Spectral Sparsificaton of Edges for Efficient Clique Counting

Aaron Schindler
School of Computing
University of Utah
Salt Lake City, UT, USA
u1117629@utah.edu

Hari Sundar
Computer Science
Tufts University
Medford, MA, USA
hari.sundar@tufts.edu

***Abstract*—Clique finding has emerged as a valuable data mining technique with applications in social computing, biology, chemistry, network analysis, and anomaly detection. In this paper, we propose a novel method for clique counting that exploits the spectral structure of the graph to sparsify the graph without affecting cliques. By employing a graph's eigenvectors, we can efficiently count edges that occur within the graph's spanning trees, thereby eliminating redundant edges. The proposed edge sparsification method is evaluated against contemporary algorithms for both time and accuracy metrics. By utilizing the spectral sparsification approach, we are able to maintain a high accuracy for clique counting even when k is large at the cost of some speed performance. On dense graphs, spectral sparsification is able to achieve nearly 20% density reduction.**

I. Introduction

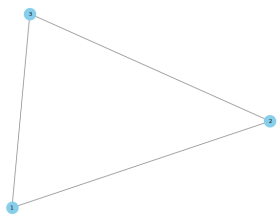
The ever increasing scale of modern network graphs presents a significant computational challenge for fundamental analysis tasks, such as the enumeration of cliques. In bioinformatics, for instance, protein-protein interaction networks can readily exceed 100,000 nodes and millions of edges [1]. Similarly, social and information networks are now approaching scales of over a million nodes and billions of edges [1]. Within these massive datasets, identifying cliques—i.e., subgraphs where every node is connected to every other node—is crucial for discovering cohesive functional modules, communities, and other higher-order structures. A variety of advanced algorithms have been developed to address the clique enumeration problem, employing techniques such as sampling [2]–[4], graph coloring [5], and exact enumeration [6]. However, each of these methods fails to produce adequate results for values of $k > 20$. While optimizing for speed, these methods have neglected to account for performance on large groups. For instance, when tested on the Wiki-Vote [1] and musae-Github [7] datasets, whose clique sizes reach 17 and 24 respectively, Turàn Shadow [4] fails to reach 50% accuracy. With clique and graph complexity growing the size of k is also increasing, indicating there is a need for methods that can accurately identify large cliques in a graph in a reasonable amount of time.

This paper introduces a novel framework designed to accurately and efficiently identify large cliques in massive graphs. Our approach is centered on a targeted pre-processing step that utilizes the graph's spectral signature to perform intelligent sparsification. Specifically, we analyze the edge frequencies across a multitude of spanning trees to approximate the structural importance of each edge. The core hypothesis is that edges integral to the graph's clique structures will appear with high frequency in this analysis, whereas less important edges that merely increase computational complexity can be pruned. This method effectively sparsifies the graph while preserving its essential clique topology. To validate our approach, the proposed spectral sparsification framework is rigorously benchmarked against several state-of-the-art clique-finding algorithms. The comparative analysis focuses on two key metrics: computational time and the accuracy of clique detection, particularly for large values of k. The objective is to demonstrate that our method provides a viable and superior solution for discovering large-scale clique structures in the complex, large-scale networks characteristic of contemporary data science challenges.

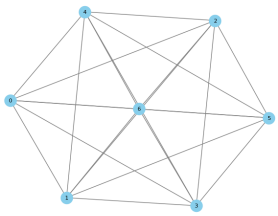
II. Background

A. Clique Problem

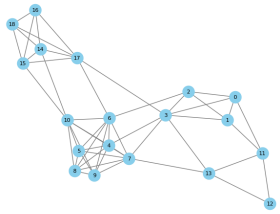
A clique by definition is any subset of vertices in an undirected graph $S \subset G = (V, E)$ such that for all vertices $u, v \in S, \{u, v\} \in E$. A simple example would be this property would be that of a clique of size 3, a triangle, noted in figure 1a. Verification of a small clique like this can be done quickly and cleanly by checking that each vertex has an edge to all other vertices included in the clique. For the triangle example, this would only require approximately 6 steps, checking each vertex for 2 edges. Naturally, the complexity in size and density of a clique increases significantly as the size of the clique grows. While a triangle only contains 3 edges, a clique of size 7 would require 21 edges as shown in figure 1b, increasing the number of verification steps to 147. Identifying cliques is further complicated by having multiple cliques of various sizes with overlapping vertex membership, shown in figure 1c. When a clique is not isolated, and contains edges to non-member vertices, the number of steps to verify increases since it is not guaranteed that the first k edges of a vertex belonging to a k-clique will be the edges corresponding to that clique.



(a) Triangle graph (3-clique)



(b) 7-clique



(c) Small graph comprised of multiple cliques

Fig. 1: Triangle (top left), 7-clique (top right), and small clique graph (bottom left)

Finding cliques in a graph can be partitioned into 3 separate use cases: Maximum clique, maximum clique listing, and k-clique counting. The maximum clique problem can be defined as finding the largest sized clique in a graph. This version of the clique finding problem is useful as it can set the upper bound on the search space and complexity for finding all cliques in a graph. A sample use case of this is predicting the most likely protein interaction in a protein product graph. Here, the largest clique represents a protein with a large amount of chemical reactions, leading to the identification of interactions as well as identifying protein structure [8]. While significant improvements have been made in this area, the fastest algorithms for this problem still require at least $\binom{n}{k} \cdot O(n^3)$ time to complete [9].

Similarly, an extension of the max clique problem is the maximum clique listing problem. The maximum clique listing problem is defined as listing all the cliques in a graph that are the maximum size. This problem is most useful for identifying major communities in social networks as well as points of interest in biological datasets. The running time of maximal clique listing algorithms typically mirrors that of the maximum clique problem because maximal clique listing commonly utilizes maximum clique algorithms as subroutines.

Lastly, there is the k-clique counting problem and its slight variation the maximal k-clique counting problem. The goal of both is to find all cliques of size k in a graph, however the maximal k-clique counting problem imposes the additional restriction of requiring each identified clique to not be a subset of any larger clique. Maximal k-clique counting is considered easier than the k-clique counting problem since there are no problems with vertex membership to multiple cliques. Alternatively, the k-clique counting problem is typically seen as the most difficult of the problems, since there are fewer restrictions on clique membership. For example, when finding cliques of size 3, a clique of size 4 will contain multiple size 3 cliques. This paper will focus mainly on this problem, by utilizing sparsification of the graph before attempting to identify cliques.

B. Related Work

There are many clique finding algorithms that have been used in practice. Most of which can be categorized into either sampling based, coloring based, or exact. Sampling based algorithms utilize some form of sampling of edges or vertices to create high probability areas for cliques like GRAFT [2], PEANUTS [3], and Turàn Shadow [4]. Similarly, another method employed to find cliques is by graph coloring, which is primarily used by DPColor, DPColorPath and DPTriPath [5]. Lastly, there is the exact counting group of algorithms, most exact counting algorithms have been derived from Chiba and Nishizeki's algorithm [10], with one of the best performing being KCList [6]. This paper will focus on testing against DPColorPath and Turàn Shadow, since they are the best performing algorithms from their respective groups. The exact counting algorithms will not be utilized in testing since they are known to not perform well for large k due to computational complexity.

1) DPColorpath: One of the modern methods to identify cliques is the DPColorpath method [5]. The defining feature of this method is the use of k-coloring a graph. By using a greedy coloring algorithm, the k-color graph is used to identify cliques by analyzing every k-colored set in the graph. Because of the large number of possible k-colored sets, an efficient verification is done using dynamic programming by keeping track of previously verified subsets so they do not need to be recomputed later. This method is also further extended in [5] to a parallel processing version which allows for the coloring and identification to be done on separate processes as they are disjoint operations. Doing so significantly improves the overall running time of the DPColorpath algorithm, however there are a few limitations.

One limitation of this approach is the necessity to build a custom representation of the graph during pre-processing that allows for efficient parallel computation. Many real world graphs are stored as simple $\{u, v\}$ edge pairings in a text format or as adjacency lists in JSON format. However, for DPColorpath, there is a need to build a custom CSR matrix format for the graph, including pre-computed information on the number of each k-clique for improved results. In addition, another limitation of this approach is that for larger values of k, the efficacy of DpColorpath diminishes significantly. This is caused by the degree of difficulty that comes with k-coloring a graph for large values of k. In order to identify large cliques, a large value of k is selected for coloring, which in turn causes

there to be a larger number of k-colored sets in the graph that act as false positives when identifying the smaller sized cliques.

2) Turán-Shadow: A second modern method of clique finding that is widely used is the Turán-Shadow algorithm [4]. This method utilizes the identification of dense subgraphs as a starting point for clique identification. The name and mechanism behind this method is derived from Turán's theorem [11] stating that a dense enough subgraph must contain a clique of a certain size.

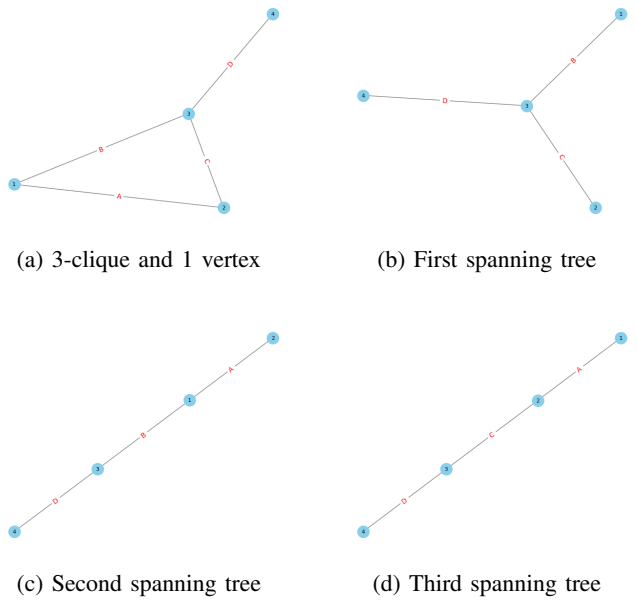
Theorem 1. (Turàn [11]) For any graph H, if $\rho_2(H) > 1 - \frac{1}{k-1}$, then H contains a k-clique.

Using this idea, the Turàn Shadow [4] method can build what is called a clique shadow by sampling vertices that come from high density areas. In order to find vertices that may belong to high density areas, Turàn Shadow [4] utilizes a degeneracy ordering of the graph. Degeneracy ordering is a way to order vertices based on their degrees, which is then used to create a DAG with edges moving lower to higher degeneracy, used in the clique shadow construction. From here, the shadow is constructed by building a subgraph from vertices with large out-neighborhoods gained from the degeneracy DAG.

One potential drawback of this method is that it has not yet been tested on large values of k. There is concern that when testing for large values of k, the performance will decrease due to the necessity for small shadow sizes. This is because for larger shadow sizes, the amount of computation required will increase due to the increased number of vertices included in the shadow. In this scenario, the computational complexity can rival exact counting methods as the primary mechanism for shadow based methods is to efficiently search subsets.

III. Methodology

One potential place to improve upon in clique finding is the use of edge sparsification. By strategically removing edges that are not related to cliques themselves, we can improve the efficiency of clique finding. The method chosen here is to utilize spanning trees of the graph in order to determine which edges to remove. Spanning trees ensure that all vertices which are connected in the original graph are connected in the spanning tree graph. To help explain this method, consider a graph with a triangle connected to one other vertex as shown in figure 2a. Now consider all of the unique spanning trees that this graph can have as shown in figures 2b, 2c, and 2d. Notice that the edge labeled D that connects the triangle to vertex 4 is the only edge that must remain constant in the graph. By removing this edge, now the triangle can easily be identified as a clique. By removing the edges that always appear in a spanning tree, we can reduce the number of operations needed to find cliques. There are however, two difficulties that arise from this method.



(a) 3-clique and 1 vertex (b) First spanning tree

(c) Second spanning tree (d) Third spanning tree

Fig. 2: 3-clique example (top left), and spanning trees (top right, bottom left, bottom right)

The first is the complexity of finding unique trees for graphs with clique sizes $k > 3$. In these cliques, the number of unique spanning trees grows exponentially. In the above triangle clique example, the number of unique spanning trees was 3, however in a similar graph where $k = 4$, the number of unique trees increases to 16. Furthermore, when dealing with graphs consisting of multiple cliques, the complexity degenerates even more. In a graph with two four cliques connected by a single edge, the number of unique spanning trees is 256. Clearly, it is not feasible to generate all of the unique spanning trees for any practical clique graph. The solution to this relates to the second problem, which will be discussed first.

The second issue to deal with is that cliques can share more than one edge with another clique or vertex. If we consider a graph of two 4-cliques, with two edges connecting the two groups, it is clear that there will not be a singular edge that appears in every spanning tree. Now, with these two issues in mind, instead of looking for a an edge that appears in every spanning tree, what if we instead look for edges that have a high frequency of appearance in the spanning trees? In the previously mentioned double 4 clique graph, each of those two edges that connect the cliques will appear in exactly $\frac{1}{2}$ of the spanning trees. However, this does not entirely solve the issue of the needing to generate all of the different spanning trees for the graph. To solve this issue, we can instead generate a smaller amount of spanning trees and then count the frequency at which edges appear.

To demonstrate this property, we can consider the graph with 2 cliques of size 4 that are connected by 2 extra edges.

Now consider the fact that the largest clique is of size 4. This means that vertices in the clique have at least 4 edges, possibly more if connected to vertices outside the clique. Now, if we generate 4 spanning trees at random, then we would expect each edge in the clique to have a $\frac{1}{4}$ probability of being selected in the tree. However, the two edges that connect the two cliques would have a $\frac{1}{2}$ probability of being selected since one of the two edges is required to create a spanning tree. That means, when generating 4 spanning trees for this graph, we would expect the two clique connecting edges to appear twice as often as all the other edges. This would imply that high frequency edges are not needed in order to identify cliques.

The first attempt to find high frequency edges was to generate random spanning trees, however this resulted in too many intra-clique edges being removed. Instead the graph's spectral properties were utilized in order to generate spanning trees. By using the top k eigen vectors, we were able to create an edge ordering based on the eigenvalues of each vertex for each eigenvector. Because the values for each vertex varied enough between eigenvector, the edge ordering when building the spanning tree offered a much nicer spread of selected edges. This in turn allowed for fewer intra-clique edges being removed as a result. Therefore, by employing the use of eigenvector based spanning trees, graph sparsification was possible while maintaining the connectivity of cliques.

It should be noted that for small values of k, utilizing spanning trees is unlikely to perform well. This is because the threshold for removing frequent edges will be much closer to the number of spanning trees generated. However, for any $k \geq 6$ this did not appear to be a significant issue when testing. Additionally, another pitfall of this method is the reliance on inter-clique edges. In graphs where cliques are mostly disjoint, sparsification is unneeded and will not be useful. Fortunately, most real world instances containing useful clique information have inter-clique connectivity and contain cliques where $k \geq 6$.

IV. Results

A. Experimental Setup and Data

To test the spectral sparsification method, various graphs were selected from Stanford's SNAP graph library [1]. Specifically, the Wiki-Vote network, the musae-Github network, and the Github Stargazers [7] dataset. Additionally, randomly generated graphs were used to measure performance on large k, these graphs are called c100k70 and c200k115. Each of these graphs is a randomly generated undirected graph containing 100 and 200 maximal cliques with maximum clique sizes of 70 and 115 respectively.

There were two experiments run, the first to test the effect sparsification had on both accuracy and time for clique finding. This involved primarily the Github Stargazers [7] dataset. The second set of experiments was to measure the performance of the spectral sparsification method with other modern methods of clique finding, namely Turàn Shadow [4] and DPColorpath [5]. Each of these experiments was run using a i7-1300H processor (14 cores) with 32 GB of memory. All code was implemented in Python, C++ or both.

B. Sparsification Results

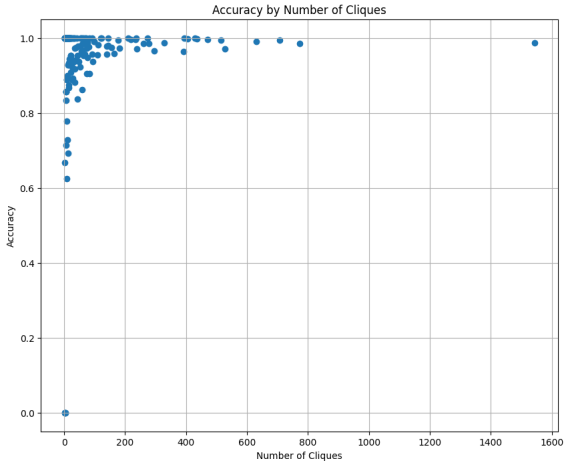


Fig. 3: Number of Cliques by Accuracy

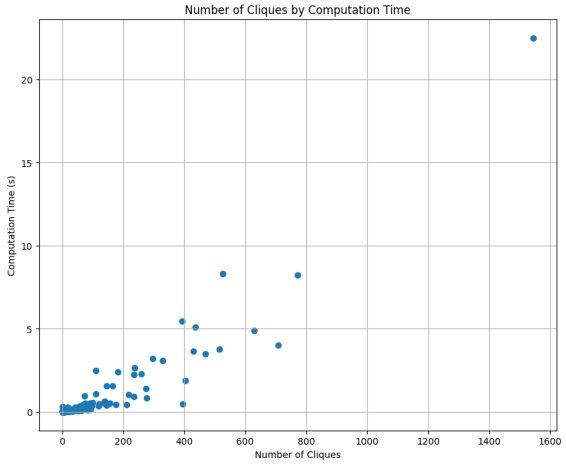


Fig. 4: Number of Cliques by Computation Time (s)

First we focus on the Github Stargazers [7] dataset. This dataset was used by only the spectral sparsification method in order to test the effectiveness of the method on small to moderately sized graphs. In figure 3 and figure 4 it is shown that the spectral sparsification method performs well. The accuracy of spectral sparsification method maintains an over 90% for the majority of the Github Stargazers [7] graphs while also keeping the running time below 20 seconds for the majority as well. The second metric to measure spectral sparsification on is the density reduction the method provides. In figure 5 and figure 6 percentage of density decrease is over 20% for the majority of the dataset. For both accuracy and density decrease, there appears to be diminished results for

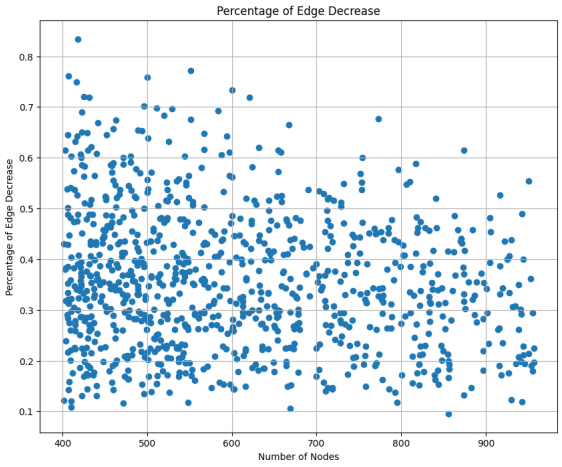


Fig. 5: Density comparison on Stargazers dataset

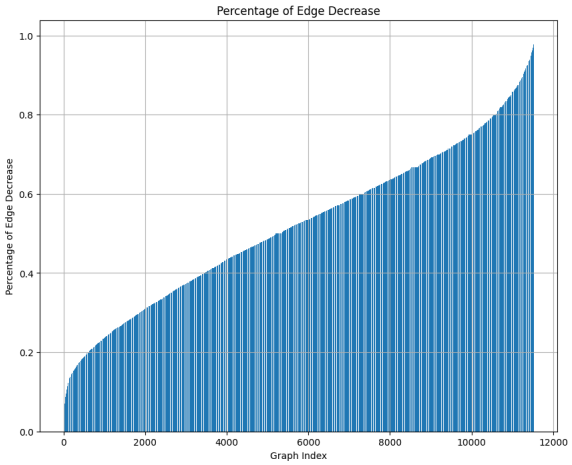


Fig. 6: Percentage edge sparsification on Stargazers dataset

smaller graphs. This result is expected since smaller graphs will have fewer edges that can be pruned as well as smaller values of k which will increase the chance of clique edges being removed. But, for the majority of graphs, sparsification worked well, showing positive results.

C. Timing and Accuracy Results

The next set of experiments was run on a variety of graphs containing a large amount of k-cliques as shown in table I. It should be noted that while both DPColorpath [5] and Tur'an Shadow [4] required pre-processing steps to create custom representations for the graph, all pre-processing steps were not included in the timing experiments. Neither method's pre-processing steps required significant enough time to alter the results. DPColorpath [5] and the spectral sparsification method were both run using 10 threads for parallelization. Turàn Shadow, which is not a parallel algorithm, was run using only a single thread. Metrics measured for each experiment include the computation time in seconds and accuracy.
Shown in table II, DPColorpath [5] performs the fastest of the three methods, with Turàn Shadow [4] being the second fastest and spectral sparsification as the slowest. This is because of the large amount of spanning trees that are required for the spectral sparsification as well as the necessity to create the edge ordering from eigenvectors which is essentially sorting the edges. However, when turning to accuracy, spectral sparsification performs better than Turàn Shadow [4] in all 4 graphs, and nearly as good or better than DPColorpath [5]. The drawback of DPColorpath [5] was that it does not perform well when dealing with larger values of k as shown with the poor accuracy on the $k = 70$ and $k = 115$ graphs. Inversely, Turàn Shadow [4] appears to see improved performance on the larger k values. However, even with the increased accuracy, spectral sparsification still performs significantly better on the larger k graphs. This suggests that in spectral sparsification offers an increase in accuracy at the expense of time complexity.

V. Conclusion

We have introduced a different approach to clique counting that utilizes edge sparsification based on spectrally generated spanning trees. Spectral sparsification has shown that there is the possibility of reducing the complexity of the graph while preserving existing cliques. This method out performs existing methods of clique finding in terms of accuracy, however does so at the cost of time performance in comparison to the current clique finding algorithms. While this opens the door for future work regarding the increased parallelism and use of GPU acceleration to improve the time complexity, the results suggest that spectral sparsification can become a useful tool when prioritizing accuracy over efficiency in clique finding operations.

References

[1] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." http://snap.stanford.edu/data, June 2014.
[2] M. Rahman, M. Bhuiyan, and M. Hasan, "Graft: An efficient graphlet counting method for large graph analysis," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, pp. 2466–2478, 10 2014.
[3] S. Jain and C. Seshadhri, "Provably and efficiently approximating near-cliques using the turán shadow: Peanuts," *Proceedings of The Web Conference 2020*, Apr 2020.
[4] S. Jain and C. Seshadhri, "A fast and provable method for estimating clique counts using turán's theorem," 2018.
[5] X. Ye, R.-H. Li, Q. Dai, H. Chen, and G. Wang, "Lightning fast and space efficient k-clique counting," in *Proceedings of the ACM Web Conference 2022*, WWW '22, (New York, NY, USA), p. 1191–1202, Association for Computing Machinery, 2022.
[6] M. Danisch, O. Balalau, and M. Sozio, "Listing k-cliques in sparse real-world graphs*," *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, p. 589–598, 2018.
[7] B. Rozemberczki, O. Kiss, and R. Sarkar, "Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, p. 3125–3132, ACM, 2020.
[8] K. Reba, M. Guid, K. Rozman, D. Janežič, and J. Konc, "Exact maximum clique algorithm for different graph types using machine learning," *Mathematics*, vol. 10, no. 1, 2022.
[9] F. Christodoulou, S. Nikoletseas, C. Raptopoulos, and P. Spirakis, "A spectral algorithm for finding maximum cliques in dense random intersection graphs," 2022.
[10] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM Journal on Computing*, vol. 14, p. 210–223, Feb 1985.

Graph	Max k Size	Graph Density	Total Number of Cliques
Wiki-Vote	17	0.003981	301214700
musae-Github	24	0.000407	190358157
c100k70	70	0.049993	3.99E+22
c200k115	115	0.02574	4.15E+36

TABLE I: Large graph metrics

Method	Graph	No. Threads	Computation time (s)	Accuracy
DPColorpath	Wiki-Vote	10	1.76	99.46
Turàn Shadow	Wiki-Vote	N/A	11.55	33.2
Spectral Sparsification	Wiki-Vote	10	147.8	99.92
DPColorpath	musae-Github	10	63.67	95.53
Turàn Shadow	musae-Github	N/A	8.94	45.86
Spectral Sparsification	musae-Github	10	12716.3	93.26
DPColorpath	c100k70	10	111.81	66.22
Turàn Shadow	c100k70	N/A	5548.97	81.09
Spectral Sparsification	c100k70	10	1460.3	99.99
DPColorpath	c200k115	10	276.31	31.88
Turàn Shadow	c200k115	N/A	21250.86	83.74
Spectral Sparsification	c200k115	10	46968.02	99.99

TABLE II: Timing and accuracy results for DPColorpath, Turàn Shadow and spectral sparsifcation

[11] P. Turán, "On an extremal problem in graph theory," *Matematikai és Fizikai Lapok*, vol. 48, pp. 436–452, 1941.

.