Acta Univ. Sapientiae, Informatica **8**, 1 (2016) 16–40

DOI: 10.1515/ausi-2016-0002

A survey of the all-pairs shortest paths problem and its variants in graphs

K. R. UDAYA KUMAR REDDY
Department of Computer Science and Engineering
NMAM Institute of Technology
Nitte–574 110, India.
email: krudaykumar@nitte.edu.in

Abstract. There has been a great deal of interest in the computation of distances and shortest paths problem in graphs which is one of the central, and most studied, problems in (algorithmic) graph theory. In this paper, we survey the exact results of the static version of the all-pairs shortest paths problem and its variants namely, the Wiener index, the average distance, and the minimum average distance spanning tree (MAD tree in short) in graphs (focusing mainly on algorithmic results for such problems). Along the way we also mention some important open issues and further research directions in these areas.

1 Introduction

1.1 Motivation

The problem of finding the shortest distance between two vertices of a graph and finding a path that causes it are classic problems in graph algorithms. It appears in countless practical applications and is an important concept in transportation (and communication) engineering, computer science, network routing, network analysis [63], image processing [37, 61], operation research [19, pages 657], VLSI design [66], DNA analysis [70], bio-informatics

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 68R10
Key words and phrases: algorithms, all-pairs shortest paths, average distance, graph algorithms, MAD tree, spanning tree, Wiener index.

[39], chemical compounds [6, 31], computational geometry and robotics [33], to mention few central areas of interest. Because of its rich in applications, the work on such problems are deep and vast (in all kinds of classic graphs, directed or undirected, weighted or unweighted), in both the scientific community and engineering community. In addition, shortest path algorithms also have applications as a subroutine in other combinatorial optimization algorithms such as network flows [19, pages 708–709]. One of the basic and key problem in transportation and network analysis is the computation of the shortest paths between any two locations on a network. In computer network, the transportation could be routing messages. For instance, given a road map (or network) on which the distance is marked between every pair of adjacent cities (or nodes), find the shortest possible route between every such pair of cities (or nodes). For such a road map, we can model the graph by representing cities as vertices, road segments between cities as edges, and road distances as edge weights.

1.2 Preliminaries and notations

Let $G = (V(G), E(G))$ be a connected and simple (i.e., without loops and multiple edges) graph on $|V(G)| = n$ and $|E(G)| = m$. The graph G may be *directed* or *undirected*, and edges of G may be *weighted* or *unweighted*. If G is weighted, then the edge weights may be *real-valued* or *integers*, of either *negative* or *nonnegative*. For a weighted graph, the *weight* of a path is the sum of the weights of its edges on that path. For an unweighted graph, the weight of an edge is taken to be one. A *shortest path* between two vertices u and v is a path of minimum weight. For $u, v \in V(G)$, the *distance* between two vertices u and v, denoted by $d(u, v)$ is defined as follows. (i) Graph G is unweighted: The distance $d(u, v) = 1$, if $uv \in E(G)$, and $d(u, v) =$ the length of a shortest path joining u and v (or smallest number of edges connecting u and v), otherwise. (ii) Graph G is weighted: Here the distance $d(u, v) =$ the sum of the weights of the edges along the shortest path joining u and v.

The *single-source shortest distance* (SSSD) problem is to compute $d(s, v)$ from a given source vertex s to all other vertices v in the graph. The *single-source shortest path* (SSSP) problem is to compute the shortest paths from a given source vertex s to all other vertices in the graph. The *all-pairs shortest distance* (APSD) problem is to compute $d(u, v)$ between all pairs of vertices u, v in the graph. The *all-pairs shortest path* (APSP) problem is to compute the shortest paths between all pairs of vertices in the graph.

Given an unweighted undirected graph G, the *Wiener index* $W(G)$ of G is defined as

$$W(G) = \frac{1}{2} \sum_{u \in V(G)} \sum_{v \in V(G)} d(u, v). \tag{1}$$

The Wiener index comes under various names such as *transmission*, *total status*, *gross status*, *graph distance*, and *sum of all distances*.

For an unweighted undirected graph G, the *average distance* $\mu(G)$ of G is defined by

$$\mu(G) = \frac{1}{n(n-1)} \sum_{u \in V(G)} \sum_{v \in V(G)} d(u, v) = \frac{2W(G)}{n(n-1)}. \tag{2}$$

From (2), we see that the quantity $\mu(G)$ is closely related to $W(G)$. For a connected undirected graph G, with a nonnegative edge weight, the *Minimum Average Distance* (MAD) spanning tree of G (MAD tree in short) is a spanning tree of G with minimum average distance.

The Wiener index of vertex-weighted graphs was introduced in [54]. If G is a graph with weight function $w : V(G) \to \mathbb{N}^+$, then the Wiener index $W(G, w)$ of a *vertex-weighted graph* (G, w) is defined as

$$W(G, w) = \frac{1}{2} \sum_{u \in V(G)} \sum_{v \in V(G)} w(u)w(v)d(u, v). \tag{3}$$

Notice that if all weights of vertices in the graph are unit (or one), then $W(G, w) = W(G)$.

It is well known that there are many variants of shortest paths problem. Typically it is categorized into the SSSP problem and the APSP problem. In this short survey we consider only the APSP problem and their related problems in graphs.

A survey by Zwick [90], the exact and approximate distances in graphs are considered. In particular, the paper gives a survey on both SSSP and APSP algorithms and the related distances such as spanners (a sparse subgraph that approximates all the distances between every pair of vertices) and approximate distance oracles (concise representation of approximate distances together with quick means of extracting these approximations). Although [90] gives a survey on the APSP problem and mentions some open problems, we revisit this problem in detail and in addition we survey the recent results on general graphs as well as on restricted family of graphs. Furthermore we also give the survey on the variants of APSP problem defined above mostly focussing on the results of algorithmic computation.

We begin with the all-pairs shortest paths problem.

2 All-pairs shortest paths problem

2.1 Background

The APSP problem is undoubtedly one of the most fundamental and classical problem in graph algorithms and is well known in the research community, yet, the complexity of this problem has remained open to date even though it runs in polynomial time. Recall that the significance of the APSP problem and each of its variants defined above are rich in applications. Almost all known shortest path algorithms are computed based on the following two types of computational models:

- $\mathit{Comparison\text{-}addition}$ model: The shortest path algorithms in this model assumes the input to be real-weighted graphs, where the only operations allowed on reals are $\mathit{comparisons}$ and $\mathit{additions}$ and no other operations are allowed. These operations are assumed to take $O(1)$ time. The comparison operation finds the larger of two real numbers, whereas the addition operation generates a new real number from the existing two real numbers. Clearly, in this model, based on the outcome of previous comparisons, the algorithm chooses the next operation. Moreover, since this model fits with the assumption of real numbers which are arbitrary values, it cannot distinguish between reals and integers–hence no integer can be produced from a real variable.

- $\mathit{Random\ Access\ Machine}$ (RAM) model: Here the shortest path algorithms assume the input to be integer-weighted graphs, where integers are manipulated by additions, subtractions, comparisons, shifts, and various logical bit operations on machine words (see [1]). These operations take $O(1)$ time. In this model, each word of memory is assumed to be w-bit wide, capable of holding an integer in the range $\{-2^{w-1}, \ldots, 2^{w-1}-1\}$. Usually, it is assumed that $w = \Theta(\log n)$, which is the standard realistic assumption, where n is the input size.

Note that most of the intricate algorithms solving shortest paths problem, work in RAM model. Given the APSP problem, it is important for us to consider the implementation of this algorithm in practice. But unfortunately, some of the algorithms are far from being practical. So, there is a great deal of interest in obtaining faster algorithms in solving the APSP problem on general graphs in practice. The progress on the APSP problem have focused mostly on the following two approaches:

- $\mathsf{Combinatorial}$ approach: Combinatorial algorithms are good in practice because they rely on the efficient computation of small subproblems, but unfortunately these algorithms are typically worse in theoretical bounds. However these algorithms can give better asymptotic bounds for sparse graphs.

- $\mathsf{Algebraic}$ approach: Algebraic algorithms are far from being practical because they suffer from large hidden constants in the running time bound and large overhead of fast matrix multiplication since they rely on matrix multiplication over a ring, but fortunately these algorithms achieve good theoretical bounds. Hence these algorithms may be viewed of only theoretical interests.

Finding the APSP problem in a graph has received a considerable attention from the research community and have been studied extensively in both theory and practice. Although the APSP problem is solvable by repeated applications of SSSP algorithms, the APSP problem can be solved efficiently for arbitrary graphs as well as for other classes of problems. The various class of problems include: algorithms for restricted families of graphs, such as interval graphs (see [3, 57, 68, 75]), circular arc graphs (see [3, 75]), strongly chordal graphs (see [5, 20, 44]) etc.; algorithms for dynamic versions of the problems (see [28, 53]); parallel algorithms (see [16, 17]). For certain applications, computing exact distances (respectively shortest paths) over all pairs of vertices of a graph may be quite expensive. Also the algorithms computing exact distances using fast matrix multiplication technique are impractical since it suffers from large hidden constants. In the last decade many researchers considered $\mathsf{all\text{-}pairs\ approximate\ shortest\ paths}$ (APASP) problem [74] where a number of subcubic algorithms have been designed using a simple and novel combinatorial ideas, and also designing an optimal (quadratic) algorithms for some restricted graph classes using simple and efficient approach. Optimal algorithms for some restricted graph classes are discussed in Section 2.4. Often in practice, instead of computing exact distance (respectively shortest paths) between any pair of vertices of a graph, computing near distance (respectively shortest paths) is good enough. An algorithm computing approximate distance (or shortest path) for any given pair of vertices may have some kind of error associated with the distance (or shortest path) and this error can be $\mathsf{additive}$ (also known as $\mathsf{surplus}$) or $\mathsf{multiplicative}$ (also known as $\mathsf{stretch}$). That is, approximate distances (or shortest paths) are longer than the actual distances (or shortest paths). The $\mathsf{all\text{-}pairs\ surplus\text{-}a\ length}$, $a \geq 0$, is to compute the length of at least $d(u, v)$ and at most $d(u, v) + a$ for all pairs of vertices $u, v \in V(G)$.

The *all-pairs stretch-t length*, $t \geq 1$, is to compute the length of at least $d(u, v)$ and at most $t \cdot d(u, v)$ for all pairs of vertices $u, v \in V(G)$.

Statement of the APSP problem: Given an input graph $G = (V(G), E(G))$, the goal is to compute the distances, and construct their corresponding shortest paths between all pairs of vertices in the graph. The output is thus a distance matrix and the shortest paths that causes it.

In the following subsections, only the exact results of the static version of the APSP problem are considered.

2.2 Arbitrary weighted graph

Suppose the problem needs to report the distances (respectively shortest paths) between all pairs of vertices of the graph. Then the time complexity for such a problem must be at least $\Omega(n^2)$ since there are $\binom{n}{2} = \Theta(n^2)$ pairs of vertices. It is well known that the straightforward approach to solve the APSP problem is to run the SSSP problem for each vertex (as source) of the graph. The classical results of SSSP problems are Bellman-Ford and Dijkstra's algorithms. Using Bellman-Ford SSSP algorithm n times [19, 651–654], the APSP problem for arbitrary graphs with real-edge weights can be solved in time $O(n^2 m)$. Using Dijkstra's SSSP algorithm n times [27], and using a Fibonacci-heap implementation Fredman and Tarjan [40] (with Johnson's [51] preprocessing step if negative weights are allowed), the APSP problem (on nonnegative edge weights) can be solved in time $O(n^2 \log n + mn)$. Then Pettie [64] has shown that the first term can be reduced to $O(n^2 \log \log n)$ time. As m can be as high as $\Omega(n^2)$, the running time of the above algorithms can be as bad as $\Omega(n^3)$. However, the APSP problem can be solved efficiently without the application of SSSP algorithms. It may be noted that all of the algorithms aforementioned in this subsection work in comparison-addition model.

2.2.1 Dense real-weighted graphs

The classical Floyd-Warshall algorithm [19, 693–697] solves the APSP problem using dynamic programming technique in time $O(n^3)$. It is one the notable algorithms that is too short, simple to implement, well understood, and works better on adjacency matrices than adjacency lists. Also note that this algorithm work in comparison-addition model. Let $A = (a_{ij})$ and $B = (b_{ij})$ be two $n \times n$ matrices. Then the *distance matrix multiplication* (DMM) $C = AB$

is an $n \times n$ matrix $C = (c_{ij})$ with $c_{ij} = \min_{k=1}^{n}\{a_{ik} + b_{kj}\}$ for $1 \leq i, j \leq n$. The distance matrix multiplication (also known as the $\mathit{min\text{-}plus\ matrix\ multiplication}$) can be naively computed using $O(n^3)$ additions and comparisons. It is well known that the APSP problem is closely related to the problem of computing distance product of matrices. That is, the time complexity of DMM is asymptotically equal to that of the APSP problem for a graph with n vertices (see [1]). Fredman [41] was the first to find the possibility of obtaining a subcubic algorithm for computing the distance product of two $n \times n$ matrices using only $O(n^{2.5})$ comparisons and additions. But there is no clear specification about as to which comparisons should be made nor how to infer the result from the outcome of these comparisons; however, Fredman was able, to use his observation to obtain an explicit subcubic algorithm for the distance product, and hence for the APSP problem. His approach was based on the construction of $\mathit{decision\ tree}$ whose depth is $O(n^{2.5})$ to solve certain small-sized subproblems. However there is no known polynomial-time implementation of his algorithm. Fredman [41] somehow was able to show that the APSP problem can be solved in $O(n^3 \log^{1/3} \log n / \log^{1/3} n)$ time. Takaoka [77] presented much simpler and efficient algorithm in $O(n^3 \log^{1/2} \log n / \log^{1/2} n)$ time based on similar ideas but using $\mathit{table\ look\text{-}up}$. Dobosiewiczs [30] obtained $O(n^3 / \log^{1/2} n)$ time algorithm by exploiting a different approach $\mathit{bit\text{-}level\ parallelism}$ (i.e., simultaneous operations on $\log n$ bits contained in a single machine word). Han [46], Takaoka [78], and Zwick [92] obtained, respectively, $O(n^3 \log^{5/7} \log n \ / \log^{5/7} n)$, $O(n^3 \log^2 \log n \ / \log n)$ and $O(n^3 \log^{1/2} \log n \ / \log n)$ time algorithms and they all involved even more complicated combinations of approaches. Chan [10] obtained $O(n^3 / \log n)$ time algorithm by using somewhat different approach. The speedup of his algorithm is obtained by using a simple $\mathit{geometric}$ approach. Han [47] obtained $O(n^3 \log^{5/4} \log n / \log^{5/4} n)$ by using a more sophisticated $\mathit{word\text{-}packing\ tricks}$. Chan [12] obtained $O(n^3 \log^3 \log n / \log^2 n)$ time algorithm using the approaches from the previous algorithms by Chan [10] and by Han [47]. That is, his algorithm combines the geometric approach by Chan [10] and more sophisticated word-packing tricks by Han [47]. The paper also gives the results for APSP problem on a large class of $\mathit{geometrically\ weighted}$ graphs, where the weight of an edge is a function of the coordinates of its vertices. His approach also extends to the case of small-integer-weighted graphs which is not as good as Zwicks algorithm [91]. However, his approach more generally extends to the case where weights are taken from small set of real values yielding the first truly subcubic result ($O(n^{2.844})$ time) for APSP in real vertex-weighted graphs, as well as an improved result ($O(n^{2.688})$ time) for the $\mathit{all\text{-}pairs\ lightest\ shortest\ path}$ (among all shortest paths connecting u and v, the $\mathit{lightest\ shortest\ path}$ is a shortest path that uses smallest number of edges) problem for small integer-weighted graphs. As Tong-Wook Shinn and Tadao Takaoka mentions in [72], currently the best known upper bound is $O(n^3 \log\log n/\log^2 n)$ due to Han and Takaoka [48].

2.2.2 Dense integer-weighted graphs

Matrix multiplication is one of the most basic problems in mathematics and computer science. For a long time, the fastest algorithm for multiplying two $n \times n$ matrices was $O(n^\omega)$ time bound, where $\omega < 2.376$ [18]. Recently, Vassilevska Williams [84], improved this long-standing bound to $O(n^\omega)$ time bound, where $\omega < 2.3727$. Very recently, Le Gall [56] achieved a faster algorithm which is better than all known algorithms for rectangular matrix multiplication. Thanks to Le Gall [56] for his achievement in rectangular matrix multiplication. The results obtained in [56] are a generalization of Coppersmith-Winograds approach [18] to the rectangular setting. Moreover the algorithms presented in [56] gives an improvement for computing the product of two sparse square matrices but for the product of dense square matrices. Thus we assume that the fastest algorithm for multiplying two $n \times n$ matrices in general is $O(n^{2.3727})$ time due to [84]. In the last decade, many researchers have shown that matrix multiplication over rings [18, 76] can be used to obtain faster algorithms (i.e., subcubic algorithms) for solving the APSP problem in dense graphs with small integer edge weights, where the weights lie in the range $\{1, \ldots, M\}$ for some constant M. In this case, Galil and Margalit [42] gave an $\tilde{O}(M^{(\omega+1)/2}n^\omega)$ time algorithm for undirected graphs, and Alon $\mathit{et\ al.}$ [2] gave an $\tilde{O}(n^{(3+\omega)/2}) = O(n^{2.688})$ time algorithm for directed graphs. Then a series of subcubic algorithms [42, 71, 73, 91] have been developed. Currently, the best known algorithm for the APSP problem over directed graphs with small integer weights is in time $O(n^{2.5302})$ due to [56] (since it relies on rectangular matrix multiplication), improving over $O(n^{2.575})$ time by Zwick [91]. For undirected graphs, currently the best known APSP algorithm using fast matrix multiplication technique is $\tilde{O}(Mn^\omega)$ time bound due to Shoshan and Zwick [73].

2.2.3 Sparse graphs

Johnson [51] observed that the APSP problem can be solved efficiently on sparse graphs. That is, if the graph has negative edge weights with no negative-weight cycles, then the new set of nonnegative edge weights allows to preserve shortest paths by $\mathit{reweighting}$ technique (see [19, pages 700–704]) in time $O(mn)$. Johnson's algorithm uses the Bellman-Ford and Dijkstra's algorithms as subroutines yielding an APSP algorithm for arbitrary weighted graphs in time $O(mn+n^2 \log n)$. Later, Pettie [64] has shown that the second term can be reduced to $O(n^2 \log\log n)$ time for directed graphs, and Pettie and Ramachandran [65] has shown that the second term can be reduced to $O(n^2\alpha(m,n))$ time for undirected graphs, where α is slow growing inverse-Ackermann function.

There are also results for solving the APSP problem on integer-weighted graphs. The existing implementations of Dijkstra's algorithm on APSP [45, 81] for integer-weighted graphs run in time $O(\min\{mn\ (\log\log n)^{1/2},\ mn + n^2 \log\log n\})$. Then Thorup [80] gave bounds on APSP problem using $\mathit{hierarchy\text{-}based}$ approach for undirected graphs in time $O(mn)$. Later Hagerup [43] was able to show the bounds on APSP problem for directed graphs using hierarchy-based approach in time $O(mn + n^2 \log\log n)$. The author generalized Thorup's approach that gave a new approach to view the commonalities between all hierarchy-type algorithms and in particular, it gives a one-line characterization for all hierarchy-type algorithms. This characterization leads to prove lower bounds on their complexities in the comparison-addition model.

2.3 Arbitrary unweighted graph

2.3.1 Dense graphs

For a weighted graph G, let the edge weights of G be a unit weight. Then, the APSP problem can be solved by the simple Floyd-Warshall algorithm [19, pages 693–697] in time $O(n^3)$. Improved results show that it runs in $O(mn + n^2 \log n)$ time (Dijkstra's algorithm [19, pages 658–662], Johnson [51], Fredman and Tarjan [40]). Using Pettie algorithm [64] the APSP problem can be solved in time $O(n^2 \log\log n + mn)$. Taking G to be an unweighted graph, the algorithm for APSP can be solved by running breadth-first search (BFS) [19, pages 594–601] once from each vertex (as source) of G. This takes a time $O(n^2 + mn)$. As m can be as high as $\Omega(n^2)$, the running time of the above algorithms can be as bad as $\Omega(n^3)$. Again all of the algorithms aforementioned in this subsection work in comparison-addition model.

For unweighted undirected graphs, Galil and Margalit [42] and Seidel [71] showed that fast matrix multiplication algorithms can be used to obtain improved algorithms for the APSP problem for graphs with small integer weights. The running time of their algorithms is $\tilde{O}(n^\omega)$ (it may be noted that $\tilde{O}(n)$

$= O(n \log^k n$; that is, $\tilde{O}(n) = O(n)$ with logarithmic factors ignored). The Seidel's algorithm is simple and elegant. However, there seems to be no simple way of using Seidel's elegant technique on weighted graphs. The algorithm of Galil and Margalit [42], on the other hand, can be extended to handle on weighted (small integer weights) graphs.

There are also results for unweighted directed graphs for solving the APSP problem that use fast matrix multiplication algorithms when the edge weights are small integers. In this case, Alon *et al.*, [2] were the first to obtain a $O(n^{2.688})$ algorithm. The authors first give a simple version of the APSP problem for directed graphs with edge weights taken from the set $\{-1, 0, +1\}$. Then they have extended their algorithm to the case of edge weights which are integers with a small absolute value. Later Zwick [91] obtained a running time of $O(n^{2.575})$. Recently, Le Gall [56] has improved the result to obtain a running time of $O(n^{2.5302})$.

By exploiting *graph compression* technique, a new combinatorial idea, Feder and Motwani [38] obtained $O(mn \log(\frac{n^2}{m})/\log n)$ time, an improved algorithm that achieves log-factor speedups for solving APSP in the unweighted undirected graphs. Generally, log-factor speedups may be possible when there is some amount of redundancy or repetition in the input or computational process. As mentioned in [38], the compressed graph G^* of a graph G encodes some aspects of the structure of G and has the following properties: (i) G^* is a graph with m^* edges, where $m^* < m$. (ii) It is computationally easy to convert G into G^*, and vice versa. Therefore, the compression may be viewed as a data structure for representing the graph G.

2.3.2 Sparse graphs

Recall that the straightforward approach for solving APSP problem is to run BFS once from each vertex (source) of G which takes a time $O(n^2 + mn)$. The algorithms of Galil and Margalit [42], and Seidel [71] do not improve over the naive approach $O(n^2 + mn)$ algorithm when $m = o(n^{\omega-1})$. As mentioned above, Feder and Motwani [38] obtained $O(mn \log(\frac{n^2}{m})/\log n)$ time algorithm which is an improvement over $O(mn)$. Then, Chan [11] gave bounds that runs in $o(mn)$ time for undirected graphs. This analysis is based on one crucial parameter, m, the number of edges in the graph. That is, the results for small values of m. In [11] the time bounds of the algorithms improves over time bounds including the naive bound, Seidel's [71] algorithm, and Feder and Motwani's [38] algorithm, for all $m \ll n^{1.376}$. Later, Blelloch *et al.* [7] presented a new combinatorial data structure method that beats Feder and Motwani's and

Chan's result. The running time of their algorithm is $O(mn \log(\frac{n^2}{m})/\log^2 n)$. The ability of this new data structure lies in computing sparse vector products quickly and tolerate matrix updates. Using their data structure, the authors give best running time bounds for four fundamental graph problems: transitive closure, APSP on unweighted graphs, maximum weight triangle, and all pairs least common ancestors. The authors also point out that by using the data structure gives the first asymptotic improvement over $O(mn)$ for all pairs least common ancestors on directed acyclic graphs.

2.4 Restricted family of graphs

When dealing with special graph classes, it is well known that the algorithmic computation of the given problem on such graph classes can be solved more efficiently. These algorithms are good enough for many practical applications since graph classes are more structured than just being general graphs. Computing exact distances (respectively shortest paths) over all pairs of vertices of a graph may be quite expensive on general graphs. So, many researchers started considering the APSP problem on special graph classes. The results on some of the restricted classes of graphs are truly overwhelming because of some optimal $O(n^2)$ algorithms are known for solving the APSP problem. These include: $\mathit{interval}$ graphs, $\mathit{circular\ arc}$ graphs, planar graphs (see [49]), $\mathit{permutation}$ graphs, $\mathit{bipartite\ permutation}$ graphs, $\mathit{strongly\ chordal}$ graphs, $\mathit{chordal\ bipartite}$ graphs, $\mathit{distance\text{-}hereditary}$ graphs, and $\mathit{dually\ chordal}$ graphs. The results of APSP problem for such graph classes [33] may be consulted.

The work by Han $\mathit{et\ al.}$ [44], presents an optimal $O(n^2)$ time algorithm for solving the APSP problem on chordal graph if G^2 ($G^2 = (V, E')$, where $\{u, v\} \in E'$ if and only if $1 \leq d(u, v) \leq 2$) is known. The authors claims that computing G^2 for chordal graphs is as hard as for general graphs. They also point out that G^2 can be computed more efficiently for special classes of chordal graphs such as planar chordal, k-chordal, and strongly chordal giving rise to optimal algorithms for the APSP problem on these classes of graphs in a more natural way than the previous results. An optimal parallel algorithm for the APSP problem on chordal graphs are also presented.

The author in [33], gives a simple and efficient approach to solve all-pairs approximate shortest paths (APASP) problem on the class of weakly chordal graphs and its subclasses. Moreover, the work in [33] presents a unified approach to solve APASP and APSP problems on graph classes including chordal, strongly chordal, chordal bipartite, and distance-hereditary graphs. A few open problems related to the distances are also suggested. Later, Mondal $\mathit{et\ al.}$ [59]

and Saha $\mathit{et\ al.}$ [69] obtained optimal algorithms for solving APSP on the class of $\mathit{trapezoid}$ graphs and $\mathit{circular\ arc}$ graphs respectively. For the definition of various families of graphs [8, 33] may be consulted.

$\mathit{An\ important\ observation}$: For each of the graph classes aforementioned, the APSP problem is solved using a simple and efficient approach which are important from the practical point of view.

2.5 Concluding remarks and open issues of APSP problem

Based on the results aforementioned, we conclude this section with few open issues and remarks on computing the APSP problem.

- Whether in general there exists a truly sub-cubic algorithm for the APSP problem in the comparison-addition model that runs in time $O(n^{3-\varepsilon})$, for some constant $\varepsilon > 0$?

- On an arbitrary unweighted graphs, the fact that the fastest combinatorial algorithm for APSP problem (despite aforementioned fast non-combinatorial algorithms based on matrix multiplication) is by running BFS [19, pages 594–601] once from each vertex (as source) of a graph. Obtaining a faster combinatorial algorithm for APSP problem on such graphs in fact will be a major breakthrough.

- Recall that the APSP problem is closely related to the problem of computing distance product of matrices. Thus the most important open issue in algebraic complexity, is finding the optimal value of the exponent of square matrix multiplication. For further information on square matrix multiplication we refer [56] and references therein.

- The author in [43] gave the bounds on APSP problem for directed graphs in the word RAM model that runs in time $O(mn+n^2 \log\log n)$. It would be desirable to obtain the bound $O(mn)$.

- For undirected graphs, currently the best known APSP algorithm using fast matrix multiplication technique is $\tilde{O}(Mn^{\omega})$ time bound due to Shoshan and Zwick [73]. As mentioned in [90], the authors in [73] show that the APSP problem for undirected graphs with edge weights taken from $\{0, 1, \ldots, M\}$ is harder than the problem of computing the distance product of two $n \times n$ matrices with elements taken from the same range by at most a logarithmic factor. Thus on undirected graphs a challenging problem for the APSP problem could be, by considering larger values of M, and obtaining truly sub-cubic algorithm for it.

- A great variety of optimal ($O(n^2)$) algorithms were developed for solving the APSP problem on restricted family of graphs as mentioned in section 2.4. It would be interesting to know for which other greater graph classes the APSP problem can be solved in $O(n^2)$ time.

Next we consider the Wiener index or average distance problem.

3 Wiener index or average distance

3.1 Background

The *Wiener index* (or *Wiener number*) problem is a well-known distance based graph invariant in mathematical chemistry. The work on the theory of Wiener indices is deep and vast, in both the biochemical community and mathematical community. In chemical graph theory, the structure of a chemical compound is usually modeled as a polygonal shape–paths, trees, graphs, etc., which is often called the *molecular graph* of this compound, where each vertex represents an atom of the molecule, and covalent bonds between atoms are represented by edges between the corresponding vertices (see [4, 67, 82]). In chemistry, much of the problems have influenced the development of graph-theory-based molecular structure-descriptors called the *topological indices*. Among all the topological indices, Wiener index $W(G)$ is the most important one (from middle 1970s), thoroughly studied and frequently used (see [87]). The Wiener index have been studied in both the mathematical and chemical literature. The majority of chemical applications lie in the study of Wiener index of acyclic (molecular) graphs. Most of the prior work on Wiener indices deals with two types of problems: Wiener index problem for graphs and the inverse Wiener index problem. The Wiener index problem deals with the efficient computation of the index, the upper and lower bounds on the index values, and the relation of the Wiener index to other quantities of the graph. The inverse Wiener index problem is: Given a positive integer k, does there exist a graph whose Wiener index is k. If so, can we compute it efficiently? For more information on the Wiener index especially for trees see [31] and references therein.

The *average distance* is one of the important parameter in metric graph theory. As mentioned in [55], it has numerous applications in many areas including computer science, cheminformatics, mathematics, and recent application in phylogenetics (see [88]). One of the fundamental parameter in computer science is to measure the cost of communication in a computer network. In a network model, the time delay or signal disgradation in sending a message between any two points is often proportional to the distance a message must travel. When G represents a network, the average distance $\mu(G)$ problem can be viewed as a tool in analyzing networks since it is a measure on the time needed to traverse the messages between two randomly chosen points in the average-case performance of a network as opposed to the diameter (maximum of all shortest-path distances), which indicates the worst-case performance time [23]. Therefore, the quantity $\mu(G)$ play a significant role in analyzing communication networks and has been studied widely in [9, 23, 24, 25, 26, 29, 62, 88]. For more information on the average distance see [9, 62] and references therein.

3.2 Computation of Wiener index or average distance

The previous and ongoing work related to Wiener index or average distance are: Wiener index problem for graphs and the Inverse Wiener index problem. As a result, the research findings on such problems include obtaining upper and lower bounds, determining relationships to other quantities, determining inverse Wiener index problem, obtaining closed form expressions, and algorithmic determination of the Wiener index. In this short survey we discuss only a few important closed form expressions and the algorithmic determination of the Wiener index or the average distance.

One motivation comes from the following problem posed by F. R. K. Chung [15]: Is there an asymptotically faster algorithm for computing average distance than computing all distances between vertices of the graph?

S. Klavžar $at\ el.$ [55], mentions that the average distance can be studied equivalently as the Wiener index (or the network distance)–hence the fundamental task would be the computation of the average distance or the Wiener index efficiently. For a general graph G, one way of computing $W(G)$ or $\mu(G)$ algorithmically is to run the APSD problem; that is it can be computed in polynomial time, [58]. Thus the results of APSP problem for unweighted undirected graphs including arbitrary graphs and restricted classes of graphs follows for computing $W(G)$ or $\mu(G)$. Besides, more efficient algorithms (i.e., linear or even sub-linear algorithms) have been developed for its computation on some restricted classes of graphs. In [13] a linear algorithm were proposed for the Wiener index of benzenoid graphs, while in [14] a sub-linear time algorithm for the same problem were proposed. Some of the results on restricted classes of graphs are mentioned below.

As Dobrynin $et\ al.$ mentions in [31], the early work by Entringer $et\ al.$

[34], closed form expressions for $W(G)$ for large classes of trees are given. Among all trees of order n, the best known are $W(P_n) = \binom{n+1}{3}$ and $W(S_n)$ $= (n-1)^2$, where P_n and S_n denote the path and star of a graph of order n respectively. It is easy to see that among all trees of order n, the Wiener indices of P_n and S_n, respectively, are maximum and minimum. In [34] it has been showed that for any tree T of order n that is different from P_n and S_n, $W(S_n) < W(T_n) < W(P_n)$. Dobrynin *et al.* [31], gives a detailed survey on the results known for the Wiener index of different class of trees. For example, the authors outlines computational methods of $W(G)$, combinatorial expressions for $W(G)$, connections between $W(G)$ and the center and centroid of a tree, and connections between $W(G)$ and the Laplacian eigenvalues. Results on the Wiener indices of line graphs of trees, on trees extremal w.r.t. $W(G)$, and on integers which cannot be Wiener indices of trees are also given. The related theory and applications are also mentioned.

Hexagonal systems (HS's) are a special type of plane graphs in which all interior regions (faces) are bounded by hexagons; that is, the two hexagons either have one common edge or have no common vertex, and no three hexagons share a common edge. HS's have applications in chemistry since they provide a graph representation of benzenoid hydrocarbons. In [32], the authors outlines the results known for Wiener index W of the HS: method for computation of W (W of a HS can be computed in time $O(n)$ and a sublinear time algorithm for simple HS's), expressions relating W with the structure of the respective HS, results on HS's extremal with respect to W, and on integers that cannot be the W-values of HS's.

An interesting conjecture on Wiener index of trees: It states that except for some finite set of integers, every integer n is the Wiener index of some tree. Ban *et al.* [6], showed that enumerating all possible trees to verify this conjecture is not required. They show that searching in a small special family of trees known as caterpillars (a tree is a *caterpillar* if the deletion of all its end vertices produces a path) suffices and hence achieving the first polynomial time algorithm up to integer n to verify the conjecture. They also provide many efficient algorithms for computing trees with given Wiener indices, and implementation results show that their performance is asymptotically better than their theoretical worst-case upper bound. The authors also point out that the approaches in their paper can be used as general techniques for tree construction problems in combinatorial biology and chemistry when it is necessary to deal with tree classes.

In [24], a sharp upper bound for $\mu(G)$ of a graph is given depending on the order of a graph and the independence number (maximum size of an independent (pair-wise nonadjacent) set of vertices of G). The author answers the question posed by Erdos asking for bounds on the independence number of a graph with a given $\mu(G)$. The author also gives the upper and lower bounds on $\mu(G)$ depending on the matching number (maximum size of an independent (pair-wise nonadjacent) set of edges of G).

Dankelmann [23], gives an algorithm for computing $\mu(G)$ on an interval graph of size $o(n^2)$ in time $O(m)$ (recall that m denote the number of edges in a graph). This implies that for such a graph and size, $W(G)$ can be computed in time $O(m)$. In [23], apart from computing $\mu(G)$ of an interval graph, it is also argued that when G is a tree instead of interval graphs, $\mu(G)$ of a tree of order n can be computed in time $O(n)$ which is optimal. Thus when G is a tree, $W(G)$ can be computed in time $O(n)$.

Iyer and Reddy [85] obtained a closed-form expression for Wiener index of binomial tree. They also provide efficient algorithms for computing the Wiener indices of Fibonacci trees and binary Fibonacci trees with F_k (k-th Fibonacci number) vertices in time $O(\log F_k)$. This bound is an improvement over the bound obtained in [23]. That is, in [23] for any tree T with n vertices, $W(T)$ can be computed by an algorithm in time $O(n)$.

Although algorithms are available for average distance on strongly chordal graphs based on the APSP problem which runs in time $O(n^2)$, the author in [83] obtained a new algorithm that is not dependent on the APSP problem for average distance on strongly chordal graphs which runs in $O(n^2)$. Though the time bound is same, but the algorithm in [83] runs better than the previous algorithms on an average.

Very recently S. Klavžar $at\ el.$ [55], proposed the average distance in interconnection networks via reduction theorems for vertex-weighted graphs. Their idea is to first shrink the original graph into smaller weighted graphs called quotient graphs. Then shrink this quotient graphs further into smaller weighted graphs called reduced graphs. During this shrinking process, a part of the Wiener index of the bigger graph is added as a corresponding weight to the smaller graph. Finally, the Wiener index of the original graph is calculated by the way of the Wiener index of the weighted reduced graphs. They have also demonstrated the significance of this technique by computing the average distance of butterfly and hypertree architectures. For other results on the average distance see [55] and references therein.

3.3 Concluding remarks and open issues of Wiener index or average distance

The problem of finding Wiener index and average distance of a graph are studied extensively in the literature. For general graphs, obtaining asymptotically faster algorithm for Wiener index (or average distance) directly without the application of APSD problem of a graph was not so obvious. So, researchers have explored on restricted family of graphs to obtain a better algorithm for computing Wiener index (or average distance) than the APSD problem. Based on the results aforementioned, we conclude this section with few open issues and future research in algorithmic computation of Wiener index (or average distance) of a graph:

- Is there a genuinely asymptotically faster algorithm for computing average distance in general than computing all distances between vertices of the graph [15]?

- Dankelmann [23] gave an algorithm for computing $\mu(G)$ of an interval graph G of size $o(n^2)$ in time $O(m)$. Thus it would be interesting to know for which other graph classes $\mu(G)$ can be computed in $O(m)$ time. One of the possible candidate could be strongly chordal graphs. In [83], an algorithm for $\mu(G)$ on strongly chordal graphs obtained $O(n^2)$ time bound. An interesting problem is, whether a similar technique can be extended to obtain that runs in linear time in the size of input graph.

- In [85], the results on Fibonacci trees and binary Fibonacci trees with F_k (k-th Fibonacci number) vertices, algorithms are obtained for computing their Wiener indices in time $O(\log F_k)$. For definitions of Fibonacci trees and binary Fibonacci trees and their applications we refer [85]. It would be desirable to obtain a closed form expression for computing their Wiener indices.

- Zmazek and Žerovnik [89] gave a linear time algorithm for weighted Wiener index on weighted cactus graphs (a graph is cactus if every edge lies on at most one cycle). The authors in [55], gave the result by introducing a new technique for computing the weighted Wiener index of butterfly networks and hypertree networks. But their method is applicable only to those families of graphs that partitions the edge set of a network into components using transitive closure which in turn enables their weighted Wiener index to compute efficiently. Thus it would be interesting to determine a unified way that is applicable on different graph classes. In addition, using the newly introduced technique in [55], the open issue is, the computation of other topological indices.

Finally we consider the minimum average distance spanning tree problem.

4 MAD trees

In addition to average distance of a network, suppose if one is interested in designing a tree subnetwork of a given network such that the delay in sending a message between any two nodes of the network is minimum on the average, then such a tree can be modeled by finding the *minimum average distance spanning tree* (MAD) trees (MAD trees are also known as *minimum routing cost spanning trees*) [22]. Thus MAD trees also play an important role in communication networks [52]. It is well known that finding a minimum-cost spanning tree is one of the classic problem in algorithmic graph theory. Also there is an interest in finding the best spanning tree with important parameters such as minimum radius, minimum diameter, and minimum average distance (see [21]). In this section the following problem is considered: *Statement of the problem*: Given a connected, undirected graph G, the goal is to simply find a spanning tree of G whose average distance is minimum.

4.1 Computation of MAD trees

The problem of finding a MAD tree in general is NP-hard [52]. So the natural question that arises is, for which special graph classes a MAD tree can be found in polynomial time. As mentioned in [21], Entringer *et al.* [36] showed that there is a spanning tree whose average distance is less than twice the average distance of the original, and that such a tree can be found in polynomial time. Later, a polynomial time approximation scheme for minimum routing cost spanning trees has been developed by Wu *et al.* [86]. For further results on MAD trees can be found in [35, 36]. In [21], an linear time algorithm is exhibited for computing a MAD tree of a given distance-hereditary graph. In [22] the average distance $\mu(G, w)$ of vertex-weighted graph (G, w) is defined as follows. If G is a graph with weight function $w : V(G) \to \mathbb{R}^+$, then

$$\mu(G, w) = \binom{w(V)}{2}^{-1} \sum_{u \in V(G)} \sum_{v \in V(G)} w(u)w(v)d(u, v),$$

where $w(V)$ is the total weight of the vertices in G. Dahlhaus *et al.* [22], show that for a given interval graph G a MAD tree can be computed in time $O(m)$.

If an interval representation of an interval graph G with n vertices is given and the left and the right boundaries of intervals are sorted, then a MAD tree of G can be computed in $O(n)$ time. Dobrynin $et\ al.$ [31] conjectured the following problem: the binomial tree is a MAD tree of the hypercube H_k. Tchuente $at\ el.$ [79] made a step towards the proof of this conjecture by showing that the binomial tree B_k is a local optimum with respect to the 1-move heuristic (A 1-move consists of adding to a tree H, an edge e and removing an edge $e^{'}$ from the unique cycle created by e).

Recently, Jana and Mondal [50], and Mondal [60], obtained efficient algorithms for computing MAD tree on permutation graphs and trapezoid graphs respectively in $O(n^2)$ time bound based on breadth-first tree.

4.2 Concluding remarks and open issues of MAD trees

Although the problem of finding a MAD tree is NP-hard in general, but fortunately there is a possibility of obtaining a polynomial time algorithm for restricted family of graphs. Based on the results aforementioned, we conclude this section with few open problems and future research in computing a MAD tree:

- Is there a polynomial time algorithm to find a MAD tree of a vertex weighted interval graph [21]? The authors in [21] point out that using their existing technique, the possible candidates for future research could be strongly chordal graphs.
- It would also be interesting to know for which other restricted graph classes a MAD tree can be computed in polynomial time. Because the result on distance-hereditary graphs is known in polynomial time, so apart from the strongly chordal graphs, the other possible candidate could also be chordal bipartite graphs.
- Hypercube is a well known and popular interconnection network for multicomputers. The question whether the binomial tree is a MAD tree of the hypercube remains open (see [31]) even though Tchuente $at\ el.$ [79] made a step towards this open problem.

5 Conclusions

In this paper, we reviewed studies on the all-pairs shortest paths problem and its variants namely, the Wiener index, the average distance, and the minimum average distance spanning tree (MAD tree) problem in graphs. Each of these problems are undoubtedly fundamental and classical problems in graph algorithms and is well known in the research community. The significance of such problems are rich in applications. From the perspective of all-pairs shortest paths problem, the discussions are focused on the exact results of the static version. From the perspective of Wiener index, average distance, and MAD trees, the discussions are focused mostly on the algorithmic computation of such problems. Finally, under each section, some of the major open issues and future research are discussed on algorithmic determination and obtaining closed form expressions for each of such problems.

References

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesely, Reading, 1974. ⇒19, 22

[2] A. Alon, Z. Galil, O. Margalit, go , *J. Comput. Sys. Sci* **54,** 2 (1997) 255–262. ⇒23, 25

[3] M. J. Atallah, D. Z. Chen, D. T. Lee, An optimal algorithm for shortest paths on weighted interval and circular arc graphs, with applications, *Proc. European Sympos. on Algorithms*, *Lecture Notes in Computer Science* **726** (1993) 13–24. ⇒20

[4] A. T. Balaban, Applications of graph theory in chemistry, *J. Chem. Inf. Comput, Sci.* **25,** 3 (1985) 334–343. ⇒28

[5] V. Balachandhran, C. Pandu Rangan, All-pairs-shortest length on strongly chordal graphs, *Discrete Appl. Math.* **69,** 1–2 (1996) 169–182. ⇒20

[6] Y. A. Ban, S. Bereg, N. H. Mustafa, A conjecture on Wiener indices in combinatorial chemistry, *Algorithmica* **40,** 2 (2004) 99–117. ⇒17, 30

[7] G. E. Blelloch, V. Vassilevska, R. Williams, A new combinatorial approach for sparse graph problems, *Proc. International Colloquium on Automata, Languages and Programming* **35** (2008) 108–120. ⇒25

[8] A. Brandstädt, V. B. Le, J. P. Spinrad, *Graph Classes: A Survey*, *SIAM Monogr. Disc. Math. Appl.* 1999. ⇒27

[9] R. M. Casablanca, Average distance in the strong product of graphs, *Util. Math.* **94** (2014) 31–48. ⇒29

[10] T. M. Chan, All-pairs shortest paths with real weights in $O(n^3/\log n)$ time, *Proc. 9th Workshop Algorithms Data Struct.*, *Lecture Notes in Computer Science* **3608** (2005) 318–324. Also available in *Algorithmica* **50,** 2 (2008) 236–243. ⇒22

[11] T. M. Chan, All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time, *Proc. ACM-SIAM Sympos. Discrete Algorithms* **17** (2006) 514–523. ⇒25

[12] T. M. Chan, More algorithms for all-pairs shortest paths in weighted graphs, In *Proc. 39th ACM Symposium on Theory of Computing (STOC)* (2007) 590–598. ⇒22

[13] V. Chepoi, S. Klavžar, The Wiener index and the Szeged index of benzenoid systems in linear time, *J. Chem. Inf. Comput. Sci.* **37,** 4 (1997) 752–755. ⇒29

[14] V. Chepoi, S. Klavžar, Distances in benzenoid systems: further developments, *Discrete Math.* **192,** 1–3 (1998) 27–39. ⇒29

[15] F. R. K. Chung, The average distance and the independence number, *J. Graph Theory* **12,** 2 (1988) 229–235. ⇒29, 32

[16] E. Cohen, Using selective path-doubling for parallel shortest-path computations, *J. Algorithms* **22,** 1 (1997) 30–56. ⇒20

[17] E. Cohen, Polylog-time and near-linear work approximation scheme for undirected shortest paths, *J. ACM* **47,** 1 (2000) 132–166. ⇒20

[18] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, *J. of Symb. Comput.* **9,** 3 (1990) 251–280. ⇒23

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms* (3rd edition), The MIT Press, 2009. ⇒16, 17, 21, 24, 27

[20] E. Dahlhaus, Optimal (parallel) algorithms for the all-to-all vertices distance problem for certain graph classes, *Proc. of the International Workshop Graph-Theoretic Concepts in Comput. Sci.*, *Lecture Notes in Computer Science* **657** (1992) 60–69. ⇒20

[21] E. Dahlhaus, P. Dankelmann, W. Goddard, H. C. Swart, MAD trees and distance-hereditary graphs, *Discrete Appl. Math.* **131,** 1 (2003) 151–167. ⇒33, 34

[22] E. Dahlhaus, P. Dankelmann, R. Ravi, A linear-time algorithm to compute a MAD tree of an interval graph, *Inf. Process. Lett.* **89,** 5 (2004) 255–259. ⇒33

[23] P. Dankelmann, Computing the average distance of an interval graph, *Inform. Process. Lett.* **48,** 6 (1993) 311–314. ⇒29, 31, 32

[24] P. Dankelmann, Average distance and independence number, *Discrete Appl. Math.* **51,** 1–2 (1994) 75–83. ⇒29, 30

[25] P. Dankelmann, Average distance and domination number, *Discrete Appl. Math.* **80,** 1 (1997) 21–35. ⇒29

[26] P. Dankelmann, Average distance in weighted graphs, *Discrete Math.* **312,** 1 (2012) 12–20. ⇒29

[27] E. Dijkstra, A note on two problems in connexion with graphs, *Numerische mathematik* **1,** 1 (1959) 269–271. ⇒21

[28] C. Demetrescu, G. F. Italiano, Fully dynamic transitive closure: breaking through the $O(n^2)$ barrier, *Proc. IEEE Sympos. on Found. Comput. Sci.* **41** (2000) 381–389. ⇒20

[29] J. K. Doyle, J. E. Graver, Mean distance in a graph, *Discrete Math.* **17,** 2 (1977) 147–154. ⇒29

[30] W. Dobosiewicz, A more efficient algorithm for the min-plus multiplication, *Int. J. Comput. Math.* **32,** 1–2 (1990) 49–60. ⇒22

[31] A. A. Dobrynin, R. Entringer, I. Gutman, Wiener index of trees: theory and applications, *Acta Appl. Math.* **66,** 3 (2001) 211–249. ⇒17, 28, 29, 30, 34

[32] A. A. Dobrynin, I. Gutman, S. Klavžar, P. Žigert, Wiener index of hexagonal systems, *Acta Appl. Math.* **72,** 3 (2002) 247–294. ⇒30

[33] F. F. Dragan. Estimating all pairs shortest paths in restricted graph families: a unified approach. *J. of Algorithms* **57,** 1 (2005) 1–21. ⇒17, 26, 27

[34] R. C.Entringer, D. E. Jackson, D. A. Synder, Distance in graphs, *Czech. Math. J.* **26,** 2 (1976) 283–296. ⇒30

[35] R. C. Entringer, Distance in graphs: trees, *J. Combin. Math. Combin. Comput.* **24** (1997) 65–84. ⇒33

[36] R. C. Entringer, D. J. Kleitman, L. A. Sžekely, A note on spanning trees with minimum average distance, *Bull. Inst. Combin. Appl.* **17** (1996) 71–78. ⇒33

[37] A. X. Falcao, J. K. Udupa, S. Samarasekera, S. Sharma, B. E. Hirsch, R. A. Lotufo, User-steered image segmentation paradigms: Live wire and live lane, *Graphical Models and Image Process.* **60,** 4 (1998) 233–260. ⇒16

[38] T. Feder, R. Motwani, Clique patritions, graph compression and speeding-up algorithms, *J. Comput. Sys. Sci*. **51,** 2 (1995) 261–272. ⇒25

[39] A. M. Fitch, and M. B. Jones, Shortest path analysis using partial correlations for classifying gene functions from gene expression data, *Bioinformatics* **25** (2009) 42–47. ⇒17

[40] M. Fredman, R. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM* **34,** 3 (1987) 596–615. ⇒21, 24

[41] M. L. Fredman, New bounds on the complexity of the shortest path problem, *SIAM J. Comput*. **5,** 1 (1976) 49–60. ⇒22

[42] Z. Galil, O. Margalit, All pairs shortest distances for graphs with small integer length edges, *Inform. Comput.* **134,** 2 (1997) 103–139. ⇒23, 24, 25

[43] T. Hagerup, Improved shortest paths on the word RAM, *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, *Lecture Notes in Computer Science* **1853** (2000) 61–72. ⇒24, 27

[44] K. Han, C. N. Sekharan, R. Sridhar, Unified all-pairs shortest path algorithms in the chordal hierarchy, *Discrete Appl. Math.* **77,** 1 (1997) 59–71. ⇒20, 26

[45] Y. Han, M. Thorup, Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space, *Symp. on Found. of Comput. Sci.* **43** (2002) 135–144. ⇒24

[46] Y. Han, Improved algorithm for all pairs shortest paths, *Inform. Process. Lett.* **91,** 5 (2004) 245–250. ⇒22

[47] Y. Han, An $O(n^3(\mathrm{loglog}n/\mathrm{log}n)5/4)$ time algorithm for all pairs shortest paths, *Proc. European Sympos. Algorithms, Lecture Notes in Computer Science* **4168** (2006) 411–417. ⇒22

[48] Y. Han, T. Takaoka, An $O(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths, *Proc. 13th SWAT*, *Lecture Notes in Computer Science* **7357** (2012) 131–141. ⇒23

[49] M. R. Henzinger, P. Klein, S. Rao, S. Subramanian, Faster shortest-path algorithms for planar graphs, *J. Comput. System Sci.* **55,** 1 (1997) 3–23. ⇒26

[50] B. Jana, S. Mondal, Computation of a minimum average distance tree on permutation graphs, *Annals of Pure and Appl. Math.* **2,** 1 (2012) 74–85. ⇒34

[51] D. Johnson, Efficient algorithms for shortest paths in sparse graphs, *J. ACM* **24,** 1 (1977) 1–13. ⇒21, 23, 24

[52] D. S. Johnson, J. K. Lenstra, A. H. G. Rinnooy-Kan, The complexity of the network design problem, *Networks* **8,** 4 (1978) 279–285. ⇒33

[53] V. King, Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs, *Proc. IEEE Sympos. on Found. of Comput. Sci.* **40** (1999) 81–89. ⇒20

[54] S. Klavžar, and I. Gutman, Wiener number of vertex-weighted graphs and a chemical applications, *Discrete Appl. Math.* **80,** 1 (1997) 73–81. ⇒18

[55] S. Klavžar, P. Manuel, M. J. Nadjafi-Arani, R. Sundara Rajan, C. Grigorious, S. Stephen, Average distance in interconnection networks via reduction theorems for vertex-weighted graphs, *To appear* ⇒28, 29, 31, 32, 33

[56] F. Le Gall, Faster algorithms for rectangular matrix multiplication, *Proc. 53rd FOCS* (2012) 514–523. ⇒23, 25, 27

[57] P. Mirchandani, A simple $O(n^2)$ algorithm for the all-pairs shortest path problem on an interval graph, *Networks* **27,** 3 (1996) 215–217. ⇒20

[58] B. Mohar, T. Pisanski, How to compute the Wiener index of a graph, *J. Math. Chem.* **2** (1988) 267–277. ⇒29

[59] S. Mondal, M. Pal, T. K. Pal, An optimal algorithm for solving all-pairs shortest paths on trapezoid graphs, *Int. J. Comp. Eng. Sci.* **3,** 2 (2002) 103–116. ⇒26

[60] S. Mondal, An efficient algorithm for computation of a minimum average distance tree on trapezoid graphs, *J. Scientific Research and Reports* **2,** 2 (2013) 598–611. ⇒34

[61] E. N. Mortensen, W. A. Barrett, Interactive segmentation with intelligent scissors, *Graphical Models and Image Process.* **60,** 5 (1998) 349–384. ⇒16

[62] S. Mukwembi, Average distance, independence number, and spanning trees, *J. Graph Theory* **76,** 3 (2014) 194–199. ⇒29

[63] W. Peng, X. Hu, F. Zhao, J. Su, A fast algorithm to find all-pairs shortest paths in complex networks, *Procedia Comp. Sci.* **9** (2012) 557–566. ⇒16

[64] S. Pettie, A new approach to all-pairs shortest paths on real-weighted graphs, *Theoret. Comput. Sci*. **312,** 1 (2004) 47–74. ⇒21, 24

[65] S. Pettie, V. Ramachandran, A shortest path algorithm for real-weighted undirected graphs, *SIAM J. Comput.* **34,** 6 (2005) 1398–1431. ⇒24

[66] S. Peyer, D. Rautenbach, J. Vygen, A generalization of Dijkstras shortest path algorithm with applications to VLSI routing, *J. Discrete Algorithms* **7,** 4 (2009) 377–390. ⇒16

[67] M. Randic, Chemical graph theory-facts and fiction, *Ind. J. Chem.* **42A** (2003) 1207–1218. ⇒28

[68] R. Ravi, M. V. Marathe, C. Pandu Rangan, An optimal algorithm to solve the all-pair shortest path problem on interval graphs, *Networks* **22,** 1 (1992) 21–35. ⇒20

[69] A. Saha, M. Pal, T. K. Pal, An optimal parallel algorithm for solving all-pairs shortest paths problem on circular-arc graphs, *J. Appl. Math. and Computing* **17** 1 (2005) 1–23. ⇒27

[70] J. P. Schmidt, All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings, *SIAM J. Comput.* **27,** 4 (1998) 972–992. ⇒16

[71] R. Seidel, On the all-pairs shortest path problem in unweighted undirected graphs, *J. Comput. Sys. Sci.* **51,** 3 (1995) 400–403. ⇒23, 24, 25

[72] T. Shinn, T. Takaoka, Combining all pairs shortest paths and all pairs bottleneck paths problems, *Lecture Notes in Computer Science* **8392** (2014) 226–237. ⇒23

[73] A. Shoshan, U. Zwick, All pairs shortest paths in undirected graphs with integer weights, *Proc. IEEE Sympos. on Found. of Comput. Sci.* **40** (1999) 605–614. ⇒23, 27

[74] C. Sommer, *Approximate shortest path and distance queries in networks*, PhD thesis, The University of Tokyo, 2010. ⇒20

[75] R. Sridhar, D. Joshi, N. Chandrasekharan, Efficient algorithms for shortest distance queries on interval, directed path and circular arc graphs, *Proc. Int'l. Conf. on Comput. and Inf.* **5** (1993) 31–35. ⇒20

[76] V. Strassen, Gaussian elimination is not optimal, *Numer. Math.* **13,** 4 (1969) 354–356. ⇒23

[77] T. Takaoka, A new upper bound on the complexity of the all pairs shortest path problem, *Inform. Process. Lett.* **43,** 4 (1992) 195–199. ⇒22

[78] T. Takaoka, A faster algorithm for the all-pairs shortest path problem and its application, *Proc. 10th Int. Conf. Comput. Comb.*, *Lecture Notes in Computer Science* **3106** (2004) 278–289. ⇒22

[79] M. Tchuente, P. M. Yonta, J. Nlong II, Y. Denneulin, On the minimum average distance spanning tree of the hypercube, *Acta Appl. Math.* **102,** 2–3 (2008) 219–236. ⇒34

[80] M. Thorup, Undirected single-source shortest paths with positive integer weights in linear time, *J. ACM* **46,** 3 (1999) 362–394. ⇒24

[81] M. Thorup, Integer priority queues with decrease key in constant time and the single source shortest paths problem, *J. Comp. Sys. Sci.* **69,** 3 (2004) 330–353. ⇒24

[82] N. Trinajstic, *Chemical Graph Theory*, CRC Press, Boca raton, FL, 1992. ⇒28

[83] K. R. Udaya Kumar Reddy, Computing average distance on strongly chordal graphs, *National J. Tech.* **8,** 1 (2012) 26–35. ⇒31, 32

[84] V. Vassilevska Williams, Multiplying matrices faster than Coppersmith-Winograd, *Proc. 44th ACM Symposium on Theory of Computing* (to appear, 2012). ⇒23

[85] K. V. Iyer, K. R. Udaya Kumar Reddy, Weiner index of binomial trees and Fibonacci trees, *Int. J. Math. Engg. with Comp.* **1** (2010) 27–34. Also available at http://arxiv.org/abs/0910.4432. ⇒31, 32

[86] B. Y. Wu, G. Lancia, V. Bafna, K. M. Chao, R. Ravi, C. Y. Tang, A polynomial-time approximation scheme for minimum routing cost spanning trees, *SIAM J. Comput.* **29,** 3 (2000) 761–778. ⇒33

[87] L. Xu, X. Guo, Catacondensed hexagonal systems with large Wiener numbers, *MATCH Commun. Math. Comput. Chem.* **55,** 1 (2006) 137–158. ⇒28

[88] S. J. Xu, R. Gysel, D. Gusfield, Minimum average distance clique trees, *SIAM J. Discrete Math.* **29,** 3 (2015) 1706–1734. ⇒28, 29

[89] B. Zmazek, J. Žerovnik, Computing the weighted Wiener and Szeged number on weighted cactus graphs in linear time, *Croat. Chem. Acta.* **76,** 2 (2003) 137–143. ⇒32

[90] U. Zwick, Exact and approximate distances in graphs: a survey, *Proc. European Symp. on Algorithms* **9** (2001) 33–48. ⇒18, 27

[91] U. Zwick, All-pairs shortest paths using bridging sets and rectangular matrix multiplication, *J. ACM* **49,** 3 (2002) 289-317. ⇒22, 23, 25

[92] U. Zwick, A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths, *Proc. Int. Sympos. Algorithms and Computation, Lecture Notes in Computer Science* **3341** (2004) 921–932. ⇒22

Received: February 13, 2016 • Revised: March 18, 2016