An Ad-hoc graph node vector embedding algorithm for general knowledge graphs using Kinetica-Graph[†]

B. Kaan Karamete*, Eli Glaser

Kinetica DB Inc.
901 North Glebe Road, Arlington, Virginia 22203

Abstract

This paper discusses how to generate general graph node embeddings from knowledge graph representations. The embedded space is composed of a number of sub-features to mimic both local affinity and remote structural relevance. These sub-feature dimensions are defined by several indicators that we speculate to catch nodal similarities, such as hop-based topological patterns, the number of overlapping labels, the transitional probabilities (markov-chain probabilities), and the cluster indices computed by our recursive spectral bisection (RSB) algorithm. These measures are flattened over the one dimensional vector space into their respective sub-component ranges such that the entire set of vector similarity functions could be used for finding similar nodes. The error is defined by the sum of pairwise square differences across a randomly selected sample of graph nodes between the assumed embeddings and the ground truth estimates as our novel loss function defined by Equation 3. The ground truth is estimated to be a combination of pairwise Jaccard similarity and the number of overlapping labels. Finally, we demonstrate a multi-variate stochastic gradient descent (SGD) algorithm to compute the weighing factors among sub-vector spaces to minimize the average error using a random sampling logic.

Keywords: *Knowledge graphs, graph embedding, vector similarity*

1. Introduction

There is not a definitive way to represent fixed dimension vector node embed-dings from variable dimension general knowledge graph connections (relations as edges). The simple reasoning is that there is really no rule in a general graph connection sense. Nevertheless, in the last decade, researchers have been pushing the envelope to apply the success of vector embedding advancements in Large Language Models (LLM) over to the general context knowledge graphs. A number of novel algorithms, namely, node2vec and word2vec [1, 2] devised with lots of success in language semantics and undoubtedly opened doors for today's many AI applications that seem to be championed as the master key solution for most of our engineering problems, if not all (exclusions are mostly in multi constraint optimization problems, such as supply chain logistics and optimal fleet routes). Though, possibly a more humble acceptance of LLM's superiority is when there is supposedly a hidden pattern among word pairings that can be put in a neural net machinery to minimize the error between the assumed solution and the known ground truth based on either some training data (supervised) or a logic of differentiation (unsupervised/reinforced) [3].

However different in its details of minimizing errors to create these sophisticated language models that would produce intended outcomes, its superiority mainly lies in the deterministic nature of the input and the output with the additions of some fuzziness so that near-reality outcomes could be achieved [4, 5]. In a general knowledge graph sense, however, there is no language ruling for how a node 'Tom' is connected to 'Bill' or 'Jane' or to the country of his/her birth place, certainly none better than mere connections as node to node 'relation's.

* *Corresponding author: Bilge Kaan Karamete, kkaramete@kinetica.com, karametebkaan@gmail.com*
† *Kinetica-Graph: https://arxiv.org/abs/2201.02136*

It is always possible however, the problem can be cast into a language model by connecting these general nodal relations around building sentences and paragraphs. These embeddings are broadly generalized into 'translational' and 'semantic' categories and extensively surveyed in [6, 7] where the former is distance based and the latter is relation centric.

The intention of this paper is not to find this mapping from unstructured knowledge graphs to language graph models so we could apply the celebrated node2vec method to create nodal embeddings for similarity analysis. Our goal, however, is to create an ad-hoc mapping framework that is based on each graph's own analytics and using as much machinery as possible from LLM technology to mimic similarities between the node pairs and combine 'translational' and 'semantic' mappings together. Perhaps, one could rephrase that a graph is its own AI model where its connections reveal the unstructured information in its most true form and any other representation is just an approximation at best.

In this spirit, we try to come up with a vector embedding in which we use a number of graph predicates, such as topological hop-patterns, common labels, transitional probabilities via Markov-chain (MC) probabilities, and clustering indices via the recursive spectral bisection (RSB) solver [8] using Kinetica-Graph [9, 10, 11, 12]. We would refer these as sub-features and explain each group in Section 2 in detail. We then describe a flattening procedure to spread these sub-feature predicates onto the sub-ranges of the vector embedding space in Section 3. A novel loss function definition is described in Section 4 where an average embedding error is assumed to be the sum of square differences between the inner product of nodal vector pairings and pairwise sum of Jaccard scores combined with pairwise common labels. Finally, we will show a stochastic gradient descent (SGD) algorithm [13] that minimizes this average error by adjusting the weights among sub-feature groups in the embedded space in Section 5.

2. Sub-vector features

The vector space is divided into sub-group range of indices that are indicative of ad-hoc graph predicates as shown in Figure 1. This is crucial since a value at an index location would have a specific meaning for every node and a share in similarity score when it is inner product-ed with that of another node. These predicates are specifically chosen to capture the local and remote affinities. The following predicates are chosen per graph node:

- Hop-patterns
- Label index associations
- Cluster index
- Transitional probability

These predicates are explained in detail below.

2.1. hop-patterns

The first feature predicate encompasses a range of indices to depict hop based pattern numbers as shown in Figure 2. Hop pattern of a node is defined by the number of forks and the number of nodes in each fork arm as shown with the respective colors per hop; e.g., second hop depicted as cyan has two forks with two nodes at each fork arm. This is not full-fledged topological pattern matching, since that would require the node indices instead of the number of nodes at breadth-first search (bfs) adjacency traversal. The reason why we can not use the node indices in the vector is that it does not have a meaning as a value subject to inner product. If we can find a better means to universally reflect node indices in vector embeddings, this sub-feature could be replaced with much accurate values, but for now, we'll be using this light weight topological feature. Maximum number of hops is added as an option to the embedding algorithm as the set of pattern based numbers can slide within the array based on this option.

2.2. Label indices

We have devised in [10] an efficient mechanism to attach multiple labels to nodes and edges. The labels are stored with their unique indexes in the graph db. This feature has as many sub-range indices in the vector as there are unique labels in the graph (that has node-associations). The idea as similar to hop patterns is for these array indexes to have an absolute meaning throughout the nodes, i.e., if a label index is common to a number of nodes, the specific array index for that label should be turned on for all those nodes that share the same label. The label indices are depicted in Figure 1 as k, m, n, p for each sub-feature, respectively.

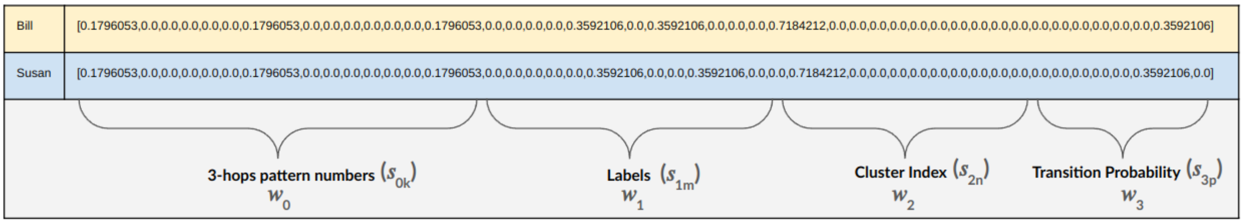


Figure 1: The layout of the sub-features within the vector embedding space; The sub-features, $s_{0..3,k..p}$ per node are hop based topology pattern, associated labels, the cluster index computed using the recursive spectral bisection (RSB) algorithm and the transitional probabilities (markov chain solver), respectively. The k, m, n, p are vector ranges per each sub-feature. Four weight factors, $w_{0..3}$ that will be used to minimize the average total loss per node are eventually multiplied with each value within the sub-range of its respective feature $s_{0..3}$ for the final embedded vector content per graph node.

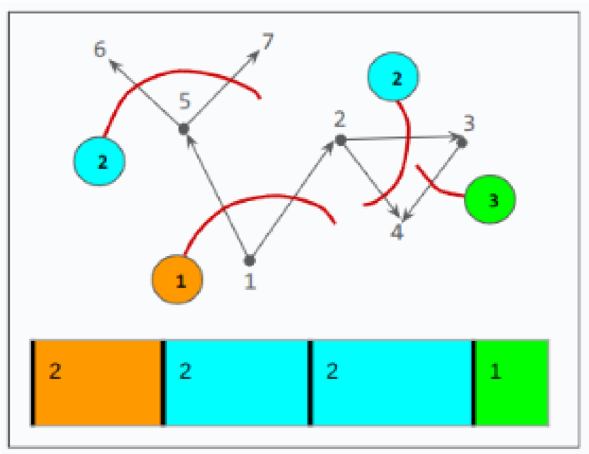
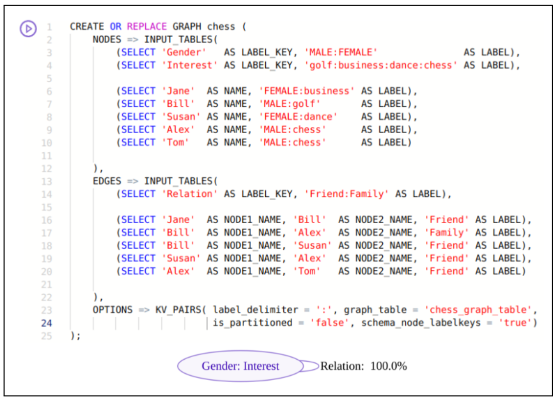


Figure 2: Hop pattern of a node is defined by the number of forks and the number of nodes in each fork arm as shown with the respective colors per hop; e.g., second hop depicted as cyan has two forks with two nodes at each fork arm as shown in the array below.

To illustrate label associations and indices, the generation of a simple wiki-graph is depicted with the SQL syntax of Kinetica-Graph's restful API in Figure 3. Node labels are listed in the response of the create-graph call in Figure 5 as a DB table. A 3D visualization of the graph with nodal label associations are also shown in Figure 4.

2.3. Cluster indices

A recursive spectral bisection algorithm is used to compute the cluster indexes of graph nodes. The idea in this Kinetica-Graph implementation is inspired from [8] where the sorted second smallest eigenvector of the graph Laplacian is used in bisecting at the median location at each recursive split as depicted in Algorithm 6. The choice of partitioning for the endpoint match/graph with 'spectral' option



```
CREATE OR REPLACE GRAPH chess (
    NODES => INPUT_TABLES(
        (SELECT 'Gender'   AS LABEL_KEY, 'MALE:FEMALE'                AS LABEL),
        (SELECT 'Interest' AS LABEL_KEY, 'golf:business:dance:chess' AS LABEL),

        (SELECT 'Jane'  AS NAME, 'FEMALE:business' AS LABEL),
        (SELECT 'Bill'  AS NAME, 'MALE:golf'       AS LABEL),
        (SELECT 'Susan' AS NAME, 'FEMALE:dance'    AS LABEL),
        (SELECT 'Alex'  AS NAME, 'MALE:chess'      AS LABEL),
        (SELECT 'Tom'   AS NAME, 'MALE:chess'      AS LABEL)

    ),
    EDGES => INPUT_TABLES(
        (SELECT 'Relation' AS LABEL_KEY, 'Friend:Family' AS LABEL),

        (SELECT 'Jane'  AS NODE1_NAME, 'Bill'  AS NODE2_NAME, 'Friend' AS LABEL),
        (SELECT 'Bill'  AS NODE1_NAME, 'Alex'  AS NODE2_NAME, 'Family' AS LABEL),
        (SELECT 'Bill'  AS NODE1_NAME, 'Susan' AS NODE2_NAME, 'Friend' AS LABEL),
        (SELECT 'Susan' AS NODE1_NAME, 'Alex'  AS NODE2_NAME, 'Friend' AS LABEL),
        (SELECT 'Alex'  AS NODE1_NAME, 'Tom'   AS NODE2_NAME, 'Friend' AS LABEL)

    ),
    OPTIONS => KV_PAIRS( label_delimiter = ':', graph_table = 'chess_graph_table',
                         is_partitioned = 'false', schema_node_labelkeys = 'true')
);
```


Figure 3: Graph-SQL syntax for Kinetica-Graph's create/graph Restful API. The nodes and edges components are depicted explicitly, i.e., with constants instead of reading from table columns (unless the example is simple the usual way is to list the nodes/edges in a DB table or stream in). Bottom is showing the chess graph ontology using the label keys; all edges (100%) are in between *Gender* and *Interest* labeled nodes via *Relation* group edge label key.

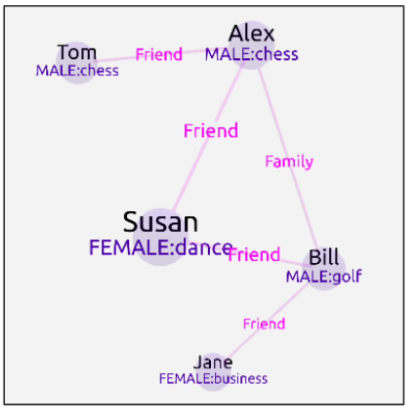


Figure 4: 3D visualization of the graph generated by tha call in Figure3 with node/edge label associations using d3's force layout

NODE_NAME	NODE_LABEL
Jane	FEMALE:business
Bill	MALE:golf
Susan	FEMALE:dance
Alex	MALE:chess
Tom	MALE:chess

Figure 5: The response of Kinetica-Graph's create/graph call depicting node-label associations as a relational DB table. E.g.: *Alex* and *Tom* has two common labels, namely, *chess* and *MALE*.

is particularly preferred for its speed of execution and low resource allocation requirement compared to the *Louvain* clustering [14] (another option in the solver) as shown in the Figure 8. A geometrical example of the RSB method with three levels of bisections can be seen in Figure 7.

The cluster index per node is pushed into the respective sub-range allocated for this feature. The width of this feature over the vector space can be scaled by overriding the default value of 8 via the maximum number of clusters option to the embedding solver. The output of the RSB clustering is shown as a DB table in Figure 9.

2.4. Transitional Probabilities

Inspired from the Pagerank algorithm [15], our novel probability ranking solver uses the same equation depicted in 1 with a modified transition probability flux p_{ij} where it is computed to be the ratio of incoming adjacent edge weights (connecting nodes i and j) within the immediate neighbor $B(i)$ to each node i. These nodal scalar p_i values are iterated at each traversal loop converging to a steady state where the maximal change in p_i is less than a small threshold. The ranking factor r is assumed to be 0.15 so that every node will have a small amount of uniform probability (r divided by the number of graph nodes NV) to account particularly for nodes with no incoming edges.

$$p_i = (1 - r) \sum_{j \in B(i)} p_{ij} + \frac{r}{NV} \tag{1}$$

These transitional probabilities are scaled between zero and one and the sub-range for this feature over the vector is allocated to a preset division. For example, if a particular node's probability is 0.25, with the default range of ten, the third index within the sub-vector range is turned on. It is also worth mentioning that this solver behaves exactly like a Pagerank solver for the uniform identical edge weights scenario where the transitional probability defined above simply becomes the incoming valence rank.

3. Flattening

Hop-pattern numbers are per each fork of a hop. Therefore, primary flattening occurs in mapping this two dimensional information over the one dimensional vector space. Furthermore, a secondary flattening happens for laying the sub-feature's own indexing after the previous feature's flattened index location. Label and cluster indices do not require flattening but their respective vector locations have to be shifted after the prior feature's index range. Transitional probability, however, would require to map the continuous scalar probability values to be bucketed over a preset number of interval indices. This is easily accomplished by partitioning the unit range equally over a preset number of intervals (can be modified by the user). The vector size formula is given in Equation 2 where each sub-feature's respective ranges are summed. The parameters *max_num_clusters*, *max_hops* are user driven and *max_forks_perhop*, *max_edges_perfork* are set implicitly to minimize the number of pattern indices within the overall vector dimension.

$$\begin{aligned} Vector\_size = {} & max\_patterns + \\ & max\_labels + \\ & max\_num\_clusters + \\ & num\_probabilities \\ where & \\ max\_patterns = {} & max\_hops \times \\ & max\_forks\_perhop \times \\ & max\_edges\_perfork \end{aligned} \tag{2}$$

Before the normalization process, these vector values within each sub-feature are multiplied by a feature specific weight parameter. We can extend the number of sub-features in our embedding framework, however, at the time of writing this manuscript, we are currently having four sub-features, hence, we have only four weight parameters that we will use for the purpose of finding out

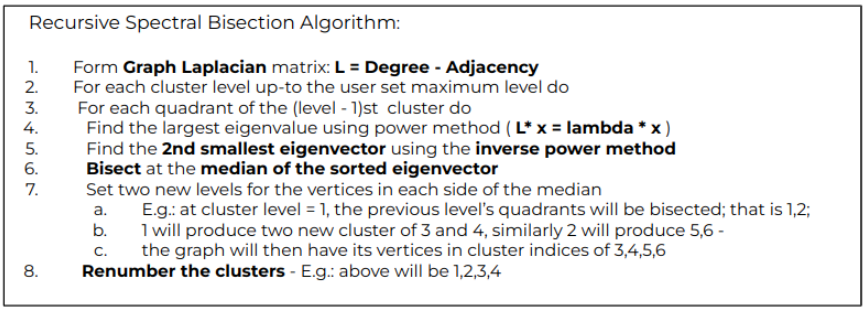


Figure 6: Steps of Recursive Spectral Bisection (RSB) Algorithm

which of these sub-features are intrinsically more dominant over each other for the specific graph in contention using an equal error distribution and minimization procedure that will be explained in Section 4 and 5.

Having multiplied by their respective weights, the vectors are then normalized so that their inner products can be used for vector similarity analysis. The unit vector embedding values for the node pair 'Bill' and 'Susan' for the simple wiki-graph is shown in Figure 1.

3.1. Quantizing

Straightforward mapping of the float predicates over the vector indices will only provide inner products for those whose predicates fall onto the exact vector index. However, this is not realistic since the integer index equivalent for a float value corresponding to a pagerank or distance (extended version of the embedding solver includes distances) score is too restrictive and would certainly miss close but off-the-index values in similarity (inner-product) computations. For instance, a value of 5.4 should not just turn on the 5^{th} index (in a 10 slot sub-range dedicated for the predicate) but the nearby indices based on its deviation from the exact index, in a hat-like diffusing behavior. Hence, we have developed a *quantizing* logic to diffuse these values over a range of nearby indices as shown in Figure 12 where in the particular example not only the 5^{th} but the nearby 4^{th}, 6^{th} and 7^{th} indices had the effect of dispersion based on the deviations of these indices from the exact value in a piece-wise fashion. *Quantizing* is a key concept that helps increase the chance of capturing potential similarities among the node embeddings.

4. Loss Function

The concept of assuming the total error distributed evenly across the nodal pairs is inspired from the computational mechanics field [16], specifically in finite element analysis defined as *z-square* where the elemental errors are aggregated over the entire domain and then divided equally over the finite elements. Similarly, in our sub-feature weight optimization, we can define the total error as the aggregated sum of the differences between the inner product of each node against every other in a subset of the graph and the ground truth estimates for each pair. Specifically, the error is defined by the sum of pairwise square differences across a randomly selected sample of graph nodes between the assumed embeddings and the ground truth estimates as our novel loss function defined by Equation 3. The ground truth is estimated to be a combination of pairwise Jaccard similarity and the number of overlapping labels.

Loss function is defined per node i such that the goal is to find the average difference aggregated over all the pairs from the node i to all other nk number of nodes. The similarity, i.e., the inner product between the vector embeddings of f_i and f_k is subtracted from the pairwise sum of Jaccard similarity score and the number of overlapping labels between the pairs as our revised ground truth estimate as shown in Equation 3(a). The α value is chosen to be 0.5 which is the mean of the two measures. The pairwise-error functions is made *L2* norm so that the derivative of the loss function with respect to the four weight parameters would have the truth estimate terms for the optimization algorithm. The vector embedding f_i or f_k is a function of w_j's. The problem is then reduced to applying the minimization procedure to the average of the total sum of the nodal losses across the sub-graph as shown in

$$Loss_i = \frac{1}{nk}\sum_{k}^{nk}\left\| < f_i, f_k > - \left(\alpha\ jac(i,k) + (1-\alpha)\frac{labels(i) \cap labels(k)}{\sum_k distinct(labels(k))}\right)\right\|_2 \quad (3a)$$

$$< f_i, f_k > = w_r^i w_r^k s_{rj}^i s_{rj}^k \quad (3b)$$

$$w_j \leftarrow min\left(\frac{\sum_i (Loss_i)}{NV}\right) \quad \forall j \ \mid \ j = 1..4 \quad (3c)$$

the Equation 3(c).

The selection of the sub-graph is done randomly such that we only grab equal number of graph nodes as batches within each cluster index (computed at the sub-feature creation) so that the random set is representative enough of the entire graph behavior. The total number of the random sampling process is a user defined parameter and usually much less than the original graph size to minimize the overall computational time.

Finally, by using this random sampling logic a multi-variate stochastic gradient descent (SGD) algorithm is devised to compute the weighing factors in minimizing the average nodal error of Equation 3(c) in Section 5.

5. Stochastic Gradient Descent

The sum of pairwise differences between the inner product of pairs for node i, node k and the ad-hoc ground truth from Jaccard scores with overlapping label count ratio as depicted in Equation 3(a) is dependent on the unknown terms w_j as the weight factor of each four sub-features. The selection of the set of graph nodes where each node is paired with every other in the set is important in finding these optimal weights. The process needs to include nodes to have a good representation of the entire graph behavior. We have opted to sample this set randomly (stochastic) with a caveat of picking the batch of nodes from each cluster index group where we have already computed in constructing the sub-features. The number of nodes in this random selection process is user specified, however, it needs to be much less than the original graph size so that the SGD iterations would not be prohibitive.

The next step is taking the derivative of the nodal average of the loss per each of these weight parameters and move against the direction of the gradient of each weight to minimize the loss. The incremental update on each unknown weigth is immediately made to reflect its impact on the next unknown weight variable computation as shown in Equation 4 and 6 in which $w_j^{(k+1)}$ is the next $(k+1)^{th}$ iteration on the j^{th} weight parameter. This approach is not a guaranteed outcome in accelerating the convergence. Other alternative approaches such as *batching* or *mini-batching* discussed extensively in [13] might provide better computational outcome. However, we think this is beyond the scope of this study and our findings are satisfactory computationally for the cases we have tried so far. The iterations are continued if the relative incremental iteration delta of all unknown weights go below a preset user threshold (default value is 0.001) or the number of epoch iterations reaches the upper limit (default is 100) which is also a user prescribed parameter of the solver as shown in Figure 13. The convergence history plot between the number of iterations and the error is also shown in Figure 14 for the knowledge graphs of varying sizes from a few hundred to 100 million nodes. The trend in all cases is with early unstable fluctuations and rapid descending to the optimal as expected. The initial weight values and the rate of iteration, also known as training or learning rate, namely, β as shown in Equation 4, can both also be overridden by the user.

$$w_j^{(k+1)} \leftarrow w_j^k - \beta \left.\frac{\partial \sum_i (Loss_i)\ /\ NV}{\partial w_j}\right|_k \quad (4)$$

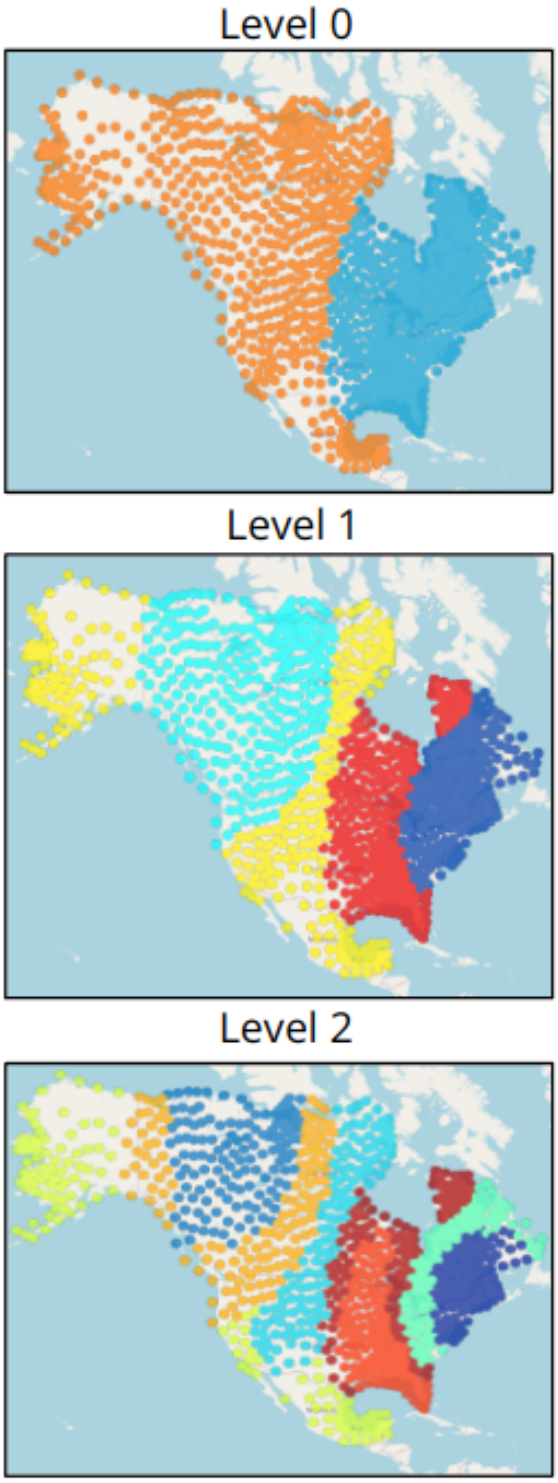


Figure 7: The application of the RSB algorithm over the mesh graph of the continental USA. The three levels of bi-section creates 8 (default number of clusters) clusters from top to bottom, respectively.

```
SELECT * FROM MATCH_GRAPH(
    GRAPH => 'chess',
    SOLVE_METHOD => 'match_clusters',
           OPTIONS => KV_PAIRS(
       cluster_quality_metric = 'spectral', max_num_clusters = '8')
)
```

Figure 8: The Graph-SQL syntax of match/graph restful API to generate clusters using RSB method. Another option is the *Louvain* clustering with more resource requirements.

NODE	CLUSTER
Jane	1
Bill	1
Susan	2
Alex	2
Tom	3

Figure 9: The output of the RSB clustering - the cluster indexes per node as a DB table for the simple wiki-graph example

NODE	PROBABILITY
Alex	0.29097033
Bill	0.29097033
Susan	0.19402967
Tom	0.11201484
Jane	0.11201484

Figure 10: The results of the probability ranking solver per node as a DB table - the probabilities are scaled between a zero (small value) and one, with the sum equal to one.

```
SELECT * FROM SOLVE_GRAPH(
    GRAPH => 'chess',
    SOLVER_TYPE => 'PROBABILITY_RANK',
    OPTIONS => KV_PAIRS( uniform_weights = '1.0')
)
```

Figure 11: Graph-SQL syntax in table function form of Kinetica-Graph's solve/graph restful API with uniform weights of 1. Equal weight distribution of the solver mimics the Pagerank results.

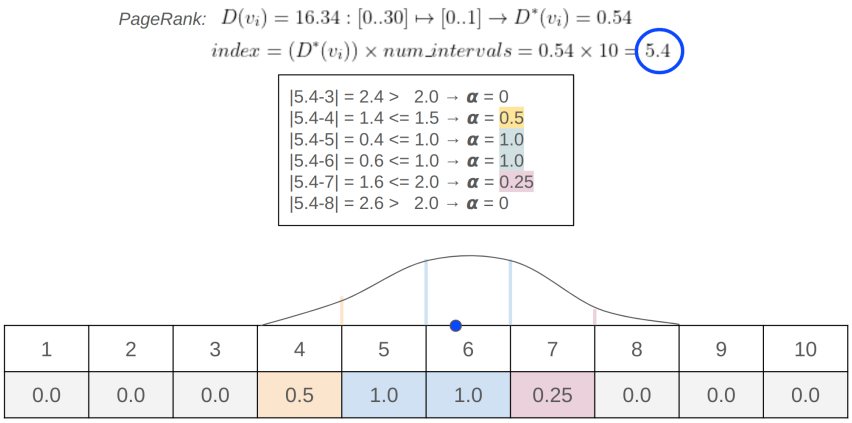


Figure 12: Quantizing: piece-wise dispersion of the float value of 5.4 for the pagerank score to a number of nearby vector indexes based on the deviations from the exact value. Spread of nodal values mapped between zero and one using a sigmoid is followed by the quantizing step that increase the chance of capturing more similarities among node embeddings.

```
DROP TABLE IF EXISTS chess_embedding;
EXECUTE FUNCTION
MATCH_GRAPH(
    GRAPH => 'chess',
    -- SAMPLE_POINTS => INPUT_TABLES(
    --     (SELECT 'Bill' AS NAME),
    --     (SELECT 'Susan' AS NAME)
    -- ),
    SOLUTION_TABLE => 'chess_embedding',
    SOLVE_METHOD   => 'match_embedding',
    OPTIONS => KV_PAIRS(
        force_undirected             = 'true',
        max_vector_dimension         = '150',
        max_hops                     = '3',
        max_num_clusters             = '8',
        embedding_weights            = '1,0,1.0,1.0,1.0',
        optimize_embedding_weights   = 'true',
        optimization_sampling_size   = '1000',
        optimization_error_tolerance = '0.001',
        optimization_max_iterations  = '1000',
        optimization_iteration_rate  = '0.2'
    )
)
```

Figure 13: Graph-SQL syntax of Kinetica-Graph's match/graph restful API with the embedding solver and options that shifts the sub-ranges of the features in the embedded vector space. If the optional *SAMPLE_POINTS* component is not commented, the solver would output only the pairs specified via the component's combo two-tuple identifiers.

$$K_{i,m} =< f_i, f_m > -\Big(\alpha\ jac(i,m) + (1-\alpha)\frac{labels(i) \cap labels(m)}{\sum_m distinct(labels(m))}\Big) \quad (5)$$

$$\frac{\partial \sum_i (Loss_i)}{\partial w_j} = 4 \sum_i \sum_m (K_{i,m} w_j s_j^i s_j^m) \quad (6)$$

6. Discussion and Conclusions

We have tried to map unlimited dimension general knowledge graph topology onto a 1−dimensional vector embeddings by constructing the vector space from features that we think would best resemble local affinity and remote structure so that any vector similarity (inner product) between a node pair would result in the same similarity behavior if we had computed the Jaccard score with the number of common labels between the two nodes of every pair. To this end, the sub-features are chosen to be the predicates of hop-pattern numbers, cluster indices (computed by the recursive spectral bisection (RSB)), associated label indices, and the transitional probability (or the Pagerank score if weights are uniform) as explained in Section 2 above. The impact of the vector component sub-features on the similarity can be found by adding a weight parameter to multiply each of

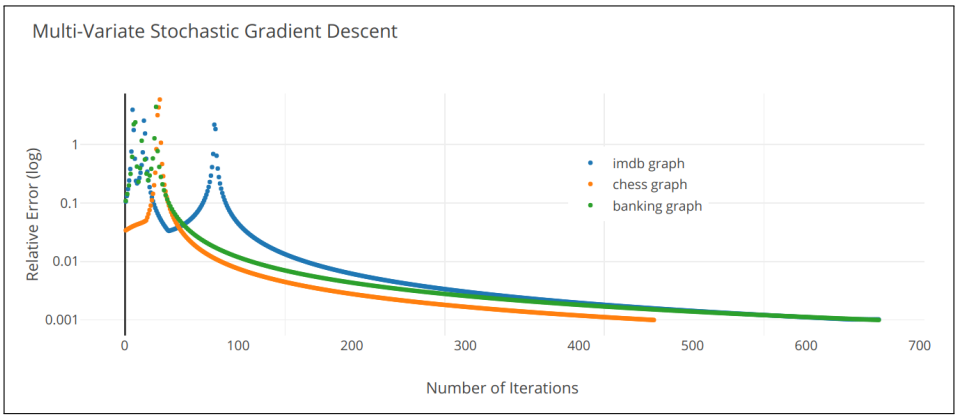


Figure 14: Stochastic Gradient Descent Convergence

the sub-feature elements within the respective vector sub-range and test the result against an estimated ground truth as explained in Section 4. We have formulated the difference between the inner product of the assumed embeddings and the combined common-labels and Jaccard score as the ground truth, as our ad-hoc *Loss* function as shown in Equation 3. We then tried to optimize the nodal average of the total loss by applying a stochastic gradient descent (SGD) algorithm to find the unknown weights so that this total average loss is minimized as explained in Section 5 and Equations 4 and 6.

Stochastic process is the selection of the random nodes that will be used in SGD to find the unknown weights. We have chosen these random nodes from each graph cluster in equally numbers. We use this smaller sub-set of nodes in computing the unknown weights. The assumption of picking this narrow set of nodes from each cluster is to increase the likelihood of better representation of the entire graph since SGD on the entire graph is computationally prohibitive. SGD converges very similarly in our testing of many graphs as shown in Figure 14. The banking graph shown, is in 10+ million range (4+ billion case is also used), and its ontology is depicted in Figure 15.

The output of the embedding solver is a database table with a vector per graph node as depicted in Figure 16. These embedding results can be used in any vector similarity functions; such as a cosine similarity as depicted in Figure 17. A common use case for vector similarity is, for instance, in recommendation engines for various industries, from friend recommendations in social networks to the next likely item in your shopping chart. The efficiency and accuracy of these embeddings, however, depends on the rich-ness of the vector sub-features and the sophistication of the randomly selected training sets in optimizing the vector contents. We argue that even the best embedding algorithm would be less accurate compared to the precise connections and labels depicted in the graph topology itself. However, mapping of graphs to vectors has a distinct advantage that they can be applied in a standard manner using simple vector functions in many AI modules. The alternative of using knowledge graph analytics has almost no standardization in many downstream AI applications provided by various graph vendors.

The stipulation of the existing four sub-features representative of the graph topology can certainly be mitigated by either adding more features or a different set of predicates. One other criterion that seems to make sense to include is the distance metric as discussed in [6, 7]; which considers similarity for nodes at an equal distance from a set source. However, this statement implies to include all nodes to be similar lying on the same ring-radius distance (hop or weight distance) from the center as the source. This is however a wrong postulate since we know that the nodes on the same ring may be at equal distance away from a source at the center, but they are no-where close to each other particularly for the nodes across each other at any section of the ring. However, along with the cluster index as is already a sub-feature, the combined effect (always consider the inner-product sense) might move the argument to a more acceptable and even preferred state. Another area of future development is in the

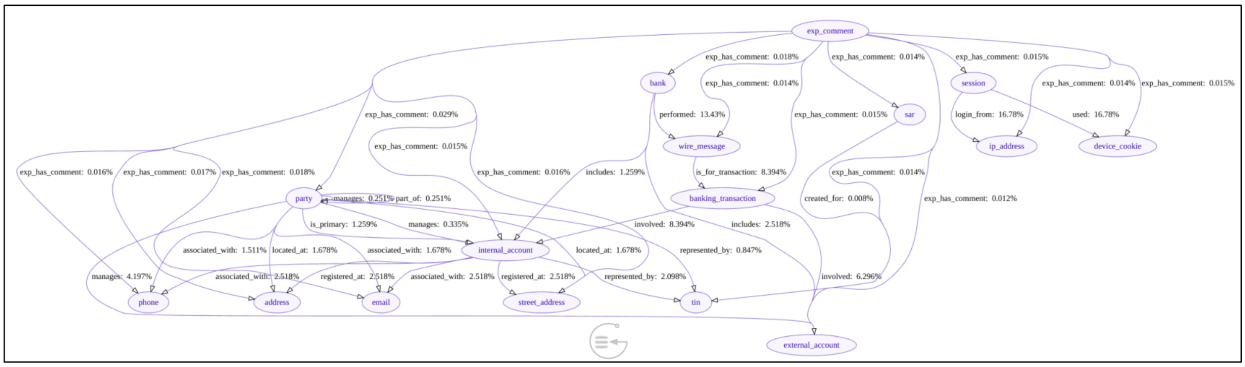


Figure 15: Banking graph ontology with 34 edge and 16 node labels with 10 million+ nodes. The percentages show how many actual graph edges are connected between each labels.

Schemas › ki_home › chess_embedding

Add | Edit | Del | Move | Conf | CSV

NODE	EMBEDDING
Jane	-0.4981874, 0, -0.4981874, 0, 0, -0.4981874, 0, 0, 0, 0, ...
Bill	0, -0.57456464, 0, 0, -0.57456464, 0, 0, 0, 0, 0, ...
Susan	0, 0, 0, 0, -0.57456464, 0, 0, 0, 0, -0.57456464, ...
Alex	0, -0.57456464, 0, 0, -0.57456464, 0, 0, 0, 0, 0, ...
Tom	-0.4981874, 0, -0.4981874, 0, 0, -0.4981874, 0, 0, 0, 0, ...

Figure 16: The vector embedding output table *chess_embedding* as the result of the Kinetica-Graph embedding solver depicted in Figure 13.

```
SELECT   t1.NODE as source,
         t2.NODE as target,
         ROUND(COSINE_DISTANCE(t1.EMBEDDING, t2.EMBEDDING),3)
         as cos_similarity
FROM
(SELECT * FROM chess_embedding) as t1
  CROSS JOIN chess_embedding as t2
 WHERE STRCMP(t1.NODE, t2.NODE) = -1
```

Figure 17: The Kinetica-SQL statement using a cross join to run the vector similarity analysis between node embeddings of each node pair from the result table, *chess_embedding* of the embeddding Kinetica-Graph solver.

dynamic additions to the graph, and how to update graph embeddings for the new additions that should be calculable instantly and ready for vector analysis in order for it to be useful in real-time simulations. We are considering to eliminate recomputing of the embeddings for new node insertions by caching and using the results of already computed weight parameters and interpolating probability and cluster indexes from adjacent nodes instead of running compute heavy cluster and probability solvers. It'll then be up-to the user to decide when to recompute for more accurate embedding values, most probably to be triggered after the number of updates reaching to a significant threshold.

Notes on Contributors

Bilge Kaan Karamete is the lead technologist for the Geospatial and Graph efforts at Kinetica. His research interests include computational geometry/algorithm development, unstructured mesh generation, parallel graph solvers. He holds PhD in Engineering Sciences from the Middle East Technical University, Ankara Turkey, and post doctorate in Computational Sciences from Rensselaer Polytechnic Institute, Troy New York. **Eli Glaser** is VP of Engineering at Kinetica. He leads the development teams concentrating in data analytics, query capability and performance. Eli holds Master's in Electrical Engineering from The Johns Hopkins University, Baltimore Maryland.

7. Software avaliability

Kinetica and Kinetica-Graph is freely available in Kinetica's Developer Edition at **https://www.kinetica.com/try** that the use cases depicted in this manuscript can easily be replicated by the readers.

References

[1] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks (2016). `arXiv:1607.00653`. URL `https://arxiv.org/abs/1607.00653`

[2] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality (2013). `arXiv:1310.4546`. URL `https://arxiv.org/abs/1310.4546`

[3] V. Zhou, Blog:machine learning for beginners: An introduction to neural networks, https://victorzhou.com/blog/intro-to-neural-networks, accessed: 2024-07-12.

[4] L. Huang, P. Zhao, H. Chen, L. Ma, Large language models based fuzzing techniques: A survey (2024). arXiv:2402.00350.
URL https://arxiv.org/abs/2402.00350

[5] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, A. Mian, A comprehensive overview of large language models (2024). arXiv:2307.06435.
URL https://arxiv.org/abs/2307.06435

[6] S. Choudhary, T. Luthra, A. Mittal, R. Singh, A survey of knowledge graph embedding and their applications (2021). arXiv:2107.07842.
URL https://arxiv.org/abs/2107.07842

[7] X. Ge, Y.-C. Wang, B. Wang, C. C. J. Kuo, Knowledge graph embedding: An overview (2023). arXiv:2309.12501.
URL https://arxiv.org/abs/2309.12501

[8] Y. Hu, R. J. Blake, Numerical experiences with partitioning of unstructured meshes, Parallel Computing 20 (1994) 815–829.

[9] B. K. Karamete, L. Adhami, E. Glaser, A fixed storage distributed graph database hybrid with at-scale olap expression and i/o support of a relational db: Kinetica-graph (2022). arXiv:2201.02136.
URL https://arxiv.org/abs/2201.02136

[10] B. K. Karamete, E. Glaser, Novel data structures for label based queries specifically efficient for billion+ property graph networks using kinetica-graph (2023). arXiv:2311.03631.
URL https://arxiv.org/abs/2311.03631

[11] B. K. Karamete, E. Glaser, Optimal routing algorithm for trips involving thousands of ev-charging stations using kinetica-graph (2022). arXiv:2206.06241.
URL https://arxiv.org/abs/2206.06241

[12] B. K. Karamete, L. Adhami, E. Glaser, An adaptive markov chain algorithm applied over map-matching of vehicle trip gps data, Geo-spatial Information Science 24 (3) (2021) 484–497.
URL https://doi.org/10.1080/10095020.2020.1866956

[13] S. Ruder, An overview of gradient descent optimization algorithms (2017). arXiv:1609.04747.
URL https://arxiv.org/abs/1609.04747

[14] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, Journal of Statistical Mechanics: Theory and Experiment 2008 (10) (2008) P10008. doi:10.1088/1742-5468/2008/10/p10008.

[15] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking : Bringing order to the web, in: The Web Conference, 1999.
URL https://api.semanticscholar.org/CorpusID:1508503

[16] O. C. Zienkiewicz, J. Z. Zhu, The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique, International Journal for Numerical Methods in Engineering 33 (1992) 1331–1364.
URL https://api.semanticscholar.org/CorpusID:120762978