total time*' shows the sum for the transfer of the data and the calculation (without the loading time).

Case	Loading	Calculation	Loading time [ms]	Transfer time [ms]	Calculation time [ms]	Total time (Tran.+Calc. times) [ms]
1	Java	Java	743	0	2 780	2 780
2	Java	C	743	2 940	2 710	5 650
3	Java	C -> CUDA	743	3 730	1 140	4 870
4	Java	C -> CUDA (Critical)	743	0	1 150	1 150
5	C	C	700	0	2 556	2 556
6	C	CUDA	700	0	1 530	1 530
7	Matlab	Matlab	18 265	0	5 725	5 725

Table 1. Benchmark comparison based on different technologies

We experimented with three different loading modules written in Java, C and Matlab. The following configurations for the tests were used:

- Java–Java: simple implementation of the benchmark algorithm in Java;
- C–C: simple implementation of the benchmark algorithm in C;
- Matlab–Matlab: simple implementation of the benchmark algorithm in Matlab;
- C–CUDA: a parallel implementation on the video device. The data is loaded in memory and then uploaded on the device memory by the C module;
- Java–C: the loading is done in the Java module, but the computation is transferred to the C module. The *Get<PrimitiveType>ArrayElements* functions are used in the C module to retrieve data from the Java module memory. This is a safer way, because the C module does not access the memory of the Java virtual machine directly, but involves copying of the data into the memory of the C module and hence the larger transfer time;
- Java–C–CUDA: the same setting as in the previous case (Java-C), with the difference that the C module uploads the data to the CUDA device, where the actual calculation is done;
- Java–C–CUDA (Critical): the same setup as above, but this time the *GetPrimitiveArrayCritical* function is used for the transfer from Java to C memory. Thus, the C module has a direct access to the Java memory and no transfer overhead occurs. This can be used only with virtual machines that support pinning and should be used carefully, because garbage collection in the JVM is stopped, while the native code executes [11].

The results presented in Table 1, show that the SA module can be implemented in Java without a time penalty for the subsequent transfer of data to the CUDA card, thus allowing easy incorporation into the existing Java implementation of DualRDF.
The evaluations demonstrate that the implementations using the CUDA device clearly outperform the C and Java implementations, despite the fact that the algorithm used is not the best candidate for parallel computations.
The difference in the computation time (see Table 1), in cases 3 and 6 (both on the CUDA device, but invoked from Java and C respectively), is most likely due to the different ways that the C module is compiled and used. In case 6, the application is standalone, while in case 3, it is compiled as a shared library, subsequently loaded and invoked by the Java Virtual Machine (JVM).
Next, Table 2 shows results from experimenting with various graph formats for storing data. Since this is a complex graph with no special structure, the graph matrix is very sparse and the degree of connectivity for each node varies a lot (there are many nodes with one connection only, but also such with more than 15 000 connections). That is why there is no special, optimal matrix format for the problem at hand.

Therefore, we have experimented with various matrix formats in order to find the one that best fits the connectivity in the LDSR dataset (see e.g. [6], [7], and [11] for more information on sparse matrix formats and their effect on computational and storage efficiency):

- List of connections – this is the simplest format in which every connection is stored as a pair of its nodes indexes in the connectivity matrix and the data is stored in a 2-dimensional array with size nx2;

- CRS (Compressed Row Storage) – this format uses two arrays - `row_ptr` and the data array - to store the graph. The ith position in the `row_ptr` array is the index of the element in the data array starting from which the neighbors of the ith node are stored. In that way, the connections going out of node i can be found in the data array - starting from the position `row_ptr[i]` to `row_ptr[i+1]`. This format has also the advantage that the count of the connections with a starting node i can be easily found (`row_ptr[i+1] - row_ptr[i]`);

- CRSB 8/16/32/64 – this format is basically the same as the CRS, but the data is not saved in single values but in blocks with size 8/16/32/64 respectively. This requires a third array with an index for the blocks;

- CRS Sparse Nodes – uses the same idea as CRS, but in the `row_ptr` array only the nodes which have outgoing connections are stored. Another array is added in as an index of the nodes in the `row_ptr` array.

No.	**Format**	**Size [MB]**	**Size reduction [%]**
1	Connections' list	6 516 MB	100 %
2	CRS	4 029 MB	62 %
3	CRSB 8	3 920 MB	60 %
4	CRSB 16	4 235 MB	65 %
5	CRSB 32	4 978 MB	76 %
6	CRSB 64	6 313 MB	97 %
7	CRS Sparse Nodes	3 639 MB	56 %

Table 2. Influence of the graph format on storage

It is seen from the results presented in Table 2, that the most compact format is the CRS Sparse Nodes format. Slightly behind are CRSB 8 and CRS. We chose the CRS format, because it requires just a little bit more space than CRSB 8 and CRS Sparse Nodes formats, but appeared to be much more computationally efficient.

Table 3 shows the results of the spreading activation algorithm implemented on a conventional CPU and on a CUDA device. The spreading activation started from node number 1 (http://dbpedia.org/resource/Berlin) and ran for 15 iterations. Column '*Links to process*' shows the number of the nodes processed in each iteration.

Iter.	Calculation time in C module [ms]	Calcualtion time in CUDA module [ms]	Connections processed
1	10	10	378
2	10	10	9 965
3	20	20	238 059
4	70	80	3 627 938
5	530	300	33 862 769
6	1 760	810	121 368 870
7	2 000	900	122 513 804
8	1 690	590	54 222 097
9	1 600	320	22 801 887
10	1 590	190	12 987 007
11	1 600	130	9 669 715
12	1 600	100	8 077 657
13	1 640	90	7 140 140
14	1 620	70	6 368 000
15	1 650	90	5 819 648
Total	**17 390**	**3 710**	**408 707 934**

Table 3. Comparison of the time per iteration in the SA algorithm implemented on a conventional CPU and on a CUDA device.

The results show (see Figure 3) that the implementation on the CUDA device outperforms considerably the implementation on the CPU (about 4 times). This effect is especially strong, when the number of nodes is larger. It is worth mentioning, that the largest amount of time is spent for fetching the data from the memory and the least is spent for the actual computation. Thus, it seems that adding additional computation on the nodes will not have a great impact on the overall speed of the CUDA device implementation.

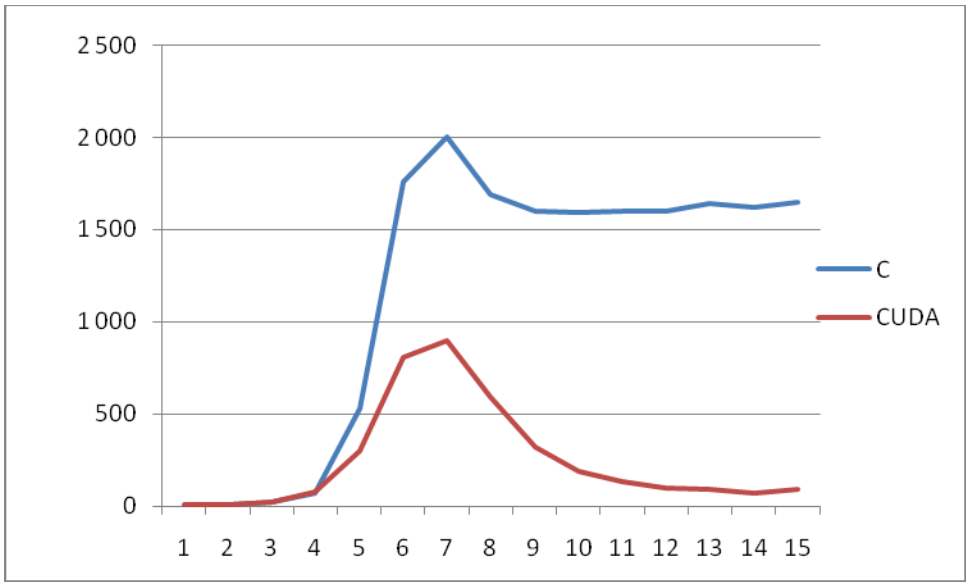

Figure 3. Comparison between the C and the CUDA device computations.

3.2. *Efficiency Tests*

Further, we have carried out an efficiency test in order to evaluate the processing time. This test is performed using the best method selected in the previous subsection – Java-C-CUDA (Critical). Two connectivity matrices were extracted. The first one took care of the direction of the connections following *<subject-predicate-object>* structure of the triples in the dataset. The second one considered symmetrical bi-directional connections. In both cases, as explained in Section 0, the weights were taken to be equal to 1. The independent activation of nodes has been taken into account by counting the number of times a node is activated from another node which has not activated it before.

The query consisted of activating the node http://dbpedia.org/resource/Kamchiya in the LDSR dataset, which contains exactly 101 005 678 nodes and 854 235 512 statements.

Iter.	Calculation time [ms]	Connections (statements)	Nodes
1	15	277	246
2	14	35 420	27 859
3	75	4 078 104	1 568 690
4	611	70 180 576	16 805 281
5	1 811	164 439 569	28 733 891
6	1 699	100 496 908	13 484 943
7	861	27 817 925	2 680 912
8	228	4 574 553	405 929
9	59	991 157	94 455

10	27	391 634	34 188
11	18	208 693	20 363
12	16	140 655	31 448
13	14	90 470	7 816
14	13	60 156	6 028
15	13	42 223	4 492
Total:	**1249**	**373 548 320**	**63 906 541**

Table 4. Time, number of connections and nodes for the NSbSA method for directed connections (101 005 678 nodes and 854 235 512 statements).

In the directed connections case, the connections were equal to the number of statements. The results for seed *"http://dbpedia.org/resource/Kamchiya"* are summarized in Table 4 and are graphically presented in Figures 4a to 4c.

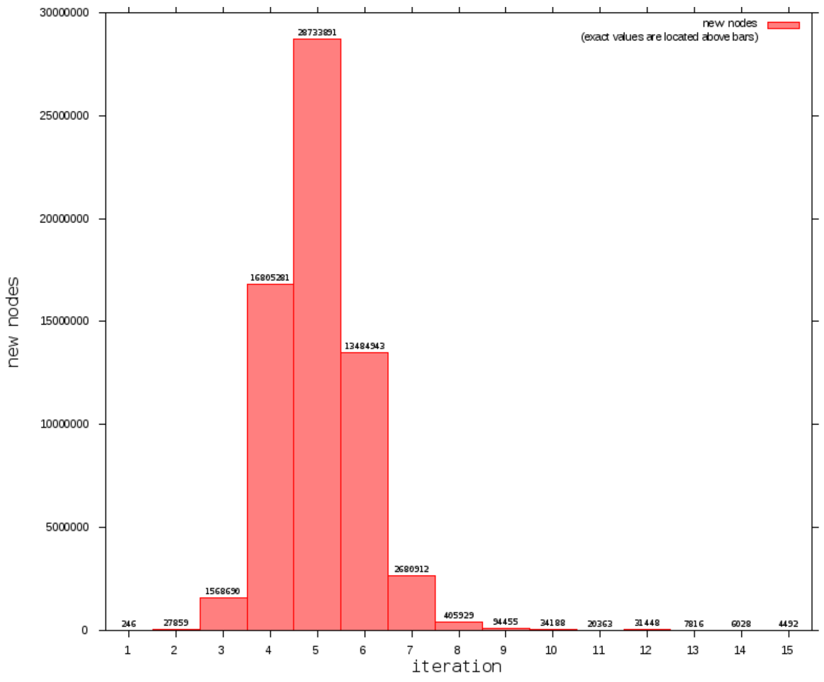

Figure 4a. NSbSA along directed connections - number of new nodes per iteration

The number of activated nodes per iteration is given in Figure 4a. As shown in Figure 4b, the number of nodes activated reaches saturation after the 6th iteration, at 63 millions.

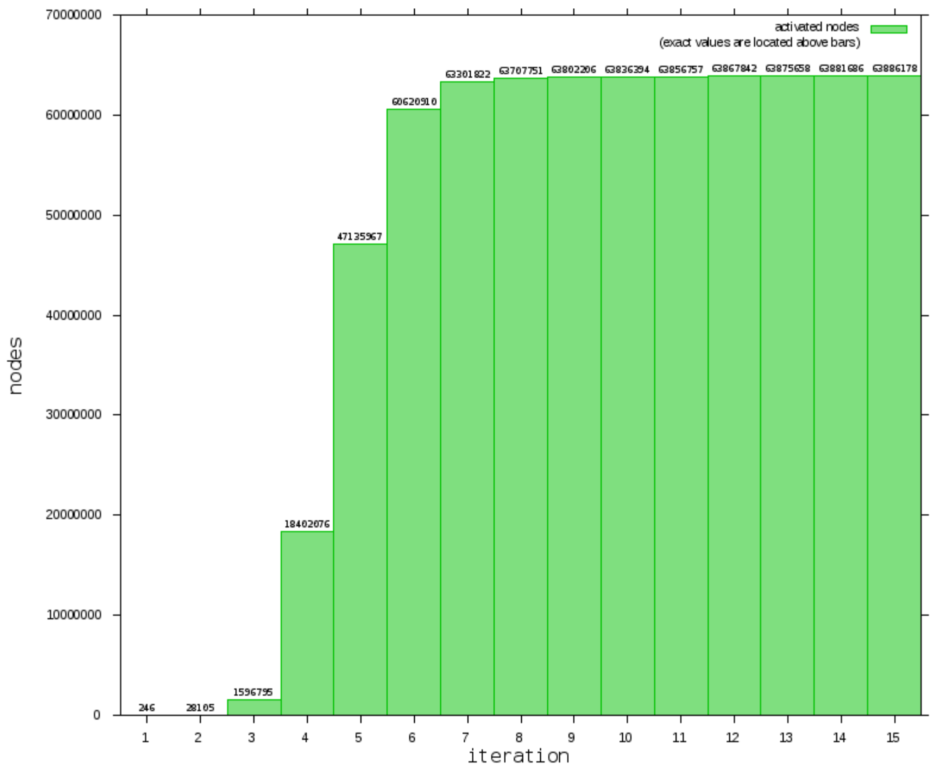

Figure 5b. NSbSA along directed connections - cumulative number of new nodes per iteration

The speed of the calculation is given in Figure 4c – saturation, in terms of the number of activated nodes, is achieved after about 5.5 seconds.

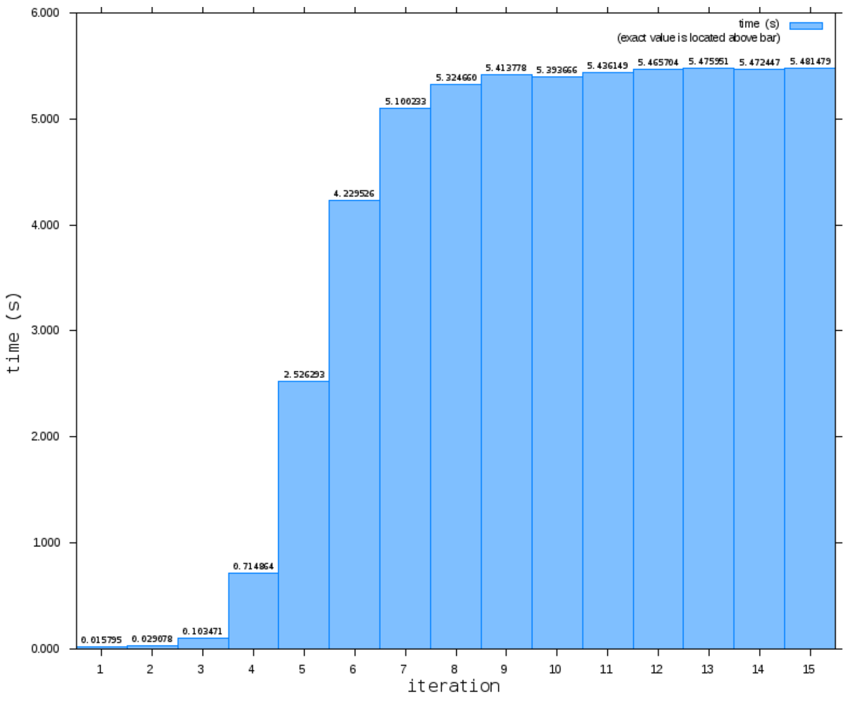

Figure 6c. NSbSA along directed connections - cumulative time per iteration

In the symmetric bi-directional connections case, the number of connections is almost twice as big as the number of connections in the directed case – 1 703 901 340. This case was considered in order to reach nodes, which cannot be reached by following only directed connections and which can be ‘close’ to the seed node in the query. The results are summarized in Table 5 and in Figures 5a to 5c.

Iter.	Calculation time [ms]	Connections (statements)	Nodes
1	16	298	263
2	526	116 486 940	23 715 191
3	17 705	1 249 286 467	58 688 259
4	2 884	309 108 884	17 736 944
5	416	28 038 118	776 593
6	53	853 096	9 720
7	13	11 368	1 106
8	12	1 198	26
9	11	30	2
10	11	2	0
11	11	0	0
12	11	0	0
13	11	0	0
14	11	0	0
15	11	0	0
Total	**21 702**	**1 703 786 401**	**100 928 104**

Table 5. Time, number of connections and nodes for the NSbSA method for symmetric connections (101 005 678 nodes and 1 703 901 340 connections).

The number of activated nodes per iteration is given in Figure 5a. Similarly to the previous case (directed connections), the number of nodes increases drastically and reaches saturation after the 6th iteration (see Figure 5b). In this case, however, almost the whole dataset gets activated. The speed of the calculation is given in Figure 5c. It can be seen that saturation is achieved after about 20 seconds (no optimization related to the symmetry of the connectivity matrix has been done).

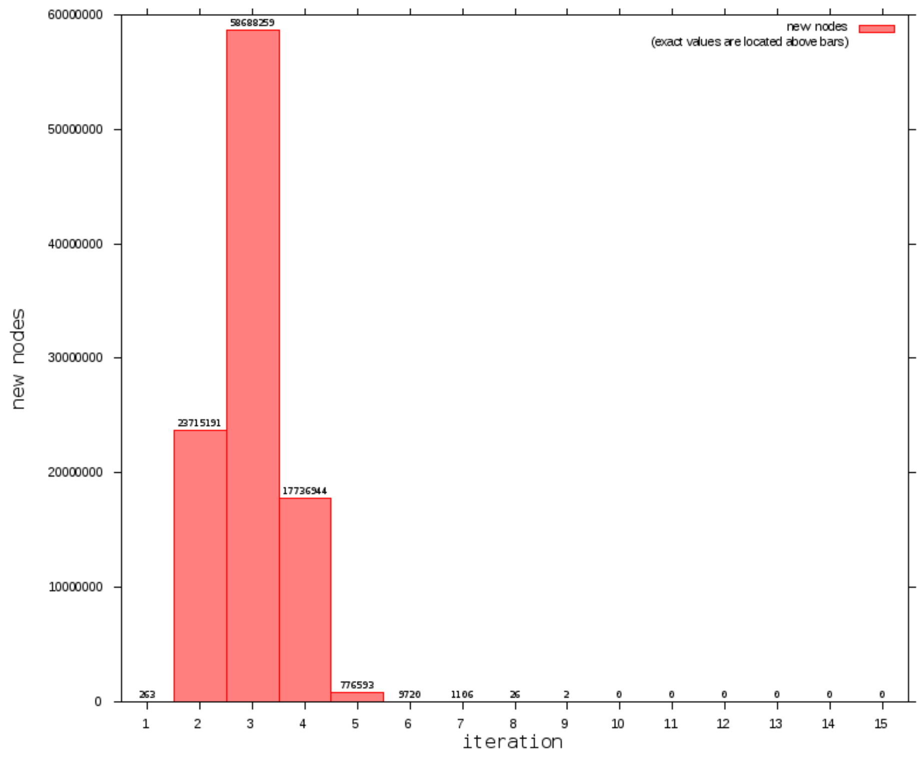

Figure 7a. NSbSA with bi-directional connections - number of new nodes per iteration

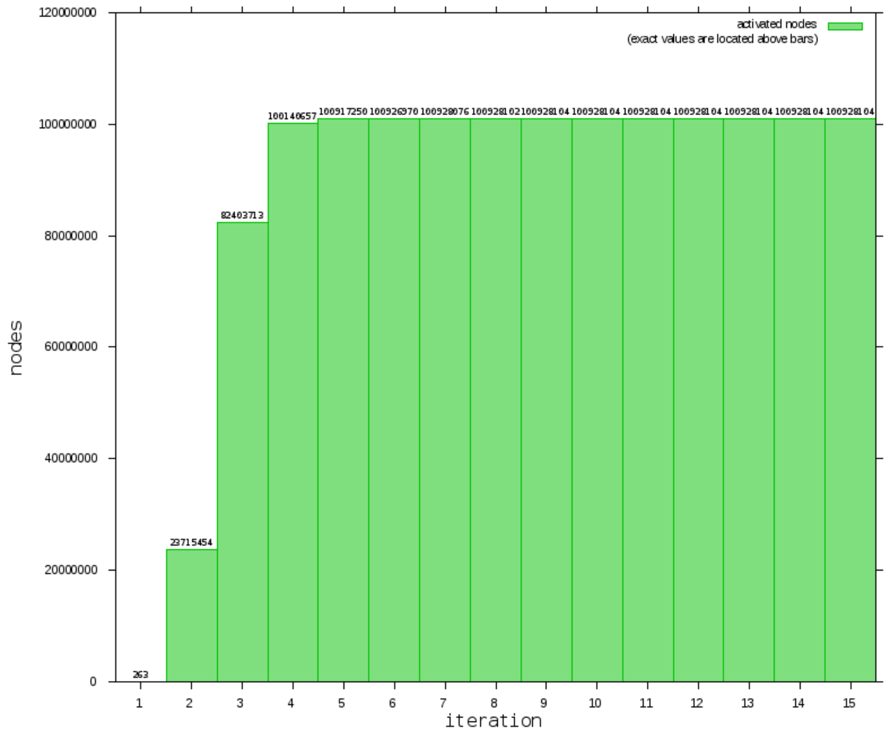

Figure 8b. NSbSA with bi-directional connections - cumulative number of new nodes per iteration

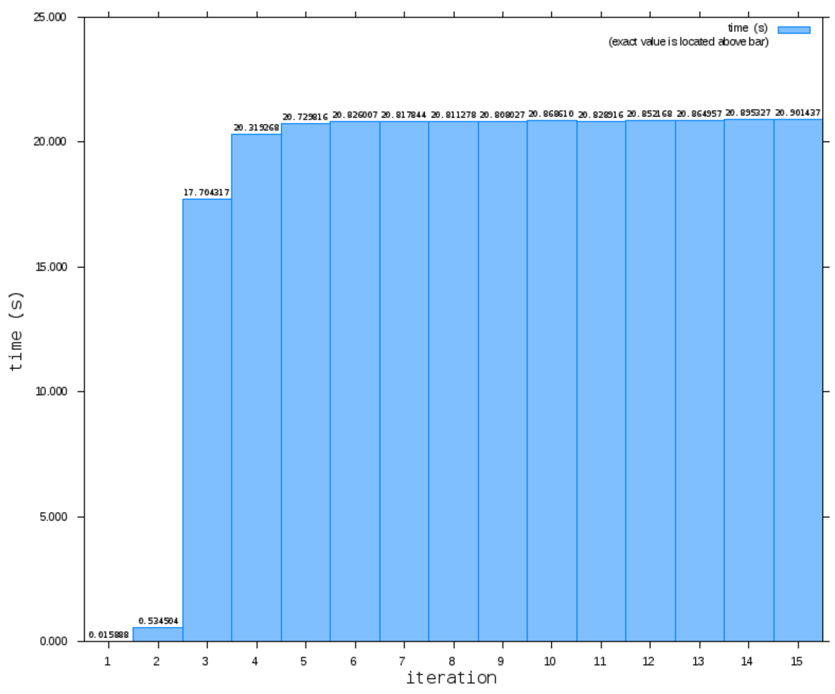

Figure 9c. NSbSA with bi-directional connections - cumulative time per iteration.

The efficiency tests presented in this section show the potential of the NSbSA approach but also highlight the problems that have to be overcome. On one hand, the speed of the processing is reasonable, and could be improved further. On the other hand, the selectivity of the method, especially in the case of bi-directional connections, is problematic, as the whole dataset gets activated. As discussed in Section 0, additional selection rules (degree of connectivity, PageRank, and similar) will be crucial for the applicability and usefulness of this method. However, they will increase the processing time and have to be evaluated carefully.

NSbSA outperforms substantially DualRDF at the level of speed of processing, as evidenced by the results reported in [2], for a similar task. The numerical evaluation seems to indicate that NSbSA is quite promising for real time applications when used with a CUDA device and optimized data storage. When applied to the LDSR dataset, NSbAS activated $65*10^6$ nodes in 15 iterations for about 5 seconds (in comparison, in similar settings DualRDF could select about 75 thousands entities in 34 seconds). However, further explorations are needed to assess the quality of retrieval with respect to the query. It is possible that the best results will be obtained by combining DualRDF, which could apply exact SA on nodes selected by the approximate but more efficient CbAS and NSbAS methods.

4. Conclusions and Future Work

In this deliverable, two approximate approaches to SA – CbAS and NSbAS – have been presented and discussed as possible alternatives and extensions to DualRDF.

The first of them (CbSA) explores the possibility to group the dataset nodes in clusters, based on relative ‘distances’ among them. Once such a clustering is found and a connectivity matrix is built using the distances among the nodes, spreading activation can be implemented much more efficiently by reducing the dimension of the weight matrix and by parallelizing the computations. More importantly, appropriate clustering of the nodes could encode important correlations which could lead to higher quality of the retrieval [2]. However, to assess the actual potential of this approach, benchmarking of DualRDF and NSbSA implementations is needed. This is work in progress and the results will be reported in the future.

The second approach (NSbSA) attempts to replace spreading activation, as a matrix-vector multiplication task, by non-zero elements search. It is assumed that the connectivity matrix (which is a sparse matrix) contains only 1’s. In this approach, starting from the query nodes as ‘seeds,’ all connected nodes are explored without repetitions. In other words, each node is used to spread activation just once. Several implementations (Matlab, Java, and C) of this approach have been benchmarked, including one based on the NVIDIA CUDA card architecture. The evaluations reveal considerably higher efficiency when the CUDA card is used (about 4 times faster computations) and reasonably low spreading activation time with respect to on-line usage (about 4-5 seconds). As discussed in more details previously in this deliverable, the performance of NSbSA can be further improved by introducing additional node selection rules which could reduce the number of nodes activated and connections followed. Currently, we are considering the nodes’ degree of connectivity and PageRank as a basis for such selection rules.

The DualRDF, CbSA, and NSbSA methods share a common goal of implementing SA. DualRDF is an exact implementation, which is the reason for its relatively lower performance. CbSA provides a method for approximate derivation of nodes’ connectivity matrix [2], which encodes some relations not explicitly present in the original connectivity matrix used in DualRDF. The two methods are similar in the computations they require, although the structure of the connectivity matrix in CbSA can be quite appropriate for considerable optimization (being a block-diagonal matrix). NSbSA proposes an approximate way of SA which turns out to be very efficient computationally. It can be used in approximate implementations of both DualRDF and CbSA.

Two flavors of NSbSA were evaluated – with directed and with symmetric connections employed to represent the triples of the dataset. For the whole LDSR dataset (101 005 678 nodes and 854 235 512 statements), the results show processing times of 5.5 seconds for the former and 20 seconds for the latter case, which is good evidence for the performance capabilities of NSbSA. The quality of the retrieval has not been evaluated in this deliverable. However, it seems clear that additional selection rules have to be applied in order to reduce the number of active nodes and choose the most meaningful subset, especially in the case of symmetric weights when practically the whole dataset gets activated. Another direction for further research is the combination of NSbSA with DualRDF (e.g. applying DualRDF on nodes selected by NSbSA).

Detailed comparison with DualRDF has not been included but, judging from the experiments presented in this deliverable, it is clear that NSbSA is much faster (NSbSA works in real time for the whole LDSR dataset). A useful comparison should consider the quality of the retrieval as well.

References

[1] Kiryakov, A., Ognyanoff, D., Peikov I., Velkov R., Tashev, Z.: 1 Spreading Activation Components. LarKC project deliverable D2.4.1 (2009)

[2] Grinberg, M. Haltakov, V.: The TRIPLE Hybrid Cognitive Architecture: Connectionist Aspects. In: Apolloni, B., Bassis, S., Morabito, C. F. (eds.), WIRN09 - Proceedings of the 19th Italian Workshop on Neural Nets, Vietri sul Mare, Salerno, Italy, May 28–30 2009, IOS Press, 2009, 204, 293-305

[3] http://en.wikipedia.org/wiki/CUDA, and references there in.

[4] Tashev, Z., Ognyanoff, D., Velkov, R., Momtchev, M., Balev, B., Peikov, I.: Validation goals and metrics for the LarKC platform. LarKC project deliverable D5.5.2 (2009)

[5] Muchnik, L., Itzhack, R., Solomon, S., Louzoun, Y.: Self-emergence of knowledge trees: Extraction of the Wikipedia hierarchies. Physical Review E (Statistical, Nonlinear, and Soft Matter Physics), APS, 76 (2007)

[6] Bell, N. & Garland, M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors. SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ACM, (2009) 1-11

[7] Aykanat C., Pinar A., Catalyurek, Ü. V.: Permuting Sparse Rectangular Matrices into Block-Diagonal Form. *SIAM Journal on Scientific Computing* 25 (2002), 1860-1879.

[8] Aldenderfer, M.S., Blashfield, R.K.: Cluster Analysis, (1984), Newbury Park (CA): Sage.

[9] Ding, Li., Finin, T., Peng, Y., da Silva, P. , McGuinness, D.: Tracking RDF Graph Provenanceusing RDF Molecules. http://ebiquity.umbc.edu/_file_directory_/papers/178.pdf, (2005)

[10] Yzelman A. N., Bisseling R. H.: Cache-Oblivious Sparse Matrix-Vector Multiplication by Using Sparse Matrix Partitioning Methods. SIAM J. Sci. Comput. 31(4) (2009), 3128-3154.

[11] Sheng Liang: The Java Native Interface Programmer's Guide and Specification, http://java.sun.com/docs/books/jni/html/objtypes.html#4099, Sun Mictosystems Ink. (2002), ch.3.