Expander Hierarchies for Normalized Cuts on Graphs

Kathrin Hanauer
kathrin.hanauer@univie.ac.at
University of Vienna
Faculty of Computer Science
Vienna, Austria

Monika Henzinger
monika.henzinger@ist.ac.at
Institute of Science and Technology
Austria (ISTA)
Klosterneuburg, Austria

Robin Münk
robin.muenk@tum.de
Technical University of Munich
Munich, Germany

Harald Räcke
raecke@in.tum.de
Technical University of Munich
Munich, Germany

Maximilian Vötsch
maximilian.voetsch@univie.ac.at
University of Vienna
Faculty of Computer Science
UniVie Doctoral School
Computer Science DoCS
Vienna, Austria

ABSTRACT

Expander decompositions of graphs have significantly advanced the understanding of many classical graph problems and led to numerous fundamental theoretical results. However, their adoption in practice has been hindered due to their inherent intricacies and large hidden factors in their asymptotic running times. Here, we introduce the first practically efficient algorithm for computing expander decompositions and their hierarchies and demonstrate its effectiveness and utility by incorporating it as the core component in a novel solver for the normalized cut graph clustering objective.

Our extensive experiments on a variety of large graphs show that our expander-based algorithm outperforms state-of-the-art solvers for normalized cut with respect to solution quality by a large margin on a variety of graph classes such as citation, e-mail, and social networks or web graphs while remaining competitive in running time.

CCS CONCEPTS

• **Information systems** → **Clustering**; • **Theory of computation** → **Sparsification and spanners**.

KEYWORDS

normalized cut, expander decomposition, expander hierarchy, graph partitioning, graph clustering

ACM Reference Format:
Kathrin Hanauer, Monika Henzinger, Robin Münk, Harald Räcke, and Maximilian Vötsch. 2024. Expander Hierarchies for Normalized Cuts on Graphs. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671978

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671978

1 INTRODUCTION

In recent years, expander decompositions and expander hierarchies have emerged as fundamental tools within the theory community for developing fast approximation algorithms and fast dynamic algorithms for such diverse problems as, e.g., routing [12], connectivity [11], (approximate) maximum flow [6, 18], or triangle enumeration [5]. Informally, an expander is a well-connected graph. Its *expansion* or *conductance* $\phi = \min_{S \subseteq V} \frac{\text{border}(S)}{\min(\text{vol}(S),\text{vol}(\bar{S}))}$ is the minimum ratio between the number of edges leaving any subset of vertices and the number of edges incident to vertices of the subset. A large value of ϕ (close to 1) indicates a graph in which no subset of vertices can be easily disconnected from the rest of the graph.

An *expander decomposition* of a graph partitions the vertices such that the subgraph induced by each part is an expander and the number of edges between different components of the partition is low.[1] Many results over the years [22, 23, 32, 34, 37] have demonstrated that such a decomposition not only exists for every graph, but can be computed in near-linear time. At a high level, the success of expander decomposition-based algorithms is due to the fact that many problems are "easy" on expanders: One first identifies regions of a graph where a problem is easy to solve (the expanders), solves the problem (or a suitable sub-problem) within each region, and then combines the individual solutions to obtain a solution for the overall problem.

Expander hierarchies [11] apply expander decompositions in a hierarchical manner. This is done by computing an expander decomposition of the graph and then contracting the individual expanders into single vertices repeatedly. This process is repeated until the entire graph has been contracted to a single vertex. This recursive decomposition has a corresponding decomposition tree T (the so-called *(flow) sparsifier*) where the root corresponds to the entire graph, the leaves correspond to vertices of the original graph, and inner vertices correspond to the expanders found during the decomposition procedure. The sparsifier approximately preserves the cut-flow structure of the original graph in a rigorous sense. This relationship has been exploited in numerous applications in static

[1] This can be seen as a special form of a graph clustering.

and dynamic graph problems [12, 16, 19] to obtain fast algorithms with provable guarantees.

While many algorithms for expander decomposition that offer strong asymptotic bounds on the running time have been suggested [22, 23, 32, 34, 37], [2] practical algorithms based on expander decomposition have not seen success thus far. This is due to these algorithms requiring one to solve many maximum flow problems, which means these algorithms are prohibitively slow in practice for graphs with many edges. Arvestad [1], e.g., reports that decomposing a graph with approximately 10^5 edges can take almost 5 min for various values of ϕ in their implementation of [32].

In practice, the *multilevel graph partitioning* framework has been the de-facto approach for computing high quality solutions for cut problems on graphs. This framework consists of a *coarsening* step, in which a smaller representation of the graph is computed, a *solving* step, where a solution is computed on the coarse graph and a *refinement* step, during which the solution is improved using local methods during the graph uncoarsening process. Multilevel graph partitioning has been successfully applied for computing balanced cuts [2, 17, 27, 31] and normalized cuts in graphs [10].

We note that algorithms based on the expander hierarchy approach may also be regarded as a variant of multilevel graph partitioning. In this approach, a graph is first coarsened by recursive contractions to obtain a smaller graph that reflects the same basic cluster structure as the input graph. Then an initial partition is computed on the sparsifier and, afterwards, in a series of refinement steps the solution is mapped to the input graph while improving it locally as we undo the coarsening. Despite the theoretical guarantees brought forward by expander decompositions and expander hierarchies, the latter have not yet led to similar breakthroughs in the field of fast, high-quality experimental algorithms. The reason is that the expander decomposition required at each coarsening step becomes a significant performance bottleneck in practice. In contrast, multilevel graph partitioners like METIS [17] and KaHiP [31] use a fast matching approach here.

In this work we mitigate the computational bottleneck by introducing a novel, practically efficient random-walk-based algorithm for expander decomposition. Based on this, we give the first implementation of the expander hierarchy and, thus, an algorithm to compute tree flow sparsifiers, allowing us to solve various cut problems on graphs effectively. Exemplarily, we show that our approach is eminently suitable to compute normalized k-cuts on graphs, where the goal is to partition the vertex set into k clusters such that the sum over the number of edges leaving each cluster, normalized by the cluster's volume, is minimized. Normalized cut is a popular graph clustering objective and particularly able to capture imbalanced clusterings. Its manifold applications include, e.g., community detection [20] and mining [4], topic reconstruction [3], story segmentation [40], bioinformatics [10, 38], tumor localization [30], and image segmentation [33, 36]. It is closely related to spectral clustering, which uses spectral properties of eigenvalues and eigenvectors of the graph's Laplacian. However, the spectral approach suffers both from very large running times and memory requirements to compute and store the eigenvectors, and thus does not scale well [10]. This problem was addressed by Dhillon et al. [10] as well as Zhao et al. [41], who presented algorithms for normalized cut that either use spectral methods only after coarsening [10] or apply them on a spectrally sparsified graph [41].

Contributions. We present the first practically efficient algorithm for expander decomposition and the first implementation to compute an expander hierarchy. Our approach is based on random walks and is justified by rigorous theoretic and empirical analysis. We report on a comprehensive experimental study on normalized cut solvers, comprising 50 medium-sized to very large graphs of various types, where we compare our expander-based algorithm, XCut, to Graclus by Dhillon et al. [10], the approach by Zhao et al. [41], as well as the state-of-the-art graph partitioners METIS and KaHiP, which do not specifically optimize towards the normalized cut objective, but are fast and in practice often used for this task. The experiments show that our algorithm produces superior normalized cuts on graph classes such as citation, e-mail, and social networks, web graphs, and generally scale-free graphs, and is only slightly worse on others. On average, it is still distinctly the best across all graphs and values of k.

If only a single value of k is desired, its running time is on average only 3 times slower than the runner-up, Graclus, and never exceeded 18 min. A notable advantage of XCut is that it can quickly compute solutions for multiple values of k once a sparsifier is computed, which can be faster than running Graclus multiple times.

2 RELATED WORK

Our work is motivated by recent theoretical results [5, 6, 11, 12, 18] building on expander decompositions [22, 23, 32, 34, 37] and the expander hierarchy [11] and inspired by non-spectral approaches [10] to tackle the normalized cut problem.

Mohar [21] showed that the computation of the so-called isoperimetric number or conductance of a graph (see section 3) is NP-hard, which implies the hardness of the normalized k-cut problem for $k \geq 2$. Normalized cut remains NP-hard on weighted trees [8].

A number of tools that have been used to solve normalized cut use spectral methods [7, 24–26, 28, 39]. This usually requires to compute k eigenvectors of the Laplacian matrix, which was shown to scale badly in practice [10, 41]. Afterwards, an additional discretization step is necessary to obtain the clustering.

Dhillon et al. [10] therefore suggest an algorithm called Graclus, which is based on the multilevel graph partitioning framework. It applies the same coarsening steps as METIS, but with a modified matching procedure. The coarsened graph can then be partitioned using different approaches, including a spectral one. In the refinement step, Graclus uses a kernel k-means-based local search algorithm for improving the normalized cut objective value. The authors evaluate their algorithm experimentally against METIS as well as a spectral clustering algorithm [39]. They show that it outperforms the spectral method w.r.t. normalized cut value, running time, and memory usage. It also produces better results than METIS and is comparable w.r.t. running time.

Zhao et al. [41] employ a joint spectral sparsification and coarsening scheme to produce a smaller representation of the graph that preserves the eigenvectors of the Laplacian in near-linear time $\tilde{O}(m)$. Afterwards, a normalized cut is computed on the sparsifier

[2] Including the near-linear-time algorithm by Saranurak and Wang [32], which finds a decomposition where each component has expansion at least ϕ in time $O(m\phi^{-1}\log^4 m)$.

using spectral clustering. Their sparsification scheme obtains a significant reduction in the number of edges and nodes, which makes it feasible to apply the spectral method on the reduced graphs, without the quadratic term in the running time growing too large. The authors again perform an experimental comparison with METIS and observe that their algorithm overall outperforms METIS w.r.t. the normalized cut value, while being slightly slower.

METIS [17], KaHiP [31], and many other graph partitioning tools [14, 35] do not solve the normalized cut problem and instead aim to find good solutions to a balanced graph partitioning problem, where the vertex set is to be partitioned into k sets of (roughly) equal size, while minimizing the number of edges cut. CHACO [14] implements a spectral graph partitioning approach, but the number of clusters is limited to at most $k = 8$.

Nie et al. [26] recently presented a spectral normalized cut solver based on the coordinate descent method along with several speedup strategies. They evaluate their algorithm against two other spectral methods [7, 25] on a number of medium-sized data sets and show that it consistently computes the best solution and is the fastest. A notable difference is that k is not an input parameter.

3 PRELIMINARIES

For an undirected graph $G = (V, E)$ we use d_v to denote the degree of vertex $v \in V$, $\vec{d}$ to denote the *degree vector*, i.e., the vector of vertex degrees, and $D = I\vec{d}$ to denote the corresponding *degree* matrix. Δ is used to denote the maximum degree of a vertex in G. For a subset S of vertices we define the *volume* $\mathrm{vol}(S)$ as the sum of vertex degrees of vertices in S. Its *border* $\mathrm{border}(S)$ (or *capacity*) is defined as $|E(S, \bar{S})|$, where $E(X, Y) = \{\{x, y\} \in E \mid x \in X, y \in Y\}$ is the set of edges between subsets $X, Y \subseteq V$, and $\bar{S} = V \setminus S$.

A *k-cut* is a partition $\mathcal{P}$ of the vertex set into k non-empty parts. Given such a partition $\mathcal{P} = (S_1, \ldots, S_k)$, its *normalized cut value* θ is defined as

$$\theta(\mathcal{P}) = \sum_{i=1}^{k} \frac{\mathrm{border}(S_i)}{\mathrm{vol}(S_i)} \quad .$$

For $k = 2$ we will usually specify a 2-cut (or just cut) by its smaller side, i.e., we will refer to the cut $(S, \bar{S})$ by just S, where $\mathrm{vol}(S) \leq \mathrm{vol}(\bar{S})$. The *conductance* of a (2-)cut is defined as $\Phi(S) = \mathrm{border}(S)/\mathrm{vol}(S)$ and the *conductance of a graph* $G = (V, E)$ is $\Phi(G) = \min_{\text{cuts } S} \Phi(S)$.[3] The problem of finding a 2-cut with minimum conductance is closely related to the problem of finding a 2-cut with minimum normalized cut value, because for any 2-cut, $\Phi(S) \leq \theta(S) \leq 2\Phi(S)$. The normalized k-cut objective can be seen as a generalization of conductance to k-cuts.

To properly describe our random walks we need some further notation. For a cut S we call $\mathrm{vol}(S)/\mathrm{vol}(V)$ the *balance* of the cut S and denote it with $b(S)$. For a subset $A \subseteq V$ we use $G\{A\}$ to denote the subgraph induced by A with self-loops added so that the vertex degrees do not change (a self-loop counts 1 to the degree of a vertex). This means the degree of a vertex $a \in A$ is the same in G as in $G\{A\}$. Whenever the graph in question is not clear from the context we use subscripts to indicate the graph, i.e., we write $\mathrm{vol}_G(S)$, $\Phi_G(S)$, $E_G(X, Y)$, etc. To avoid notational clutter we write $\mathrm{vol}_A(S)$, $\Phi_A(S)$, $E_A(X, Y)$ when we are referring to the graph $G\{A\}$.

[3] If there are no cuts in G (i.e., $|V| = 1$) we define its conductance to be 1.

We say a graph $G = (V, E)$ is a *ϕ-expander* if $\Phi(G) \geq \phi$, and we call a vertex partition $(V_1, \ldots, V_t)$ a *ϕ-expander decomposition* if $\Phi(G\{V_i\}) \geq \phi$ for all i.

4 EXPANDER DECOMPOSITION USING RANDOM WALKS

In this section we describe our random-walk-based approach for obtaining expander decompositions and present its theoretical guarantees. The complete proofs for these guarantees can be found in the full version of the paper.

A natural approach for computing expander decompositions is to find a low conductance cut to split the graph into two parts and then to recurse on both sides. If no such cut exists, we have certified the (sub-)graph to be an expander. In the end, the whole procedure terminates if each subgraph is an expander, i.e., it terminates with an expander decomposition. Saranurak and Wang [32] used this general approach to obtain an expander decomposition that runs in time $O(m \log^4 m/\phi)$ and only cuts $O(\phi m \log^3 m)$ edges. While their flow-based techniques give very good theoretical guarantees, the hidden constants do not seem to allow for good practical performance (see e.g. [1]).

In this work we base the cut procedure of the expander decomposition on random walks. As a consequence, we can only guarantee that our decomposition cuts at most $\tilde{O}(\sqrt{\phi}\, m)$ edges[4], since we are limited by the intrinsic Cheeger barrier of spectral methods. However, random walks have a very simple structure, which leads to a simple algorithm with good practical performance. The weaker dependency on ϕ ($\sqrt{\phi}$ instead of ϕ) is not crucial for our graph partitioning application because when using the expander decomposition to build an expander hierarchy one chooses ϕ as large as possible anyway.

Theorem 1 (Expander Decomposition). *Given a graph G with m edges and a parameter ϕ, there is a random-walk-based algorithm that with high probability finds a ϕ-expander decomposition of G and cuts at most $O(\sqrt{\phi}\, m \log^{5/2} m)$ edges. The running time is $O\big(\frac{m+n \log n}{\phi} \log^3 m\big)$.*

The main part of our algorithm is the cut procedure described in Section 4.1. This procedure is then plugged into the framework of Saranurak and Wang to find an expander decomposition. Our cut procedure gives the following guarantees.

Theorem 2 (Cut Procedure). *Given a graph $G = (V, E)$ with m edges and a parameter ϕ, the cut procedure takes $O\big((m + n \log n)/\phi\big)$ steps and terminates with one of these three cases:*

1. *We certify that G has conductance $\Phi(G) \geq \phi$.*
2. *We find a cut $(A, \bar{A})$ in G that has conductance at most $\Phi_G(A) \in O(\sqrt{\phi} \log^{3/2} m)$. Then one of the following holds:*
   (a) *either $\mathrm{vol}(A), \mathrm{vol}(\bar{A})$ are both $\Omega(m/\log^2 m)$, i.e., $(A, \bar{A})$ is a relatively balanced low conductance cut;*
   (b) *or $\mathrm{vol}(\bar{A}) \in O(m/\log^2 m)$ and A is a near 6ϕ-expander.*

Given the cut procedure, an expander decomposition is computed as follows. On the current subgraph G, execute the cut procedure, to either find a low conductance cut S or certify that none exists. If no

[4] $\tilde{O}$ hides polylogarithmic factors.

such cut exists, then G is a certified ϕ-expander and we terminate. Otherwise we check whether S is sufficiently balanced, i.e., the volume of the smaller side is at least $\Omega(m/\log^2 m)$. In that case we cut the edges across $(S, \bar{S})$ and recurse on both parts. As both parts are substantially smaller than G we obtain a low recursion depth.

Otherwise, S is very unbalanced but the larger side $\bar{S}$ is a so-called *near expander* – a concept introduced by Saranurak and Wang [32]:

Definition 1 (Near ϕ-expander). *Given a graph $G = (V, E)$. A subset $A \subseteq V$ is a near ϕ-expander in G if for all sets $X \subseteq A$ with* $\mathrm{vol}(X) \leq \mathrm{vol}(A)/2 : |E(X, V \setminus X)| \geq \phi \,\mathrm{vol}(X)$.

Note that if the LHS in the above equation were $|E(X, A \setminus X)|$ then $G\{A\}$ would be a ϕ-expander.

If $G\{\bar{S}\}$ (for $\bar{S}$ returned by the cut-procedure) is indeed a ϕ-expander we can just recurse on the smaller side S and would obtain a low recursion depth. Saranurak and Wang introduced a trimming procedure that, given a subset B that is a near 6ϕ-expander with volume $\mathrm{vol}(B) \geq 9\,\mathrm{vol}(V)/10$ and $|E(B, \bar{B})| \leq \phi\,\mathrm{vol}(B)/10$, computes a subset $B' \subseteq B$ that is a proper ϕ-expander and has volume $\mathrm{vol}(B') \geq \frac{1}{2}\,\mathrm{vol}(B)$. By applying this trimming step to $\bar{S}$ we can return $G\{\bar{S}'\}$ as a proper ϕ-expander and recurse on the remaining graph – still with a small recursion depth. Overall, using our cut procedure within this framework gives Theorem 1.

4.1 Finding Low Conductance Cuts

We now give a detailed description of the cut procedure that forms the basis of Theorem 2. The goal is to either certify that G is a ϕ-expander or to find a low conductance cut that is as balanced as possible. The idea is to exploit that random walks converge quickly on expanders and hence when they don't, we know there must be a low conductance cut. See Algorithm 1 for an outline.

We employ a concurrent random walk, where each node distributes its unique commodity in the graph. We are interested in the probability that after t steps a particle that started say at node i is at some other node j. The walk has converged if this distribution is essentially identical for *every* starting vertex. If we quickly reach this stationary distribution, there cannot be a low conductance cut and hence the graph must be an expander. Otherwise we can use information gathered from the walk to find a low conductance cut.

Such a cut may however be very unbalanced. The procedure therefore accumulates low conductance cuts until the combined cut is a balanced low conductance cut or the graph that remains does not have a low conductance cut anymore. Here we call a cut S *balanced* if its balance $b(S) \geq \beta := 2/\log^2 m$.

More precisely, the algorithm maintains a partition of V into two sets A, L for each iteration t with initial values $A := V, L := \emptyset$. We repeat the following for $T = 1/(12\phi)$ steps: In iteration t, we generate a new random unit vector r and execute $t-1$ steps of the random walk, initialized according to r. This walk yields a vector u, on which we analyze the conductance of all sweep cuts, i.e., cuts of the form $S_c := \{v \in A : u_v \leq c\}$ for some value c. Note that these conductance values can be calculated in linear time after sorting the entries of u.

We consider a cut to have low conductance if the value is below the threshold $\gamma = O(\sqrt{\phi}\log^{3/2} m)$. A lower threshold value γ would give better guarantees in case (2) of Theorem 2 but at the same time it would increase the number of rounds required to converge and hence worsen the guarantee of case (1) in Theorem 2. The value is chosen to ensure we can guarantee $\Phi(G) \geq \phi$ in case (1) of the Theorem.

Algorithm 1 Cut Procedure

Input: Graph $G = (V, E)$, Target expansion ϕ
Output: Expander or Balanced$(S, \bar{S})$ or Unbalanced$(S, \bar{S})$

$T \leftarrow 1/(12\phi)$
$\gamma \leftarrow 343\sqrt{\phi \log(32m^3)\log^2(n)}$ ▷ cut threshold
$\beta \leftarrow 2/\log^2 m$, ▷ balance
$W \leftarrow I$ ▷ random walk matrix
$A \leftarrow V, L \leftarrow \emptyset$
for $t = 1, \ldots, T$ **do**
  $\vec{r} \leftarrow$ *random unit vector in* $\mathbb{R}^n$
  $\vec{u} \leftarrow WD^{-1}r$ ▷ apply random walk
  $\vec{u} \leftarrow \vec{u} - (\vec{u}^\top \vec{d})/\mathrm{vol}(V) \cdot \mathbb{1}$ ▷ ensure $\vec{u} \perp \vec{d}$
  $\mathcal{A}, D_A \leftarrow$ *adjacency and degree matrix of* $G\{A\}$
  $W \leftarrow \left(\frac{1}{2}I + \frac{1}{2}\mathcal{A}D_A^{-1}\right)W$ ▷ extend walk matrix
  $S \leftarrow \mathrm{SweepCut}(\vec{u}, \gamma)$
  if $\mathrm{vol}(S) \geq \beta m$ **then return** Balanced$(S, \bar{S})$
  else if $S \neq \emptyset$ **then**
    $L \leftarrow L \cup S, A \leftarrow A \setminus S$
    if $\mathrm{vol}(L) \geq \beta m$ **then return** Balanced(A, L)
if $L = \emptyset$ **then return** Expander
else return Unbalanced(A, L)

The analysis of the sweep cuts yields one of these three cases:

(1) If all sweep cuts have conductance at least γ, we continue with the next iteration.
(2) If there exists a sweep cut with conductance $< \gamma$ and balance $\geq \beta$ we return this low conductance cut.
(3) Otherwise we consider the two-ended sweep cuts of the form $S_{a,b} := \{v \in A : u_v \notin [a, b]\}$ for values $a \leq b$ and find the one with largest volume among those with $\Phi_A(S_{a,b}) < \gamma$. If this cut has balance $\geq \beta$ we return it, otherwise we move it from A to L and check whether L has become balanced.

The random walk can be interpreted as the projection of a much higher dimensional random walk onto the randomly chosen direction r. This projection step is crucial for the implementation to become computationally feasible and we show that the projections approximate the original structure sufficiently well.

In order to argue the correctness of the cut procedure we have to show that it is highly unlikely that the procedure does not find a cut on a graph that has expansion less than ϕ. For this we argue that after T random walk steps (without finding a cut) the walk will have "converged" to its stationary distribution w.h.p. Because of the choice of parameters such a quick convergence is only possible if G is a near 6ϕ-expander. Whenever the cut procedure returns a cut, it is guaranteed to have conductance at most γ. From this it follows that the expander decomposition cuts at most $O(\gamma m \log m)$ edges. An outline of the proof can be found in Section A in the appendix, the entire argument is laid out in the full version of the paper.

5 XCut – A NEW NORMALIZED CUT ALGORITHM

In this section, we introduce the algorithm XCut, which is based on the previous section's novel random walk-based expander decomposition. We note the apparent similarity between multilevel graph partitioning and the expander hierarchy and use this as the basis of XCut. As an outline, we use the novel random walks to construct the expander hierarchy to obtain a coarse representation of the graph, the tree flow sparsifier. We then compute an initial solution on the tree. Finally, we use an iterative refinement step while descending the hierarchy to improve the solution we found. Compared to other contraction schemes, which lead to each vertex in the coarsest graph representing roughly the same number of nodes in the base graph, the subtrees on each level in the tree flow sparsifier can represent a vastly different number of vertices.

Expander Decomposition. While the expander decomposition outlined in section 4 is much simpler to implement than that of [32], as we do not rely on maximum flow computations at all, we made several choices in the implementation to speed up computation. We iterate a single random walk, and after each iteration, we check whether we can find a sparse cut. If we find a suitable cut, we disconnect the edges going across it, but contrary to the algorithm in section 4, we do not restart the random walk, as we find we can extract further information about the cut structure of the graph from the state of the random walk. For example, if the random walk has mixed very well on one of the new components, it is likely to be an expander, while if there is another sparse cut in the component, then the random walk will likely not have mixed well on the component. One may think that reducing the number of random walks might lead to a loss of guarantees and higher variance of the algorithm, but in experiments conducted while designing the algorithm, we found that on real graph instances running multiple concurrent random walks is not necessary. In fact, only a single graph in our 50 graph benchmark had noticeable variance. See also the discussion in subsection 6.3.

The main parameter of the expander decomposition is the cut value γ, which is the minimal sparsity of the cuts our algorithm makes. Additionally, we introduce a parameter ρ, to be used as a threshold for "certifying" that a component is an expander, as we found that choosing the threshold to be $1/(4\,\mathrm{vol}(V)^2)$ does not offer any benefits over a much larger value. See also subsection 6.2 for details on choosing the value for this parameter. We make a final modification to the theoretical algorithm in that we omit the trimming step on unbalanced cuts, since it does not provide any further speedup of the expander decomposition routine in practice.

Automatically Choosing γ. The theoretical analysis in [11] suggests a choice for γ and ϕ that is sufficient to prove the theoretical results. In preliminary experiments we found this choice to be too pessimistic and, in fact, by adapting ϕ and γ to the graph we can obtain better results. However, there is a trade-off to be made: If γ is too small, the expander decomposition will not find many sparse cuts, and we obtain sparsifiers of low quality. On the other hand, if γ is too large, most cuts will be sparser than γ, which leads to many cuts being made and increases the running time. In the worst case, this can even prevent the algorithm from terminating.

Thus, our goal is to choose γ such that it offers a good quality vs. time trade-off. When choosing γ, we can only observe whether this was a good choice in a post hoc fashion. A naive strategy would be to start with a large γ and decrease it until the expander hierarchy terminates within a reasonable amount of time. However, this approach is wasteful, as we discard previously computed decompositions, even if they were good. Instead, we decrease γ by multiplying it with constant factor $\epsilon < 1$ whenever the expander decomposition on a specific level cuts too many edges in G_i. We then use the new $\gamma' = \epsilon\gamma$ for the remaining expander decompositions, decreasing it further as required.

Solving on the Sparsifier. To solve normalized k-cut on the tree sparsifier obtained from the hierarchy, we want to remove $k-1$ edges to decompose it into a forest of k trees. Given a solution, i.e., a tuple of edges $(e_1, \ldots, e_{k-1})$, we assign vertex $v \in V$ to the cluster C_j associated with e_j if e_j is the first edge we encounter on the path from v to the root. If no edge in the solution lies on the path to the root, we assign v to cluster C_k. As normalized cut is NP-hard also on trees (see section 2), we introduce two heuristic approaches that take time $O(nk)$ each:

Greedy: The simple greedy heuristic picks the edge in the sparsifier which minimizes the increase in the normalized cut objective in every step. By simply computing the cost of cutting each remaining edge, it takes $O(n)$ time to find this edge, assuming the number of vertices in the sparsifier is $O(n)$. To partition a graph into k clusters, we repeat this process $k-1$ times. For $k = 2$, this algorithm produces the optimal solution on the tree as we pick the edge that minimizes the cut objective.

Dynamic Programming: Each row of the dynamic program corresponds to a level, and we make one cell for each pair (v, i) of the row, where v is a vertex in the sparsifier and $i \in [0, k]$. The value of each cell is the normalized cut value θ of decomposing the subtree rooted at v into i parts. We write $DP(v, i)$ for this value. Additionally, we write $cut(v, i)$ for the weight of cut edges in the solution incident to the subtree rooted at v, as well as $vol(v, i)$ for the volume remaining in the subtree.

Without loss of generality, assume the tree is binary, as otherwise we can binarize the tree by inserting edges of infinite cost (see Henzinger et al. [15] for details). The value of a cell is computed according to the following rule:

$$\mathrm{DP}(v, j) = \min(\ \mathrm{cutParent}(\mathrm{DP}(v, j-1)),\ \min_{0 \le i \le j} (\mathrm{DP}(v_l, i) + \mathrm{DP}(v_r, j-i)))),$$

where $\mathrm{cutParent}(\mathrm{DP}(v, j-1)) = \mathrm{DP}(v, j-1) + \frac{w(v,v') + \mathrm{cut}(v, j-1)}{\mathrm{vol}(v, j-1)}$ is the best solution where the edge going to the parent is cut. For each vertex v of G_0, we initialize the bottom row of the program with $\mathrm{DP}(v, 0) = 0$, $\mathrm{DP}(v, 1) = 1$ and $\mathrm{DP}(v, j) = \infty$ for $j \in [2, k]$. In the root vertex we use the special rule $\mathrm{DP}(v, j) = \min_i \mathrm{DP}(v_l, i) + \mathrm{DP}(v_r, j-i) + \frac{\mathrm{cut}(v_l, i) + \mathrm{cut}(v_r, j-i)}{\mathrm{vol}(v_l, i) + \mathrm{vol}(v_r, j-i)}$, where the last term ensures we do not produce a solution with $\mathrm{vol}(v, i) = 0$, leading to cluster j being empty.

The Refinement Step. Finally, we perform an iterative refinement as we descend the hierarchy. While descending, we introduce new clusters according to the edges in the solution. On each level we then perform vertex swaps that improve the normalized cut objective,

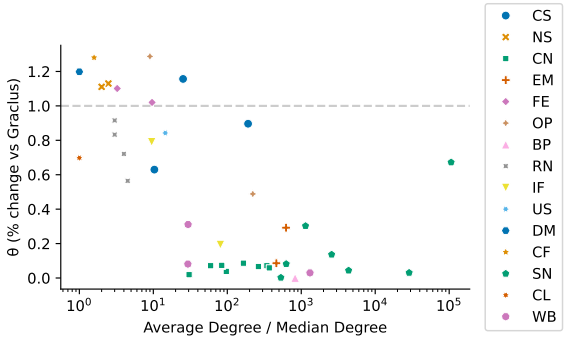

Figure 1: Relative improvement over Graclus (y-axis) vs. the ratio of the maximum degree and the median degree (x-axis). The value on the x-axis is larger if the graphs exhibit a distribution with large outliers. Note the negative trend, except for the outlier towards the right corresponding to instance SN7.

by either reducing the number of cut edges or making the partition more balanced. For details see the full version of the paper.

Support for Variable Number of Partitions k. A notable strength of our algorithm is that with both heuristics, it solves the problem for any value $k' \leq k$ during their execution. Suppose we are still determining the number of clusters needed. In that case, we can compute the solution for the maximum k we are interested in and obtain solutions for all smaller numbers of partitions during exploratory data analysis. The only step that needs to be rerun is the refinement step.

6 EXPERIMENTAL EVALUATION

In the previous sections, we have shown that our approach provides provable guarantees on the approximation ratio for the value θ of the normalized cut[5] (and its relatives, sparse cut, and low-conductance cut) if $k = 2$. We now turn to evaluate XCut experimentally in different configurations. We compare the objective value of normalized cuts produced by XCut and its running time against the normalized cut solver Graclus by Dhillon et al. [10] and the state-of-the-art graph partitioning packages METIS [17] and KaHiP (kaffpa) [31], all of which are available publicly. These algorithms are based on the multilevel graph partitioning framework and produce disjoint partitions of the vertices into k clusters, where k is a freely choosable parameter. We note that METIS and KaHiP solve the balanced k-partitioning problem rather than normalized cut. Nevertheless, we include these solvers in our comparison, as they are used for this task in practice and we found that they can outperform Graclus on some of the graphs in our benchmark dataset. By this, we follow the methodology of [10] and [41].

In addition, we compare our results to the values reported for the normalized cut algorithm by Zhao et al. [41][6]. We omit comparisons to solvers employing spectral methods such as the recent works by Chen et al. [7] and Nie et al. [26], as this approach does not scale well to large datasets of millions of nodes [10]. In preliminary experiments we found that the solver of Nie et al. [26] uses over 330GB of memory on instance CN3, whereas our solver used less than 400MB of memory. Furthermore, the algorithm presented in [26] does not necessarily produce k clusters, thus making direct comparisons difficult. Lastly, the space complexity of spectral methods becomes prohibitive for larger values of k, as noted by [10].

6.1 Experimental Setup

Instances. Our setup includes 50 graphs from various applications. See Table 1 for an overview and the full version of the paper for details. To facilitate comparability, our collection contains the eight instances used by Dhillon et al. [10] (BP1, CF1, CS1–3, DM1, OP1, and OP2) as well as the 21 instances used by Zhao et al. [41]. In addition, we selected 21 real-world networks that cover various application areas, including some of larger sizes. All instances are available publicly in the Network Repository [29] or the SuiteSparse Matrix collection [9].

Note that a graph with k or more connected components always has a (normalized) k-cut of size 0. Surprisingly, we found that XCut is the only solver tested here that finds the trivial optimal solution if k is less than the number of connected components. However, testing connectivity before starting a solver remedies this problem, which is why we decided to exclude graphs with more than 128 connected components except for one (graph ID 0), which we keep for consistency reasons as it was used in previous comparisons [10].

Methodology. As XCut, KaHiP, and METIS are randomized, we ran each of them $\ell = 10$ times per instance with different seeds. Graclus is deterministic, and we ran it three times to obtain a stable value for the running time. We use the arithmetic mean over the ℓ runs for each instance to approximate the expected value of θ and the running time. When reporting values, we write $\text{XCut}_{\text{mean}}$ for the mean value across these 10 runs, and XCut_{min} for the minimum. As KaHiP and METIS behaved almost identically over all ℓ runs, we only report mean values for them. For each algorithm and each graph, we compute a partitioning consisting of $k \in \{2, 4, 8, 16, 32, 64, 128\}$ clusters as well as θ (smaller is better). As a second criterion, we compare the algorithms' running times.

All experiments were conducted on a server with an Intel Xeon 16 Core Processor and 1.5 TB of RAM running Ubuntu 22.04 with Linux kernel 5.15. XCut is implemented in C++ and compiled using gcc 11.4 with full optimization[7]. For all other solvers, we followed the build instructions shipped with their code. As Graclus is single-threaded, we ran the single-threaded version of every algorithm. METIS was run in its default configuration. We used KaHiP with the `fsocial` flag, which is tailored to quickly partitioning social network-like graphs[8] and Graclus with options `-l 20 -b` to enable the local search step and only consider boundary vertices during local search, as suggested by the authors for larger graphs [10].

6.2 Configuring XCut

Greedy vs. Dynamic Programming (DP). Section 5 describes two heuristics for computing normalized cuts on the sparsifier. We

[5] See Section 3 for the precise definition.

[6] Unfortunately, the code is not available publicly and also could not be provided by the authors upon request before the submission deadline.

[7] `-O3 -march=native -mtune=native`

[8] We chose this setting as we are especially interested in social network-like graphs.

Table 1: Types and number of graphs in our benchmark dataset. Δ is the maximum degree, k and M are shorthand notations for 10^3 and 10^6, respectively. See the full version of the paper for a detailed list.

Type (Abbreviation)	#	$\|V\|$	$\|E\|$	Δ
Bipartite (BP)	1	1.4M	4.3M	1.7k
Computational Fluids (CF)	1	17k	1.4M	269
Clustering (CL)	2	4.8k-100k	6.8k-500k	3-17
Citation Network (CN)	9	226k-1.1M	814k-56M	238-1.1k
Circuit Simulation (CS)	3	5k-30k	9.4k-54k	31-573
Duplicate Materials (DM)	1	14k	477k	80
Email Network (EM)	2	33k-34k	54k-181k	623-1383
Finite Elements (FE)	2	78k-100k	453k-662k	39-125
Infrastructure Network (IF)	2	2.9k-49k	6.5k-16k	19-242
Numerical Simulation (NS)	2	11k-449k	75k-3.3M	28-37
Optimization (OP)	2	37k-62k	131k-2.1M	54-8.4k
Random Graph (RD)	1	14k	919k	293
Road Network (RN)	4	114k-6.7M	120k-7M	6-12
Social Network (SN)	7	404k-4M	713k-28M	626-106k
Triangle Mixture (TM)	7	10k-77k	54k-2M	22-18k
US Census Redistricting (US)	1	330k	789k	58
Web Graph (WB)	3	1.3k-1.9M	2.8k-4.5M	59-2.6k

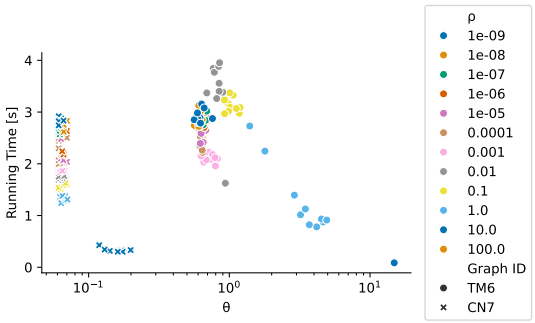

Figure 2: Running time vs. normalized cut for different choices of ρ on two different graphs for $k = 16$. Colors denote different levels of ρ, while shapes indicate the graph.

compare for both heuristics the running time and θ across ten precomputed sparsifiers each for 19 representative graphs. The quality returned by DP was never better than that of Greedy. While the running times for both heuristics are linear in the size of the sparsifier, the DP approach scaled worse in k. For example, for $k = 32$, DP was three times slower than Greedy, and seven times slower for $k = 128$. See the full version of the paper for a plot with the results for $\rho = 10^{-4}$. Greedy was faster and produced no worse quality than DP for all values of ρ. Thus, we only report results for Greedy for all further experiments.

Parameter Choice for the Expander Decomposition (ρ, γ). In Figure 2, we examine the effect of the threshold parameter ρ on the quality of solutions and running time. If the parameter is chosen too large, especially greater than one, the entire graph will likely be certified as an expander before any cuts are made, leading to very large θ. Furthermore, we no longer obtain any significant

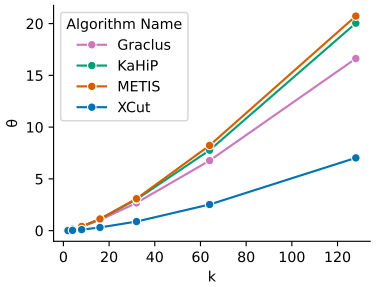

Figure 3: Geometric mean of the cut value θ across all graphs for each k for $\text{XCut}_{\text{mean}}$, Graclus, METIS, and KaHiP.

improvement in θ for values of ρ below 10^{-4}. At the same time, the running time increases inversely with ρ as extra iterations are needed to converge, which is shown by the vertical arrangement of the dots in Figure 2. For graph instance TM6, we also find that non-Pareto-optimal choices of ρ exist, namely 0.1 and 0.01, where both the running time and the returned value are worse than 10^{-3} and 10^{-4}. This suggests that the choice of ρ is an important design decision. On all our instances, $\rho = 10^{-4}$ produced Pareto-optimal results, so we conclude that we can configure this parameter to be a constant independently of the graph's structure.

For the automatic tuning of γ, we chose a starting threshold of 0.3, as this is around the largest value for which the expander decomposition finds structurally interesting cuts. Whenever γ is too large, i.e., the node reduction $|G_{i+1}|/|G_i| > 0.95$, we multiply γ by a factor of $\epsilon = 0.8$ and restart the expander decomposition.

6.3 Comparison to Graclus, METIS, and KaHiP

Figure 5 depicts the solution quality of every solver relative to that produced by Graclus on all instances for $k = 32$, showing both the mean and the minimum of the ten runs for XCut. For other values of k, the overall picture remains the same, see Figure 3 and Appendix B. We group the graphs by type, with Figure 5 containing two plots, the upper showing the disconnected IMDB graph and email, citation, and social network graphs as well as infrastructure networks, as these are the graphs on which XCut is particularly strong.

Looking at absolute values of θ, we find that across all instances, the geometric mean is at least 70 % lower than our competitors, see Table 2. Interestingly, when we only consider the seven graphs from [10], the geometric means become 0.84 for XCut_{min} and 0.90 for $\text{XCut}_{\text{mean}}$, while they are 1.11, 1.19 and 1.03 for Graclus, METIS, and KaHiP respectively, which implies that KaHiP slightly outperforms Graclus in terms of θ on their benchmark (but not XCut).

One point of note is that XCut does not always find small normalized cuts on the social network SN7 representing user-user interactions in the Foursquare social network. This appears to be due to a very high-degree node that connects to approximately 15 % of all vertices, leading to fast convergence of the random walk. Due to the averaging effect of such a vertex, many nodes v have almost identical values u_v that vary only slightly between runs. Thus, when sorting by u-value, their order can vary greatly depending on the initial values, leading to very different candidate cuts. This is the only graph where the minimum and mean of the ten

Table 2: Cut value (θ) of different algorithms. The geometric mean is taken across all graphs and values of $k \in \{2, 4, 8, 16, 32, 64, 128\}$ for each graph type. *For the overall geometric mean, instances (graph + k) with $\theta = 0$ were omitted. Only XCut detected such cases.

Type	$\text{XCut}_{\text{mean}}$	XCut_{min}	Graclus	METIS	KaHiP
BP	**0.00**	**0.00**	1.18	1.38	1.58
CF	4.61	4.18	3.44	3.51	**3.15**
CL	0.80	**0.72**	1.04	1.13	1.02
CN	**0.07**	**0.07**	1.42	1.72	1.58
CS	0.44	**0.41**	0.51	0.59	0.57
DM	2.58	2.42	**2.11**	2.12	2.23
EM	0.44	**0.43**	3.44	3.78	3.80
FE	0.56	**0.53**	0.54	0.54	0.55
IF	**0.00**	**0.00**	0.93	1.11	1.12
NS	0.85	0.78	0.77	**0.76**	0.80
OP	0.71	**0.66**	1.47	1.48	0.99
RD	13.37	13.37	12.08	12.01	**11.94**
RN	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**
SN	0.25	**0.19**	3.43	3.98	4.00
TM	2.04	**1.93**	2.87	3.29	3.04
US	0.04	**0.03**	0.06	0.06	0.05
WB	**0.01**	**0.01**	0.10	0.14	0.14
All*	0.25	**0.22**	0.89	1.0	0.94

runs of XCut differ significantly (−33 % for XCut_{min} and +39 % for $\text{XCut}_{\text{mean}}$ relative to Graclus). This indicates that the theoretical algorithm sketched in section 4, which uses multiple concurrent random walks (corresponding to more attempts to find a good cut), would likely have reduced the variance here. This is also the only graph where the fact that we only use a single random walk impairs the quality of the result.

The graph classes on which XCut does not perform as well have fairly homogeneous degree distributions and often appear grid-like when drawn. We conjecture that in these grid-like graphs, there are no good expanders (which XCut is trying to find). Instead, sparse cuts arise mainly from the fact that the cut is balanced, i.e., the components we disconnect are all large enough, rather than being sparsely interconnected, which is exploited by the other solvers.

6.4 Comparison to Zhao et al.

In Table B.3 we compare θ for XCut and Graclus to the values reported by Zhao et al. [41] for $k = 30$. We observe a similar but weaker pattern as in the previous section. On graphs arising from numerical simulation and finite element problems, XCut performs worse than Zhao et al., but never by more than 36 %. On the triangle mixture instances, XCut achieves better θ on four instances, while their solver outperforms XCut on two instances. On citation networks, clustering instances, and those based on maps (RN1 and US1), XCut outperforms the algorithm by Zhao et al., with θ being up to 2.5 times lower on US1 and CN7. Altogether, XCut is better than Zhao et al. on roughly 2/3 of the instances, while Graclus does best on one instance. The geometric mean across all instances is 1.46 for $\text{XCut}_{\text{mean}}$, 1.39 for XCut_{min}, 1.64 for Zhao et al., and 3.06 for Graclus. We note that ours is 11 % lower for $\text{XCut}_{\text{mean}}$ and 15 % lower for XCut_{min}, while Graclus's value is almost twice that of

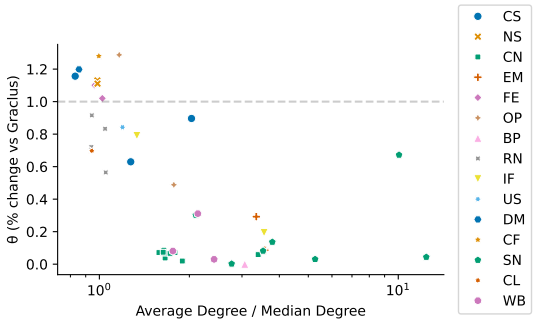

Figure 4: Relative improvement over Graclus (y-axis) vs. the average degree divided by the median degree (x-axis). Larger x-values signify that the graph exhibits a skewed, power-law-like degree distribution. Note the negative trend, except for the outlier towards the right corresponding to instance SN7.

the other solvers in the comparison due to it producing 5–30 times greater θ on the citation network (CN) instances.

While it is difficult to draw good conclusions for the running times from the values reported in [41] as no source code is available, we note that on some instances XCut takes less than 10 % of the reported time while producing higher-quality solutions, which might be indicative of a running time advantage of our solver.

6.5 Running Time

In our experiments, we found that on many graphs, the running time is spent mainly on computing the expander hierarchy, where on some instances, this step accounts for over 80% of the running time, even for $k = 128$. See Figure B.6 for some examples. However, even on the largest instances in our benchmark, the absolute running time never exceeded 18 minutes.

Overall, XCut is on average three times slower than Graclus, 20.8 times slower than METIS and 6.7 times slower than KaHiP for the mean execution time across all choices of k and all instances. Interestingly, we find that XCut's running time is lower than Graclus's on several social network graphs and the triangle mixture instances while it produces a better solution. See the full version of the paper for detailed running times on some exemplary instances.

Finally, recall that once our algorithm has computed a sparsifier, we can obtain a solution on the sparsifier for different values of k without recomputing the sparsifier, which is a unique feature among the solvers tested here. If we are interested in all seven values of k, e.g., and only count the time to compute the sparsifier once for XCut, our experiments take 0.56, 3.82, and 1.24 times the running time to compute partitions across all graphs and values of k using Graclus, METIS, and KaHiP, respectively. In particular, XCut then is 44 % *faster* than Graclus. This demonstrates the utility of XCut as a tool for exploratory data analysis. We could achieve further speedups by only computing a solution for $k = 128$ and then choosing the initial subsets of edges for the other values of k, only performing the refinement.

Figure 5: Percentage deviation of the returned normalized cut value relative to Graclus for $k = 32$. This means that a value of -75% indicates that the normalized cut value is 75% lower (i.e., better). The thin black bars indicate the standard error across our runs. The top graph shows the disconnected IMDB graph (BP1), citation network instances (CN), email networks (EM), infrastructure graphs (IF), social networks (SN), and web graphs (WB), while the bottom shows the remaining instances. See the full version of the paper for details.

6.6 Discussion

XCut outperforms other software when computing normalized cuts on social, citation, email, and infrastructure networks and web graphs. It performs slightly worse on graphs arising from specific computational tasks, such as finite elements, circuit, or numerical simulations. The graphs on which this behavior occurs tend to have degree distributions concentrated around the average degree, suggesting that they are not graphs with a scale-free structure.

In Figure 4 we plot the relationship between our improvement over Graclus and the value of $\frac{\frac{1}{n}\sum_{v\in V} d_v}{\text{median}_{v\in V} d_v}$ for the non-synthetic graphs of our benchmark. This value measures how much the mean and median diverge due to outlier nodes with very high degrees. The graphs with power-law distributions tend to have a higher value on this measure, and we find that there appears to be a negative correlation, with one outlier due to instance SN7.

7 CONCLUSION

In this work we introduced XCut, a new algorithm for solving the normalized cut problem. It is based on a novel expander decomposition algorithm and to the best of our knowledge, it is the first practical application of the expander hierarchy. XCut clearly outperforms other solvers in the experimental study on social, citation, email, and infrastructure networks and web graphs, and also in the geometric mean over all instances. It scales to instances with tens of millions of edges and can produce solutions for multiple numbers of clusters k with little overhead by comparison. We are confident that with further optimization and the use of parallelism it will be possible to scale our algorithm to even larger graphs, while further improving the solution quality, especially since computing the expander decomposition appears to be highly parallelizable.

We also believe that the expander hierarchy and our expander decomposition can be applied to other graph cut problems in the future, as the tree flow sparsifiers approximate *all* cuts in the graph, and there are theoretical results that suggest this might be the case. XCut is open source software and its code is freely available on GitLab [13].

ACKNOWLEDGMENTS

Monika Henzinger: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564) and the Austrian Science Fund (FWF) grant DOI 10.55776/Z422, grant DOI 10.55776/I5982, and grant DOI 10.55776/P33775 with additional funding from the netidee SCIENCE Stiftung, 2020–2024.

Harald Räcke, Robin Münk: This project has received funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 498605858 and 470029389.

REFERENCES

[1] Isaac Arvestad. 2022. Near-linear time expander decomposition in practice.

[2] Bas Fagginger Auer and Rob H Bisseling. 2012. Graph coarsening and clustering on the GPU. *Graph Partitioning and Graph Clustering* 588, 223 (2012), 2.

[3] Charles-Edmond Bichot. 2010. Co-clustering Documents and Words by Minimizing the Normalized Cut Objective Function. *J. Math. Model. Algorithms* 9, 2 (2010), 131–147. https://doi.org/10.1007/S10852-010-9126-0

[4] Deng Cai, Zheng Shao, Xiaofei He, Xifeng Yan, and Jiawei Han. 2005. Mining hidden community in heterogeneous social networks. In *Proceedings of the 3rd international workshop on Link discovery, LinkKDD 2005, Chicago, Illinois, USA, August 21-25, 2005*, Jafar Adibi, Marko Grobelnik, Dunja Mladenic, and Patrick Pantel (Eds.). ACM, 58–65. https://doi.org/10.1145/1134271.1134280

[5] Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. 2021. Near-optimal Distributed Triangle Enumeration via Expander Decompositions. *J. ACM* 68, 3 (2021), 21:1–21:36. https://doi.org/10.1145/3446330

[6] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. 2022. Maximum Flow and Minimum-Cost Flow in Almost-Linear Time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 612–623. https://doi.org/10.1109/FOCS54457.2022.00064

[7] Xiaojun Chen, Feiping Nie, Joshua Zhexue Huang, and Min Yang. 2017. Scalable Normalized Cut with Improved Spectral Rotation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, Carles Sierra (Ed.). ijcai.org, 1518–1524. https://doi.org/10.24963/IJCAI.2017/210

[8] Amir Daneshgar and Ramin Javadi. 2012. On the complexity of isoperimetric problems on trees. *Discret. Appl. Math.* 160, 1-2 (2012), 116–131. https://doi.org/10.1016/J.DAM.2011.08.015

[9] Timothy A Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1–25.

[10] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. 2007. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence* 29, 11 (2007), 1944–1957.

[11] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. 2021. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2212–2228.

[12] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. 2022. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 1325–1338.

[13] Kathrin Hanauer, Monika Henzinger, Robin Münk, Harald Räcke, and Maximilian Vötsch. 2024. XCut/XCut v1.0.0. https://doi.org/10.5281/zenodo.12108189. https://doi.org/10.5281/zenodo.12108189

[14] Bruce Hendrickson and Robert W. Leland. 1995. A Multi-Level Algorithm For Partitioning Graphs. In *Proceedings Supercomputing '95, San Diego, CA, USA, December 4-8, 1995*, Sidney Karin (Ed.). ACM, 28. https://doi.org/10.1145/224170.224228

[15] Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid. 2023. Dynamic Maintenance of Monotone Dynamic Programs and Applications. In *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany (LIPIcs, Vol. 254)*, Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 36:1–36:16. https://doi.org/10.4230/LIPICS.STACS.2023.36

[16] Yiding Hua, Rasmus Kyng, Maximilian Probst Gutenberg, and Zihang Wu. 2023. Maintaining expander decompositions via sparse cuts. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 48–69.

[17] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

[18] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. 2014. An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, Chandra Chekuri (Ed.). SIAM, 217–226. https://doi.org/10.1137/1.9781611973402.16

[19] Jason Li. 2021. Deterministic mincut in almost-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 384–395.

[20] Jia Li, Honglei Zhang, Zhichao Han, Yu Rong, Hong Cheng, and Junzhou Huang. 2020. Adversarial Attack on Community Detection by Hiding Individuals. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 917–927. https://doi.org/10.1145/3366423.3380171

[21] Bojan Mohar. 1989. Isoperimetric numbers of graphs. *J. Comb. Theory, Ser. B* 47, 3 (1989), 274–291. https://doi.org/10.1016/0095-8956(89)90029-4

[22] Danupon Nanongkai and Thatchaphol Saranurak. 2017. Dynamic spanning forest with worst-case update time: adaptive, las vegas, and o (n1/2-ε)-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 1122–1129.

[23] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. 2017. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 950–961.

[24] Maxim Naumov and Timothy Moon. 2016. Parallel spectral graph partitioning. *NVIDIA, Santa Clara, CA, USA, Tech. Rep., NVR-2016-001* (2016).

[25] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 2001. On Spectral Clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani (Eds.). MIT Press, 849–856. https://proceedings.neurips.cc/paper/2001/hash/801272ee79cfde7fa5960571fee36b9b-Abstract.html

[26] Feiping Nie, Jitao Lu, Danyang Wu, Rong Wang, and Xuelong Li. 2024. A Novel Normalized-Cut Solver With Nearest Neighbor Hierarchical Initialization. *IEEE Trans. Pattern Anal. Mach. Intell.* 46, 1 (2024), 659–666. https://doi.org/10.1109/TPAMI.2023.3279394

[27] Vitaly Osipov and Peter Sanders. 2010. n-Level graph partitioning. In *Algorithms–ESA 2010: 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I 18*. Springer, 278–289.

[28] Richard Peng, He Sun, and Luca Zanetti. 2017. Partitioning Well-Clustered Graphs: Spectral Clustering Works! *SIAM J. Comput.* 46, 2 (2017), 710–743. https://doi.org/10.1137/15M1047209

[29] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. https://networkrepository.com

[30] Tapasmini Sahoo and Kunal Kumar Das. 2023. Brain Tumor Localization Using N-Cut. *International Journal of Online & Biomedical Engineering* 19, 15 (2023).

[31] Peter Sanders and Christian Schulz. 2012. High quality graph partitioning. *Graph Partitioning and Graph Clustering* 588, 1 (2012), 1–17.

[32] Thatchaphol Saranurak and Di Wang. 2019. Expander Decomposition and Pruning: Faster, Stronger, and Simpler. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 2616–2635. https://doi.org/10.1137/1.9781611975482.162

[33] Jianbo Shi and Jitendra Malik. 2000. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 8 (2000), 888–905. https://doi.org/10.1109/34.868688

[34] Daniel A Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. 81–90.

[35] Chris Walshaw, Mark Cross, and Martin G. Everett. 1995. A Localized Algorithm for Optimizing Unstructured Mesh Partitions. *Int. J. High Perform. Comput. Appl.* 9, 4 (1995), 280–295. https://doi.org/10.1177/109434209500900403

[36] Yangtao Wang, Xi Shen, Yuan Yuan, Yuming Du, Maomao Li, Shell Xu Hu, James L. Crowley, and Dominique Vaufreydaz. 2023. TokenCut: Segmenting Objects in Images and Videos With Self-Supervised Transformer and Normalized Cut. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 12 (2023), 15790–15801. https://doi.org/10.1109/TPAMI.2023.3305122

[37] Christian Wulff-Nilsen. 2017. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 1130–1143.

[38] Eric P. Xing and Richard M. Karp. 2001. CLIFF: clustering of high-dimensional microarray data via iterative feature filtering using normalized cuts. In *Proceedings of the Ninth International Conference on Intelligent Systems for Molecular Biology, July 21-25, 2001, Copenhagen, Denmark*. 306–315.

[39] Stella X. Yu and Jianbo Shi. 2003. Multiclass Spectral Clustering. In *9th IEEE International Conference on Computer Vision (ICCV 2003), 14-17 October 2003, Nice, France*. IEEE Computer Society, 313–319. https://doi.org/10.1109/ICCV.2003.1238361

[40] Jin Zhang, Lei Xie, Wei Feng, and Yanning Zhang. 2009. A Subword Normalized Cut Approach to Automatic Story Segmentation of Chinese Broadcast News. In *Information Retrieval Technology, 5th Asia Information Retrieval Symposium, AIRS 2009, Sapporo, Japan, October 21-23, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5839)*, Gary Geunbae Lee, Dawei Song, Chin-Yew Lin, Akiko N. Aizawa, Kazuko Kuriyama, Masaharu Yoshioka, and Tetsuya Sakai (Eds.). Springer, 136–148. https://doi.org/10.1007/978-3-642-04769-5_12

[41] Zhiqiang Zhao, Yongyu Wang, and Zhuo Feng. 2018. Nearly-linear time spectral graph reduction for scalable graph partitioning and data visualization. *arXiv preprint arXiv:1812.08942* (2018).

A THEORETICAL ANALYSIS OF THE EXPANDER DECOMPOSITION

We now give an outline of the proof of correctness of our cut procedure which culminates in Theorem 2 and forms the basis of our expander decomposition algorithm. The full argument and detailed proofs of each lemma can be found in the full version.

The key ingredient for the analysis is to show that it is very unlikely that the cut procedure does not return a cut within T iterations when started on a graph with conductance less than ϕ. Consequently, if the algorithm does not return a cut for T iterations we can declare G to be a ϕ-expander (or actually $G\{A\}$ to be a near 6ϕ-expander), with a small probability of error.

We analyze the random walks in terms of flows. Each node v injects d_v units of flow of a unique commodity. This flow is distributed according to the random walk. Let $F_{ij}(t)$ denote the amount of flow from node j that has reached node i after t steps. We define $P_{ij}(t) \coloneqq \frac{F_{ij}(t)}{d_i d_j}$ for all t. Whenever clear from context, we may omit the explicit indication of the round t.

Let $A(t) \subseteq V$ be the set of nodes in the subgraph in round t. We define a natural average vector $\mu(t) \coloneqq \frac{1}{\mathrm{vol}(A(t))} \sum_{i \in A(t)} d_i P_i(t)$ and track the convergence of our random walk to this stationary distribution with a potential function:

$$\varphi(t) \coloneqq \sum_{i \in A(t)} d_i \cdot \|P_i(t) - \mu(t)\|^2 \quad .$$

Intuitively, a low potential indicates that the $P(t)$ vectors have mixed well in the set $A(t)$. The following lemma shows that a low potential implies that $A(t)$ is a near 6ϕ-expander in the graph G. $G\{A(t)\}$ may not be a proper expander because the random walk may have used edges outside of $G\{A(t)\}$.

Lemma 1. *If $\varphi(t) \leq \frac{1}{4\,\mathrm{vol}(V)^2}$ in any step $t \leq T$, then $A(t)$ is a near 6ϕ-expander in G.*

The intuition for this lemma is that the convergence of the random walk (i.e., low potential) implies the existence of a low-congestion all-to-all multicommodity flow which in turn implies good expansion of the graph.

Next, we show that with constant probability during one iteration we either return a balanced low conductance cut or the potential decreases significantly. For this we first have to analyze by how much the potential decreases during a random walk step.

Potential Decrease by Random Walk. Fix a round t. Note that the random walk for round t only takes the $t-1$ random walk steps for the graphs $G\{A(1)\}, \ldots, G\{A(t-1)\}$, the step for $G\{A(t)\}$ follows in the next round. In the following we develop an expression for how much the potential will decrease due to the random walk step in the current iteration t. Let $\delta(t) \coloneqq (\varphi(t) - \varphi(t+1))/\varphi(t)$ be the relative factor by which the potential of round t decreases after the random walk step in graph $G\{A(t)\}$.

Lemma 2. *The relative potential decrease due to the random walk step in iteration t is at least*

$$\delta(t) \geq \frac{1}{2} \frac{\sum_{\{i,j\} \in E(A(t))} \|P_i(t) - P_j(t)\|_2^2}{\sum_{i \in A(t)} d_i \|P_i(t) - \mu(t)\|_2^2} \quad .$$

Projected Potential Decrease. The algorithm does not maintain the P_i vectors or the potential explicitly, as that would be computationally infeasible. Rather it operates on their projections onto some random vector r. The entries in the vector u of the algorithm are actually $u_i = (P_i - \mu)^\top r$. This follows since performing the random walk and then projecting the resulting P_i vectors onto r is equivalent to first projecting onto r and then running the random walk on the vector of projections. Ultimately, subtracting the weighted average of the u values corresponds to subtracting μ from the P_i vectors before the projection. Consider the quotient

$$R(u) \coloneqq \frac{\sum_{\{i,j\} \in E(A(t))} (u_i - u_j)^2}{\sum_{i \in A(t)} d_i u_i^2} \quad ,$$

and compare it to the bound $\delta(t)$ for the relative potential decrease in Lemma 2. Up to a factor of $1/2$, $R(u)$ is obtained by individually performing a random projection on each vector in the expression for $\delta(t)$. Using properties of random projections, we will show that $R(u)$ is a good indicator for the potential decrease $\delta(t)$.

Lemma 3 (Projection Lemma). *Let $v_1, \ldots, v_n$ be a collection of d-dimensional vectors. Let r denote a random d-dimensional vector with each coordinate sampled independently from a Gaussian distribution $\mathcal{N}(0, 1/d)$. Then*

(1) $\Pr[\exists i, j : (v_i - v_j)^\top r \geq 11 \log n \cdot \|v_i - v_j\|^2/d] \leq 1/n$
(2) $\Pr[\sum_i (v_i^\top r)^2 \geq \frac{1}{20 \log n} \sum_i \|v_i\|^2/d] \geq 1/2$ *for $n \geq 8$.*

We call an iteration *good* if the vector of projections u retains sufficient information of the higher dimensional P_i vectors. Formally, we require the following.

Definition 2. *An iteration t of the algorithm is* good *if*

- $\sum_{i \in A(t)} d_i u_i^2 \geq \frac{1}{20\, n \log n} \cdot \sum_i d_i \|P_i - \mu\|^2$ *and*
- $(u_i - u_j)^2 \leq \frac{11 \log n}{n} \left\|P_i - P_j\right\|^2 \quad \forall \{i,j\} \in E(A(t))$.

From Property 1 and Property 2 of the Projection Lemma we can directly infer the following.

Claim. *An iteration is good with probability at least $1/4$ for $n \geq 4$.*

The next lemma shows that in a good round, a large $R(u)$ value implies a large $\delta(t)$ value, i.e., a large potential decrease.

Lemma 4. *In a good round t, we have $R(u) \leq 440 \log^2(n) \cdot \delta(t)$.*

For the further analysis we always assume that we are in a good round.

Finding a Cut. With the above lemma we have established that a large $R(u)$ value ensures that the next step of the random walk reduces the potential a lot. Now we show that a small $R(u)$ value ensures that the algorithm finds a low conductance cut.

Indeed the quotient $R(u)$ (the Rayleigh quotient of u using the graph Laplacian), and its analysis lies at the heart of the proof for the celebrated Cheeger inequality. It states that a small $R(u)$ value implies the existence of a low conductance sweep cut.

Lemma 5 (Cheeger). *For any $x \in \mathbb{R}^n$, with $x \perp \vec{d}$ there is a value c, s.t. there is a non-trivial set $S_c = \{i \in V : x_i \leq c\}$ with conductance $\Phi_G(S_c) \leq \sqrt{2R(x)}$.*

Note that $u \perp \vec{d}$ holds by design of the algorithm due to Line 10 of the algorithm. With this Cheeger Lemma we thus get the guarantee that there has to be a sweep cut through u whose conductance Φ satisfies $\Phi \leq \sqrt{2R(u)}$. Together with Lemma 4 this now implies the following: If we will only make little progress in the potential with the next random walk step, i.e., $\delta(t) \leq \alpha$ for some small α, then we find a sweep cut through the vector of projections u with conductance $\Phi \leq O(\sqrt{\alpha} \log n)$. By contraposition, if all sweep cuts have conductance at least Φ, then $\delta(t) \geq \Omega(\Phi^2/\log^2 n)$.

This motivates our approach of analyzing the sweep cuts on u and differentiating three cases: 1) No low conductance sweep cut exists, 2) A balanced low conductance cut exists, or 3) An unbalanced low conductance cut exists.

If we find a balanced sweep cut, we return immediately as this ensures the recursion depth of the surrounding expander decomposition remains small. If no sweep cut has conductance below γ, the above reasoning ensures we make sufficient progress with the next random walk step. Next we prove that even if we find an unbalanced cut, the potential still decreases sufficiently.

Handling Unbalanced Cuts. Let (S, B) be an unbalanced cut, where $\mathrm{vol}(S) \leq 2\log^2(\mathrm{vol}(A))/\mathrm{vol}(A)$ and there is no low conductance sweep cut through B in u. We show that if $R(u)$ is small and we thus cannot ensure sufficient progress from the random walk, then the set S must contain a large fraction of the potential. Hence we instead make progress by removing S when we set $A(t+1) = B$.

First we observe that the $R(u)$ value cannot increase too much by removing a small cut. In fact, with the above bound on the volume of the smaller side S, we can show that $R(u)$ increases by at most a factor 2. This leads to the following lemma, where φ_B denotes the potential generated by summing only over the nodes in B.

Lemma 6. *If $R(u) < \gamma^2/4$ and we find an unbalanced cut (S, B) with $\mathrm{vol}(S) \leq 2m/\log^2 m$, then $\varphi_B(t) \leq \left(1 - 1/(880\log^2 n)\right) \cdot \varphi(t)$.*

The above results show that a good round ensures sufficient progress in decreasing the potential or produces a balanced cut.

Claim. *In a good round we either find a balanced cut, or the potential reduces by a factor of at least $1 - \gamma^2/(1760\log^2 n)$.*

Proof. Let Φ be the minimum value of any sweep cut on u. If $\Phi \geq \gamma$, then Lemmas 4 and 5 give $\delta \geq \gamma^2/(880\log^2 n)$ since

$$\gamma^2 \leq \Phi^2 \leq 2R(u) \leq 880\log^2(n) \cdot \delta \ .$$

If $\Phi < \gamma$, we will find some low conductance cut S. If S is balanced, we can return it, otherwise we move S from A to L. Then either

- $R(u) \geq \gamma^2/4$ and Lemmas 4 and 5 give $\delta \geq \gamma^2/(1760\log^2 n)$, or
- $R(u) < \gamma^2/4$ and Lemma 6 gives a factor of $1 - 1/(880\log^2 n)$.

In any case, the decrease is at least $1 - \gamma^2/(1760\log^2 n)$. □

Iterations. An iteration takes $O(m)$ time for the random walk and $O(n \log n)$ for sorting the u values. It only remains to prove the upper bound on the number of iterations. The threshold for the potential is chosen to guarantee the existence of a near 6ϕ-expander in Lemma 1.

Lemma 7. *After $T = 1/12\phi$, iterations of the cut procedure, with high probability $\varphi(T) \leq \frac{1}{4\,\mathrm{vol}(V)^2}$.*

Altogether, the above arguments imply the correctness of our Theorem 2. Analogous to the reasoning by Saranurak and Wang [32], we can conclude that our modified guarantees yield Theorem 1 when using our cut step procedure in their expander decomposition framework.

B ADDITIONAL FIGURES

Table B.3: Cut values and running time of XCut vs the values reported by Zhao et al [41]. All values are for $k = 30$. The Cut-columns contain the normalized cut value (lower is better). The minimum value in each row is marked bold. The bottom row contains the geometric mean of all values.

GID	$\mathrm{XCut}_{\mathrm{mean}}$	$\mathrm{XCut}_{\mathrm{min}}$	Zhao	Graclus
CL1	0.87	**0.77**	1.05	1.15
CL2	6.00	**5.87**	7.05	7.61
CN1	0.31	**0.29**	0.52	4.06
CN3	**0.27**	**0.27**	0.49	3.66
CN5	**0.13**	**0.13**	0.14	3.24
CN7	0.17	**0.16**	0.41	2.21
CN8	**0.04**	**0.04**	0.06	1.88
FE1	2.06	1.96	**1.68**	1.74
FE2	1.58	1.55	1.50	**1.45**
NS1	5.73	5.35	**4.71**	**4.71**
NS2	1.45	1.39	**1.08**	1.17
RD1	28.98	28.98	**23.80**	26.48
RN4	**0.06**	**0.06**	0.07	0.07
TM1	9.53	8.91	**6.85**	8.48
TM2	12.34	**12.06**	13.55	16.63
TM3	2.78	**2.65**	2.72	2.78
TM4	10.09	**9.58**	10.48	14.45
TM5	7.77	**7.48**	7.88	13.25
TM6	2.11	**1.94**	2.09	2.1
TM7	17.51	16.90	**12.83**	15.6
US1	0.17	**0.16**	0.41	0.19
All	1.46	**1.39**	1.64	3.06

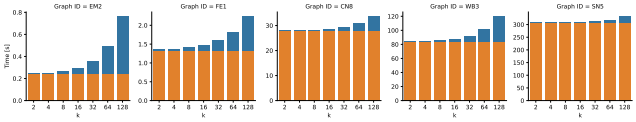

Figure B.6: Plot showcasing the time taken to compute the expander hierarchy in orange and the total time to compute a normalized cut in blue over different values of k for five graph instances of different sizes.