

Neighborhood-Preserving Graph Sparsification

Supplementary Material

Abd Errahmane KIOUCHE

Univ Lyon, Université Lyon 1, LIRIS
UMR CNRS 5205
Lyon, France
abd-errahmane.kiouche@univ-lyon1.fr

Julien BASTE

Univ. Lille, CNRS, Centrale Lille, UMR
9189 - CRISAL -
Lille, France
julien.baste@univ-lille.fr

Mohammed HADDAD

Univ Lyon, Université Lyon 1, LIRIS
UMR CNRS 5205
Lyon, France
mohammed.haddad@univ-lyon1.fr

Hamida SEBA

Univ Lyon, Université Lyon 1, LIRIS
UMR CNRS 5205
Lyon, France
hamida.seba@univ-lyon1.fr

Angela BONIFATI

Univ Lyon, Université Lyon 1, LIRIS
UMR CNRS 5205
Lyon, France
angela.bonifati@univ-lyon1.fr

1 LINK TO CODE AND DATA

The source code, data, and/or other artifacts have been made available at <https://gitlab.liris.cnrs.fr/coregraphie/ptspar>

2 PROOF OF THEOREM 2

To prove Theorem 2, we show here that a k -spanner is a particular case of (p, t) -sparsification. We first recall the definition of a k -spanner.

DEFINITION 1. A k -spanner S of a graph G , where $k > 1$, is a subgraph of G that has fewer edges than G and maintains the property that the shortest-path distance between every pair of vertices in S is at most k times the shortest-path distance between those vertices in the original graph G .

THEOREM 1. Given a graph $G = (V, E)$ and a sparsification function p_s where $p_s(t) = 1$ for some $t > 1$ and $p(i) > 0$ for all $i < t$, a k -spanner of G is a $(p = p_s, t = k)$ -sparsification of G .

PROOF. This theorem asserts that a k -spanner, which ensures that the shortest path between any two nodes in the sparsified graph is at most k times the shortest path in the input graph, is a specific instance of (p, t) -sparsification. The sparsification function p here is defined such that full neighborhood information is preserved at the k -hop distance (since $p(t) = p_s(k) = 1$), and some positive amount of information is preserved at distances less than t (since $p(i) > 0$ for $i < t$). \square

3 PROOF OF THEOREM 3

Let $\pi : E \rightarrow \{1, \dots, |E|\}$ be any bijective function (i.e., permutation) that associates to each edge index i its processing rank $\pi(i)$ by Algorithm *ptSpar*. We denote by E^π any permutation of the set of edges E (i.e., $E^\pi = (e_{\pi(1)}, \dots, e_{\pi(i)}, \dots, e_{\pi(|E|)})$). Algorithm *ptSpar* constructs a simpler graph G_s from the original graph G **incrementally** by processing all edges in a certain ordering E^π . For simplicity reasons, we will use $G'(k) = (V, E'(k))$ to refer to the constructed graph G' at the k -th iteration containing the k first edges of E^π (i.e., $G' = (V' = V, E'(k) = \{e_{\pi(1)}, \dots, e_{\pi(k)}\})$ and $G_s(k) = (V_s = V, E_s(k))$ the sparsified graph after the k -th iteration. Note that $G'(E)$ at the final iteration (i.e., $G'(|E|)$) is nothing else than the original graph $G = (V, E)$ and $G_s(|E|) = G_s$ the output of Algorithm *ptSpar*. Let $e_{\pi(k)}$ be the edge to be tested (to be inserted or dropped) at the k -th iteration, the algorithm checks whether the deletion of $e_{\pi(k)}$ will violates the (p, t) -sparsification constraints or not (lines 8-14), if that is the case, the edge $e_{\pi(k)}$ must be inserted into G_s , otherwise it will be dropped (lines 16-17). The (p, t) -sparsification constraints verification consists in computing the proportion of preserved neighbors at each depth $i \leq t$ in $G_s(k)$ compared to the neighborhood set of the graph constructed so far $G'(k)$ (lines 8-15).

To prove that the output G_s of Algorithm *ptSpar* is a (p, t) -sparsification of the original graph G , we need to prove that for each $0 \leq k \leq |E|$, $G_s(k)$ is a (p, t) -sparsification of $G'(k)$. To do so, we proceed by **induction**.

Base case: $G_s(0) = (V, \emptyset)$ is a (p, t) -sparsification of $G'(0) = (V, \emptyset)$. This statement is true, since all neighborhood set are empty because there is no edge in the graph $G'(0)$.

Induction step: We assume that $G_s(k)$ is a (p, t) -sparsification of $G_s(k)$ and let's prove that $G_s(k+1)$ is a (p, t) -sparsification of $G'(k+1)$.

Let us proceed by **contradiction**. Suppose that $G_s(k+1)$ is not a (p, t) -sparsification of $G'(k+1)$ and let $e_{\pi(k+1)} = (u, v)$ be the edge to be processed at the $(k+1)$ -th iteration. Notice that $G'(k+1) = (V', E'(k+1)) = (V, E'(k) \cup \{e_{\pi(k+1)} = (u, v)\})$. Knowing that the arrival of the edge $e_{\pi(k+1)} = (u, v)$ impacts only the neighborhood sets $N_{G'(k+1)}^1(v)$ and $N_{G'(k+1)}^1(u)$, we can deduce the following statements:

- $N_{G'(k+1)}^1(u) = N_{G'(k)}^1(u) \cup \{v\}$
and
 $|N_{G'(k+1)}^1(u)| = |N_{G'(k)}^1(u)| + 1$ (s1)
- $N_{G'(k+1)}^1(v) = N_{G'(k)}^1(v) \cup \{u\}$
and
 $|N_{G'(k+1)}^1(v)| = |N_{G'(k)}^1(v)| + 1$ (s2)
- $\forall s \in V - \{u, v\} : N_{G'(k+1)}^1(s) = N_{G'(k)}^1(s)$
and
 $|N_{G'(k+1)}^1(s)| = |N_{G'(k)}^1(s)|$ (s3)

On the other hand, $G_s(k+1)$ is not a (p, t) -sparsification of $G'(k+1)$ signifies that $\exists i \leq t \exists s \in V, |N_{G'(k+1)}^i(s) \cap N_{G_s(k+1)}^i(s)| < |N_{G'(k+1)}^i(s)| p(i)$ (s4). We distinguish two possible cases:

1. $s \in V - \{u, v\}$: by applying (s3) and (s4), we deduce $\exists i \leq t \exists s \in V, |N_{G'(k)}^i(s) \cap N_{G_s(k+1)}^i(s)| < |N_{G'(k)}^i(s)| p(i)$. Since $N_{G_s(k)}^i(s) \subseteq N_{G_s(k+1)}^i(s)$ we deduce that:

$\exists i \leq t \exists s \in V, |N_{G'(k)}^i(s) \cap N_{G_s(k)}^i(s)| < |N_{G'(k)}^i(s)| p(i)$ (s5).

Therefore, $G_s(k)$ is not (p, t) -sparsification of $G'(k)$, which odds the induction assumption.

2. $s \in \{u, v\}$: without loss of generality we suppose $s = u$ (the proof remains correct if we consider $s = v$), by applying (s1) ((s2) if $s = v$) and (s4), we deduce $\exists i \leq t$, $\left| N_{G'(k)}^1(u) \cap N_{G_s(k+1)}^i(u) \right| + \left| \{v\} \cap N_{G_s(k+1)}^i(u) \right| < \left(\left| N_{G'(k)}^1(u) \right| + 1 \right) p(i)$ (s6). Since $N_{G_s(k)}^i(u) \subseteq N_{G_s(k+1)}^i(u)$, we deduce that $\exists i \leq t$, $\left| N_{G'(k)}^1(u) \cap N_{G_s(k)}^i(u) \right| + \left| \{v\} \cap N_{G_s(k+1)}^i(u) \right| < \left(\left| N_{G'(k)}^1(u) \right| + 1 \right) p(i)$ (s7). Note that the node u does not satisfy the (p, t) -sparsification constraints (i.e., $s = u$), therefore the edge $e_{\pi(k+1)} = (u, v)$ must be inserted into $G_s(k+1)$ (line 11). In this case, $N_{G_s(k+1)}^i(u)$ must contains v since (u, v) was inserted, that indicates that $\left| \{v\} \cap N_{G_s(k+1)}^i(u) \right| = 1$ (s8). By applying (s7) and (s8) and knowing that $p(i) \leq 1$, we deduce $\exists i \leq t$ $\left| N_{G'(k)}^1(u) \cap N_{G_s(k)}^i(u) \right| < \left| N_{G'(k)}^1(u) \right| p(i)$ (s5). **Therefore, $G_s(k)$ is not (p, t) -sparsification of $G'(k)$, which odds the induction assumption.**

Therefore, we deduce that if $G_s(k)$ is a (p, t) -sparsification of $G_s(k)$ then $G_s(k+1)$ is a (p, t) -sparsification of $G'(k+1)$ \square .

4 PROOF OF THEOREM 4

At each iteration, Algorithm *ptSpar* processes an edge in the given ordering. We first prove that if at the k -th iteration the current output $G_s(k) = (V_s = V, E_s(k))$ is a (p, t) -sparsification of the original graph $G = (V, E)$, then all edge that will be processed in the following iterations will be rejected, i.e., not included in the output subgraph $G_s = G_s(|E|)$. This means that if $G_s(k)$ is (p, t) -sparsification of G then $G_s(k) = G_s(k+1) = \dots = G_s(|E|) = G_s$. The proof of this claim is quite straightforward; if at the k -th iteration the current output is a $G_s(k)$ is a (p, t) -sparsification of G then we have $\forall x \in N - \{0\} : x \leq t, \forall v \in V \left| N_G^1(v) \cap N_{G_s}^x(v) \right| \geq \left| N_G^1(v) \right| p(x)$ (1).

Thus, let $e = (u, v)$ be an edge of $E - E_s(k)$. When Algorithm *ptSpar* considers e , it will checks if vertices u and v have all their neighborhood constraints satisfied. By property (1), all vertices already have all their neighborhood constraints satisfied, hence edge e will be rejected.

Now, let $G_s^* = (V, E_s^*)$ be a (p, t) -sparsification of minimum size of the graph G . Consider a permutation function π^* such that the processing rank of any edge of E_s^* is lower than the processing rank of any edge of $E - E_s^*$. If Algorithm *ptSpar* processes the edges of E in the order defined by π^* , by the previous claim, once all edges of E_s^* are treated, all following iterations will reject the remaining edges \square .

5 COMPLEXITY ANALYSIS

THEOREM 2. *The average time complexity of the *ptSpar* algorithm is $O(|E|d^t)$ where d is the average degree in the graph G . In the worst-case scenario, where $d = |V|$, this complexity rises to $O(|E||V|^t)$.*

PROOF. If we consider that Algorithm *ptSpar* utilizes BFS (Breath First Search) to traverse the graph up to a depth of t to compute the set of neighbors. The average complexity depends on the average degree of the graph d . In each level of the BFS, the potential number of nodes to explore grows exponentially based on d . Therefore, for a depth t , the traversal of nodes (and consequently the operations count) is proportional to d^t , yielding an average time complexity of $O(|E|d^t)$, where $|E|$ denotes the number of edges in the graph.

In a complete graph scenario, every vertex is connected to every other vertex, effectively making the average degree d approximate the total number of vertices $|V|$. This situation represents the worst-case complexity, as the BFS would potentially need to explore nearly all vertices at each level up to t , culminating in a complexity of $O(|E||V|^t)$. \square

THEOREM 3. *The time complexity of computing the LP-based Edge Ordering in a graph $G = (V, E)$ is $O(\text{poly}(|E| + |V|d^{t-1}))$, where $|E|$ is the number of edges, $|V|$ is the number of vertices, d is the average degree, and t is the parameter of the sparsification.*

PROOF. The LP-based Edge Ordering algorithm involves formulating and solving a linear programming (LP) problem. The significant factors contributing to its complexity are the number of variables and constraints in the LP formulation.

The number of edge variables, x_e , is equal to the number of edges, $|E|$. Moreover, the algorithm considers path variables, which are influenced by the number of paths of length less than or equal to $t - 1$ from each vertex. The average number of such paths in a graph with an average degree d is $O(d^{t-1})$. As each of the $|V|$ vertices can be the start point of these paths, the total count of path variables is on the order of $|V|d^{t-1}$.

Therefore, the LP formulation has $O(|E| + |V|d^{t-1})$ variables. As solving a linear programming problem is polynomial in the number of variables, the time complexity of the LP-based Edge Ordering algorithm is $O(\text{poly}(|E| + |V|d^{t-1}))$. \square

THEOREM 4. *The average time complexity of algorithm 3 is $O(|E|(d^t + \log |E|))$.*

PROOF. Consider the following points:

- The average number of paths of length $\leq t$ between two connected nodes u and v , starting from u , is of order $O(d^t)$, where d is the average degree of the graph.
- The number of edges in the graph is denoted by $|E|$.

Hence, the time complexity of computing all the scores $s(e)$ for each edge is $O(|E|d^t)$. Furthermore, since sorting all these scores requires $O(|E| \log |E|)$ time, the overall time complexity of the algorithm is $O(|E|(d^t + \log |E|))$. In the worst-case scenario, such as in a complete graph, this complexity escalates to $O(|E|(|V|^t + \log |E|))$. \square

6 DISCUSSION ABOUT THE APPROXIMATION RATIO

While the presented algorithms are efficient in practice, from a theoretical point of view, these algorithms do not provide a constant factor approximation. We argue here for the ordering based on the linear programming.

For each integer $k \geq 2$, we construct the graph G_k as follows. We start from a clique K of $k + 1$ vertices v_0, \dots, v_k . For each $0 \leq i < j \leq k$, we add 2 vertices ($a_{i,j}^1$ and $a_{i,j}^2$), each of them having exactly 2 neighbors, v_i and v_j . Finally, we add d vertices b^1, \dots, b^d , each of them having exactly k neighbors, the vertices v_1, \dots, v_k . Note that the vertices $a_{i,j}^z$, $0 \leq i < j \leq k$, $z \in \{1, 2\}$, and b^z , $z \in \{1, \dots, d\}$, form an independent set.

Let discuss about the form of an optimal (p, t) -sparsifiers of G_k where $p(1) = 0\%$ and $t = 2$. First for each $0 \leq i < j \leq k$, with regard to the neighbors of $a_{i,j}^1$ and $a_{i,j}^2$, one can easily check that an optimal solution should contains the edge $\{v_i, v_j\}$ together with one edge incident to $a_{i,j}^1$ and one edge incident to $a_{i,j}^2$. This allows to satisfy the neighborhood constraint of $a_{i,j}^1$ and $a_{i,j}^2$ with 3 edges where any other solution needs strictly more edges. We obtain that all the edges in the clique K should be in any optimal solution. Adding exactly one edge incident to b^i for each $i \in \{1, \dots, d\}$ provides an optimal (p, t) -sparsifiers that contains $m_{opt} = 2 \cdot \binom{k+1}{2} + d = d + k \cdot (k + 1)$ edges.

Let discuss about what can append with the use of the LP ordering when applied to G_k . Lets assume that the LP provides a weight 1 for each edge of the clique K , the edges $\{a_{i,j}^1, v_i\}$, $0 \leq i < j \leq k$, and the edges $\{b^z, v_1\}$, $z \in \{1, \dots, d\}$, corresponding to the edges of an optimal solution and 0 to all other edges. Let assume that the edges of the set $E_0 = \{\{v_0, v_j\} \mid 0 < j \leq k\}$ come first in the created ordering, that is a possibility, followed directly by the other edges, named E_1 , of K . Then when applying Algorithm 1, we obtain that the edges of E_0 will be added to the solution and the other edge $\{v_i, v_j\}$, $0 < i < j \leq k$, will not as the condition of line 11 of the algorithm will already be satisfied. Because of this, when we will deal with the edges incident to b^z , $z \in \{1, \dots, d\}$, we will have to add each of them as none of the edges of E_1 has been selected. This creates a solution of size at least $m_{LP} = d \cdot k + k$. Using $d = k \cdot (k + 1)$, we obtain that $m_{LP} > \frac{k}{2} \cdot m_{opt}$. Thus no constant factor approximation can be obtained here.

7 TABLES 5 WITH ALL DATASETS

Because of space limitation, we did not include the ENZYMES and CA-HEPTH datasets in Table 5. In the following, we provide this table with all the datasets.

Table 5: Effect of the sparsification on the entropy loss

	COLLAB	IMDB BINARY	MSRC_ 21C	PROTEINS	PUBMED	CITSEER	CA-HEPTH	CORA	FLICKR JOURNAL	LIVE ASTROPH	CA- CATALOG	BLOG- TO	ENZYMES	FRIENDSTER 2015	GSH- TO	Average
ptSpar	1.00%	1.50%	0.60%	1.50%	0.71%	0.65%	0.62%	0.40%	0.26%	0.78%	0.90%	1.67%	1.58%	0.71%	0.69%	0.85%
SLB	20.80%	22.70%	4.40%	4.10%	0.29%	0.77%	4.28%	0.51%	0.48%	OT	0.94%	TO	21.8%	TO	TO	-
AD	11.80%	3.70%	1.50%	4.90%	0.52%	0.27%	1.43%	0.14%	0.07%	2.70%	2.89%	0.10%	4%	0.62%	0.78%	2.59%
LS	7.70%	6.40%	0.60%	1.30%	0.07%	0.17%	0.63%	0.21%	0.24%	1.09%	1.56%	1.28%	6.7%	0.48%	0.61%	2.03%
QSB	2.90%	1.20%	1.70%	5.30%	0.60%	1.00%	1.75%	1.12%	0.07%	0.31%	0.75%	1.60%	1.5%	0.60%	0.84%	1.61%
SB	1.70%	1.10%	1.80%	4.40%	0.63%	1.08%	1.77%	1.18%	0.07%	0.64%	0.98%	4.45%	1.45%	0.63%	0.92%	1.60%
EFF	7.40%	6.60%	2.90%	3.80%	0.78%	1.87%	3.07%	1.48%	0.47%	2.48%	3.15%	3.76%	8.3%	0.78%	0.99%	3.23%
LD	21.30%	19.40%	6.40%	4.80%	0.57%	0.56%	6.57%	1.30%	0.44%	0.46%	2.84%	6.74%	17.93%	1.26%	1.13%	5.94%
RE	6.80%	5.90%	2.70%	2.50%	0.88%	1.50%	2.66%	1.58%	0.72%	1.73%	1.85%	7.26%	5.87%	0.68%	0.79%	2.99%