Graph Stores based on Spatial Computers

Dominic Pacher, Robert Binna, Günther Specht

Databases and Information Systems
Institute of Computer Science
University of Innsbruck, Austria
{*firstname.lastname*}*@uibk.ac.at*

***Abstract*—This paper presents a novel concept of a Spatially Aware Graph Store, which combines a Graph Store and a Spatial Computer to manage graphs in one, two or three physical dimensions. In this environment, the physical distance between graph nodes strongly affects graph traversal performance. Consequently, a Spatially Aware Graph Store needs to minimize these distances to operate efficiently. We show that this minimization can be achieved by increasing the dimensionality of the Spatial Computer. Furthermore, a novel optimization method is introduced to further minimize distances by rearranging nodes in the data space. Our results show that the overall distances between nodes can be reduced by three orders of magnitude for 3-D in comparison to 1-D Spatially Aware Graph Stores. Moreover, the suggested optimization method achieves a reduction by another order of magnitude.**

I. Introduction

Over the last years the number of available linked data sets as well as the size of each single dataset has undergone an exponential growth[1]. In the beginnings, stores designed for linked data query and retrieval, could easily deal with the amount of data available by either using in-memory stores or by mapping the graph data onto existing relational database technologies. With the amount of linked open data surpassing the computing power of a single central processing unit, it became inevitable to distribute the workload onto several processing units. Furthermore, dedicated main memory systems were developed to ensure low access times. However, future graph stores will have to cope with significantly larger data sets due to exponential growth. Eventually, this development will surpass the performance of todays main memory graph stores. The main reasons for this lie in the access delay of main memory, which did not evolve at the same rate as computation power. Consequently, the so called memory wall issue [19] emerged. On contemporary systems this means that a single cpu executes about 100 instructions until it receives previously requested data from the main memory. For computationally intensive problems, which need many instructions per single data element, this performance penalty can be neglected. However, in graph stores computationally cheap filter operations are dominant. This makes such systems much more vulnerable to the memory wall issue. Furthermore, the problem gets even worse if distribution is introduced, as in multi core environments all cpus need to share the main memory hardware. Hence, we identify the memory wall effect as the main problem of future graph stores. This issue needs to be addressed to enable both, higher performance and scalability.

Spatial Computers present a promising alternative to address this issue as it introduces a new way of how data is stored and processed. In more detail, in Spatial Computers physical space is not an issue that is neglected, but is exploited to enable more efficient storage and distributed processing [3]. A graph store operating on this architecture would store its graph in physical space. In the course of this endeavor, an important property needs to be considered. In Spatial Computers access delay depends on the actual physical distance between a processing unit and the requested data element. This represents a major difference to todays non spatial computers, which provide constant random access delay. Consequently, linked graph nodes should be located closely to each other to enable efficient node-to-node traversing.

This leads to the following two challenges. First, how much can the distances between linked nodes be decreased by operating in a 2-D or 3-D rather than on a 1-D physical space? Second, is it possible to rearrange nodes in the store aiming at further decreasing distances? The latter question is particularly important as graph data is usually high dimensional. Consequently, even if all three physical dimensions are used, the distances between linked nodes will not be sufficiently minimized. As there are no more than three physical dimensions, an alternative way to further decrease the distances between linked nodes is required.

To address these challenges the main contributions of this work are: *1*) A hybrid concept of a Spatially Aware Graph Store, which combines a Graph Store with a Spatial Computer *2*) The Mid Point Optimization method to decrease distances between linked nodes and *3*) An agent based approach to query data.

The remainder of this paper is structured as follows. Section II introduces the concept in detail along with its basic operations and explains the origin of the distance dependent access delay. Section III discusses the effect of graphs embedded in 1-D, 2-D and 3-D Spatial Computers on the distances between linked nodes and presents the Mid Point Optimization to improve embeddings. In Section IV we evaluate how the distance between linked nodes decreases through higher dimensional Spatial Computers and how the suggested optimization method can enhance results even more. In addition, the effect on graph queries is evaluated. In Section V we present related work in the area of spatial computing, graph stores and optimization of

[1]Freie Universität Berlin, The Linking Open Data cloud diagram, http://lod-cloud.net/, last visited February 2014

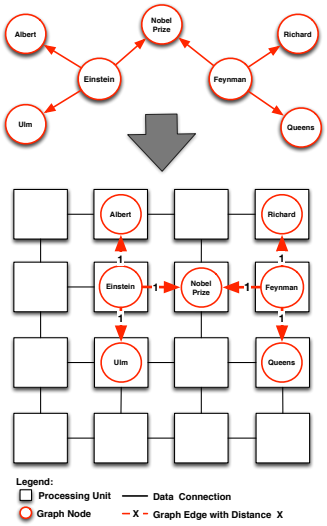
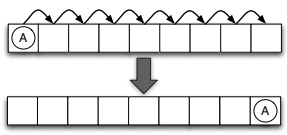
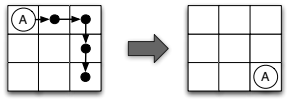


Fig. 1: Design of a Spatially Aware Graph Store



(a) 1-D Grid: Node A requires 8 Node Swap Operations



(b) 2-D Grid: Node A requires 4 Node Swaps Operations

Fig. 2: Worst Case Comparison of Node Rearrangement Operation

graph data. Finally, section VI gives a summary and outlines potential future work based on the results presented in this paper.

II. Spatially Aware Graph Stores

The concept of a Spatially Aware Graph Store is shown in Figure 1. In the interest of simplification, a 2-D representation is chosen. However, the presented concept can easily be applied to 1-D or 3-D physical space. Each square in Figure 1 represents a simple processing unit that is capable of storing a single node of the entire graph. All processing units are connected by a regular grid structure. In this way, a processing unit can communicate only directly with its adjacent neighbours. To access nodes stored to non-adjacent neighbours, the request needs to be routed over intermediate processing units to its destination. Consequently, the delay of a node to access a linked node strongly depends on the distance between these two units. In contrast to this delay, the time to execute an operation on the node stored in a processing unit is considered to be negligible. In fact, this assumption can be made as query operations are usually composed of computationally cheap filter operations. Due to the strong dependency of access time on access distance and the ability to process data in one, two or three physical dimensions, the concept resembles a Spatial Computer.

A. Foundations

The architecture underlying our concept represents an n-dimensional regular grid (cf. Figure 1) of one, two or three dimensions. More formally, the grid is represented by a Graph $G = (V, E)$ with edge set $E = \{\{(x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n)\} : \sum_{i=0}^{n} |x_i - y_i| = 1\}$ and vertex set $V = \{1, 2, ..., k\}^n$.
Distances between two nodes (processing units) in the grid with coordinates $(x_1, x_2, \ldots, x_n)$ and $(y_1, y_2 \ldots, y_n)$ are defined by the Manhattan Distance $d(x, y)$:

$$d(x, y) = \sum_{i=0}^{n} (|x_i - y_i|)$$

Furthermore, the number of adjacent neighbours with distance $d(x, y) = 1$ can be controlled by the number of dimensions used. In particular, in a 2-D grid, each inner processing unit can directly communicate with four adjacent neighbours, whereas in a 3-D grid, this number increases to six units.

B. Node Rearragement Operation

The ability to rearrange nodes presents a basic operation in a Spatially Aware Graph Store. In particular, during the optimization process described in the following Section III, nodes frequently need to pass through the entire data space of the spatial computer. As there is no central instance to coordinate this movement, a decentralized method is required, which is executed independently on each processing unit. Therefore, each processing unit can request a swap of its stored graph node with the one of its adjacent neighbour. This swap operation is executed iteratively, until each graph node is moved to its desired position. The complexity of this operation depends on the dimensionality of the spatial computer, as the maximum number of swap operations is determined by the diameter of the grid. Figure 2 illustrates this case with an example. In a store with nine nodes, node A is linked to other nodes (for the sake of simplicity these links are omitted in Figure 2) in a way that it needs to be moved to the other side of the data space. As the store can only move nodes by local swap operations within its neighbourhood, the number of needed steps is bound by the dimensionality of the spatial computer. In the example, eight flip operations are needed in a 1-D architecture to reach the other side of the data space (Figure 2a). However, in a 2-D architecture, the number of operations is reduced to four steps (Figure 2b). More formally, the complexity of this move operation can be expressed as $\mathcal{O}(n * k^{1/n})$ with n being the dimensionality of the spatial computer and k being the number of nodes. For example, in a store with 1,000,000 nodes, the worst case number of swap operations is 1,000,000 for a 1-D, 2,000 for a 2-D and 300 for a 3-D Spatial Computer.

C. Query Operation

To achieve efficient query execution on graph data, both primitive operations such as edge traversal as well as filtering need to be efficiently executed. Applying a traditional approach where a single coordinator distributes and coordinates all work units is not feasible. This is due to the following two facts. First, the coordination of all participating processing units is computationally too expensive for a single processing unit. Second, in a Spatial Computer the access delay depends on the actual physical distance. Consequently, routing all communication to a single processing unit results

in an unacceptable overhead. Therefore, an execution model is required, which on the one hand reduces the amount of data transferred and on the other hand is independent of a central coordinator. To fulfill these requirements, we suggest an agent based model. This model allows the execution of a query to flow along the graph's edges. Whenever an edge needs to be traversed, the agent is transferred to the destination node's processing unit. Therefore, every filter operation of a node occurs at its processing unit, which is a computationally cheap operation. Thus, this execution model fosters the distribution of work among the participating processing units and reduces the amount of data to be transferred. More specifically, it prevents long access paths to distant nodes, as all computation is done on the respective processing units. However, in spatial computers the time needed to transfer the agent itself cannot be neglected. It is therefore important to reduce the distance between the participating nodes.

III. Linked Node Distance Optimization

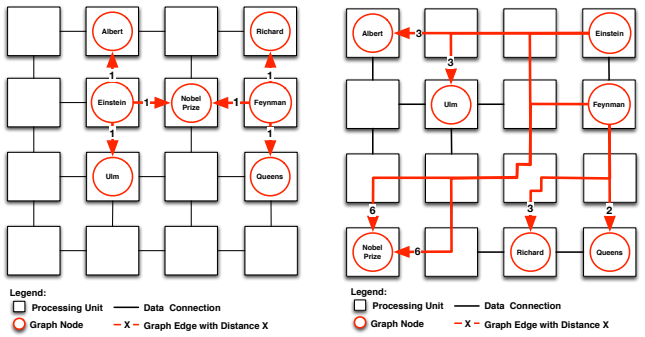
In a Spatially Aware Graph Store efficient graph traversal can only be achieved, if linked nodes are closely located to each other. To ensure this, a twofold approach is applied. On the one side, the maximum distance between two linked nodes is reduced by increasing the dimensionality of the spatial computer. On the other side, locality is improved by rearranging nodes to reduce the distance to non-adjacent processing units.

A. Spatial Computer Dimensionality

As previously mentioned in Section II-B, the diameter of the Spatial Computer and therefore the maximum distance between processing units is reduced with each additional dimension. In this way, the average distance between linked nodes decreases as well.
Consequently, it can be assumed that by increasing the dimensionality the linked node locality is improved. However, only the physical three dimensions can be used for this approach. This poses a problem as graphs are usually high dimensional. In this case, a one-to-one embedding is not possible. Nevertheless, most real world graphs are sparse graphs with heterogeneous dimensionality. More specifically, parts of the graph can be two dimensional while other parts are very high dimensional. This property can be used to calculate an optimized graph embedding to reduce the overall distance between linked nodes. How such an embedding can be obtained is described in the next section.

B. Graph Embedding

As shown in Figure 1, the embedding of the graph on the processing units represents the first step to build a Spatially Aware Graph Store. In the course of this process, many solutions are possible. However, for efficient query processing an embedding that minimizes the distances between linked nodes is required. Consequently, the quality of an embedding can be estimated by its overall sum of distances between linked nodes. Figure 3 shows this with an example graph,



(a) Optimal Solution. Overall Sum of Distances is 6.

(b) Worse Solution. Overall Sum of Distances is 22.

Fig. 3: Embeddings of the same Graph with different Overall Distances between Linked Nodes

which is embedded in two different ways. Although the same graph is embedded, it becomes apparent that solutions can differ significantly in relation to the achievable traversing performance. In more detail, Figure 3a shows the optimal solution. The length of each edge is one or, in other words, only one hop is needed to reach its linked node. This way, the overall sum of edge distances is 6 for the entire embedding. In contrast, Figure 3b shows a worse embedding with an overall distance of 22.

C. Optimization of Embeddings

In distributed graph databases *Graph Partitioning* [1] is commonly applied to minimize the number of cuts between partitions. However, partitioning does not provide physical locality between partitions when mapped onto the fixed dimensional space of a spatial computer. More specifically, according to our concept, the graph is maximally partitioned already. Each node represents one partition, stored on a single processing unit. In this case, a partitioning algorithm cannot improve the embedding any further, as it does not determine how the partitions need to be placed optimally in a Spatial Computer.
Consequently, this method is unable to increase locality in our environment. However, there is a solution to the problem. Optimal linear arrangement solvers map graphs onto n-dimensional space in a way that the overall sum of edge distances is minimal [7]. When applied on a Spatially Aware Graph Store, the distances between linked nodes are minimized and the number of intermediate processing units between two linked nodes is reduced as shown in Figure 4.

More formally, the overall sum of distances can be denoted as *costs*. The minimization problem [7] of these costs are defined as follows. Given an n-dimensional grid $G = (V, E)$ as defined in Section II-A the problem is to find a mapping f from V onto $\{1, 2, \ldots, n\}$ that minimizes:

$$cost(G, f) = \sum_{\{i,j\} \epsilon E} (|f(i) - f(j)|)$$

Furthermore, the term $(|f(i) - f(j)|)$ can be replaced by the

Manhattan distance of Equation II-A leading to

$$cost(G, d) = \sum_{\{i,j\} \epsilon E} d(i, j)$$

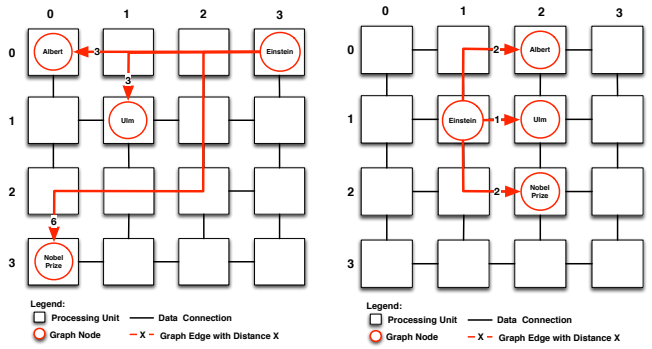
It can be shown that this problem is NP-hard for directed and undirected graphs [8], which causes the computational requirements to become prohibitive for an optimal solution. However, heuristics can be applied to significantly optimize the node ordering of larger data sets in reasonable processing time [16].

D. Mid Point Optimization

To approximate the Optimal Linear Arrangement Problem, various different approaches exist. Rodriguez and Tello used *Local Optimization* and *Simulated Annealing* [17] methods. Also multi grid approaches have been suggested in [13] and [18] to obtain results of similar quality, but with less processing time. However, for a Spatially Aware Graph Store, two additional issues have to be considered. First, data sets even for main memory graph stores, consist of up to one billion edges. Therefore, a faster optimization method than Local Optimization and Simulated Annealing is required to achieve better embeddings in feasible time. Second, the method needs to be simple enough to be applicable on each processing unit of a spatial computer. This poses a problem on complex algebraic multi grid approaches.

Therefore, we introduce our novel method called *Mid Point Optimization*. The basic principle behind this algorithm is to iteratively search for a new embedding with lower costs. This is done by moving a node to the geometrical mean position of all its linked nodes. This position can be calculated, as each node already knows the position of each linked node to process agent bases queries (cf. Section II-C). The algorithm is executed on each node of the graph, leading to a new overall node arrangement with reduced costs. In more detail, the method consists of the following three steps. First, the new mid point positions are calculated for each node. Second, all nodes are moved to the mid points using the node rearrangement operation as described in Section II-B. Finally, all linked nodes are notified of the new positions.

The algorithm is repeated iteratively until the improvement drops below a predefined threshold. An example of this method is shown in Figure 4. The position of each node is denoted by its two-dimensional coordinates. For example the node *Albert* has coordinates (0,0) and the node *Einstein* (3,0). The first Figure 4a shows the initial embedding with an overall distance of 12. The new mid point coordinates of the Einstein node are: $(0,0) + (1,1) + (0,3) + (3,0))/4 = (1,1)$. By applying this procedure on all nodes (results are rounded to the next bigger integer) the new embedding shown in Figure 4b can be obtained. This embedding features a reduced overall distance of 5.



(a) Non-Optimized Graph. The Overall Distance of the Embedding is 12

(b) Optimized Graph. The Overall Distance of the Embedding are reduced to 5

Fig. 4: Mid Point Optimizations Example

IV. Evaluation

To evaluate our concept of a Spatially Aware Graph Store we conducted three experiments. First, we surveyed the influence of two- and three- dimensional storage of graph data on the edge distances. Second, the decrease of edge distances by applying the Mid Point Optimization is evaluated. Finally, the effects of optimization on query execution is examined. The evaluation is carried out on the following three datasets, namely web-Google[2], p2p-Gnutella25[2] and DBpedia[3]. These datasets were selected to provide a balanced overview on optimization results in the best (web-Google), average (DBpedia) and worst case (p2p-Gnutella25). To the best of our knowledge, no hardware exists, on which a Spatially Aware Graph Store could be evaluated. Therefore, we implemented a simulation engine of a Spatially Aware Graph Store which is able to load, optimize and query data sets. Hence, all experiments were conducted by using this simulation engine on a server equipped with two Intel Xeon L5520 Quad Core CPUs, 2.27 GHz, 64-bit architecture and 96 GB main memory.

In the first experiment each data set was loaded into the Spatially Aware Graph Store in a 1D, 2D and a 3D embedding. To avoid any pre-existing locality, the nodes of each data set were inserted in random order. The column $dist_{org}$ in Table I contains the sum of all distances between linked nodes after this insertion process. It can be observed that in the 2D case the distances between linked nodes are reduced by two to three orders of magnitude in comparison to the 1D case (e.g. the *web-google* data set improves from 2.5×10^{12} to 5.3×10^{9}). Furthermore, the 3D case improves by another order compared to the 2D case (the *web-google* data set improves from 5.3×10^{9} to 8.3×10^{8}). It can be inferred that by embedding a graph in a higher dimensional Spatial Computer, distances between linked nodes are significantly reduced.

The second experiment executes the Mid Point Optimization on each data set for the one-, two- and three dimensional case. The resulting distances between linked nodes after this

[2]Stanford University, Stanford Large Network Dataset Collection, http://snap.stanford.edu/data/, last visited February 2014

[3]The DBpedia Data Set, http://wiki.dbpedia.org/Datasets, last visited February 2014

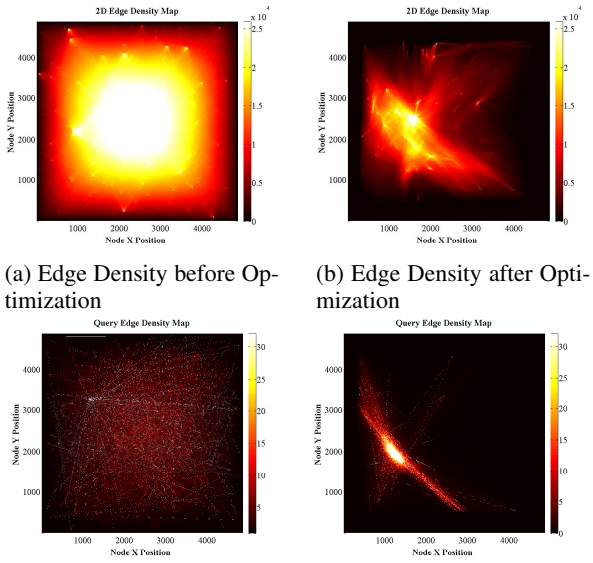
Dataset	$\|V\|$	$\|E\|$	Graph Type	Dim	$dist_{org}$	$dist_{opt}$	δ_{dist}	M_{opt}	MS_{opt}	t_{opt}
web-Google	875,713	8,644,102	web topology	1-D	2.5×10^{12}	6.5×10^{10}	0.02	2065	0.002	1348 min
				2-D	5.3×10^{9}	2.6×10^{8}	0.04	13	0.006	24 min
				3-D	8.3×10^{8}	8.8×10^{7}	0.1	6	0.02	16 min
DBpedia	23,617,067	126,756,890	knowledge	1-D	1.2×10^{13}	-[4]	-	-	-	-
				2-D	4.1×10^{11}	1.0×10^{11}	0.25	589	0.06	36 h
				3-D	3.6×10^{10}	9.6×10^{9}	0.26	56	0.06	13 h
gnutella25	62,586	295,784	peer to peer	1-D	8.2×10^{8}	3.2×10^{8}	0.39	2471	0.10	165 s
				2-D	1.0×10^{7}	5.0×10^{6}	0.46	40	0.13	15 s
				3-D	3.1×10^{6}	1.5×10^{6}	0.48	13	0.15	15 s

TABLE I: Reduction of Distances for a 1-D, 2-D and 3-D Spatially Aware Graph Store with and without Optimization

optimization can be found in Table I in colum $dist_{opt}$. In addition, the relative difference between $dist_{opt}$ and $dist_{org}$ is labeled as δ_{dist}. As a result, distances were reduced by a factor 0.02 to 0.48 in relation to the original costs (cf. *web-Google* with δ_J = 0.04 for the 2-D and 0.1 for the 3-D case). Furthermore, a significant reduction of the median distance between linked nodes can be observed in column M_{opt}. The value MS_{opt} denotes the median in relation to the maximum possible distance.

The results of the optimization can be visualized with Edge Density Plots. These plots are heat maps created by counting the number of edges crossing a location in the data space. In particular, dark colors denote few and brighter colors denote frequent edge crossings. Figure 5 shows such edge density plots for the DBpedia data set. In this Figure the white color denotes 25,000 edges crossing a single location. Moreover, Subfigure 5a visualizes the edge density of the original and 5b of the optimized embedding. It becomes apparent that in 5b the number of hot areas decreases and distinct clusters become visible. Beside the optimization quality also the time needed to perform the optimization has been observed. The execution times can be found in the last column t_{opt} of Table I. From this values the predicted speedup (cf. Section II-B), which can be achieved by a higher dimensional Spatial Computer, can be observed. This is most evident for the DBpedia dataset, which could not be optimized in feasible time using a 1-D Spatial Store. However, in 2-D and 3-D the runtime drop significantly from 36 to 13 hours. Due to the overhead of the simulation engine the runtime of the gnutella25 dataset could not be improved for the 3-D case. By comparing the times between the datasets it becomes clear that the computational effort does not only depend on the size of the dataset but on its dimensionality as well.

In the last experiment, the impact of the Mid Point Optimization on the agent based query engine is evaluated. Therefore a query selecting all actors with a Kevin Bacon number below or equal to two is executed[5] on an unoptimized as well as on an optimized embedding in a two dimensional Spatially Aware Graph Store. The related edge density plots, restricted on the edges access by this query, are shown in Figure 5c and in Figure 5d. It becomes clear, that in the unoptimized case, the nodes accessed by the query agent are randomly scattered over the entire data space. However, in the optimized embedding the access pattern is concentrated on a distinct hotspot. In this way, the overall distance an agent needs to travel is significantly reduced. In particular, 106,427,753 hops between processing units are needed in the unoptimized case and 33,439,669 in the optimized case.



(a) Edge Density before Optimization

(b) Edge Density after Optimization

(c) Query Access Heat Map before Optimization

(d) Query Access Heat Map after Optimization

Fig. 5: The DBpedia Data Set Stored in 2-D Spatially Aware Graph Store

V. Related Work

To the best of our knowledge, there are only the works of deLorimier and Kapre [4], [5] that followed a similar approach to combine the concept of Spatial Computers with Graph Based Processing. These works resulted in the *GraphStep* system, a novel Field Programmable Arrays (FPGAs) based compute model. However, this work differs significantly from their approach and focuses on a specialized concept for graph stores instead of a general compute model. In addition, the optimization of access times using optimal linear arrangement solvers presents an important cornerstone of this work, which has not been discussed by deLorimier

[4]Skipped after 72 hours

[5]Two degrees of separation from Kevin Bacon Query, RDF Store Benchmarks with DBpedia, http://wifo5-03.informatik.uni-mannheim.de/benchmarks-200801/, last visited February 2014

et al. Furthermore our approach simulates a one-, two- and three-dimensional spatial computer and is therefore not restricted to the two dimensional FPGA design. In the area of non-spatial computers several approaches for distributed graph stores exist. Zeng et al. presented Trinity.RDF, a dedicated main memory store [20]. Huang et al. combined RDF3x [14] with the Hadoop Map Reduce Framework in [11]. Finally, Erling presented Virtuoso, a hybrid store combining relational and graph concepts [6]. First in-depth experiments on the *Minimum Linear Arrangement* (MinLA), also known as *Optimal Linear Arrangement* problem, were conducted by Petit [16] in 2003. *Simulated Annealing* is the most traditional approach to optimize data sets, which leads to good results as it successfully escapes local minima [12]. However, this method features non-linear runtime complexity and therefore cannot be applied on large-scale datasets [17]. Consequently, faster concepts using multigrid techniques [2] became more popular. Safro and Koren presented such multigrid algorithms to solve the MinLA problem [13], [18]. Furthermore, *Graph Visualization* algorithms [10], [9] are related to the MinLA problem as their objective is to optimize node-to-node locality in a two-dimensional space. However, the problem behind graph visualization is much broader than MinLA, as it needs to consider the full range of human perception characteristics to achieve good results. As an example, modern algorithms also consider crossing angles of edges to improve human readability [15].

VI. Conclusion and Future Work

This work introduced a novel concept of Spatially Aware Graph Stores to address contemporary memory wall and scalability issues. It has been shown that the combination of a Graph Store and a Spatial Computer represents a way to enhance existing architectures by taking advantage of storing information in one, two or three physical dimensions. Furthermore, a realization of the concept needs to locate nodes as close to their linked nodes as possible to enable efficient graph querying. Consequently in this work, the distance between linked nodes has been minimized with a twofold approach. First, the effect of increasing the dimensionality of the Spatially Aware Graph Store on the overall distances between linked nodes has been evaluated. In fact, in the case of 2-D and 3-D stores, these distances are reduced by up to three orders of magnitude in comparison to a 1-D realization. Second, we have shown that the Mid Point Optimization is able to further optimize locality, which decreases the overall distances between linked nodes by another order of magnitude. Through the combination of the optimization method with a two or three dimensional Spatially Aware Graph Store, we could achieve a major speedup. Indeed our results show, that without this speedup it is not possible to optimize the linked node locality of large data sets in reasonable time. Finally the results on querying the DBpedia knowledge network showed comparable improvements, as on optimized datasets query agents need to travel significantly less through the Spatial Computer.

We conclude that our concept of a Spatially Aware Graph Store in combination with optimization algorithms provides a sound basis for efficient graph stores that are less vulnerable to the memory wall issue and enable distribution on a massive scale. In future work, many challenging issues have to be addressed to further develop the concept. The next steps will be to refine the optimization method to further improve linked node locality. In addition, the decomposition of very large nodes that appear in scale free graphs represents an interesting opportunity to further improve linked node locality.

References

[1] A. Abou-Rjeili and G. Karypis. Multilevel algorithms for partitioning power-law graphs. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, IPDPS'06, page 103, Rhodes Island, Greece, 2006. IEEE.

[2] A. Brandt, D. Ron, and D. Amit. Multi-level approaches to discrete-state and stochastic problems. *Multigrid Methods II*, pages 65–98, 1986.

[3] A. DeHon, J.-L. Giavitto, and F. Gruau. 06361 Executive Report – Computing Media Languages for Space-Oriented Computation. In *Computing Media and Languages for Space-Oriented Computation*, number 06361 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007.

[4] M. Delorimier and N. Kapre. GraphStep: A system architecture for sparse-graph algorithms. In *Field-Programmable Custom Computing Machines, 2006. FCCM'06. 14th Annual IEEE Symposium on. IEEE*, number FCCM, pages 143–151, 2006.

[5] M. Delorimier, N. Kapre, N. Mehta, and A. Dehon. Spatial hardware implementation for sparse graph algorithms in GraphStep. *ACM Transactions on Autonomous and Adaptive Systems*, 6(3):1–20, Sept. 2011.

[6] O. Erling and I. Mikhailov. Towards web scale RDF. *Proc. SSWS*, 2008.

[7] P. Fishburn, P. Tetali, and P. Winkler. Optimal linear arrangement of a rectangular grid. *Discrete Mathematics*, 213(1-3):123–139, Feb. 2000.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[9] D. Harel and Y. Koren. Graph Drawing by High-Dimensional Embedding. *Journal of Graph Algorithms and Applications*, 8(2):195–214, 2004.

[10] Y. Hu. Efficient, High-Quality Force-Directed Graph Drawing. *The Mathematica Journal*, 10(1):37–71, 2006.

[11] J. Huang, D. Abadi, and K. Ren. Scalable SPARQL querying of large RDF graphs. *Proceedings of the VLDB Endowment*, 4(11):1123–1134, 2011.

[12] D. S. Johnson, C. R. Aragon, L. A. Mcgeoch, C. Schevon, and R. Aragon. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Optimization*, pages 865–892, 1989.

[13] Y. Koren and H. David. A multi-scale algorithm for the linear arrangement problem. In *Graph-Theoretic Concepts in Computer Science*, pages 296–309, 2002.

[14] T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.

[15] Q. Nguyen, P. Eades, S. Hong, and W. Huang. Large crossing angles in circular layouts. In *Proceedings of the 18th international conference on Graph drawing*, pages 397–399, 2011.

[16] J. Petit. Experiments on the minimum linear arrangement problem. *Journal of Experimental Algorithmics*, 8(3):112–128, Jan. 2003.

[17] E. Rodriguez-Tello, J.-K. Hao, and J. Torres-Jimenez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research*, 35(10):3331–3346, 2008.

[18] I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, July 2006.

[19] W. Wulf and S. McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news*, pages 20–24, 1995.

[20] K. Zeng, J. Yang, H. Wang, B. Shao, and Z. Wang. A distributed graph engine for web scale RDF data. *Proceedings of the 39th international conference on Very Large Data Bases*, pages 265–276, 2013.