SYNOPSIS OF

A LINEAR SIZE APPROXIMATE DISTANCE ORACLE FOR CHORDAL GRAPHS

A THESIS

to be submitted by

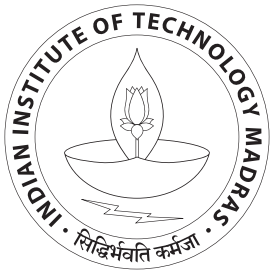
GAURAV SINGH

for the award of the degree

of

MASTER OF SCIENCE

(by Research)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

July 2014

1 INTRODUCTION

The All Pairs Shortest Path (APSP) problem is one of the most fundamental problems in graph theory. For a given graph $G = (V, E)$, where V is the set of vertices and E is the set of edges with $|V| = n$ and $|E| = m$, the *all pairs shortest path* problem on G is to compute shortest path between all pairs of vertices of G. A *distance oracle* is a data structure that stores the APSP information and can respond to queries efficiently. The time to set up the oracle, the amount of space it uses, the query response time, and update times are among the most well-studied problems in the area of data structures. Many questions still remain open and such data structures are also of immense practical relevance.

DISTANCE ORACLES

Input A graph $G = (V, E)$, where V is the vertex set of graphs and E is set of edges of graph.

Output A set $\mathcal{D}$ of data structures that can efficiently answer queries of the distance between any pair of vertices of the graph.

In most of the real world applications we are not interested in queries for all the distances, but we query for distance between some pairs of nodes more frequently than others. For example, in a big city like Chennai, distance between many pairs of addresses are of no interest to most of the people, we want to access distances between some pairs of nodes quickly. Distance oracle of small size suits best for these kind of applications.

In this thesis, we restrict our attention on *chordal graphs* with an intent to explore how the underlying graph structure can play an important role in setting up efficient distance oracles. We exploit the structural properties of clique tree representation of chordal graph to build our distance oracle. We essentially prove the following theorem.

Theorem 1 *Given a chordal graph G, it can be preprocessed in $O(n^2)$ time and $O(n^2)$ space to build a data structure of size $O(n)$ such that for any query pair $u_i, u_j \in V(G)$, a distance response upper bounded by $2d_G(u_i, u_j) + 8$ can be given in constant time.*

2 MOTIVATION

Consider a network of n nodes and m connections. Usually $m \ll n^2$. We need to extract the distance information of the network such that subsequent distance queries can be answered quickly by using this distance information. This problem can be addressed by solving APSP for the network and storing APSP matrix. The size of APSP matrix is n^2 which is too large to be stored efficiently. Also, for networks having $m \ll n^2$, the size of matrix is even larger than the network itself. Furthermore, despite being one of the oldest problems of graph theory, there does not exist any algorithm that can solve APSP in truly sub cubic time, i.e., $O(n^{3-\epsilon})$ time for some $\epsilon > 0$.

These issues have motivated researchers to design sub cubic time algorithms using distance oracles of small size that can approximately answer distances between every pairs of vertices efficiently.

Thorup et al. [15] have proved that any distance oracle for general graphs, which gives distances with stretch factor strictly less then $2k + 1$, requires $\Omega(n^{1+1/k})$ space. In particular, if the stretch is strictly less than 3, then size of the oracle must be $\Omega(n^2)$. However, for chordal graphs with diameter at least ten, our stretch is strictly less than 3 and the size of data structure is $O(n)$. Secondly, algorithms designed by both Seidel [13] and Han et al. [8] compute the square of the input graph G as a subroutine. Given a graph $G = (V, E)$, the square of the graph is defined as $G^2 = (V, E')$ where $(u, v) \in E'$ if and only if $1 \leq d_G(x, y) \leq 2$. Methods known for computing G^2 for a general graph or a chordal graph are as hard as matrix multiplication,

hence have time complexity of $O(n^{2.376})$. Therefore, computation of G^2 acts as a bottleneck for solving APSP on chordal and general graphs. Han et al.[8] also proved that computing G^2 for split graphs, which is a subclass of chordal graphs, is as hard as the problem for general graphs. In this thesis we exploit structural properties of chordal graphs to bypass matrix multiplication and achieve an $O(n^2)$ running time.

3 CONTRIBUTION OF THE THESIS

3.1 Preliminaries

A *chord* of a cycle is an edge between two non adjacent vertices of the cycle. A graph $G = (V, E)$ is *chordal* if every cycle of length greater than three has a chord. Let K_G be the set of maximal cliques of a chordal graph G. It is well known that there exists a tree T, called the clique tree of G, whose vertices correspond bijectively to the elements of K_G and for each $v \in V(G)$. We can construct a clique tree T of G in linear time [14]. For a set of cliques in G, the hitting set of the cliques is a subset of $V(G)$ that has a non-empty intersection with each clique in the given set.

Least Common Ancestors. Let T' be a rooted tree on n vertices. Harel et al. [9], Scheiber et al. [12] and Powell [10] have given three different algorithms to preprocess T' in $O(n)$ time to build a data structure D of $O(n)$ size. Subsequent queries of the least common ancestor of two vertices an be answered in constant time. We will use the algorithm presented in [12] as a subroutine in our algorithm.

Throughout this document, we assume that $G = (V, E)$ is a connected, unweighted chordal graph with $|V| = n$ and $|E| = m$ and T is a clique tree of G. Further, when we refer to a clique of a clique tree T, we actually mean the clique in G that a node in T corresponds to. For a

$u \in V(G)$, we use T_u to refer to the set of cliques in T that contain u. In other words, T_u is the subtree of T whose nodes correspond to $K_G(u)$. Finally, when we refer to a vertex $u_i \in V(G)$, we mean that the subscript i satisfies $1 \leq i \leq n$.

3.2 Distance information from clique trees

We first show the connection between the distance in the graph between u and v and the hitting set of the cliques on the path connecting T_u and T_v in T.

Lemma 2 *Let $u, v \in V$ such that $d_G(u, v) > 1$, and let $S \subseteq V$ be a minimum set of vertices that hits all the cliques in the path in T joining T_u and T_v. Then $|S| \leq d_G(u, v) - 1$, i.e., $|S|$ is upper bounded by the number of internal vertices in any shortest path joining u and v in G.*

Corollary 3 *Let u and v be non-adjacent vertices in G, and C_u and C_v be two maximal cliques of T containing them, and let S be a minimum hitting set that hits all the cliques in the path joining C_u and C_v in T. Then $|S| \leq d_G(u, v) + 1$.*

Theorem 4 *Let $S \subseteq V$ be a set of vertices that hits all the cliques in the path joining two cliques C_u and C_v in T, then there is a shortest path P in G between u and v such that P contains all the vertices of S and $|P| \leq 2|S|$.*

Proof Since S is a hitting set of the path joining C_u and C_v in T, it follows that in G there is a path from u to v of length at most $2|S|$. The factor 2 is the worst possible and is achieved in the case when S is an independent set, and since any two consecutive cliques on the path from C_u to C_v share at least one common vertex, two consecutive vertices in the independent set S can be connected by a path of length 2 via this common vertex. The hitting set vertices are now joined by a path of length at most $2|S| - 2$, and therefore, by adding edges to u and v from the

ends of this path, there is a path of length at most $2|S|$ from u to v in G. Hence the claim. ■

We now show that a minimum hitting set of cliques in the path connecting T_u and T_v can be found efficiently.

Structure of a Minimum Hitting Set. Let $P = [C_1, C_2, \ldots, C_b]$ be the path joining T_u and T_v in T, and $u \in C_1, v \in C_b$. We reiterate that C_1 and C_b are the only cliques in T_u and T_v, respectively. In this part it is useful to visualize this path as a sequence of cliques. Therefore a prefix of the path P is a prefix of the corresponding sequence. Let S be a minimum hitting set of the cliques in P. Let v_m be a vertex in C_1 such that it occurs in a maximum number of cliques in P. Consequently, there is a minimum hitting set S of cliques in P such that $v_m \in S$. Indeed this is true because if we have a minimum hitting set S' that does not satisfy this property, then we can replace $v \in S' \cap C_1$ by v_m, and we will have a minimum hitting set satisfying the desired property. This gives us an algorithm to find a minimum hitting set:

Data Structure for a Minimum Hitting Set:

Partition the cliques in P into sub-paths $IS_1, IS_2, \ldots, IS_r$ such that for $1 \leq i \leq r$, sub path IS_i is the *maximum length prefix* of cliques in $P \setminus \{IS_1, \ldots, IS_{i-1}\}$ that have a common element. To compute this maximum length prefix, we start with the empty prefix and the empty set M of common elements. Let C_j be the clique considered in the j-th iteration- clearly, it is the first clique in the remaining sequence. If $M \cap C_j = \phi$, then we have found the current prefix as the maximum length prefix, and if it is non-empty then C_j is added to the current prefix. Let M_i denote the set of common elements in $IS_i, 1 \leq i \leq r$. Consider the set $S = \{v_i \mid v_i \in M_i, 1 \leq i \leq r\}$.

Theorem 5 *The* hitting set S *computed above is a minimum hitting set of cliques in* P*. Further, it can be computed in* $O(n^2)$ *time.*

Lemma 6 *Let* C_k *be a clique on the path* P *connecting* T_u *and* T_v*, and let* P_1 *be the path from*

C_1 to C_k, and P_2 is the path from C_k to C_b. Let S, S_1 and S_2 be the minimum hitting sets for P, P_1, and P_2 respectively, then $|S| \leq |S_1| + |S_2| \leq |S| + 1$.

Lemma 6 is used to find an efficient and good approximation to the hitting set of cliques in many paths in a clique tree. The efficiency is gained by using the exact hitting set of cliques in some paths in the clique tree to calculate approximate hitting sets of cliques in other paths.

3.3 Distance oracle for chordal graphs

In this section we present the details of the data structures involved in setting up the distance oracle. The preprocessing algorithm receives a chordal graph $G = (V, E)$ as an input adjacency matrix and sets up the distance oracle. It first constructs a clique tree T of the graph G. Construction of clique tree T of a chordal graph require $O(n)$ time. These is achieved using known algorithms to construct a clique tree [7].

Vertex to Maximal Clique Mapping: This is very crucial, as the idea is to approximate the distance between two vertices u_i, u_j in the graph by the hitting set of the maximal cliques in the path connecting the corresponding maximal cliques in T. Towards this end, an n-element array is used. In this array the i-th element corresponds to u_i and it points to a maximal clique of clique tree T that contains u_i. This is achieved by inspecting each clique of T in an arbitrary order, and whenever the array entry corresponding to a vertex u_i is empty and is encountered in the current clique C_j, the array entry is updated to point to C_j. . We refer to this array as $\mathcal{H}$. To populate the array $\mathcal{H}$ the algorithm inspects each clique of the clique tree T. The size of a clique is bounded by n and also number of maximal cliques in a chordal graph is at most n. Hence it takes $O(n^2)$ time to populate $\mathcal{H}$. Further, the array $\mathcal{H}$ contains a constant size entry corresponding to each vertex, therefore, the space occupied by $\mathcal{H}$ is n.

Hitting Set Oracle for a rooted Clique Tree: We now present an algorithm to calculate the hit-

ting set of cliques on all paths originating from a single vertex in the clique tree. The algorithm is presented by way of a function definition $DFS(C, X, l)$ in Algorithm 1. This information is then used to calculate the approximate distance information efficiently on a query.

Algorithm 1 DFS(C,X,l)- Used to Set up the Hitting Set Oracle

```
1: Let i be the index of maximal clique C.
2: X ← C ∩ X
3: if X ≠ φ then
4:    HS[i] = l {/* no increase in hitting set to hit C */}
5: else
6:    HS[i] = l + 1 {/* increase in hitting set to hit C */}
7: end if
8: if C is a leaf node in T then
9:    return
10: else
11:    for each C' ∈ CHILDREN(C) do
12:      {/* Find hitting sets of cliques on paths to descendants
         of C */}
13:      if X ≠ φ then
14:        DFS(C', X, l)
15:      else
16:        X ← C'
17:        DFS(C', X, l + 1)
18:      end if
19:    end for
20: end if
```

Lemma 7 *On termination of the function call $DFS(C_r, C_r, 1)$, for each $1 \leq i \leq n$ the array $HS[i]$ has the value of the minimum hitting set of the cliques in the path from C_r to clique C_i in the clique tree T.*

Proof The correctness proof works based on Theorem 5. The proof is a direct application of Theorem 5 if the tree rooted at C_r is a path. In case the tree is not a path, let us focus on one particular node C_i in the tree. When the DFS traversal visits C_i, owing to the careful use of recursion, we can consider just the path from C_r to C_i in our analysis. Now we appeal to Theorem 5 to conclude that $HS[i]$ contains the minimum hitting set of the cliques on the path

from C_r to C_i. ■

The array HS has one element per maximal clique in G. Therefore, it has at most n elements in it. Since a DFS traversal is used to populate HS using an $O(n)$ time set intersection performed at each node, it follows that the entries of HS are calculated in $O(n^2)$ time. Further, the clique tree itself requires $O(n^2)$ space and therefore, the setup time and space of the algorithm is $O(n^2)$.

Setting up the rooted T for LCA queries: To be able to answer distance queries efficiently, our aim is to use the hitting set of the cliques on the path between a pair of cliques in T. For this, we will be using Lemma 6. In particular for the distance between u and v, we propose to use the hitting set of the cliques on the path between C_u and C_v in clique tree T. To compute the value of this hitting set, we will take the least common ancestor C of C_u and C_v in the rooted tree, and compute the hitting set of the cliques on the path from C and C_u, and similarly between C and C_v. Using these hitting set values, we apply Lemma 6. Therefore, it is crucial that we can perform LCA queries efficiently. Towards this end we preprocess the rooted tree T, using algorithm presented in [12], to build a data structure D of linear size in linear time. Using D we can compute the least common ancestor of any two cliques C_u and C_v in T in constant time.

As a consequence of the above results in Lemma 7 and the LCA algorithm in [12], we have the following theorem about the time to construct our distance oracle and the space occupied by it.

Theorem 8 *Given a chordal graph G, the data structure, HS, $\mathcal{H}$ and D use $O(n)$ space and*

as described above can be constructed in $O(n^2)$ time and $O(n^2)$ space.

3.4 Constant time response to distance queries

Algorithm 2 $Query(u_i, u_j)$- uses the Oracle $\mathcal{H}, HS, D$ as in Theorem 8

if $\{u_i, u_j\} \in E(G)$ **then**
  return 1
  /* distance is 1 in constant time by probing adjacency matrix */
end if
$p = \mathcal{H}[i]$;
$q = \mathcal{H}[j]$;
/* these are indices of two maximal cliques containing u_i and u_j */
if $p == r$ (or $q == r$) **then**
  /* r be the index of root clique C_r */
  $h_{pq} = HS[q]$ (or $HS[p, q] = HS[p]$)
  /* one of the two queried vertices is in root clique C_r */
else
  Query D with p and q.
  Let s be the least common ancestor of p and q in T.
  if $p == s$ (or $q == s$) **then**
    $h_{pq} = (HS[q] - HS[p] + 1)$ (or $HS[p, q] = (HS[p] - HS[q] + 1)$)
    /* r, p, q are in a single path */
  else
    if $s == r$ **then**
      $h_{pq} = HS[q] + HS[p]$
    else
      $h_{pq} = HS[q] + HS[p] - 2HS[s] + 2$
    end if
  end if
end if
return $d_{ij} = 2 * h_{pq}$
/* Approximate Distance Returned */

The query consists of $u_i, u_j \in V$ as input and the response is a value d_{ij} which is the approximate distance between u_i and u_j in G. The algorithm given below in Algorithm 2 uses the oracle set up earlier in this section. Recall that $\mathcal{H}[i]$ gives the index of a maximal clique in a clique cover that contains vertex u_i. Similarly, $HS[i]$ gives the size of the minimum hitting set

of the maximal cliques on path connecting C_r and C_i. Finally, D can respond to LCA queries on T in constant time.

The most important benefit of using the hitting sets of the cliques on a path between two cliques in T is that for any pair of cliques this can be computed within an additive error of 3. Using this approximate hitting set value, we can then provide an estimate of the distance, as shown in Algorithm 2.

Lemma 9 *For two cliques C_p and C_q in T, let S_{pq} denote a minimum hitting set of the cliques on the path connecting C_p and C_q in T. Then value h_{pq} calculated in Algorithm 2 is such that $h_{pq} \leq |S_{pq}| + 3$.*

Lemma 10 *For any two vertices $u_i, u_j \in V(G)$, the value d_{ij} output by Algorithm 2 is such that $d_{ij} \leq 2d_G[u_i, u_j] + 8$. Further, there is a path in G of length at most d_{ij}.*

Proof We know that $d_{ij} = 2h_{pq}$. From Lemma 9, $h_{pq} \leq |S_{pq}| + 3$. Further, from Corollary 3, we know that $|S_{pq}| \leq d_G(u_i, u_j) + 1$. Therefore, it follows that $d_{ij} \leq 2(d_G(u_i, u_j) + 1 + 3) = 2d_G(u_i, u_j) + 8$. Since there is a hitting set of size at most h_{pq} that hits the cliques on the path from C_p to C_q, from Theorem 4 it follows that there is a path of length $d_{ij} = 2h_{pq}$. Hence the lemma. ■

We have now proved all the properties of our distance oracle for chordal graphs and these complete the proof of Theorem 1.

3.5 Response to a query of shortest path

Lemma 10 proves that a path of length at most d_{ij} exists between u_i and u_j in G. We can modify the distance oracle in this paper to store entire hitting sets instead of the size of the

hitting set. We essentially keep the hitting sets of cliques on all paths originating from a single vertex in the clique tree. Now, by including clique tree also in our modified distance oracle we can respond to distance queries with paths of length at most $2d_G(u, v) + 8$. The query time would take length of the path being output. Size of the clique tree is $O(n^2)$. Also, it requires $O(n^2)$ space to keep hitting set of cliques on all paths originating from a single vertex in the clique tree. Therefore, the size of the modified distance oracle is $O(n^2)$.

4 CONCLUSION AND FUTURE WORK

4.1 Conclusion

In this work we studied distance oracle for chordal graphs. We explored the structural properties of underlying clique tree structure of chordal graphs to build a set of data structures that can answer distance queries in constant time. We proved that the hitting set for the path joining cliques C_i and C_j encodes the distance between the pair of vertices present in C_i and C_j respectively. Lemma 2 and Theorem 4 represent an interesting connection between the hitting set for a path of cliques in clique tree and the distance between the vertices present in the end cliques of the path. We also proved that by keeping the size of the hitting set of cliques on all paths originating from a single vertex in the clique tree, we can compute hitting set for the path joining any pair of cliques in constant time. This observation is important to answer queries in constant time. We essentially proved that a linear size distance oracle for chordal graphs can be build in $O(n^2)$ time and $O(n^2)$ space. Subsequent queries for the distance between any pair of vertices of the graph can be answered approximately in constant time. Finally, by maintaining the approximate hitting sets too, along with the whole clique tree, we can modify the distance oracle in this paper to use $O(n^2)$ space, and respond to distance queries with paths of length at most $2d_G(u, v) + 8$.

4.2 Future Work

A tree t-spanner of a graph can be visualized as a distance oracle of the graph that can answer distance queries with stretch of at most t. We know that the hitting set encodes the distance information of the graph. Therefore, the idea of hitting set can be further extended to compute the tree t-spanner for chordal graphs. The other area of work is to use the idea of hitting set to compute G^2 matrix for split graphs without using matrix multiplication. Due to a reduction by Han et al.[8], we know that, if G^2 matrix of a split graph can be computed efficiently then it can be efficiently computed for general graphs as well. Therefore, by using this procedure we can remove the dependency of solving APSP for unweighted general graphs [13] on matrix multiplication. Hence it will reduce the time complexity to compute APSP matrix for general unweighted graph.

5 PROPOSED CONTENTS OF THE THESIS

The outline of the thesis is as follows:

1 Introduction

- **1.1** Overview and Motivation
- **1.2** Graphs Theoretic Preliminaries
- **1.3** Contribution of the Thesis
- **1.4** Organization of the Thesis

2 Chordal Graph Family

- **2.1** Chordal Graphs
  - **2.1.1** Minimum Vertex Separator
  - **2.1.2** Perfect Elimination Ordering
  - **2.1.3** Clique Tree Structure
- **2.2** Interval and Split Graphs
- **2.3** More Subclasses of Chordal Graphs

6 PUBLICATIONS BASED ON THE THESIS

1. Gaurav Singh, N. S. Narayanswamy and G. Ramakrishna "Approximate Distance Oracle in $O(n^2)$ Time and $O(n)$ Space for Chordal Graphs", *The 25th International Symposium on Algorithms and Computation, 2014*, Jeonju, Korea (under review).

REFERENCES

[1] Surender Baswana and Telikepalli Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 591–602. IEEE, 2006.

[2] Jean RS Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.

[3] Edith Cohen and Uri Zwick. All-pairs small-stretch paths. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 93–102. Society for Industrial and Applied Mathematics, 1997.

[4] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990.

[5] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[6] Fanica Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[7] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.

[8] K Han, Chandra N Sekharan, and R Sridhar. Unified all-pairs shortest path algorithms in the chordal hierarchy. *Discrete Applied Mathematics*, 77(1):59–71, 1997.

[9] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2):338–355, 1984.

[10] Patrick Powell. further improved LCA algorithm. *Plant Genome Data and Information Center collection on computational molecular biology and genetics*, 1990.

[11] R Ravi, Madhav V Marathe, and C Pandu Rangan. An optimal algorithm to solve the all-pair shortest path problem on interval graphs. *Networks*, 22(1):21–35, 1992.

[12] Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: simplification and parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.

[13] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[14] Yukio Shibata. On the tree representation of chordal graphs. *Journal of Graph Theory*, 12(3):421–428, 1988.

[15] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.