On the Approximate Computation of Graph Diameter

A Project Report

submitted by

AAHLAD MANAS (EE10B075)

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY and MASTER OF TECHNOLOGY

under the guidance of
Dr. Narayanaswamy N. S.

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

May 2015

THESIS CERTIFICATE

This is to certify that the thesis entitled **On the Approximate Computation of Graph Diameter**, submitted by **Aahlad Manas (EE10B075)**, to the Indian Institute of Technology, Madras, for the award of the degree of **Dual Degree**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Narayanaswamy N. S.	**Andrew Thangaraj**
Research Guide	Co-guide
Associate Professor	Associate Professor
Dept. of Computer Science and Engineering	Dept. of Electrical Engineering
IIT-Madras, 600 036	IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

First and foremost I would like to thank my Dual Degree project guide Prof. Narayanaswamy for insights into the many techniques imparted to me during our meetings. I thank him for the certain flair to approaching problems that elucidates the most important aspects of the underlysing strutures.

I thank my parents and family for supporting me into shifting my focus of major. I thank my department and Faculty Advisor for allowing the academic freedom and letting me explore everything that I was interested in. I thank the institute for provinding such a stimulating atmosphere needed in the noble pursuit that is research.

Finally I thank my friends who are akin to family and without whose help I couldn't have begun to do what I now set out to be. I would like to name you all but then adding the font size corresponding to the impact you've had on my life would make this report cross a 100 pages.

ABSTRACT

The best algorithms till date to compute the diameter of a graph are the all-pairs-shortest-paths which runs in $O(nm)$ for general graphs and fast matrix multiplication(for unweighted undirected graphs) which result in a complexity upper bound of Õ$(n^{2.373})$. To approximate the diameter additively throught APSP approximations, the best time is Õ$(min(n^{3/2}m^{1/2}, n^{7/3})$.In this report an approach to the problem of approximating the diameter to within $D-4$ which partitions the graph, contracts it and then computesa modified version all-pairs-shortest-paths to compute the diameter is given. The result is a complexity of O$((max(min(n^2 n_c, \delta^2 m_c), min(nn_c^2, \delta n_c m_c))$, where the terms δ, n_c, m_c and n are the maximum degree in the original graph, number of vertices and the number of edges in the contracted graph and the vertex-set of the original graph. A simple analysis of the algorithm is shown over graphs with have a new notion of density appropriate for the algorithm. This works by generalizing from a family of graphs to showcase the way the complexity varies from sparse graphs to dense graphs.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

The diameter of a graph is the length $max_{(u,v)}d(u,v)$; the ”longest shortest path” between any two graph vertices (u,v) of a graph, where $d(u,v)$ is a shortest distance between u and v. In other words, a graph’s diameter is the largest number of vertices which must be traversed in order to travel from one vertex to another when paths which backtrack, detour, or loop are excluded from consideration. It is one of the fundamental properties of a graph with applications in fields ranging from networks, social and otherwise, to linguistics.

Rather than the path itself, computing the length is adequate. This is so because many networking protocols start of with allocation of resouces based of the diameter of a graph. Being able to compute the diameter, center and radius of a graph efficiently has become an increasingly important problem in the analysis of large networks [16].The diameter of the web graph for instance is the largest number of clicks necessary to get from one document to another, and Albert et al. were able to show experimentally that it is roughly 19 [17].

Some trivial diametric results are:

1. A disconnected graph has infinite diameter.
2. A complete graph has diameter 1.
3. A complete bipartite graph has diameter 2.

The best algorithm to compute the diameter in general till date is the All-pair-shortest-paths(APSP) algorithm. It runs in Õ(nm) time for general weighted directed graphs.

For dense graphs, fast matrix multiplication, the fastest implementation of which runs in Õ$(n^{2.373})$[4] time, can be used to compute diameter. This algorithm stems from the consideration of the problem from an algebraic graph theoretic approach. The diameter $D(G)$ is the least integer t such that the matrix $M = I + A$ has the property that all entries of Mt are nonzero. One can find t by $O(\log n)$ iterations of matrix multiplication. The diameter algorithm then requires $O(M(n)\log n)$ time, where $M(n)$ is the complexity upper bound for matrix multiplication. The matrices I and A above are the identity matrix and the adjacency matrix respectively. APSP becomes asymptotically faster for sparse graphs with the size of the edge-set $m = \mathrm{o}(n^{1.373})$.

A lot of work has gone into computing the diameter for restricted families of graphs. For suitable graph class, the computation of graph diameter can be solved using implementations of APSP problem in optimal $\mathrm{O}(n^2)$ time. These families include, but not limited to, interval graphs, circular arc graphs, permutation graphs, bipartite permutation graphs, strongly chordal graphs, chordal bipartite graphs, distance-hereditary graphs, and dually chordal graphs.[3]

It is unclear why super-quadratic complexity is required to compute the diametric length from a graph. Hence, attempts were made to approximate the diameter. The best multiplicative approximation algorithm for unweighted graphs runs in $\tilde{\mathrm{O}}(\sqrt{n}m)$ time with an estiamte guarentee of $\frac{2d}{3}$. It was proved by Roddity et al[2]. to be the best possible, given the Exponential Time Hypothesis. For weighted graphs ,with arbitrarily large edge-weights, on the other hand, this approach falls short. In a subsequent paper [1], an algorithm running in $\mathrm{O}(mn^{\frac{2}{3}})$ was shown to give the same estimate as above $\frac{2d}{3}$.

Eccentricity and Radius

A closely related parameter is the eccentricity of a graph. Graph diameter is also the largest eccentricity over all vertices. The smallest eccentricity over all vertices, on the other hand, is called the radius of the graph. The problem of computing a 'center' vertex and the radius of a graph is often studied as a facility location problem for networks: pick a single vertex facility so that the maximum distance from a demand point (client) in the network is minimized. The algorithms described here and henceforth in the other chapter all have the property that they be easily modified to compute the radius.

This Report

In this report, an algorithm that beats APSP in dense unweighted, undirected graphs is shown. The algorithm also works for a few related problems and computes approximately graph parameters like the raidius and eccentricity. It works by partitioning the graph into subgraphs of radius 1 and then computing an APSP over a graph in these subgraphs are themselves the vertices. The information required to compute the diameter that is lost during clustering, i.e edges inside cluster, is partially recovered by cheking every vertex and it's neighbours.

In Chapter 2, the diameter results over the last 2 decades are set in perspective with the changing research trend; how it went from diameter computation to diameter approximation to APSP approximation and then back to APSP independetn diameter approximation. A brief introduction to the techniques presented in this paper is also included in this paper.

Chapter 3 consists of the algorithm itself and the analysis. As the bound on the complexity in terms of the parameters of the contracted graph is not easily reducible to the parameters of the original graph, a different approach to the analysis is demonstrated. The rest of the chapters are decicated to other immediate applications and possible extensions of the algorithm respectively.

CHAPTER 2

Diameter Computation Results: Old and New

2.1 Previous results

2.1.1 General Graphs

Till date there has been no algorithm for general weighted graphs that solves Diameter problem asymptotically faster than APSP and the existence of such an algorithm is an open problem (see, e.g., [5][6] [7]). In the case of graphs with arbitrary real edge weights, no truly sub-cubic algorithm for APSP or for Diameter is known.

Sub-cubic algorithm

The presently fastest algorithm for both is an algorithm of Chan [8] that runs in O$(n^3 \log^3 \log n / \log^2 n)$ time, where n in the number of vertices of the graph. For sufficiently sparse graphs, the APSP algorithm of Johnson [9] performs better as it runs in O$(mn + n^2 \log n)$time where m is the number of edges.

2.1.2 Edge restrictions and fast matrix multiplication

When the edge weights are integers, which is the case for undirected unweighted graphs, fast matrix multiplication techniques are used. For undirected graphs with integer edge weights in $\{1, \cdots, M\}$ an APSP (and thereby a Diameter) algorithm of Shoshan and Zwick [10] runs in O(Mn^w) time, where $w < 2.376$ (recently upper bounded by 2.373 in a paper by Vassilevska Williams) is the exponent of matrix multiplication. This algorithm generalizes earlier Õ(n) algorithms of Seidel and of Galil and Margalit for the unweighted case [12] [11].

The situation becomes even more dire as the techniques become more involved in directed graphs, as negative edge weights are now allowed. In the presence of weights in $\{M, \cdots, 0, \cdots, M\}$, the fastest APSP algorithm (and consequently the fastest Diameter algorithm) is by Zwick [13]. It runs in $\tilde{O}(M^{1/(4-w)}n^{2+1/(4-w)})$ time and an additional small speedup is obtained if fast rectangular matrix multiplication is used. For bounded M, this speedup results in a running time of O$(n^{2.575})$. Interestingly, Zwick's algorithm is also the fastest known algorithm for unweighted APSP in directed graphs.

The directed unweighted setting is also the only known setting where Diameter has an algorithm that is presently faster than APSP.

2.2 Approximate diameter computation

One of the natural questions that can be asked is whether substantially faster diameter and radius algorithms are possible if we loosen the problem from exact to approximate copmutation of the diameter. It is easy to devise and algorithm that results in a multiplicative 2-approximation for both the diameter and the radius in an undirected graph that achieves this in O$(m + n)$ time using BFS from an arbitrary node. This is easy to see as the longest path, the eccentricity, from a vertex satisfies the following inequality: $D/2 < r < \epsilon(v) < D$.

2.2.1 Relation to Matrix Multiplication

For APSP, to set the problem at hand in perspective, Dor et al.[14] show that any $(2 - \epsilon)$-approximation algorithm in unweighted undirected graphs running in $T(n)$ time would imply an O$(T(n))$ time algorithm for Boolean matrix multiplication (BMM). This hardness result could make sure that a (2)-approximation algorithm for the diameter and radius of a graph may also require solving Boolean Matrix Multiplication.

In a seminal paper from 1996, Aingworth et al.[5] showed that it is in fact possible to get a sub-cubic (2ϵ)-approximation algorithm for the diameter in both directed and undirected graphs without utilization of fast matrix multiplication. They designed an $O(m\sqrt{n}+n^2)$ time algorithm computing an estimate $\hat{D}$ that satisfies $2D/3 \leq \hat{D} \leq D$. Their algorithm was important as it had many interesting properties that would lead to a change in research trend. It is the only known algorithm for approximating the diameter polynomially faster than O(mn) for every m that is super linear in n. It always runs in truly sub-cubic time even in dense graphs, and does not explicitly compute all-pairs approximate shortest paths.

2.2.2 Research Trend: APSP approximation

For the radius problem, Berman and Kasiviswanathan [15] showed that the approach of Aingworth et al. can be used to obtain in $\tilde{O}(m\sqrt{n} + n^2)$ time an estimate $\hat{r}$ that satisfies $r \leq \hat{r} \leq 3/2r$, where r is the radius of the graph. Thus both radius and diameter admit $\tilde{O}(m\sqrt{n} + n^2)$ time $3/2$-approximations. Aingworth et al. later also developed an algorithm that solves the APSP problem with additive 2-approximation and runs in $\tilde{O}(n^{2.5})$ time, that is for every $u, v \in V$ the algorithm returns a value $\hat{d}(u, v)$ such that $d(u, v) \leq \hat{d}(u, v) \leq d(u, v) + 2$, where $d(u, v)$ is the distance between u and v. Their paper spawned a long line of research on distance approximation in the properties of graph-spanner algorithms or approximate distance oracles.

However, none of the work in the next few years considered the problems specific to the approximation of diameter and radius. Instead they focused on approximation algorithms for APSP. Dor, Helperin, and Zwick [14] came up with an algorithm that gave an additive 2-approximation for APSP in unweighted undirected graphs with a running time of $\tilde{O}(min(n^{3/2}m^{1/2}, n^{7/3})$, and henceforth establishing the best time complexity bound for for additive approximation, and improving on Aingworth et al.s work on the APSP approximation algorithm.

2.2.3 Existing APSP Approximation Results

Baswana et al.[19] came up with a las-vegas algorithm for unweighted undirected graphs with an *expected* running time of $O(m^{2/3}n \log n + n^2)$ that computes an approximation of all distances with a multiplicative error of 2 and an additive error of 1. Elkin [23] improved partially upon this result with an algorithm for unweighted undirected graphs with a running time of $O(mn\rho + n^2\gamma)$ that approximates the distances with a multiplicative error of $(1 + \epsilon)$ and an additive error that is a function of γ, ρ and ϵ.

Cohen and Zwick [22] extended the results of [14] to weighted graphs. Baswana and Kavitha [15] presented an $\tilde{O}(m\sqrt{n}+n^2)$ time multiplicative 2-approximation algorithm and an $\tilde{O}(m^{2/3}n+n^2)$ time 7/3-approximation algorithm for APSP in weighted undirected graphs. Since Aingworth et al.s paper, the only paper that considers the diameter approximation problem directly is by Boitmanis et al.[21]. They presented an algorithm with $\tilde{O}(m\sqrt{n})$ running time that computes the diameter with an additive error of $\sqrt{n}$. Even though such an additive error could be neglected for graphs with large diameter, it cannot be put into practice when it comes to graphs with small diameter. A simple random sampling argument shows that for all graphs with diameter at least n^{δ}, there is an Õ$(mn^{1-\delta}/\epsilon)$ time $(1+\epsilon)$-approximation algorithm for all $\epsilon > 0$. This is the reason diameter approximation is hardest for graphs with small di-ameter. For such graphs the additive approximation of Boitmanis et al.presents no significant approximation guarantee.

2.2.4 Recent Diameter Approximation Algorithms

In the last 4 years, there have been two papers that established algorithms with a 3/2 approximation factor that do not explicitly compute All-pairs-shortest paths. The first one [2] by Liam Roddity and Virginia Vassilevska Williams works best for unweighted graphs. The second one [1] was by Shiri Chechik, Daniel Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, Virginia Vassilevska Williams and this improves upon the first paper for weighted graphs and eccentricity computation.

In [2] Roddity and Vassilevska Williams present an algorithm that guarantees an estimate of $\hat{D} \geq 2D/3$ and runs in O$(m\sqrt{n})$expected running time. Then they prove that given the Strong exponential time hypothesis, it'll be hard to improve this result. More specifically,they show that if there exists a positive constant $\delta > 0$ so that there is an algorithm for undirected unweighted graphs that runs in O$(m^{2-\epsilon})$ time and produces an approximation $\hat{D}$ such that $(2/3+\epsilon)D \leq \hat{D} \leq D$, then SAT for CNF formulas on n variables can be solved in $O^{*}((2-\delta)^{n})$ time for some constant $\delta > 0$, and consequently strong exponential time hypothesis of Impagliazzo, Paturi, Zane is false.

Brief Outline of $\frac{3}{2}$-approximation Algorithm

The first s closest vertices of every vertex need to be found for the algorithm; this is the neighbourhood of a vertex. A sampling technique is used which hits the neighbourhood of every vertex with high probability.

1. The neighbourhood of every vertex is hit by sampling vertices with a certain probability and a set S of size $\Theta(n/s \log n)$.
2. The closest $p_S(v) \in S$ to every vertex is found. This is done by adding another vertex to every vertex in S and computing a BFS from it.
3. The main part of the algorithm is computing the farthest vertex w to any vertex in S, i.e. such that $d(w, p_S(w)) \geq d(u, p_S(u))$, for every $u \in V$
4. Compute A BFS from every vertex in S and in the neighbourhood of w.

5. Return the maximum distance found.

Proof of correctness: Let there be diametric vertices $a, b \in V$ such that $d(a,b) = D = 3h + z$. If $d(w, p_S(w)) \leq h$ then also $d(a, p_S(a)) \leq h$. $BFS(a)$ is computed hence, an estimate atleast $2h + z$ is obtained because of shortest path proerties. Assume that $d(w, p_S(w)) > h$. It can also be assumed that the depth of $BFS(w)$ $d(w) < 2h + z$ since the algorithm computes $BFS(w)$ and if $s(w) \geq 2h + z$ then it computes a BFS tree of depth at least $2h + z$ which achieves the estimate guarantee. Since $d(w) < 2h + z$ it follows that $d(w, b) < 2h + z$.

Given that it is assumed $d(w, p_S(w)) > h$ and S hits neighbourhood of w with high probability, that the neighbourhood of w contains a node at distance $> h$ from w, and hence $BFS(w)$ till depth h is a subset of the neighbouhood of (w). This directly implies that there is a vertex w' in the niehgbourhood of w on the path from w to b such that $d(w, w') = h$ and hence $d(w', b) < h + z$. Since $d(a, b) = 3h + z$, we also have that $d(a, w') \geq 2h + 1$.The algorithm computes BFS for every vertex u for every u in the neighbourhood of w; Thus the estimate is obtained with at least $d(a, w') \geq 2h + 1$.

The runnnin time comes from setting the parameter $s = |S| = \sqrt{n}$, thus optimizing the time complexity $\tilde{O}(m(n/s + s))$ =$\tilde{O}(\sqrt{n}m)$

Optimization for dense and sparse graphs

Using the same techniques as the general algorithm they also present separate algorithms that work better on dense and sparse graphs respectively. In the case of dense graphs there set $|S| = s = (m/n)^{1/3}$ and obtain a time complexity of $\tilde{O}(m^{2/3}n^{4/3})$.

A similar setting of parameters and some more computation with different sets of vertices bound by another parameter h, they achieve a time complexity of $\tilde{O}(m^{2-1/(2h+3)})$.

Further Improvement

In [1], extending the results from the earlier paper,Shiri Chechik, Daniel Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, Virginia Vassilevska Williams present two deterministic algorithms that run in $O(m^{3/2})$ and $O(mn^{2/3})$ respectively with the same estimate as the previous paper. These are the first deterministic $(2 - \epsilon)$-approximation algorithms for the diameter that takes sub-quadratic time in sparse graphs.

They also consider the problem of additive c-approximation for the diameter, i.e. an estimate Dsuch that $D - c \leq \; D \leq D$. An additive n^{ϵ}-approximation. They show that for any $\delta > 0$, getting an additive n^{ϵ}-approximation algorithm for the diameter running in O$(n^{2-\delta})$ time for any $\delta > 2\epsilon$ would falsify the Strong Exponential Time Hypothesis. Thus, the algorithm might be essentially tight for sparse graphs and a subquadratic time additive c-approximation seems unlikely.

Brief Outline of n^{ϵ} Additive Approxiamtion

This is an extremely simple probabilistic algorithm

1. Sample $cn^{1-\epsilon} \log n$ nodes S uniformly at random.
2. Run Dijkstra's algorithm from and to each vertex in S.
3. Return the maximum distance found.

Proof of Correctness: The above algorithm runs in Õ$(mn^{1-\epsilon}$ time and returns a good estimate $\hat{D}$ such that $\hat{D} > D - n^{1-\epsilon}$ with high probability. The running time is obvious. The approximation guarantee: Let $\hat{D} = max_{s \in S; v \in V} max\{d(s,v), d(v,s)\}$ be the estimate that the algorithm returns. Let a and b be the end points of the diametric path. Consider the first case when the diameter D is atmost $2n^{epsilon}$. Then, consider an arbitrary vertex $s \in S$. $D \leq d(a,s) + d(s,b) \leq 2\hat{D}$, and hence $D/2 \leq \hat{D} \leq D$. However, $D/2 = D - D/2 \geq D - n^{\epsilon}$, and hence $D - n^{\epsilon} \leq \hat{D} \leq D$.

Now suppose that $D > 2n^{\epsilon}$. Then with high probability there is a node $s \in S$ such that $d(a,s) \leq n^{\epsilon}$. Thus, $d(s,b) \geq D - d(a,s) \geq D - n^{\epsilon}$, and hence again $D - n^{\epsilon} \leq \hat{D} \leq D$.

2.2.5 Diameter computation and the SETH

This section relates approximate computation to the strong exponential hypothesis. Both [2] and [1] establish that given the strong exponential hypothesis and a particular approximation factor guarantee, there is a lower bound on the time complexity of the algorithm.

SETH

In [24] and [25], Impagliazzo, Paturi and Zane presented a hypothesis so as to give new methods for establishing lower bounds for algorithms. For the sake of completeness they are defined here.

Definition

k-SAT is the problem of testing whether a Boolean formula, in conjunctive normal form(CNF) with at most k variables per clause, can be made to be true by some assignment of Boolean values to it's variables. For each integer $k \geq 2$, define a real number s_k to be the infimum of the real numbers δ for which there exists an algorithm solving k-SAT in time O$(2^{\delta n})$, where n is the number of variables in the given k-SAT instance. Then s_2 = 0, because 2-SAT can be solved in polynomial time.

Exponential Time Hypothesis (ETH): The exponential time hypothesis is the conjecture that, for every $k > 2, s_k > 0$.

Clearly, $s_3 \leq s_4 \leq s_5 \leq \cdots$, so it is equivalent to assume that $s_3 > 0$; the positivity of the remaining numbers s_k follows automatically from this assumption. The Exponential Time Hypothesis also implies that, for some constant d, $s_k \leq s_\infty 1(1 - \frac{d}{k})$[24]. In particular, the sequences k increases infnitely often.

There exist definitions that stateo the exponential time hypothesis to be a slightly weaker statement that 3-SAT cannot be solved in time $2^{o(n)}$. If there existed an algorithm to solve 3-SAT in time $2^{o(n)}$, then clearly s_3 would equal zero. However, it is consistent with current knowledge that there could be a sequence of 3-SAT algorithms, each with running time O(2in) for a sequence of numbers i tending towards zero, but where the descriptions of these algorithms are so quickly growing that a single algorithm could not automatically select and run the most appropriate one.

Strong Exponential Hypothesis (SETH): For all $\delta < 1$ there is a k such that k-CNF-SAT cannot be solved in O$(2^{\delta n})$time.

SETH implies that CNF-SAT cannot be solved in time O$(2^{\delta n})$; It cannot be solved even in time O$(n^{f(k)} 2^{\delta n})$,for any function f and $\delta < 1$. Because the numbers $s_3, s_4, s_5, \cdots$ form a monotonic sequence that is bounded above by one, they must converge to a limit s_∞. The strong exponential time hypothesis is the conjecture that the limiting value s_∞ of the sequence of numbers s_k equals one.

Multiplicative Approximation and relation to SETH

Theorem 4[2]
Let it be so that there exists a constant $\epsilon > 0$ so that there is a $(3/2 - \epsilon)$-approximation algorithm for the diameter in m-edge undirected unweighted graphs that runs in $O(m^{2\epsilon})$ time for every m. This directly implies that CNF-SAT formulas on n variables can be solved in $O^*((2\delta)n)$ time for some constant $\delta > 0$.

Proof:
As any $(3/2 - \epsilon)$-approximation algorithm can distinguish between diameter 2 and 3, the following relation to the S problem (NP-complete) is made. Given an instance of CNF-SAT on n variables and m clauses, partition the variables into two sets S_1 and S_2 on $n/2$ variables each. Create a vertex for every one of the $2^{n/2}$ partial assignments to the variables in S_1 and similarly a vertex for every assignment to the variables in S_2.

Create two nodes t_1 and t_2 and add an edge to t_i from each assignment to the variables of S_i. Create a node for every clause, and connect all clause-nodes together with t_1 and t_2 into a clique of size $m + 2$. Connect every assignment-node to the clauses that it does not satisfy.

This graph has diameter 3 if and only if there are two partial assignments, ϕ_1 to S_1 and ϕ_2 to S_2 that together form a satisfying assignment to the CNF formula, i.e. the distance between ϕ_1 and ϕ_2 inthe graph is 3 iff they form a satisfying assignment, and allother node distances are ≤ 2. The graph has $O(m + 2^{n/2})$ nodes and $O(m2^{n/2})$ edges. Since one can solve the diameter problem in $O(m^{2-\epsilon})$ time, the statement follows.

Additive approximation and relation to SETH

Theorem 4.1[1].Assuming the SETH, approximating the diameter of a graph with N edges to within an additive error of N^δ requires time $\Omega(N^{2-\epsilon})$for all $\epsilon > 2\delta$.

Proof Outline: The reduction is similar in spirit to the reduction of Roditty and Vassilevska W.[2]. The construction proceeds in two phases. In the first, a graph G' for which the diameter is 3 if ϕ is unsatisfiable and 4 if it is satisfiable, is constructed. Then, for any integer, we construct a graph G'_l by subdividing the edges of G' . G'_l has the property that if ϕ is unsatisfiable, then G'_l has diameter $3(l+1)$ and if ϕ is satisfiable, then G'_l has diameter $4(l+1)$. The construction is itself very involved and so is avoided here. Any interested readers are invited to read the full paper [1]. The construction is very interesting.

For the sake of con-tradiction, let $\epsilon > 2\delta$ such that there exists an $N^{2-\epsilon}$ time algorithm which approximates the diameter ofa graph with N edges to within additive error N^{δ}. Let

$$\gamma = \frac{2 - \left(1 + \frac{\delta}{1-\delta}\right)(2-\epsilon)}{5} > 0$$

because

$$\left(1 + \frac{\delta}{1-\delta}\right)(2-\epsilon) < \left(1 + \frac{\delta}{1-\delta}\right)(2-2\delta) = 2$$

By SETH, there exists a k such that solving k-SAT requires time $2^{(1-\gamma)n}$. Given a k-SAT instance ϕ on m clauses and n variables, let $l = (4m2^{n/2})^{\delta/(1-\delta)}$, and create the graph G'. The number of edges N is at most $2lm2^{n/2} + 2 + 2n^{n/2} < 4lm2^{n/2}$. Thus the error of the algorithm makes is at most

$$N < (4lm2^{n/2})^{\delta} = (4m(4m2^{n/2})^{\delta/(1-\delta)}2^{n/2})^{\delta} = l$$

Thus computing the diameter within an additive error of N will solve the k-SAT instance ϕ. The time it will take to solve the instance is $N^{2-\epsilon}$ as creating the graph G'_l takes only $O(N)$ time. However,

$$N^{2-\epsilon} = (4m(4m2^{n/2})^{\delta/(1-\delta)}2^{n/2})^{2-\epsilon} \leq (4m2^{n/2})^{2-5\gamma}$$

Now it is known that $m \leq (2n)^k$ (otherwise one can remove the duplicate clauses before the reduction) and so for large enough n $(4m)^2 < 2^{\gamma n}$. Thus, $(4m2^{n/2})^{2-5\gamma} < 2^{(1-2\gamma)n}$ Thi is a SETH contradiction.

2.3 New approach

2.3.1 Overview

The algorithm is fairly intuitive to start with. Partitioning the graph into clusters and then contracting the graph should improve the complexity of the All-pairs-shortest-paths algorithm, as and when computed over the contracted graph G_c. The main motivation for trying this approach is to make sure that going through every edge for every run of APSP is avoided. Through local computation of that shortest paths need to be computed. And so, the following problems arise.

1. Computing path lengths of G using APSP information from G_c. Partially solved by running **ModAPSP** instead of normal APSP.
2. The first and last clusters on the diameter are not taken into account in **ModAPSP**.
3. There could just be 1 cluster. This directly Means a diameter of $D \leq 2$.

If the diameter is 1 it can be checked with one run of counting the number of edges. If there are $n(n-1)/2$ edges, then the diameter is 1 as only a complete graph can have diameter of 1. If not we have $D \geq 2$. The irregularity of the algorithm lies in the fact that it reverse the complexity trend from sparse graphs to dense graphs. This is easily solved by crossing over to APSP whenever the complexity of the algorithm goes over APSP's.

2.3.2 Preliminaries

1. **Clusters:** A cluster C is a set of a vertices. It has a center $r(C)$ and is a collection of some of it's neighbours. The set of all the clusters in a graph will be $\mathcal{C}$. $C(v)$ is the cluster v belongs to. A vertex that is inside a cluster is *clustered*; One that is oustide any cluster is called *Unclustered.*

   In this report, the clustering, i.e partitioning the graph into clusters, is done greedily. This means that in every iteration the vertex with the maximum number of unclustered vertices is made a *center* and all it's neighbours are added to a new cluster.

2. **Contraction:** All the clusters in the given graph are 'contracted' into a vertices. V_c and E_c are the vertex set and the edge set of the contracted graph G_c with sizes $|V_c| = n_c$ and $|E_c| = m_c$ respectively.

   This can be done efficiently by taking a copy of the adjacency list and replacing all the occuerences of every vertex in a cluster by the name of the cluster. The complexity of this is O(m) as each vertex replacement takes exactly the number of steps as the number of times it appears in the adjacency list, i.e the degree of the vertex; The sum of all degrees in a graph is the size of the edge set.

3. **1-step-BFS** A check of a vertex's neighbours. This is used here to find clusters adjacent to the vertex in question and it's cluster.

   This is used to create a data structure $\mathcal{T}$ which will have objects of form $(c_i, c_j, c_k, \delta), \delta \in \{0, 1, 2\}$. The objects basically inform the algorithm about the distance between clusters c_i and c_j
   This data structure is used to compute the APSP over the contracted graph but will help output the distances between clusters in the actual graph. This is required so that the diameter can be computed without looking inside each cluster and go through all the edges that need to be avoided to get a improve the complexity bound.

4. **ModAPSP:** A modified version of APSP (Dijkstra's) informed by $\mathcal{T}$. The distance query is $d(c_j, c_k) = 1 + \delta$ where δ is from (c_i, c_j, c_k, δ) such that c_i was the last vertex to be removed. The correctness doesn't change because the only thing being modified from the APSP algorithm is the distance query. **ModAPSP** outputs the distances in the G.

2.3.3 Results

To computer the diameter approximately with an estimate $\hat{D} > max(D - 4, 2)$ of a graph with $D > 2$, a time complexity bound of O$((max(min(n^2 n_c, \delta^2 m_c), min(nn_c^2, \delta n_c m_c))$ (construction of $\mathcal{T}$) is obtained in this report. The $D > 2$ limit arises because for any graph with $D \geq 3$, one can guarantee $|\mathcal{C}| \geq 2$. Intuitively, the number of clusters increase with the sparsity of graphs; This directly implies a small edge-set between the clusters themselves. Hence this bound cannot be easily reduced to be in the form $f(n, m)$ as V_c and E_c are related inversly a graph.

To sidestep this, a family of dense graphs called near-complete graphs is constructed to show the polynomial speed up. However, the speed-up vanishes as the graph becomes sparser. This is obvious when the graph is a tree . As V_c =O(n) and E_c =O(n), O(nn_c^2) and O$(n_c^2 n)$ turn out to be O$(n * n^2)$ which is the more than standard APSP time for a tree. There could be better bounds based upon simple variants of the algorithm but such approaches seem to require a generalized analysis of the approach itself.

CHAPTER 3

The Algorithm

1. Perform clustering greedily until all vertices are covered.
2. **1-step-BFS:**
3. Make dictionaries of types $(c_i, c_j, \{v_i\})$, $(c_i, \{v_i\})$, $(v_j, \{v_i\})$. These are neighbouring vertices of clusters in another cluster. vertices inside a cluster and vertices neighbouring a clusters, and neighbouring vertices of a vertex inside it's own cluster.
4. In a cluster, get sets of neighbouring vertices of all it's neighbouring clusters $N_c(c_i)$ and make the set $M = \{\{v_j\}\}$. Find neighbouring vertices and clusters of all vertices in all elements of M. Call them M' and M''.
5. Extend the data strcutre $\mathcal{T}$ with $\{(c_i, C(v), c_j, 0) | \forall c_i, c_j \in N_c(c_i), M'\}$
6. Extend $\mathcal{T}$ with $\{(c_i, C(v), c_j, 1) | \forall (c_i, c_j) \in N_c(c_i), M''\}$
7. Mark c_i. Repeat the steps 3-5 for unmarked clusters.
8. $\forall (c_i, c_j, c_k) \notin \mathcal{T}$ such that they are on a path add $(c_i, c_j, c_k, 2)$ to $\mathcal{T}$.
9. Contract the clusters into singular vertices; Make the **contracted** graph G_c.
10. Run **ModAPSP** over G_c informed by $\mathcal{T}$. Compute max. path length d.
11. Output d =2 if and only if $d \leq 2$ and $m < n(n-1)/2$.

3.1 Analysis

With respect to a particular path p, there are 3 kinds of clusters corresponding to the number of edges of the path that are inside: 0, 1, 2. Let the respective sets be called C_{p0}, C_{p1} and C_{p2} respectively. As all the vertices were assigned to clusters, every path will have all of it's vertices in set of clusters C_p such that $\cup_{C\ C_p} C = p$.

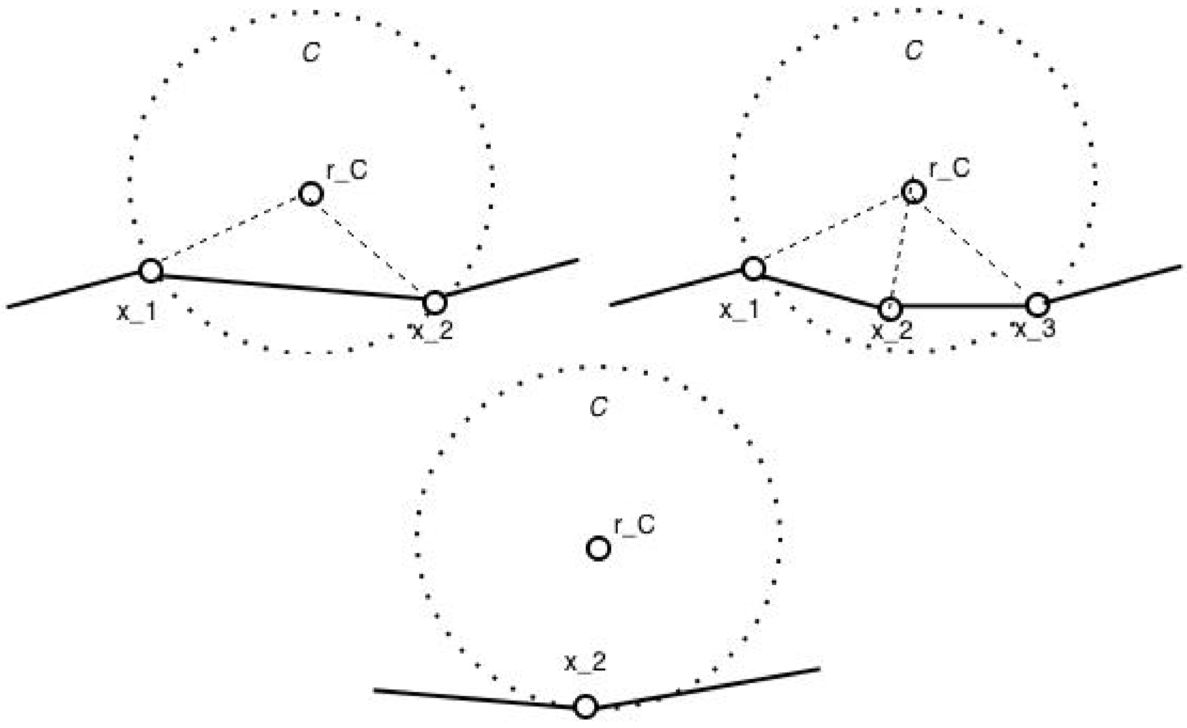

Figure 3.1: Possible clusters that hit a path

3.1.1 Construction of $\mathcal{T}$

When the **ModAPSP** over G_c is done, we need $\mathcal{T}$ so that we have the distance between clusters in G, not in G_c. For every the neighbouring clusters $N(v) \setminus \{C(v)\}$ of a single vertex v there are no edges inside the cluster. So for distance between any adjacent clusters $(C(v), c_k)$, $\delta = 0$ is added. Similarly for a cluster c_i adjacent to vertex v and another cluster c_j adjacent to one of v's vertex neighbours inside $C(v)$, we have $\delta = 1$. For any other query for adjacent c_x and c_y, we use $\delta = 2$. It would serve us best to store the object with the smallest δ for all the objects with the same (c_i, c_j, c_k).

The complexity of this part of the algorithm comes from two kinds of operations

1. Computation of vertex neighbours of vertex neighbours M. Complexity: O($|c_i|^2$).
2. Computation of the cluster neighbours of vertices in M. Complexity: O($|c_i| deg(c_i) = |c_i| d_i$).
3. Computation of the cluster neighbours of the vertices in M, which is M''. Complexity: O($|c_i| d_i$).

size of the cluster $\times$ the degree in G_c of the the cluster. Each cluster is marked after computation that requires time, ignoring log factors and d_i is degre in the contracted graph.

$$\sum_{j\in[d_i]} |c_i|(d_i - j) = O(|c_i|d_i^2)$$

This over the whole graph becomes, (where δ is maximum degree in the original graph):

$$\sum_{i\in V_c} |c_i|d_i^2 = \delta \sum d_i^2$$

This turns out to be O($\delta n_c m_c$)

Claim :

$$\sum d_i^2 = O(n_c m_c)$$

Proof: The technique here is proof by induction. The claim is true for $d_i = 1$. Following this, assume for some graph the claim is true. Adding an edge between two vertices k and l leads to the following,

$$\sum d_i^2 + 2(d_k + d_l) + 1 = O(n_c m_c) + O(n_c)$$

The proof follows by induction.

There is another way of bounding this function by bounding $d_i^2 = n_c^2$ instead of the $|c_i|$ term.

$$\sum_{i\in V_c} |c_i| = n_c^2 \sum_i |c_i| = nn_c^2$$

Similarly, bounding the time complexity O($\sum_i |c_i|^2 d_i$)= O($min(\delta^2 m_c, n_c n^2)$). We have the observations that $n > n_c$ and $m > m_c$ The complexity of this part of the algorithm is finally O($f(n, n_c, m_c, \delta)$)=:

$$\text{O}((max(min(n^2 n_c, \delta^2 m_c), min(nn_c^2, \delta n_c m_c))$$

3.1.2 Diameter of G_c

Claim 1: The output of **ModAPSP** is $D_c \geq D - 4$

Proof: It suffices to prove that after **ModAPSP**, the exact distance between two clusters in G, not

G_c, is available. After this the only edges missing from the diameter would be the edges in the first and the last clusters of the path; each of which contributes to a maximum of 2 (minimum of 1) edges extra to the diameter. The claim follows from here.

For every cluster trio on a path $(c_i, c_j, c_k,)$ we know the distance between c_i, c_k through, i.e from inside, c_j from $\mathcal{T}$. This means when **ModAPSP** querys for the distance c_j, c_k, given the last vertex on the path is c_i, the response makes sure that the distance being considered is the shortest path in G between c_i and c_k. The same applies for the next trio of vertices on the path c_j, c_k, c_l. This extends to the whole path. Thus, the claim follows.

3.1.3 Time Complexity

1. **O**(n^2). Remove all edges to clustered vertices each turn. Clustering and making Adjacency list for Clustered graph.
2. Making dictionaries is O(δm_c)
3. **O**(m) for the **1-step-BFS**. Each edge counted only twice.
4. **O**$(f(n, n_c, m_c, \delta))$. For Construction of $\mathcal{T}$.
5. **O**(m). Replace vertices in a cluster with it in the adjacency list.
6. **O**$(n_c m_c)$. APSP over contracted graph.
7. Overall time complexity.

$$\text{O}((max(min(n^2 n_c, \delta^2 m_c), min(n n_c^2, \delta n_c m_c))$$

3.2 Graph Family and Polynomial Speed-up

The algorithm works by partitioning the graph in clusters. It's trivial to observe that it works best when the graphs are dense in a way that makes sure that there are very few clusters. This in turn minimizes the edges between pairs of clusters and the time required to run the APSP over the constracted graph will be polynomiall faster. One possible setting in which this speed-up happens is when the graph has disjoint(for ease of analysis) complete subgraphs. To capture this a family of graphs is constructed that consists of subgraphs that are complete which in turn are connected by paths that can be used as a control parameter for the diameter.

3.2.1 Graph family

Let the family on n vertices be called M_n. Each graph is t complete graphs K_k on k vertices connected by k paths more per pair. These paths can be used to control the length d of the diameter D. Each vertex on these paths has degree 2 other than the vertices at either end. Each K_k pair has the corresponding vertices connected via these paths. This gives an edge set of size O(dkt^2+tk^2) over the n =O$(tk+dkt^2)$ vertices. **Note:** the parameter k controls the density of the graph.

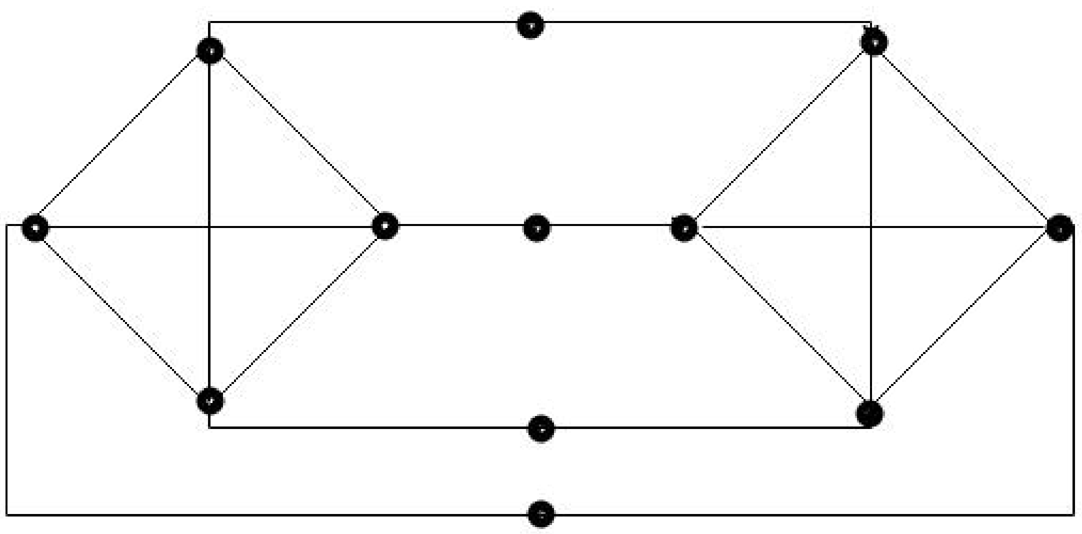

Figure 3.2: Example: graph family instance with t=2, k=4, d=1

Checking for Polynomial Speed-up

After clustering, we have n_c =O$(t + dkt^2)$ $\sim$ O(dkt^2) and m_c =O$(t^2 + dkt^2)$. **Observation:** k^2 *was eliminated; This leads to the polynomial improvement* compared to Õ$(n^{2.373})$(Fast Matrix Multiplication) and without big constants. It's compared to exact diameter computation here because the author believes that there is a non-trivial but simple step that can be added so that the approximation becomes exact computation. This is of valid concern as the APSP step is not limited for the formulation of the bound. We have $\delta = O(t+k), n_c = O(n), m_c = O(n)$: O$((max(min(n^2n_c, \delta^2 m_c), min(nn_c^2, \delta n_c m_c))$, the terms of the complexity turn out to be,

$$\tilde{O}((t+k)^2(t+dkt^2)) \sim \tilde{O}((t+k)^2(dtk^2)) \sim \tilde{O}((max(t,k))^2 n)$$

$$O(n^2 n_c) = O(nn_c^2) = O(n^2 n), O(\delta n_c m_c) = O(max(t,k)n^2)$$

To bound $t + k$ =O$(max(t, k))$ the following observations need to be made.

1. n =O$(dkt^2) = O((max(t,k))^3)$ when $t = \Theta(k)$. $\implies t + k = O(n^{\frac{1}{3}})$. This leads to O$(n^{\frac{7}{3}})$ time.
2. n =O$(dkt^2) = O(t^2)$ when $k = O(1)$. $\implies t + k = O(n^{\frac{1}{2}})$. This leads to O$(n^{2.5})$ time.
3. n =O$(dkt^2) = O(k)$ when $t = O(1)$. $\implies t + k = O(n)$. This leads to O(n^3) time.

This analysis hints that the sparser the graph gets the worse the algorithm gets. A quick comparision to Õ$(min(n^{3/2}m^{1/2}, n^{7/3})$ shows that the approach does not work well without a bound on the degree.

3.2.2 Approach to complexity in terms of n

It's hard to get from a general graph to number of clusters, which is basically a minimal dominating set. It's a natural direction to look for the bounds from the clusters point of view. As the algorithm seems to work better for dense graphs, it's natural to go a step farther and assume there are n_c clusters and n_c^2 cluster-edges. The following claim can be made:

Claim: There are $\Theta(n_c)$ clusters each with $\Theta(n_c)$ cluster-edges.

Proof: The proof is simple and follows from contradiction argument. If either the number of clusters or the number of cluster-edges incident on these is o(n_c), there are only o(n_c^2) cluster-edges in the contracted graph. This is a contradiciton. Hence, the claim follows.

Looking inside each cluster and becaue of the property of degree based clustering that is done, another claim can be made.

Claim: Each of the clusters with the property as mentioned above has $\Omega(\sqrt{n_c})$ vertices inside them.

Proof: Let's assume there are t vertices inside with a degree $> d$. Because of the degree-based clustering it's obvious $t \geq d$. If it wasn't so then another vertex would be the cluster center. So, without loss of generality one can assume that $t \geq d$. From the framing, $\Theta(n_c)$ edges to go out. This means td =O$(t^2) = \Theta(n_c)$. The claim follows.

Claim: n_c =O$(n^{\frac{2}{3}})$.

Proof: There are now $\Theta(n_c)$ clusters each with $\Omega(\sqrt{n_c})$ vertices inside. This means there are $\Theta(n_c^{3/2})$

vertices in this graphs. As this can't be more than n, we have the result that $n_c = O(n^{2/3})$.

Plugging in the time complexity for degree based bounds, we get O($max(\delta^2 n^{\frac{4}{3}}, \delta n^2)$) which is better than the [14] Dor, Helperin, and Zwick's time upto logarithmic factors when the degree of any vertex is $\Theta(\delta)$ where the following inequality is satisfied: $O(1) < \delta < n^{\frac{2}{9}}$. Considering the assumption made is not very specific, the approach seems to be paying off in the case of degree bounded that result in near-complete cluster contracted graphs.

New notion of density

The assumption in the claim above could be true even for sparse graphs with O(n) edges. So this notion of density showcases the properties of the algorithm. Trivially, there is time left after the APSP over the contracted graph is computed upto a factor of the max degree as the APSP is not the step in the algorithm that is the most time consuming. This gap can be used to get from an estimate to the exact diameter.

CHAPTER 4

Eccentricity and Periphrey Computation

The following distance-based properties are also approximately computed by the algorithm.

1. The eccentricity $\epsilon(v)$ of a vertex v is the greatest geodesic distance between v and any other vertex. It can be thought of as how far a node is from the node most distant from it in the graph.
2. The radius r of a graph is the minimum eccentricity of any vertex or, in symbols, $r = \min_{v \in V} \epsilon(v)$.
3. A central vertex in a graph of radius r is one whose eccentricity is rthat is, a vertex that achieves the radius or, equivalently, a vertex v such that $\epsilon(v) = r$.
4. A pseudo-peripheral vertex v has the property that for any vertex u, if v is as far away from u as possible, then u is as far away from v as possible. Formally, a vertex u is pseudo-peripheral, if for each vertex v with $d(u, v) = \epsilon(u)$ holds $\epsilon(u) = \epsilon(v)$.

The algorithms work because of APSP, as elicited in the following list:

1. We compute APSP and hence we know the eccentricity of every vertex to within an additive approximation of $\epsilon(v) - 4$. This is trivial to see. The rest of the problems are similarly solved. It's just a matter of storing different distances once APSP is done.
2. A direct result from the algorithm as we compute APSP leading from the last one. Results in a estimate of $\hat{r} > r - 4$. This is obvious because of the missing edges inside the polar clusters of the diameter.
3. Both periphery and pseudo-periphery are also solved by APSP in graphs. Hence, they will be solved approximately by the algorithm in report. The approximation factor is $\epsilon(u) \geq \epsilon(v) - 4$. The same applies when $\epsilon(u) > D - 4$, when u is a near-diametric vertex.

CHAPTER 5

Conclusion and Future Work

5.1 Conclusion

The approach presented in this algorithm works by partially removing the edges between clusters from consideration in APSP. This works when a notion of density of graph in terms of the edge-density of the contracted graph is set as an underlying principle. It's not hard to see that the sparser the cluster graph the worse the bound on the number of clusters and this directly leads to bad bounds on APSP. So, the algorithm could perform very differently for two different graphs with the same number of edges. The difference is in the new notion of density. It's trivial that the algorithm works best when there are very few clusters which means that there is a minimum degree based analysis that can be done if there is a direct relation of minimum degree to number of clusters. When the graph gets really dense and there are O($n^{o(1)}$) clustesrs, the algorithm asympotically hits O(n^2) time bound.

5.2 Future work

5.2.1 Exact diameter computation

As we are using a lot more time than is required by the APSP over the contracted graph, it makes sense to ask for the exact diameter rather than an approximation. A possible approach is to unravel certain clusters and run APSP over the partially unravelled graph again. This has to happen without running too many BFSs in which case it will become APSP over tha original graph anyway.

Another approach is to go path-wise and and check only the insides of certain clusters. This should be better when the paths are few in number. But the number of near-diametric paths increase(not always, but over a large increase) with density of the graph. At which point there will be a cluster that most of the paths pass through and the problem becomes finding that cluster more than finding the diametric paths. One property of the algorithm described in this report that will help in this case would be the fact that if we find any path that has had 4 edges added to it, it stops.

5.2.2 Generalization

Directed graphs

Extension of the above algorithm to directed graphs seems possible with a modification of the **1-step-bfs**. Making it the **2-step-BFS** one can check for the all the paths through a cluster. In this case there have to two sets of edges for the **2-step-bfs** to run through.

The problem presents itself when the the construction of $\mathcal{T}$ needs to happen. We need to compute the in- edges to one set of clusters and the out-edges from the other. So it's not one-sided so that we can bound the complexity the way we did. This part blows-up the complexity is attempted brute-force.

Weighted graphs

Weighted graphs are much worse off than directed graphs. The main advantage of the algorithm presesnted in this report, which is avoiding counting repeatedly the edges that fall inside the cluster, is lost when the setting is shifted to general graphs. One would have too go through every edge inside every cluster. There is not even a hint of how to extend this algorithm to general graphs.

5.2.3 Sparsification

The clustering step directly leads to a procedure for sparsification. A trivial observation leads to a $3 - multiplicative$ spanner; Keep the edges that connect all the vertices in the cluster to it's center. This results in an edge set of size O($n + m_c$) but the multiplicative stretch is bad.

A better stretch resulting in a 2-multiplicatives spanner could be acheieved if we were to keep the edges between non-adjacent clusters that are incident of a single vertex. Thi could results in a bigger edge-set size of O($\sum_{i=[n_c]} |c_i| deg(c_i)$). The denser the graph gets, the better this bound as is the spirit of the algorithm.

REFERENCES

[1] Shiri Chechik, Daniel Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, Virginia Vassilevska Williams
Better Approximation Algorithms for the Graph Diameter.
SODA 2014: 1041-1052

[2] Liam Roditty, Virginia Vassilevska Williams
Fast approximation algorithms for the diameter and radius of sparse graphs.
STOC 2013: 515-524

[3] Feodor F. Dragan
Estimating all pairs shortest paths in restricted graphfamilies: a unified approach
Journal of Algorithms 57 (2005): 1-21

[4] Virginia Vassilevska Williams:
Multiplying matrices faster than coppersmith-winograd.
STOC 2012: 887-898

[5] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani.
Fast Estimation of Diameter and ShortestPaths (Without Matrix Multiplication).
SIAM Journal on Computing, 28:1167-1181, 1999.

[6] N. Alon, Z. Galil, and O. Margalit.
On the exponent of the all pairs shortest path problem.
Journal of Computer and System Sciences, 54:255-262, 1997.

[7] T.M. Chan.
All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time.
Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM/SIAM,514-523, 2006.

[8] T.M. Chan.

More algorithms for All-Pairs Shortest Paths in weighted graphs.

Proceedings ofthe 39th ACM Symposium on Theory of Computing (STOC), ACM Press, 590-598, 2007.

[9] D.B. Johnson.

Efficient algorithms for shortest paths insparse graphs.

Journal of the ACM,24:1-13, 1977.

[10] A. Shoshan and U. Zwick.

All pairs shortest paths in undirected graphs with integer weights.

Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 605-614, 1999.

[11] Z. Galil and O. Margalit.

All pairs shortest distances for graphs with small integer length edges.

Information and Computation, 134:103- 139, 1997.

[12] R. Seidel.

On the All-Pairs-Shortest-Path problem in unweighted undirected graphs.

Journalof Computer and System Sciences, 51(3):400-403, 1995.15

[13] U. Zwick.

All Pairs Shortest Paths using bridging sets and rectangular matrix multiplication.

Journal of the ACM, 49(3):289-317, 2002

[14] D. Dor, S. Halperin, and U. Zwick.

All-pairs almost shortest paths.

SIAM J. Comput., 29(5):17401759,2000.

[15] P. Berman and S. P. Kasiviswanathan.

Faster approximation of distances in graphs.

InProc. WADS,pages 541-552, 2007.

[16] D. J. Watts and S. H. Strogatz.

Collective dynamics ofsmall-world networks.

Nature, 393:440-442, 1998.

[17] R. Albert, H. Jeong, and A.L. Barabasi.

Diameter ofthe world wide web.

Nature, 401:130 131, 1999.

[18] G. Blelloch, V. Vassilevska, and R. Williams.

A newcombinatorial approach to sparse graph problems.

InProc. ICALP, pages 108120, 2008.

[19] S. Baswana, V. Goyal, and S. Sen.

All-pairs nearly 2-approximate shortest paths in $o(n^2 poly \log n)$ time.

Theor. Comput. Sci., 410(1):8493, 2009.

[20] T. M. Chan.

All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time.

ACM Transactionson Algorithms, 8(4):34, 2012.

[21] K. Boitmanis, K. Freivalds, P. Ledins, and R. Opmanis.

Fast and simple approximation of thediameter and radius of a graph.

In WEA, pages 98-108, 2006.

[22] E. Cohen and U. Zwick.

All-pairs small-stretch paths.

J. Algorithms, 38(2):335-353, 2001.

[23] M. Elkin.

Computing almost shortest paths.

ACM Transactions on Algorithms, 1(2):283323, 2005.

[24] R. Impagliazzo and R. Paturi. *On the complexity of k-SAT.*

J. Comput. Syst. Sci., 62:367-375, 2001.

[25] R. Impagliazzo, R. Paturi, and F. Zane.

Which problems have strongly exponential complexity?

J.Comput. Syst. Sci., 63:512-530, 2001.

[26] T. Kavitha, S. Baswana, K. Mehlhorn, and S. Pettie

Efficient Construction of (α, β)-Spanners and Purely Additive Spanners.

In SODA 2005 (full version in ACM Transactions on Algorithms).

[27] Telikepalli Kavitha, Nithin M. Varma:
Small Stretch Pairwise Spanners.
ICALP (1) 2013: 601-612

[28] Marek Cygan, Fabrizio Grandoni, Telikepalli Kavitha:
On Pairwise Spanners.
STACS 2013: 209-220

[29] Telikepalli Kavitha:
Faster Algorithms for All-Pairs Small Stretch Distances in Weighted Graphs.
Algorithmica 63(1-2): 224-245 (2012)

[30] D.G. Corneil, F. Dragan, M. Habib and C. Paul,
Diameter determination on restricted graph families,
Discrete Applied Mathematics 113, 2-3 (2001), pp.143-166. (Extended Abstract in Proceedings of 24th International Workshop on Graph Theory, LNCS 1517 (Springer) (1998),pp.192-202.)