Fast graph similarity search and motif mining via locality sensitive hashing

Agnishom Chattopadhyay, Nicolae Sapoval
Rice University, Houston, TX 77005

1 Abstract

Motivation. Graphs are ubiquitous throughout many application areas of computer science including computer systems (network topologies), programming languages and program design, and bioinformatics (assembly graphs [1, 2], interaction and coexistence networks). Two common tasks that arise in the computational analyses of graphs are graph structure comparison and motif mining. Both of these tasks are computationally hard in their exact forms (e.g. the graph isomorphism problem is quasipolynomial and the subgraph isomorphism problem is NP-hard), therefore motivating the need for approximate methods that can achieve high accuracy at lower computational costs.

Problem statement. We are aiming to address two problems in our work. (1) Given a database of graphs $D = \{G_1, ..., G_N\}$ and a query graph G return a subset $\mathcal{T} \subseteq \mathcal{D}$ of the nearest neighbors G_i to G under some fixed graph distance metric. (2) Given a single graph G and a query graph Q, with $|V(Q)| \ll |V(G)|$, determine with high probability if Q occurs as a subgraph (or induced subgraph) of G and estimate the count of its occurrences.

Prior approaches. For efficient estimation of pairwise graph similarity, the concept of graph kernels has been introduced [3]. There are several major categories of graph kernels that are based on random walks, motif counts [4, 5], and subtree patterns [6]. In the setting described in (1), one can leverage fast graph kernels and a set of discriminating graph samples from the database D (called prototypes) to provide an embedding from the space of graphs $\mathcal{G}$ into $\mathbb{R}^n$ and subsequently use locality sensitive hashing to perform efficient similarity queries with respect to the database D [7].

Study goals. Given the two outlined problems and the prior work done in the area the aims of our study are threefold.

1. We plan to investigate impact of different graph kernels and different locality sensitive hash function families on both accuracy and runtime of graph similarity queries with respect to a large graph database D.

2. We want to examine if the problem (2) can be reduced to problem (1) by considering the database D as inferred (for example via sampling) from the graph G, and subsequently if the approaches studied in goal 1 can be translated into approaches for problem (2).

2 Final Report

2.1 Problem description

Recall that our primary statement is as follows. We wish to preprocess a database of graphs $D = \{G_1, ..., G_N\}$ so that given a query graph G, we can return a subset $\mathcal{T} \subseteq D$ of the nearest neighbors G_i to G under some fixed graph distance metric κ. In this study we aim to investigate three metrics induced by graph kernels: Weisfeiler-Lehman subtree kernel of depth h, graphlet kernel for graphlets of size up to k, and connected graphlet kernels of size up to k.

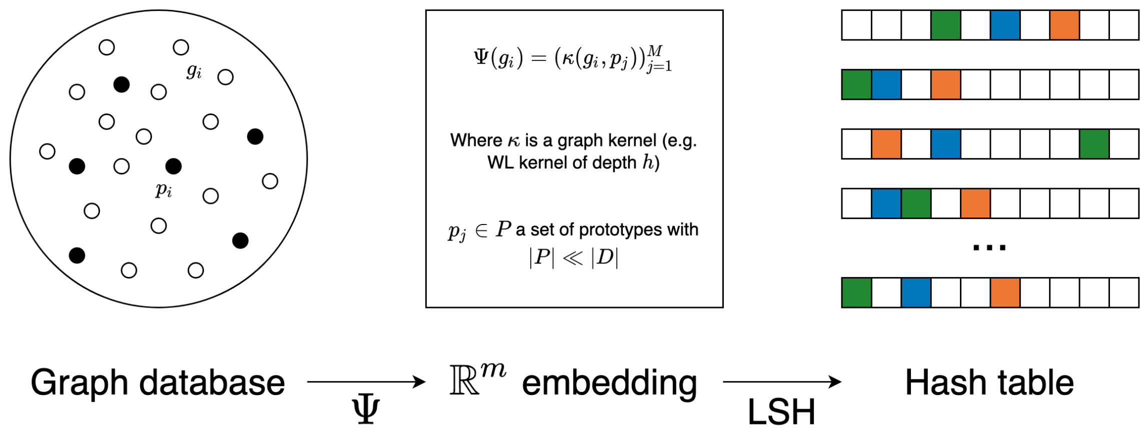

g_i
p_i
$\Psi(g_i) = (\kappa(g_i, p_j))_{j=1}^{M}$
Where κ is a graph kernel (e.g. WL kernel of depth h)
$p_j \in P$ a set of prototypes with $|P| \ll |D|$
T
Graph database
Ψ
$\mathbb{R}^m$ embedding
LSH
Hash table

Figure 1: An illustration of the two step process for building a hashtable H that stores graph database D. First, we pick a set of prototypes P, and embed our graphs into $\mathbb{R}^m$ via a kernel κ based embedding Ψ. Next, we perform LSH on the embeddings to construct a hashtable H which consists of T individual hashtables, each indexed into by B hashes.

The general approach we take for this problem is based on the work of Zhang *et al.* [7]. The overview of the main idea is provided in Figure 1. Briefly, the approach proceeds in the following steps:

1. Given the database D of graphs, pick a set of prototypes $P = \{p_1, p_2, \cdots p_m\} \subset D$
2. Define the function $\Psi : \mathcal{G} \to \mathbb{R}^m$ as
$$\Psi(g) = (\kappa(g, p_1), \kappa(g, p_2), \ldots, \kappa(g, p_m))$$
3. Represent each graph $g \in D$ by $\Psi(g)$
4. Use a family of LSH functions h_i, $i \in \{1, ..., B\}$ to insert graph g into the bucket indexed by $h_i(\Psi(g))$, repeat for all T tables
5. Given a query graph q return the union of all buckets that q is hashed into
6. Filter the returned candidate set $C = \{c_1, ..., c_L\}$ to its top-K members via explicit computation of $\kappa(q, c_i)$

Thus, we specifically want to investigate if: (a) we can improve the accuracy of the nearest neighbor retrieval by selecting our prototypes as centroids of the $|P|$ clusters formed under the metric κ in D; (b) what are the accuracy and search time trade-offs between the three considered kernels (WL, graphlet, connected graphlet); (c) how do parameters of the actual hash table (table count, hash function count, range) impact the empirical performance.

2.2 Literature review

In their work [7] Zhang *at al.* introduce the framework that combines graph kernel computations with locality sensitive hashing for fast near neighbor search in graph databases. Their key idea relies on the two step process of first creating an embedding from graph space into $\mathbb{R}^m$ and then using locality sensitive hashing in $\mathbb{R}^m$ to efficiently retrieve near neighbors. Authors aimed to approximate graph edit distance metric, but since graph edit distance computations are too slow, instead they relied on much faster Weisfeiler-Lehman subtree kernel [6]. However, authors did not provide any specific way for picking prototype set within the database, which motivates a part of our study.

The original paper that proposes using modified version of Weisfeiler-Lehman isomorphism test as a graph kernel [6] motivates it's contribution, as a computationally efficient alternative to the graph edit distance. Authors provide a mathematical argument for why WL-kernel efficiently approximates the graph edit distance, and provide two algorithms for efficient single pair kernel computation and bulk pairwise kernel computation.

Additionally, there have been other graph kernels proposed that combine efficient computation with good approximation of the graph edit distance metric. In particular, the graphlet based kernels have been proposed [5, 4] as a way of estimating similarity in the graph space. Since, the original paper we considered, only looked at WL-kernels, we have been interested in exploring graphlet kernels as an alternative approach for the same problem, in order to evaluate their computational performance.

2.3 Experimental results

Key hypothesis: *An optimized selection of prototype graphs at the preprocessing stage, can improve the accuracy of nearest neighbor search in graph space*.

We have used the Weisfeiler-Lehman (WL) kernel for our experiments, as suggested by the authors of [7]. We follow the method from [7]. We investigate how tuning the number of hash functions K, range of each hashtable R and the number of tables T affect the performance of this data structure.

In [7], a specific form of LSH is used that is based on the p-stable distributions [8]. We implement the LSH family that they prescribe.

To analyze the performance of our algorithm, we query our database with a graph g randomly picked from $\mathcal{D}$ itself. Then, we also pick the graphs from $\mathcal{D}$ which are truly similar to g based on the value of the kernel itself. Then, we check the size of this intersection. Our aim is to maximize the size of this intersection as a ratio of the total number of neighbors queried for.

For computing kernels of these graphs, we use the graphkernels library which represents graph as igraph objects.

For our experiments, we use the NCI1 and NCI-H23 databases that were downloaded from `https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets`. NCI1 has 4110 graphs with mean vertex count of 30 and mean edge count of 28. NCI-H23 has 40353 graphs with mean vertex count of 26 and mean edge count of 28.

For all experiments we have fixed the depth for WL kernel computation to $h = 10$ for the hash table building stage, and increased it to $h = 60$ for the evaluation stage. Thus, the ground truth set of top-50 nearest neighbors is computed with higher precision than the one used for deciding candidate set from the hash table.

In the first set of experiments we have selected 300 random graphs from our dataset to serve as prototypes, and 500 random graphs from the dataset to serve as query graphs. For each query we have computed it's true top-50 neighbors under the WL kernel ($h = 60$). We then proceeded to vary the number of hashtables used T, number of hash functions per table B, the weight parameter W of the LSH function, and the number of top hits to be returned from candidate set K. For each set of parameters we have computed the average proportion of reported top-K hits from the candidate set that are present in the true top-50 set for this query. Additionally we report average time per 500 queries to our hashtable.

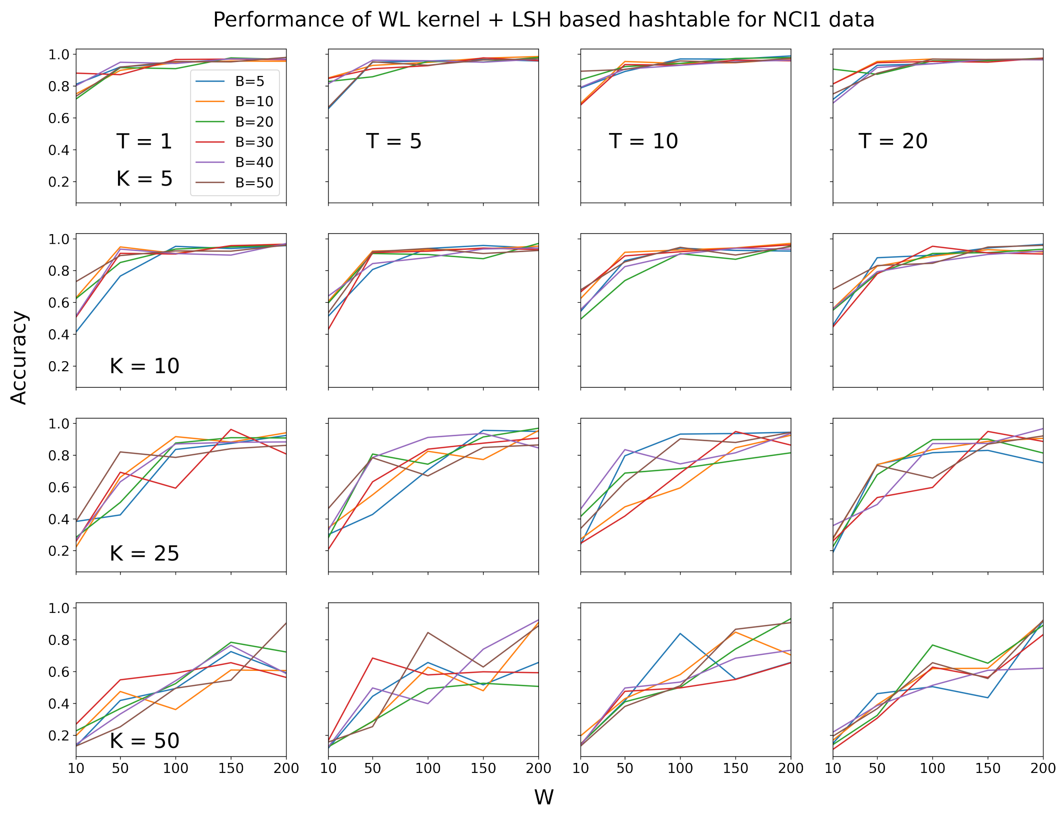


Figure 2: Performance of WL Kernel + LSH based hashtable for NCI1 data

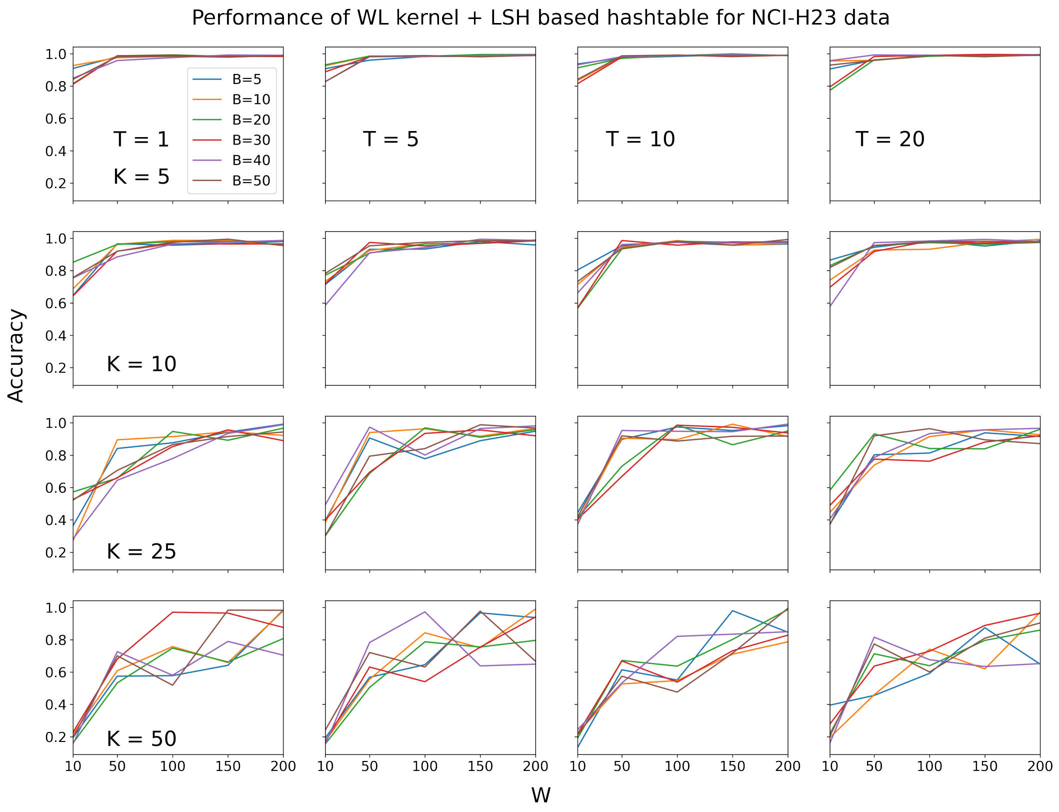


Figure 3: Performance of WL Kernel + LSH based hashtable for NCI-H23 data

No. of hashes **No. of tables**	5	10	20	30	40	50
1	86.90s	103.94s	140.43s	178.80s	207.98s	248.72s
5	272.21s	377.29s	573.05s	757.69s	961.83s	1125.76s
10	535.59s	688.94s	1046.69s	1371.27s	1767.09s	2177.06s
20	905.69s	1327.08s	2048.34s	2744.13s	3422.81s	4220.89s

Table 1: Time per 500 queries for the NCI1 dataset.

No. of hashes **No. of tables**	5	10	20	30	40	50
1	596.12s	668.54s	651.45s	701.62s	629.90s	700.56s
5	1848.05s	2181.73s	2223.67s	2437.88s	2726.57s	2993.99s
10	3945.37s	3785.79s	4063.03s	4634.10s	5047.63s	5652.16s
20	6682.92s	6756.61s	7860.36s	8760.91s	9293.99s	9566.74s

Table 2: Time per 500 queries for the NCI-H23 dataset.

Additionally, we have evaluated two extensions of the initial experimental setup. Namely, we considered a varying number of prototype graphs p which took values 50, 100, 200, 300 and 500, and we also compared the performance obtained by applying spectral clustering to the data and picking prototypes as graphs closest to the inferred cluster centroids.

We observe that with the increase of the number of prototypes used the overall performance of our hashtable degrades (Fig. 4). Additionally, we note that contrary to our hypothesis choosing near-centroids of clusters decreases the performance of our algorithm. We have conjectured two possible causes of this phenomenon: (1) Since the size of the NCI1 dataset is on the scale of $\sim$4,000 graphs it is possible that oversampling the prototypes can introduce too much bias into the embedding function; (2) Spectral clustering tends to produce even sized clusters, which might bias sampling towards dense regions of the graph space, and skewing the embedding.

To test the frst hypothesis we have performed a similar evaluation on the NCI-H23 dataset which contains 10 times more graphs that NCI1 dataset.

We note that the general trend of decreased performance with higher number of prototypes holds for the NCI-H23 dataset (Fig. 5). However, the absolute impact of the increased number of prototypes is lower than in the NCI-H23 case, suggesting that hypothesis (1) might hold partially. Additionally, we also note that similarly to the NCI1 dataset we get on average worse performance from employing a clustering pre-processing step.

To further investigate the reason why performing clustering can lead to worse performance, we have visualized our randomly selected prototypes and prototypes inferred form cluster centroids by using WL-kernel based distances, and projecting into 2 dimensional space via tSNE.

We note that at the lower count of prototypes random sampling tends to yield a less biased selection of prototypes (Fig. 6), while the clustering based method tends to oversample the dense regions of the space. As the number of prototypes increases both methods tend to select comparably similar sets of prototype graphs (Fig. 7). Furthermore, the clustering based method tends to better resolve dense clusters once the total number of prototypes is high enough.

2.4 Conclusion

Thus, we have found that empirically in this scenario clustering pre-processing step yields no observable advantage, and in fact tends to slightly decrease the performance of the method. We have also observed that the number of prototype graphs can adversely affect the performance, and therefore needs to be tuned to the estimated dataset size.

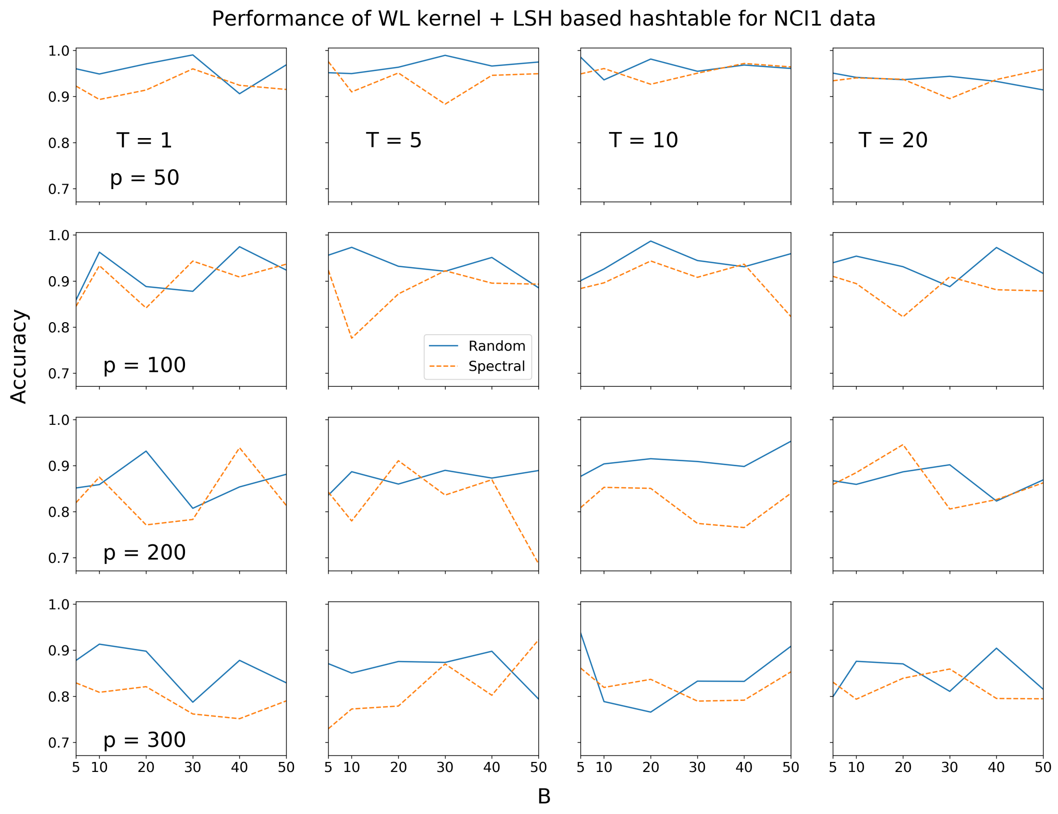


Figure 4: Performance of WL Kernel + LSH based hashtable for NCI1 data for top-25 nearest neighbors. Results are averaged over the range of W values used for hash functions.

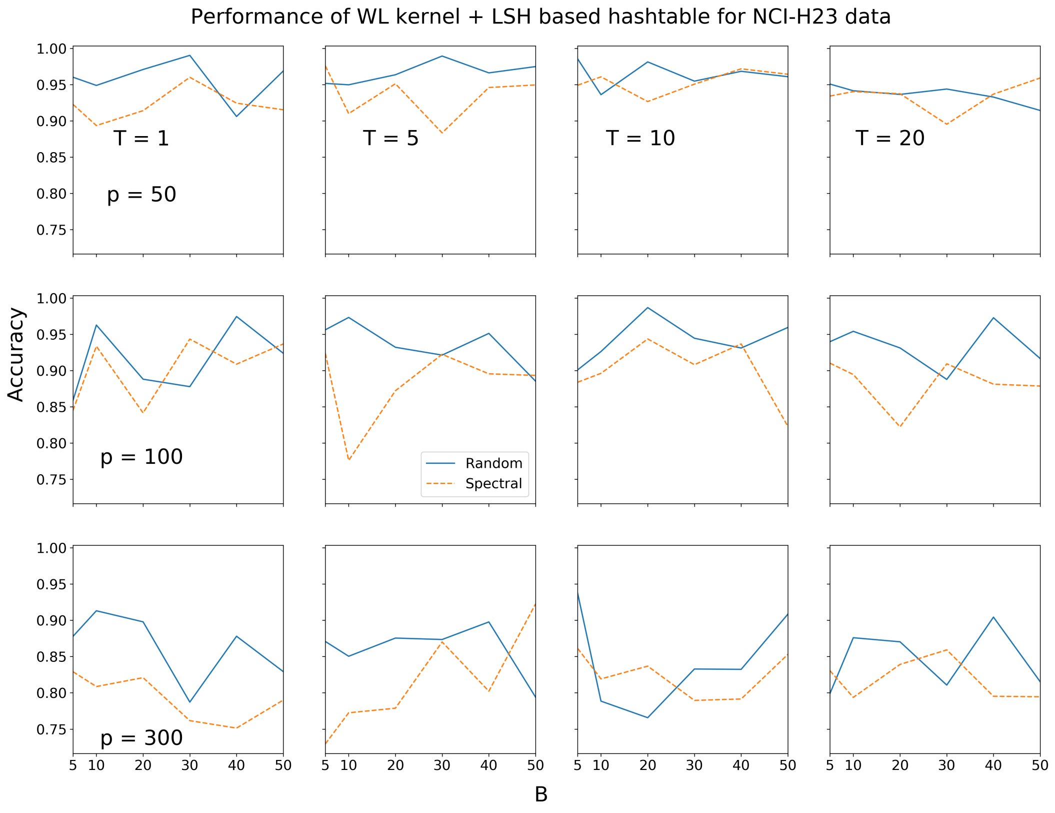
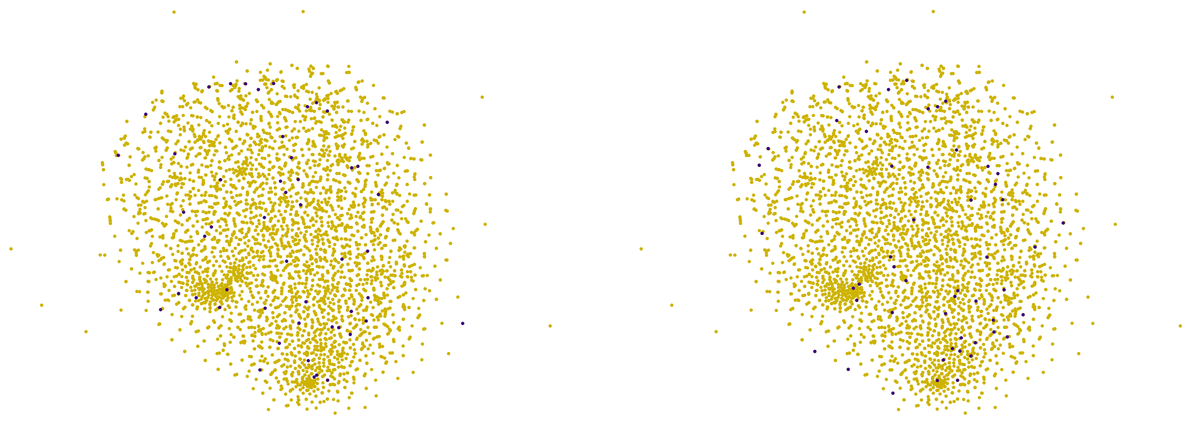
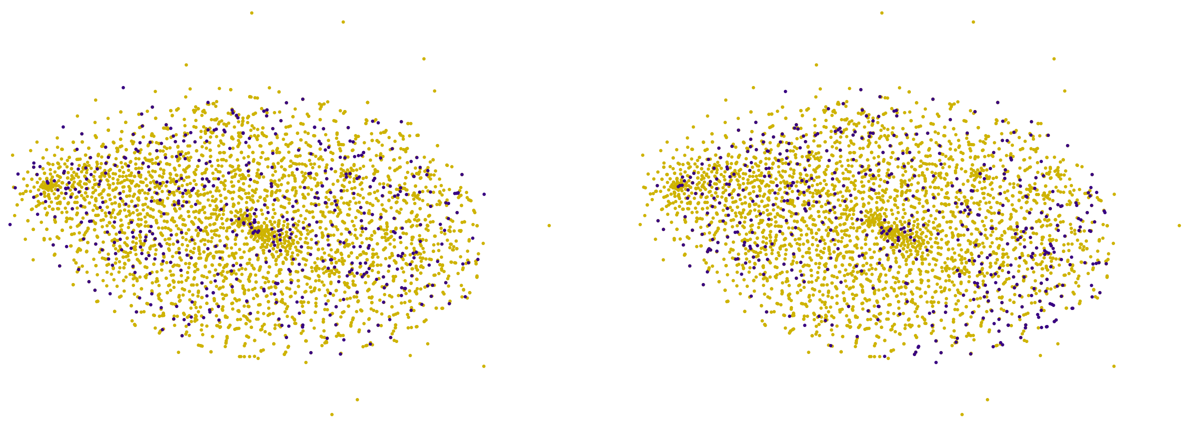


Figure 5: Performance of WL Kernel + LSH based hashtable for NCI-H23 data for top-25 nearest neighbors. Results are averaged over the range of W values used for hash functions.



(a) Randomly selected prototypes (b) Prototypes as approximate cluster centroids

Figure 6: Selected prototypes ($p = 50$, purple) visualized against other graphs (yellow) via tSNE based projection from WL-kernel metric space into $\mathbb{R}^2$.



(a) Randomly selected prototypes

(b) Prototypes as approximate cluster centroids

Figure 7: Selected prototypes ($p = 500$, purple) visualized against other graphs (yellow) via tSNE based projection from WL-kernel metric space into $\mathbb{R}^2$.

References

[1] Jurgen F. Nijkamp, Mihai Pop, Marcel J. T. Reinders, and Dick de Ridder. Exploring variation-aware contig graphs for (comparative) metagenomics using marygold. *Bioinformatics*, 29(22):2826–2834, 2013.

[2] Jay Ghurye, Todd Treangen, Marcus Fedarko, W. Judson Hervey, and Mihai Pop. Metacarvel: linking assembly graph motifs to biological variants. *Genome Biology*, 20(1):1–14, 2019.

[3] Swarnendu Ghosh, Nibaran Das, Teresa Gonçalves, Paulo Quaresma, and Mahantapas Kundu. The journey of graph kernels through two decades. *Computer Science Review*, 27:88–111, 2018.

[4] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 158–167, 2004.

[5] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495. PMLR, 2009.

[6] Nino Shervashidze and Karsten M Borgwardt. Fast subtree kernels on graphs. In *NIPS*, pages 1660–1668, 2009.

[7] Boyu Zhang, Xianglong Liu, and Bo Lang. Fast graph similarity search via locality sensitive hashing. In *Pacific Rim Conference on Multimedia*, pages 623–633. Springer, 2015.

[8] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.