State-of-the-art review

Hugo Abreu[†], Fanny Terrier[†] and Hugo Thomas[†]

[†]Master in Quantum Information, Sorbonne Université

June 3, 2022

Abstract

This report presents a review of a paper written by Apers and de Wolf [AW19] in which a new quantum algorithm for graph sparsification relying on nearly linear classical algorithms is introduced, leading to quantum speedups for several problems such as extremal cuts and Laplacian solving.

1 Introduction

«Graphs are nice [...], but sparse graphs are nicer.»

Graphs are a very common data structure in many areas of computer science, such as optimization and networks. Many practical problems can indeed be reduced to graph problems, and as such are of interest to computer scientists. Recent works, such as that by Chen *et al.*, yielded a near-linear time classical algorithm for the exact maximum-cost flow problem [Che+22]. It is nevertheless possible to get an even better speedup by considering approximate algorithms. The paper contribution is the creation of a quantum algorithm for ε-spectral sparsification of graphs in time $\tilde{O}(\frac{\sqrt{nm}}{\varepsilon})$, proving by the way the lower bound of their algorithm. Taking into account the algorithm of Chen *et al.*, it results in an algorithm generalizable to most graph problems.

1.1 Graphs

Let $G = (V, E, \omega)$ be a weighted graph, where V is a set of vertices, E a set of edges, and $\omega : V \times V \to \mathbb{R}$ a weight function, with $|V| = n$ and $|E| = m \leq \binom{n}{2}$. G is said to be undirected if for all $i, j \in V$ then $(i, j) \in E$ implies $(j, i) \in E$.

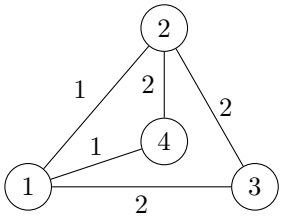

Figure 1: Example of a weighted graph

The access to the graph are done via the *adjacency list*.

1.2 Graph Laplacian

Let $G = (V, E, \omega)$ be a weighted graph, the Laplacian of G is an $n \times n$ matrix defined as

$$L_G = D - A \tag{1}$$

where D is the degree matrix and A is the adjacency matrix, defined such that $(D)_{ii} = \sum_j \omega(i, j)$ and $(A)_{ij} = \omega(i, j)$. The graph shown in Figure 1 has the following adjacency and degree matrices

$$A = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 2 & 2 \\ 2 & 2 & 0 & 0 \\ 1 & 2 & 0 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix},$$

which yield the Laplacian

$$L = \begin{pmatrix} 4 & -1 & -2 & -1 \\ -1 & 5 & -2 & -2 \\ -2 & -2 & 4 & 0 \\ -1 & -2 & 0 & 3 \end{pmatrix}.$$

An interesting property of the Laplacian that arises from those definitions is that L_G is invertible if and only if the graph G is connected. Equivalently, the Laplacian of a graph can be expressed in terms of a weighted sum of its edges Laplacian:

$$L_Q = \sum_{(i,j)\in E} \omega(i, j) L_{(i,j)} \tag{2}$$

where $L_{(i,j)}$ denotes the Laplacian of the edge (i, j), defined as

$$L_{(i,j)} = (\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^T \tag{3}$$

$\boldsymbol{e}_i$ is a unit vector with a 1 in coordinate i and zeros everywhere else. One can remark that $L_{(i,j)}$ is a very sparse matrix, with only 4 nonzero entries.

The Laplacian is a positive semidefinite matrix i.e. the eigenvalues of the Laplacian are non-negative.

The pseudo inverse of a Laplacian L denoted L^+, is such that $LL^+L = L$ and $L^+LL^+ = L^+$.

1.3 Quadratic forms of a Laplacian

The quadratic form of a Laplacian has a number of nice properties, and can be used to calculate quantities associated to the graph. All quadratic forms of a Laplacian can be expressed, by linearity of the sum, in terms of a weighted sum of its edges Laplacian:

$$\begin{aligned} \chi^T L_G \chi &= \sum_{(i,j)\in E} \omega(i, j) \chi^T L_{(i,j)} \chi \\ &= \sum_{(i,j)\in E} \omega(i, j)(\chi(i) - \chi(j))^2 \end{aligned} \tag{4}$$

An interesting example, showing how quadratic forms underlie graphs properties, is that if χ_s is an indicator vector on $S \subseteq V$, the quadratic form $\chi_s L_G \chi_s$ is equal to the value of the cut (S, S^c).

1.4 Spectral Sparsification

Spectral sparsification of graphs aims to reduce the number of edges, while keeping an approximation of interesting quantities i.e., approximately preserving all quadratic forms.

Definition 1.1 (ε-sparsifier)**.** H is an ε-sparsifier of G if and only if $\forall \chi \in \mathbb{R}^n, \chi^T L_H \chi = (1 \pm \varepsilon)\chi^T L_G \chi$.

Using the pseudo-inverse of the Laplacian, this definition can be equivalently formulated as $\chi L_H^+ = (1 \pm O(\varepsilon))\chi L_G^+ \chi$. It is also possible to define an ε-spectral sparsifier taking into account the positive semidefinite property of the Laplacian, such that $(1-\varepsilon)L_G \preccurlyeq L_h \preccurlyeq (1+\varepsilon)L_G$, where $\preccurlyeq$ denotes the partial ordering on symmetric matrices. The three above definitions are equivalent and one should use one or the other depending on the context.

Theorem 1.2 (Graph Sparsifier)**.** *Every graph G has an ε-spectral sparsifier H with a number of edges in $\tilde{O}(\frac{n}{\varepsilon^2})$. Moreover, H can be found in time $\tilde{O}(m)$.*

One should note that this is relevant only when $\varepsilon \leq \sqrt{\frac{n}{m}}$

The existence of such ε-spectral sparsifier was proved by Spielman and Teng [ST11]. Additional work of Batson, Spielman and Strivastava [BSS12] reduced the lower bound on the number of edges in the sparsifier to $O(\frac{n}{\varepsilon^2})$.

2 Classical Sparsification Algorithm

The classical algorithm for graph sparsification is based on edge sampling, where each edge is added to the sparsifier according to a fixed probability distribution.

In order to be sure L_H effectively approximates L_G, the choice of each p_e cannot be done at random. A nearly-linear classical sparsification algorithm was introduced by Spielman and Srivastava [SS11], by approximating effective resistance between any two edges in the graph efficiently, and thus introducing a way to correctly sampling the edges.

2.1 Effective resistance

In the case of unweighted graphs, the effective resistance of an edge can be related to the connectivity of the graph: edges belonging to strong components have a low effective resistance, and vertex cut (whose removal renders G disconnected) tends to have a high effective resistance.

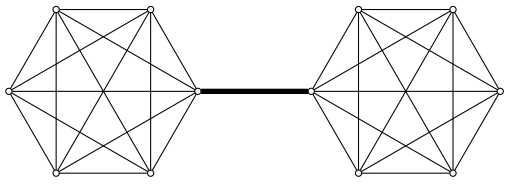

Figure 2: Graph illustrating edge of high effective resistance.

The effective resistance of the bold edge is roughly $r_e = 1$, while the effective resistance of the other is $r_e \in O(\frac{1}{n})$. Spielman and Srivastava associated then the probability for an edge to be kept while constructing the sparsifier proportionally to $r_e \omega_e$.

In a graph $G = (V, E)$ the effective resistance r_e of an edge $e = (u, v)$, with u, v two nodes, can be expressed with the quadratic form

$$R_{u,v} = (\chi_u - \chi_v)^T L_G^+ (\chi_u - \chi_v) \ , \tag{5}$$

where χ_i is the i^{th} vector of the canonical basis. [SS11]

2.2 Graph spanner

The *distance* between two nodes u and v with respect to G is defined as:

$$\delta_G(u, v) = \min_{u \to v} \sum_i \omega(p_i, p_{i+1})^{-1},$$

which is consistent with the previous definition of effective resistance of an edge, where $\{u \to v\}$ is the set of all paths from u to v in G, each element $u \to v = \{p_0, \cdots, p_k\}$ is a set of vertices of V. A spanner H of a graph G is a subgraph of G with fewer edges, where a trade-off is made between the number of edges and the stretching of distances.

Definition 2.1. An (α, β)-spanner of the graph $G = (V, E)$ is a subgraph $H = (V, E_H)$ with $E_H \subseteq E$, such that $\forall u, v \in V$,

$$\delta_G(u, v) \leq \delta_H(u, v) \leq \alpha \delta_G(u, v) + \beta.$$

This definition holds for weighted graphs, in which case the weight of the kept edges stay unchanged. In the following only multiplicative spanners are considered, i.e. $\beta = 0$ and $\alpha = 2 \log n$, namely, $2 \log n$-spanners. Furthermore, key objects of the algorithm for graph ε-spectral sparsification described below are r-packings spanners.

Definition 2.2. Let G be a graph, an r-packings spanner of G is an ordered set $H = (H_1, \cdots, H_r)$ of r edge-disjoint subgraphs of G such that H_i is a spanner for the graph $G - \bigcup_{j=1}^{i-1} H_j$.

Koutis and Xu proposed the following algorithm [KX16], using the effective resistance of each edge as exhibited by Spielman and Srivastava to construct t-packings spanner of the input graph:

Algorithm 1 ClassicallySparsify(G, ε)

1: *construct an $O(\frac{\log n}{\varepsilon^2})$-packing spanner H of G*
2: $\tilde{G} \leftarrow H$
3: **for all** $e \notin H$ **do**
4: *w.p. $\frac{1}{4}$ add e to $\tilde{G}$, with weight $4\omega_e$*
5: **return** $\tilde{G}$

and provided the following theorem:

Theorem 2.3 (Classical sparsifier)**.** *The output $\tilde{G}$ of **ClassicallySparsify** on inputs G and ε satisfies with probability $1 - \frac{1}{n^2}$*

$$(1-\varepsilon)L_G \preccurlyeq L_{\tilde{G}} \preccurlyeq (1+\varepsilon)L_G$$

Moreover, the expected number of edges in $\tilde{G}$ is at most $\tilde{O}(\frac{n}{\varepsilon^2} + \frac{m}{2})$.

The proof of Theorem 2.3 [KX16] ensures that the output of **ClassicallySparsify** is an ε-spectral sparsifier. A single iteration of the above procedure divides the number of edges in the output graph by roughly two. Hence, repeating $t \in O(\log \frac{m}{n})$ times **ClassicallySparsify**(G, ε') with $\varepsilon' \in O(\frac{\varepsilon}{t})$ results in an ε-spectral sparsifier with $\tilde{O}(\frac{n}{\varepsilon^2})$ edges.

Complexity-wise, the execution time of the provided algorithm is mostly dominated by the construction of the $\tilde{O}(\frac{1}{\varepsilon^2})$ spanners, each of which requires time $\tilde{O}(m)$, giving a total time complexity of $\tilde{O}(\frac{m}{\varepsilon^2})$.

3 Quantum speed-up for graph sparsification

Apers and de Wolf propose a quantum analog to the sparsification algorithm described in Section 2. They build on results from classical and quantum algorithms, in particular the classical algorithm for sparsification by Koutis and Xu (Algorithm 1), the spanner algorithm by Thorup and Zwick [TZ05], the quantum algorithm for single-source shortest-path trees by Dürr et al. [Dü+06], and an efficient k-independent hash function by Christiani, Pagh, and Thorup [CPT15].

3.1 Quantum spanner algorithm

The quantum spanner algorithm proposed by Apers and Wolf is heavily inspired by the best classical introduced by Thorup and Zwick [TZ05]: as such, the classical algorithm will be introduced before the quantum one.

3.1.1 Classical spanner algorithm

In order to efficiently construct a graph spanner, Thorup and Zwick [TZ05] designed a classical algorithm based on shortest-path trees.

Definition 3.1 (shortest-path tree)**.** Inside a graph $G = (V, E)$, a shortest path tree $\mathcal{T}$, rooted at a node v_0 and covering a subset $S \subseteq V$ of vertices, is a subgraph of G such that for all nodes $v_S \in S$, the distance between v_0 and v_S is the same as in the original graph G, and is minimal in G.

Their algorithm constructs a $(2k-1)$-spanner H of G with $O(kn^{1+1/k})$ edges, for some $k \in \mathbb{N}$. To do so, a family $\{A_0, \dots, A_k\}$ of node subsets is generated at random such that $A_0 = V$, $A_k = \emptyset$ and for all $i < k$,

$$A_i = \{v \in A_{i-1} \ w.p.\ n^{-1/k}\} \ , \tag{6}$$

i.e., A_i contains each edge of the previous subset with probability $n^{-1/k}$. At each iteration $i \leq k$, for all nodes $v \notin A_i \cap A_{i-1}$, a shortest path tree $\mathcal{T}(v)$ spanning the ensemble of nodes V' is built from node v. V' is defined such that for all $v' \in V'$, the distance between v and v' is smaller than the distance between v' and all the nodes that belongs to A_i. The resulting spanner is the union of all the shortest path trees created thereby.

3.1.2 Quantum spanner algorithm

The runtime of Thorup and Zwick's algorithm is dominated by the construction of the shortest path trees. A quantum algorithm speeding up this construction exists [Dü+06], and is strongly inspired by Dijkstra's algorithm. In the latter, a tree $\mathcal{T}$ rooted at a node v_0 is recursively grown by adding the cheapest border[1] edge, i.e the edge (i, j) such that

$$cost(i,j) = \min\{cost(u,v) | u \in \mathcal{T}, v \notin \mathcal{T}\} \ ,$$

where $cost(i,j) = \delta(v_0, i) + \frac{1}{w(i,j)}$. The quantum time improvement arises from a speedup for the selection of the cheapest border edge. The quantum routine called in the quantum shortest path tree algorithm is the *minimum finding* quantum algorithm MINFIND(d, f, g), which takes as inputs

- a *value* function $f : [N] \to \mathbb{R} \cup \{\infty\}$
- a *type* function $g : [N] \to \mathbb{N}$
- an integer $d \leq \dfrac{N}{2}$

and outputs a subset $I \subseteq [N]$ of size $|I| = \min\{d, M\}$, where $M = |Im(g)|$, such that every distinct elements of I have a different type, i.e. for all $i, j \in I$

$$g(i) \neq g(j) \ ,$$

and for $j \notin I$ and $i \in I$, having $f(j) < f(i)$ implies that there exists an $i' \in I$ so that

$$f(i') \leq f(i) \text{ and } g(i') = g(j) \ ,$$

i.e., j and i' have the same type.

Let P_L be a subset of nodes and $E(P_L)$ the set of edges such that $\forall (u,v) \in E(P_L)$, $u \in P_L$ or $v \in P_L$. In Algorithm 2, the functions f and g are both defined on $E(P_L)$, in such a way that $g((u,v)) = v$ and

$$f((u,v)) = \begin{cases} cost(u,v) = dist(u) + \frac{1}{w(u,v)} & \text{if } u \in P_L,\ v \notin T \\ \infty & \text{otherwise.} \end{cases}$$

[1] A border edge (i,j) of $\mathcal{T}$ is so that $i \in \mathcal{T}$ and $j \notin \mathcal{T}$.

In other words, we are looking for a subset of the border edges of the set of nodes P_L that contains at most one edge for each node in P_L, and if several edges are possible, the least costly is kept. A brief explanation follows.

Algorithm 2 QuantumSPT$(G = (V, E), v_0)$

Require: $T = (V_T = \{v_0\}, E_T = \emptyset)$ ▷ Shortest path tree to be grown.
Require: $P_1 = \{v_0\}$ and $L = 1$
1: set $\text{dist}(v_0) = 0$ and $\forall u \in V, u \neq v_0$, $\text{dist}(u) = \infty$.
2:
3: **while** $|V_T| < n$ **do**
4: $B_L = \text{MINFIND}(|P_L|, f, g)$
5: Let $(u, v) \in B_1 \cup \cdots \cup B_L$ have minimal $\text{cost}(u, v)$ with $v \notin P_1 \cup \cdots \cup P_L$. ▷ (u, v) is a border edge
6: **if** $w(u, v) = 0$ **then**
7: **return** $\mathcal{T}$
8: **else**
9: $V_T \leftarrow V_T \cup \{v\}$, $E_T \leftarrow E_T \cup \{(u, v)\}$
10: $\text{dist}(v) = \text{dist}(u) + 1/w(u, v)$
11: $P_{L+1} \leftarrow \{v\}$, $L \leftarrow L + 1$
12: **while** $L \geq 2$ and $|P_L| = |P_{L-1}|$ **do**
13: *merge* P_L *into* P_{L-1}
14: $L \leftarrow L - 1$

In Algorithm 2, a set of L partitions $\{P_l\}_{l=1}^{L}$ of the vertices covered by the shortest path tree $\mathcal{T}$ is generated, and the algorithm stops only when $\mathcal{T}$ covers the connected component of v_0.

Step 1 initializes the distances, as does Dijkstra's algorithm. In Step 4, a set B_L containing the $|P_L|$ cheapest border edges with disjoint target vertices is generated by the quantum routine $\text{MINFIND}(|P_L|, f, g)$. Step 10 updates the distance of the selected vertex, in a same manner as in Dijkstra's. After all the merges of Step 13, the P_k are sets of vertices of the growing tree, so that $|P_k| = 2^{L-k}$. This ensures that since B_k contains $|P_k|$ edges, then at least one of these edges has its target outside of $\cup_{j=1}^{L} P_k$, implying in Step 5 at least one border edge exists, and is effectively selected, thus the correctness of the algorithm (see [AW19, Appendix A, Proposition 5]). As a side note, at each step $V_{\mathcal{T}}$ contains the growing tree.

Theorem 3.2. *In the worst case, Algorithm 2 returns a shortest path tree covering the graph $G = (V, E)$ in time $\tilde{O}(\sqrt{mn})$.*

More precisely, the running time depends on the size of the connected component in which the starting node v_0 is. Taking into account Theorem 3.2, one can conclude on the overall time complexity of Algorithm 3.

Theorem 3.3. *There exists a quantum algorithm that outputs in time $\tilde{O}(kn^{1/k}\sqrt{mn})$ with high probability a $(2k-1)$-spanner of G with an expected number of edges $O(kn^{1+1/k})$.*

Algorithm 3 QuantumSpanner$(G = (V, E), k)$

Require: $A_0 = V$ ans $A_k = \emptyset$
Ensure: *H is initially an empty graph*
1: **for** $i = 1, \cdots, k$ **do**
2: **if** $i < k$ **then** *set A_i such as defined in Equation 6*
3: **for all** $v \in A_{i-1} - A_i$ **do**
4: $\mathcal{T} \leftarrow$ **QuantumSPT**(G, v)
5: $H \leftarrow H \cup \mathcal{T}$
6: **return** H

To conclude, setting $k = \log n + 1/2$, one can construct $(2 \log n)$-spanners of an input graph with n nodes and m edges in time $\tilde{O}(\sqrt{mn})$.

3.2 Implicit construction of the graph though a *string*

In order to stay within a sublinear runtime, one cannot use an explicit representation as used by Koutis and Xu in Algorithm 1: indeed, after a single iteration, the outputted graph could have up to $\frac{m}{2} \in O(n^2)$ edges (see e.g., Theorem 2.3).

Apers and de Wolf address this issue by constructing a random string $r \in \{0, 1\}^m$ encoding the discarded edges at some iteration with 0-valued bits, and later implicitly setting the corresponding weights in the graph to 0, as shown in Algorithm 4. This enables the construction of a spanner in the remaining graph. One can then use a Grover search to the undiscarded $\tilde{O}(n/\varepsilon^2)$ edges, whose union forms the spectral sparsifier. In addition, it is possible to further improve the classical complexity, since a k-query quantum algorithm cannot distinguish a $2k$-wise independent strings from a uniformly random one [Zha15].

At first, a family of independent random bit-strings

$$r_i \in \{0, 1\}^m, \ \ i \in \left[\log \frac{m}{n}\right] ,$$

is considered, such that all bits are *independent* and *equal to 1 with probability 1/4*.

Thus the graph is represented throughout the execution with a bit-string r, where each bit b_e is sampled only when edge e is queried.

However, thanks to the result of Zhandry, it is possible to discard the random strings by considering k-independent hash functions, whose definition is recalled in Appendix B, Definition B.1. Hence, such a structure allows to query for a bit-string element in time $\tilde{O}(1)$ without even having to store the bit-string, but still being able to retrieve it. It is important to stress that it is a purely classical result.

A quantum oracle that keeps track of the weight updates is easily constructed. Considering the i^{th} iteration; given an edge e, let k denote the number of spanners in which e appears before this iteration.

If $k = 0$, the weight of the edge e is re-weighted as

follows:

$$\omega'_e = \begin{cases} 4^i \omega_e & \text{if } \bigvee_{l=1}^{i} r_l(e) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Otherwise, in the case where $k > 0$, the weight of the edge e is re-weighted in a different manner, so that

$$\omega'_e = \begin{cases} 4^{i-k} \omega_e & \text{if } \bigvee_{l=1}^{j+1} r_l(e) = 1 \\ 0 & \text{otherwise,} \end{cases}$$

where $\vee$ is the logical disjunction.

Algorithm 4 QuantumSparsify(G, ε)

Require: $\forall e,\ w'_e = w_e$ and $l = \lceil \log \frac{m}{n} \rceil$

Require: $\forall i \in [\log(m/n)],\ r_i \in \{0,1\}^m$, ▷ A family of random strings such that all bits are independent and equal to 1 w.p. $\frac{1}{4}$.

1: **for** $i = 1, 2, ..., l$ **do**

2: *create H_i, union of an $O(\frac{\log^2 n}{\varepsilon^2})$-packing of spanners of $G' = (V, E, w')$*

3: **for all** $e \notin H_i$ **do**

4: **if** $r_i(e) = 1$ **then** $w'_e \leftarrow 4\, w'_e$

5: **else** $w'_e \leftarrow 0$

6: *use repeated Grover search to find $\tilde{E} = \{e \in E | w'_e > 0\}$ the edges of $\tilde{G}$*

7: **return** $\tilde{G}$

Intuitively, the unions of the $O(\frac{\log^2 n}{\varepsilon^2})$-packing of spanners select the *most* important edges of the graph, and the conditional reweighting (Steps 4,5) is a way of keeping a fraction of the remaining edges in order to *spectrally* preserve the graph (i.e., asserts that in the end it effectively $(1+\varepsilon)$-approximates the input graph). In each iteration, the remaining graph is classically sparsified using Algorithm 1. The sparsified graph is the one induced by the vertices of the initial graph and the edges whose weight w'_e is greater than 0.

Proposition 3.4. *The probability that all* $\log \frac{m}{n}$ *iterations succeed is* $1 - \mathrm{O}(\frac{\log n}{n^2})$.

See proof on page 8.

The overall time complexity of the algorithm depends on whether the $(r_i)_i$ are represented with a random string. By Definition 2.2, the set of spanners is assumed ordered, allowing to binary search through the set in time $\tilde{O}(1)$, and there is $O(i)$ calls to the aforementioned oracle. The algorithm requires $O(\log n)$ qubits, which is the number of qubits needed for the quantum spanner algorithm and the repeated Grover search. In addition a QRAM [2] of $\tilde{O}(n/\varepsilon^2)$ bits is required since the classical space complexity is dominated by the output size, i.e. the size of the graph.

It is possible to simulate the random strings in Algorithm 4 with k-independent hash functions, and hence improve the classical space complexity from $\tilde{O}(n/\varepsilon^2)$ to $\tilde{O}(\sqrt{mn}/\varepsilon^2)$.

Considering the efficiently computable search function $f : E \to \{0,1\}$ such that

$$f(e) = \begin{cases} 1 & \text{if } w'_e > 0 \\ 0 & \text{otherwise} \end{cases},$$

Grover's algorithm finds a single edge in time $\tilde{O}(\sqrt{\frac{m}{n/\varepsilon^2}})$. Therefore, retrieving $\tilde{O}(n/\varepsilon^2)$ edges belonging to $\tilde{G}$ takes time $\tilde{O}(\frac{\sqrt{m\,n}}{\varepsilon})$.

As stated in Theorem 3.3, one can construct a $(\log^2 n/\varepsilon^2)$-spanner in time $\tilde{O}(\sqrt{mn}/\varepsilon^2)$.

The overall time complexity is the sum of the runtimes needed to simulate the random string, to construct a spanner and for the repeated Grover search. Therefore, the total runtime is

$$2\tilde{O}(\sqrt{mn}/\varepsilon^2) + \tilde{O}(\sqrt{mn}/\varepsilon) = \tilde{O}(\sqrt{mn}/\varepsilon^2)\ .$$

Theorem 3.5 (Quantum Spectral Sparsification)**.** *The algorithm* ***QuantumSparsify***(G, ε) *returns with probability* $1 - \mathrm{O}(\log n/n^2)$ *an ε-spectral sparsifier of G with $\tilde{O}(n/\varepsilon^2)$ edges, in time $\tilde{O}(\sqrt{m\,n}/\varepsilon^2)$ and using a QRAM of $\tilde{O}(\sqrt{m\,n}/\varepsilon^2)$ bits.*

3.3 Time improvement of quantum sparsification

3.3.1 Approximate resistance oracle and spectral sparsification

From the result of Spielman and Srivastava [SS11], one can compute a matrix Z such that for all pairs (s,t) of edges in G,

$$(1-\varepsilon)R_{s,t} \leq ||Z \cdot (\chi_s - \chi_t)^2|| \leq (1+\varepsilon)R_{s,t} \qquad (7)$$

in time $\tilde{O}(m/\varepsilon^2)$. Z is defined as $Z = QW^{\frac{1}{2}}BL^{+}$, where $L = B^T W B$ with B the incidence matrix and W a diagonal matrix such that $(W)_{ii} = \omega_{e_i}$, and Q a random $\pm 1/\sqrt{k}$ matrix (i.e., independent Bernoulli entries). Consequently, thanks to Equation 5, the matrix Z helps ε-approximate the effective resistance between any edge $e = (s,t)$ of the initial graph.

The proof of the existence of such a Z matrix allows one to efficiently create an oracle for the quantum algorithm.

Theorem 3.6 (Sparsification with approximate resistances [SS11])**.** *Let $R_e/2 \leq \tilde{R}_e \leq 2R_e$ be a rough approximation of R_e, for each $e \in E$ and $p_e = min(1, Cw_e\tilde{R}_e \log(n)/\varepsilon^2)$. Then, with probability $1 - 1/n$, an ε-spectral sparsifier H with $O(n \log(n)/\varepsilon^2)$ edges can be obtained by keeping every edge e independently with probability p_e and rescaling its weight with $1/p_e$.*

Theorem 3.6 allows one to efficiently define the $\{p_e\}$ according the effective resistance approximations $\{\tilde{R}_e\}$.

[2] See Appendix A for details about the QRAM model used herein.

Since H is an ε-spectral sparsifier of G, we have that for all edge e of H,

$$(1-1/\varepsilon)R_e^G \leq R_e^H \leq (1+1/\varepsilon)R_e^G \ ,$$

where R_e^G and R_e^H are effective resistances in G and H, respectively. From Equation 7, the effective resistances R_e can be approximated with the matrix Z in such a way that for an edge $e=(s,t)$, the approximated resistance is

$$\tilde{R}_e = ||Z\cdot(\chi_s-\chi_t)^2|| \ .$$

The probability p_e of keeping an edge e is taken to be proportional to $\tilde{R}_e$, since an edge will be more important if it belongs to a weak component, i.e. if it has a high effective resistance. Thanks to the result of Bollobás [Bol98, Theorem 25],

$$\sum_e w_e R_e = n-1$$

for connected graphs of order n [3], and thus, one has that $\sum_e w_e\tilde{R}_e = O(n)$. Since $\sum_e p_e$ represents the number of edges in the sparsifier, if one wants to end up with $O(n\log n/\varepsilon^2)$ edges in the resulting graph, p_e should be taken proportional to $w_e\tilde{R}_e\log(n)/\varepsilon^2$.

In order to keep a satisfying approximation of the weights $\{w_e\}$ in the sparsifier, we want to keep unchanged the expectation value of the weight of each edge. Hence[4], every weight w_e is re-scaled by $1/p_e$ i.e., $\tilde{w_e}=\frac{w_e}{p_e}$.

3.3.2 Edge sampling

Classically, Algorithm 5 shows how one can sample a subset of edges that contains every edge e independently with probability p_e, in time $\tilde{O}(m+\sum_e p_e)$.

Algorithm 5 ClassicalEdgeSampling(G,ε)

Require: $S=\emptyset$
1: *approximate* $\{p_e\}_{e\in E}$ *using Equation 7*
2: **for all** $e\in E$ **do**
3: *add edge* e *to* S *with probability* p_e
4: **return** S

A quantum algorithm could sample a subset of edges more efficiently. We assume we have access to a random string $r\in\{0,1\}^{\tilde{O}(m)}$ through the hash function $h_r: E\times[0,1]\to\{0,1\}$, such that for all $e\in E$, $h_r(e,p_e)=1$ with probability p_e and $h_r(e,p_e)=0$ otherwise. From this, it is possible to construct the following oracle

$$O_s : |e\rangle|v\rangle|w\rangle \longmapsto |e\rangle|v\oplus p_e\rangle|w\oplus h_r(e,p_e)\rangle \ .$$

Due to the fact that the expected number of edges e for which $h_r(e,p_e)=1$ is $\sum_e p_e$, a repeated Grover search finds the desired edges in time $\tilde{O}(\sqrt{m\sum_e p_e})$.

[3] i.e. with n edges

[4] Let $\tilde{W}$ be the random variable such that $P(\tilde{W}=\tilde{w_e})=p_e$ and $P(\tilde{W}=0)=1-p_e$. Then the expectation value of $\tilde{W}$ is $\mathbb{E}(\tilde{W})=p_e\tilde{w_e}+(1-p_e)\times 0=p_e\tilde{w_e}$.

3.3.3 Refined quantum sparsification algorithm

The runtime of Algorithm 4 can be improved to $\tilde{O}(\sqrt{mn}/\varepsilon)$ by creating a first "rough" ε-sparsifier H, estimating the effective resistances of G from H using Laplacian solving, and then using quantum sampling in order to sample a subset containing $\tilde{O}(n/\varepsilon^2)$ edges.

Algorithm 6 RefinedQuantumSparsify(G,ε)

1: *use Algorithm 4 to construct a* $(1/100)$*-spectral sparsifier* H *of* G
2: *create a* $(1/100)$*-approximate resistance oracle of* H *using Theorem 3.6, yielding estimations* $\tilde{R}_e$
3: *use quantum sampling to sample a subset of the edges, keeping every edge with probability* $p_e = min(1, Cw_e\tilde{R}_e\log(n)/\varepsilon^2)$

The Step 1 of Algorithm 6 requires for $\tilde{O}(\sqrt{mn})$ to construct the 1/100-spectral sparsifier H, in which each edge e is such that its effective resistance R_e^H satisfies

$$(1-1/100)R_e^G \leq R_e^H \leq (1+1/100)R_e^G \ .$$

According to Equation 7, there exists an oracle to derive approximated resistances $\{\tilde{R}_e\}$ in Step 2 such that, for all edges $e=(s,t)$,

$$(1-1/100)R_e^H \leq \tilde{R_e^H} \leq (1+1/100)R_e^H \ ,$$

where $\tilde{R_e^H}=||Z\cdot(\chi_s-\chi_t)^2||$. One can then deduce that

$$(1-1/100)^2R_e^G \leq \tilde{R_e^H} \leq (1+1/100)^2R_e^G \ .$$

Supposing that each edge e is kept with probability

$$p_e = \min(1, Cw_e\tilde{R_e^H}\log(n)/\varepsilon^2) \ ,$$

an ε-spectral sparsifier can be constructed with $O(n\log(n)/\varepsilon^2)$ edges according to Theorem 3.6. The approximate oracle needed for this step requires time $\tilde{O}(n)$ to be constructed.

The quantum routine of Step 3 takes time $\tilde{O}(\sqrt{m\sum_e p_e})$ where

$$\begin{aligned}\sum_e p_e &\leq \frac{C\log(n)}{\varepsilon^2}\sum_e w_e\tilde{R_e^H} \\ &\leq \frac{(1+1/100)^2C\log(n)}{\varepsilon^2}\sum_e w_eR_e^G \ .\end{aligned}$$

As stated in [Bol98], one always has that for a connected graph of order n, $\sum_e w_eR_e^G = n-1$. Therefore, we can conclude that $\sum_e p_e \in \tilde{O}(n/\varepsilon^2)$ which implies that the total runtime of the quantum sampling routine is $\tilde{O}(\sqrt{mn}/\varepsilon)$.

One can notice that Step 2 only succeeds with probability $1-\frac{1}{n}$ as claimed by Theorem 3.6. According to that, one can abort the algorithm as soon as the runtime exceeds $\tilde{O}(\sqrt{mn}/\varepsilon)$ and start again, yielding a runtime of

$\tilde{O}(2\sqrt{mn}/\varepsilon) = \tilde{O}(\sqrt{mn}/\varepsilon)$ in the worst case.

The total runtime of Algorithm 6 is the sum of the runtimes of the three steps and it is therefore $\tilde{O}(\sqrt{mn})$.

Theorem 3.7 (Quantum Spectral Sparsification)**.** ***RefinedQuantumSparsify****(G,ε) returns with high probability an ε-spectral sparsifier H with $\tilde{O}(n/\varepsilon^2)$ edges, and has runtime $\tilde{O}(\sqrt{mn}/\varepsilon)$. The algorithm uses $O(\log n)$ qubits and a QRAM of $\tilde{O}(\sqrt{mn}/\varepsilon)$.*

4 Lower bound of the query complexity

The proof of the lower bound on the query complexity relies on random graphs

Having arrived to Theorem 3.7, the main result of the paper was made explicit. Apers and de Wolf's algorithm thus yields a quantum speedup for solving Laplacian systems and for approximating a range of cut problems such as min cut and sparsest cut.

This result can probably be combined with recent classical results such as [Che+22] to yield even faster algorithms. Stay tuned...

References

[AW19] Simon Apers and Ronald de Wolf. "Quantum Speedup for Graph Sparsification, Cut Approximation and Laplacian Solving". In: (2019). DOI: 10.48550/ARXIV.1911.07306.

[Bol98] Béla Bollobás. *Modern Graph Theory*. en. Vol. 184. Graduate Texts in Mathematics. New York, NY: Springer New York, 1998. ISBN: 9780387984889 9781461206194. DOI: 10.1007/978-1-4612-0619-4.

[BSS12] Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. "Twice-Ramanujan Sparsifiers". en. In: *SIAM Journal on Computing* 41.6 (Jan. 2012), pp. 1704–1721. DOI: 10.1137/090772873.

[Che+22] Li Chen et al. "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time". In: (2022). DOI: 10.48550/ARXIV.2203.00671.

[CPT15] Tobias Christiani, Rasmus Pagh, and Mikkel Thorup. "From Independence to Expansion and Back Again". en. In: *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*. Portland Oregon USA: ACM, June 2015, pp. 813–820. ISBN: 9781450335362. DOI: 10.1145/2746539.2746620.

[Dü+06] Christoph Dürr et al. "Quantum Query Complexity of Some Graph Problems". In: *SIAM Journal on Computing* 35.6 (2006), pp. 1310–1328. DOI: 10.1137/050644719. eprint: https://doi.org/10.1137/050644719.

[GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. "Quantum Random Access Memory". In: *Phys. Rev. Lett.* 100 (16 2008), p. 160501. DOI: 10.1103/PhysRevLett.100.160501.

[KX16] Ioannis Koutis and Shen Chen Xu. "Simple Parallel and Distributed Algorithms for Spectral Graph Sparsification". en. In: *ACM Transactions on Parallel Computing* 3.2 (Aug. 2016), pp. 1–14. DOI: 10.1145/2948062.

[SS11] Daniel A. Spielman and Nikhil Srivastava. "Graph Sparsification by Effective Resistances". en. In: *SIAM Journal on Computing* 40.6 (Jan. 2011), pp. 1913–1926. DOI: 10.1137/080734029.

[ST11] Daniel A. Spielman and Shang-Hua Teng. "Spectral Sparsification of Graphs". en. In: *SIAM Journal on Computing* 40.4 (Jan. 2011), pp. 981–1025. DOI: 10.1137/08074489X.

[TZ05] Mikkel Thorup and Uri Zwick. "Approximate Distance Oracles". In: *J. ACM* 52.1 (2005), 1–24. DOI: 10.1145/1044731.1044732.

[Zha15] Mark Zhandry. "Secure identity-based encryption in the quantum random oracle model". en. In: *International Journal of Quantum Information* 13.04 (June 2015), p. 1550014. DOI: 10.1142/S0219749915500148.

A QRAM Model

To achieve the speed-up promised by the quantum algorithms presented hereby, we assume the existence of a quantum device able to run quantum subroutines on at most $O(\log N)$ qubits, where N is the size of the problem or the input.

Besides, we assume an access to a Quantum Random Access Memory (QRAM) which is, as its classical analog, composed of an *input* register, a *memory* array and an *output* register. The main variations are that the input and output registers are composed of qubits rather than bits. Thus, the quantum computer can address memory in superposition meaning that a superposition of inputs returns a superposition of outputs, so that one can design the following quantum unitary

$$\sum_j \lambda_j |j\rangle_{in} |0\rangle_{out} \xrightarrow{QRAM\ access} \sum_j \lambda_j |j\rangle_{in} |v_j\rangle_{out} \quad ,$$

where *in*, *out* represent respectively the *input* and the *output* registers and v_j the value contained in the $j-th$

register. Hence, a reading operation corresponds to a quantum query to the classical bits stored in the memory array, whereas the operation of writing a bit in the QRAM stays classical.

Within this computational model, the complexity of an algorithm can have several definitions. One can consider either the *time complexity*, which counts the number of elementary gates (classical and quantum), of quantum queries to the input and of QRAM operations, or the *query complexity* which only counts the number of quantum queries to the input. As an example of actual QRAM, a quantum optical implementation is presented in [GLM08].

B k-independent hash functions

Definition B.1 (k-independent hashing)**.** Let $\mathcal{U}$ be the set of keys. A family $\mathcal{H} = \{h : \mathcal{U} \to [m]\}$ is said to be k-independent if for all keys $x_1, \cdots, x_k$ in $\mathcal{U}$ pairwise distinct and for all values $v_1, \cdots, v_k$ in $[m]$,

$$\big|\{h \in \mathcal{H}\ ;\ h(x_1) = v_1, \cdots, h(x_k) = v_k\}\big| = \frac{|\mathcal{H}|}{m^k}\ ,$$

in other words, by providing $\mathcal{H}$ with the uniform probability, for any $h \in \mathcal{H}$

$$\mathbb{P}\big(h(x_1) = v_1, \cdots, h(x_k) = v_k\big) = \frac{1}{m^k}\ .$$

C Proofs

Proof of Proposition 3.4. Let p_s be the probability of success and p_f be the probability of failure. If $p_s = 1 - \frac{1}{n^2}$ then $p_e = \frac{1}{n^2}$, since $\log \frac{m}{n}$ are done, the global probability of failure P_f is the sum of each p_f, such that

$$P_f = \frac{1}{n^2} \times \log \frac{m}{n}.$$

Since m is the number of edges of the input graph,

$$m \leq \binom{n}{2} \in O(n^2),$$

thus

$$\log \frac{m}{n} \in O\big(\log \frac{n^2}{n}\big) = O(\log n)\ ,$$

hence

$$p_e = O\big(\frac{\log n}{n^2}\big)\ ,$$

the result follows. □