MINAR International Journal of Applied Sciences and Technology
ISSN: 2717-8234
Article type: Research Article

Received: 23/03/2023 **Accepted:** 22/04/2023 **Published:** 01/06/2023

FRAMEWORK AND TOPOLOGY OF SHORTEST ROUTE PROBLEM IN A MAZE UNDIRECTED GRAPH

Khalid Abdulkalek ABDU [1]

Al-Iraqia University, Iraq

Saad M. Saleh AHMED [2]

University of Baghdad, Iraq

Abstract

The problem of serving the on-time client query of shortest path between two vertices in road network has always been solved using an algorithm that avoid reading all input of the large graph because of the limited capacity of memory and time. Such a method does not always give the realistic shortest path and cannot be used for all types of the shortest path problem that consists of vertex to vertex, all pairs and single-source shortest problems. The realistic shortest path is obtained only when all vertices are visited. The memory requirement and the speed of the computation are challenges that need to be considered. Paths in graphs, represented in terms of sets obtained from applying topology on the edges, are all identified in the undirected graph. Therefore, no edge is possibly distracted or left by searching. Such data of sets contains the realistic shortest path. We presented a framework that includes applying the edge topology on the undirected graph and the process of the finding the shortest path from a large data are presented and discussed. The provision of an immediate response to the client query is proposed within a simple framework that consists of the Internet webserver. The evaluation of the proposal, the topology with the framework, showed that it was successful and applicable and steered the researches towards finding the best and fastest service for the problems through the offline techniques of graph data manipulations at the webserver.

Keywords: *shortest route problem, graph topology, undirected graph, framework.*

http://dx.doi.org/10.47832/2717-8234.15.13

[1] khalid.abdu@aliraqia.edu.iq

[2] saad.mohammed@hiuc.edu.iq

Introduction

The determination of the shortest route in graphs has still been one of the problems that need to be solved with more efficient method. In graphs such as those for road network, GPS service is provided to drivers (clients) for their queries of the shortest path or the least-duration arrival to arrive the destination. Searching and finding the shortest path between two points (or nodes) was always chosen to be solved by an algorithm that fans out the graph and implemented in the handheld GPS or mobile device. This scheme was proposed a few decades ago because of the limited capacity and slow processes of computations. Dijkstra algorithm [1] is one of the basic algorithm that was used and modified to find the shortest path between two nodes. Such algorithms don't need to read all nodes of the network but only those in the way between two nodes. Therefore, its technique is based on choosing the next node till it arrives the destination. Such a technique cannot ensure that the other next nodes are connected to the destination (cut-off) and when it is connected there is no guarantee whether the chosen path is exactly shortest. However the backtrack is a good technique to avoid the cut-off, it slows down the process. And, although more than one path was found and the shortest one was chosen, it is still no guarantee whether the shortest path was found or not. Since, in every step, one next (nearest) vertex out of others is chosen. This step leads to one path (the current one is being found by the algorithm) and leaves other paths. In the next steps, it is not certain whether the next nodes of the current path are shorter than the nodes of the other paths left because the comparison and the choice is based on only the next edge but not the whole path. This is why some modified the algorithm by using backtracks and trying other paths to be compared [2–6]. In real road networks (large graphs), so many paths exist between two nodes when they are relatively apart. If the backtrack is fully extended to extract all possible node-node paths then the shortest is chosen using comparisons between them, it takes too long to finish and needs large memory that is not available in the recent devices. Therefore, the size of memory and the speed of processing were the two factors that should be optimized to serve the real-time node-node query.

Since several decades, intensive research works have been developed to serve real-time node-node queries. Dantzig (1960) developed an algorithm by saving the least distance between the nodes and not to revisit them again to decrease the computations [2]. The probability theory based on people rumour was used to speed up the algorithm that finds all-pairs shortest paths [7]. Some targeted decreasing the number of the visited nodes during the implementation of the algorithm using the landmarks on roads or using pre-processing data that has the weight of every edge between every two nodes. So when the weight (length) of an edge is over the budget it is distracted. A* search algorithm (ATL) was proposed by [8] to solve the point to point (P2P) problem with searching only a smaller portion of the graph than Dijkstra algorithm. ATL algorithm for finding P2P in a road network based on landmarks and with Euclidean bounds was proposed by [8]. The memory needed and suggested for pre-processing used in this algorithm is limited and linearly increasing with the graph size. However the landmarks helps to decrease the duration of the applying the algorithm, these landmarks are only on the main roads and not available everywhere especially inside the districts. The use of external memory such as a flash memory or sd-card for the pre-processing data is still slow when the graph is very large and the use of internal memory (built-in) is faster. Ding et al. (2008) proposed an algorithm to find the minimum time of travel from a point to another point in a large graph (road network) and discussed the storage model to suggest an easier implementation in a data system [9]. Goldberg and Werneck (2005) studied the shortest path problem, from point to point, for road network using ALT algorithm. They also suggested an external memory, flash memory, for the graph data and the RAM in the PC pocked for the visited graphs of the current shortest problem [10]. Ananthanarayanan et al. 2017 proposed a multi-pass sequential localized search technique for the path planning to avoid the complexity of manipulators for the shortest path in real-time due to obstacles [11]. So many researches, not mentioned here, targeted the improvement of the algorithm for the shortest problem and its derivations. In all works, the suggested algorithms are used to compensate the lack of memory and the low speed of computations. Therefore, the solution of the shortest path problem is not optimally solved as we mentioned above. The classic Dijkstra's algorithm, which has been around for several decades, is incapable of handling on-line queries with acceptable response time for massive road networks [12]. Even when an algorithm with some suggestions about the memory increment is succeeded to find the shortest path, it will not be applicable for the single-source shortest problem [8]**.** All algorithms proposed and applied were not evaluated because their results cannot be compared with the real shortest that was not available. The realistic shortest path is found when only all paths and their weights are found and then the shortest is chosen.

It is important to remember that the all pairs and node-node problems are essential similar [13]. In fact, the node-node problem is also a part of the single-source problem that finds all paths between one node and all other nodes. When all paths between every two nodes (Node-node) in the graph are found the problem is called all-pairs shortest path. When all-pairs shortest problem is solved and stored as data the node-node problem is solved because it is a part of it and solution already exist in the data. But this problem was always avoided because it needs reading all data of the graph (nodes) whereas very small data is needed by the algorithm in the shortest path between only two nodes. Such a process takes too long to be applicable in the clients' devices. Even the long duration is ignored the replication of the data of the graph is still a serious issue. Then a too large amount of memory is needed, which are not available in the mobiles or the GPS devices.

Very recently, most works have also considered the problem using. A novel algorithm for the shortest path have been proposed. The length of a path is represented by an interval value that is represented by neutrosophic number, trapezoidal and triangular interval [14]. A new approach for the one-way road network using genetic algorithm has been adapted. It is based on searching out and determining appropriate directions [15]. Zhang, Z and Li, M

(2023) have developed a model for a real traffic jam travel to estimate the travel time with least uncertainty within an approach of sophisticated route planning. The stochastic and time-varying model was based on empirical observations [16]. Delivery time issue has been considered by Maha G. et al. (2021) to propose a solution with providing time windows [17]. The approach is based on a tabu search heuristic of several short paths withing 200 nodes and 580 arcs. A BCloud-IFog framework have been proposed by Liu Q. et al (2019) to include the usability and intelligence in large route planning for large network. From previous works, no recent works gave up the use of algorithms for the shortest path problems. Our proposal is to solve the problem without using the algorithms with perfect solutions.

Update data broadcasting centrally at the road server has recently been one of the suggestions as approach for processing the queries of workload to the clients making the system more scalable and fault tolerant. It is assumed that the road server has much more powerful computation capability and larger memory size than the clients [12]. But such a server with all equipment and installations for the broadcast to only road network is too expensive. Lately, the speed of the processing units and the memory has been increased million times more in the commercial devices and billions in the servers. Most companies follow their products to provide more services as maintenance and tracking the operations of their products. This is compatible with the future cloud computing [18]. Besides, the edge topology on undirected graph describes the graph in terms of paths in details, so all possible edges and paths with their weights (lengths) are listed. Extracting the shortest path between any two vertices is just an operation of reading from the data base, which does not take long especially when it is performed on the computer of the webserver.

Hence, the Internet webserver within a framework and the edge topology on graph were proposed to efficiently evaluate the solution of the vertex to vertex, all pairs and single-source problems. The graph representation by edges topology is presented in section 2. The proposed framework and considering the issues related to the data construction by computations is shown in section 3.

1. Topology on graph

Some basic notions of graph theory [6,19] and the definition of edges topology on graphs [20,21] are presented with the proofs, but only theorems are given. Simple examples on simple graph are shown to explain the theorems.

1.1 Preliminaries

A graph G consists of a non-empty set $V(G)$ of nodes (or vertices), a set $E(G)$ of arcs (or edges), and an incidence function φ_G that joins each edge of G with an unordered pair of nodes of G. Usually the graph is denoted by $G = (V, E, \varphi_G)$. If a vertex v is not incident with any edge, then v is called isolated vertex. If $\varphi_G(e) = vv$, then the edge e is called a loop or self-loop. A multiple edges are two or more edges that join the same pair of vertices. A graph which has no loops or multiple edges is called simple graph. A graph with multiple edges but no loops is allowed called multigraph. In a pseudograph, loops and multiple edges are allowed. If each pair of distinct nodes in a graph G is connected by exactly one edge, then G is called complete graph. If all vertices and edges of a graph H are in G, then H is a subgraph of G. A path is a finite sequence of distinct nodes and distinct arcs in which the sequence begins and ends with nodes and its consecutive elements are incident. A cycle C is a path that begins and ends at the same node.

1.2 Definition of edges topology

Edges topology on graphs was presented in Ref. [20,21]. Suppose that $G = (V, E, \varphi_G)$ is a simple, multi or pseudograph without isolated vertex. Let I_v be the set of all edges incident with the vertex v. Then S_{EG} is defined as $S_{EG} = \{I_v / v \in V\}$. Since each edge is incident with at least one vertex, we have $E = \bigcup_{v \in V} I_v$. For this reason S_{EG} forms subbases for the topology $\mathcal{T}_{EG}$ on E, called edges topology of G.

Example 1. Let $G = (V, E, \varphi_G)$ be a simple graph as in Figure 1 such that $V = \{v_1, v_2, v_3\}$, $E = \{e_1, e_2, e_3\}$ and $\varphi_G(e_1) = v_1v_2$, $\varphi_G(e_2) = v_2v_3$, $\varphi_G(e_3) = v_1v_3$.

We have $I_{v_1} = \{e_1, e_3\}$, $I_{v_2} = \{e_1, e_2\}$ and $I_{v_3} = \{e_2, e_3\}$.

By taking finitely intersection the basis obtained is

$\{e_1\}$, $\{e_2\}$, $\{e_3\}$, $\{e_1, e_3\}$, $\{e_1, e_2\}$, $\{e_2, e_3\}$, $\emptyset$.

Then by taking all unions the topology can be written as

$\mathcal{T}_{EG} = \{\emptyset, E, \{e_1\}, \{e_2\}, \{e_3\}, \{e_1, e_2\}, \{e_1, e_3\}, \{e_2, e_3\}\}$.

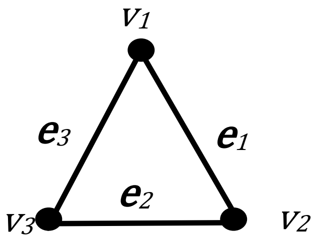

Figure 1 A simple graph of three edges

It is clear that, the edges topology of every simple graph is discrete. This means that the edges topology on simple graphs represents all edge induced subgraphs, i.e. each member of $\mathcal{T}_{EG}$ with the exception of the empty set is an edge induced subgraph. In the last example, it has been shown that applying topology produces all possible subsets (or subsets, or subgraphs) and no path is left or ignored. In the next example, we will show later that some of these subsets are not valid paths and should be distracted. Identification of valid paths from the edges topology (subsets) is the next step and given in the next subsection.

1.3 Identifications of paths and shortest path tree

In the paths that are made from more than one edge, every two adjacent edges are connected and have a common vertex. Theorem 1 shows that P is a path between two distinct vertices in a simple graph G.

Theorem 1. Let $(E, \mathcal{T}_{EG})$ be an edges topology on a simple connected graph with n vertices. For all $P \in \mathcal{T}_{EG}$, if P is a sequence of edges (each edge in the sequence is adjacent with the edges preceding and following) such that two distinct vertices $u, v \in V$ occur one time in P and any other vertices occur two times, then P is a path between u and v.

The proof of the theorem is given in the appendix. In the next theorem, in a connected simple weighted graph, if we have the paths of minimum weights (shortest paths) between every two distinct vertices in G, then we can construct for each vertex in G the shortest-path tree which is a spanning tree rooted at a vertex u such that the path from u to any other vertex v in the tree is the path on minimum weight (shortest path) from u to v in G.

Theorem 2. Let $(E, \mathcal{T}_{EG})$ be an edges topology on a simple connected weighted graph with n vertices. For all $v \in V$ if $P_1, P_2, \cdots, P_{n-1}$ are the paths of minimum weights (shortest paths) from v to all other vertices in the graph $v_1, v_2, \cdots, v_{n-1}$ respectively such that $P_1, P_2, \cdots, P_{n-1} \in \mathcal{T}_{EG}$, then $P_1 \cup P_2 \cup \cdots \cup P_{n-1} \in \mathcal{T}_{EG}$ and construct a shortest-path tree rooted at v.

The last two theorems can be summarized in the following steps that determine all paths between any two distinct vertices and construct the shortest path-tree for any vertex from edges topology of a simple graph G with n vertices are shown below:

Step1: Find the edges topology $\mathcal{T}_{EG}$ for G.

Step2: Examine each element $P \in \mathcal{T}_{EG}$ with length not greater than $(n-1)$ as follows:

a. Represent the edges in P in terms of the vertices incident with it.

b. If P is a sequence of edges (each edge in the sequence is adjacent with the edges preceding and following) such that u and v occur one time in P and any other vertices occur two times, then P is a path between u and v.

Step3: Calculate the weight of each path by taking the sum of the weights on its edges.

Step4: Construct the shortest-path tree rooted at $v \in V$ by taking the union $P_1 \cup P_2 \cup \cdots \cup P_{n-1}$ such that $P_1, P_2, \cdots, P_{n-1}$ are paths of minimum weights (shortest paths) from v to all other vertices in the graph $v_1, v_2, \cdots, v_{n-1}$ respectively.

Example 2. Let $G = (V, E, \varphi_G)$ be a simple graph as in Figure 2 such that $V(G) = \{v_1, v_2, v_3, v_4\}$ and $E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6\}$.

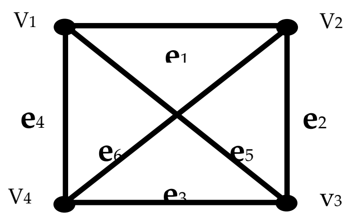

Figure 2

Firstly, we apply the steps for determining all paths between any two distinct vertices and construct the shortest path-tree for any vertex from edges topology of the graph G in Figure 2.

Step1: Since G is a simple graph, then $\mathcal{T}_{EG}$ is discrete. Therefore, $\mathcal{T}_{EG}$=$\{\emptyset$, E, $\{e_1\}$, $\{e_2\}$, $\{e_3\}$, $\{e_4\}$, $\{e_5\}$, $\{e_6\}$, $\{e_1, e_2\}$, $\{e_1, e_3\}$, $\{e_1, e_4\}$, $\{e_1, e_5\}$, $\{e_1, e_6\}$, $\{e_2, e_3\}$, $\{e_2, e_4\}$, $\{e_2, e_5\}$, $\{e_2, e_6\}$, $\{e_3, e_4\}$, $\{e_3, e_5\}$, $\{e_3, e_6\}$, $\{e_4, e_5\}$, $\{e_4, e_6\}$, $\{e_5, e_6\}$, $\{e_1, e_2, e_3\}$, $\{e_1, e_2, e_4\}$, $\{e_1, e_2, e_5\}$, $\{e_1, e_2, e_6\}$, $\{e_1, e_3, e_4\}$, $\{e_1, e_3, e_5\}$, $\{e_1, e_3, e_6\}$, $\{e_1, e_4, e_5\}$, $\{e_1, e_4, e_6\}$, $\{e_1, e_5, e_6\}$, $\{e_2, e_3, e_4\}$, $\{e_2, e_3, e_5\}$, $\{e_2, e_3, e_6\}$, $\{e_2, e_4, e_5\}$,

$\{e_2,e_4,e_6\}$, $\{e_2,e_5,e_6\}$, $\{e_3,e_4,e_5\}$, $\{e_3,e_4,e_6\}$, $\{e_3,e_5,e_6\}$, $\{e_4,e_5,e_6\}$, $\{e_1,e_2,e_3,e_4\}$, $\{e_1,e_2,e_3,e_5\}$, $\{e_1,e_2,e_3,e_6\}$, $\{e_1,e_2,e_4,e_5\}$, $\{e_1,e_2,e_4,e_6\}$, $\{e_1,e_2,e_5,e_6\}$, $\{e_1,e_3,e_4,e_5\}$, $\{e_1,e_3,e_4,e_6\}$, $\{e_1,e_3,e_5,e_6\}$, $\{e_1,e_4,e_5,e_6\}$, $\{e_2,e_3,e_4,e_5\}$, $\{e_2,e_3,e_4,e_6\}$, $\{e_2,e_3,e_5,e_6\}$, $\{e_2,e_4,e_5,e_6\}$, $\{e_3,e_4,e_5,e_6\}$, $\{e_1,e_2,e_3,e_4,e_5\}$, $\{e_1,e_2,e_3,e_4,e_6\}$, $\{e_1,e_2,e_3,e_5,e_6\}$, $\{e_1,e_2,e_4,e_5,e_6\}$, $\{e_1,e_3,e_4,e_5,e_6\}$, $\{e_2,e_3,e_4,e_5,e_6\}\}$.

Step2: From G we have, $\varphi_G(e_1)=v_1v_2$, $\varphi_G(e_2)=v_2v_3$, $\varphi_G(e_3)=v_3v_4$, $\varphi_G(e_4)=v_1v_4$, $\varphi_G(e_5)=v_1v_3$, and $\varphi_G(e_6)=v_2v_4$. By theorem 2, $P\in\mathcal{T}_{EG}$ is a path of length not greater than three between two distinct vertices $u,v\in V$ if P is a sequence of edges such that u and v occur one time in P and any other vertices occur two times.

$\{e_1\}=\{v_1v_2\}$ is a path between v_1 and v_2, $\{e_1,e_2\}=\{v_1v_2,v_2v_3\}$ is a path between v_1 and v_3. $\{e_1,e_2,e_3\}$ = $\{v_1v_2,v_2v_3,v_3v_4\}$ is a path between v_1 and v_4.

$\{e_1,e_2,e_4\}$ = $\{v_1v_2,v_2v_3,v_1v_4\}$ the vertices v_3 and v_4 occur one time and the edges need to rearrange, we put first an edge contain a vertex occur one time. After that the edges arranged in a sequence such that each edge in the sequence is adjacent with the edges preceding and following in the sequence. then $\{v_1v_2,v_2v_3,v_1v_4\}$ = $\{v_3v_2,v_2v_1,v_1v_4\}$ is a path between v_3 and v_4.

$\{e_1,e_2,e_5\}$ = $\{v_1v_2,v_2v_3,v_1v_3\}$ is not a path between two distinct vertices since every vertex occurring twice.

$\{e_1,e_2,e_6\}$ = $\{v_1v_2,v_2v_3,v_2v_4\}$ is not a path since the edge v_2v_4 is not adjacent with the preceding edge also the vertex v_2 occur three times.

By theorem 2, we can determine that $P\in\mathcal{T}_{EG}$ is a path between two distinct vertices u and v by deleting all vertices that repeated twice and the remaining vertices will be u and v in different edges such that any subset of P is not a path between u and v. Otherwise, P is not a path between u and v.

$\{e_1,e_3,\cancel{e_4}\}$=$\{v_1\cancel{v_2},v_1v_3,v_1v_4\}$ is not a path.

$\{e_2,e_4,\cancel{e_6}\}$=$\{v_2\cancel{v_3},v_1\cancel{v_4},\cancel{v_2}v_4\}$ is a path between v_1 and v_3. After testing all elements $P\in\mathcal{T}_{EG}$ of length not greater than three, all paths between distinct vertices are shown in table 1:

Table 1. All paths between distinct vertices of the simple graph in Figure 2.

v_1-v_2	v_1-v_3	v_1-v_4	v_2-v_3	v_2-v_4	v_3-v_4
$P_1=\{e_1\}$	$P_6=\{e_5\}$	$P_{11}=\{e_4\}$	$P_{16}=\{e_2\}$	$P_{21}=\{e_6\}$	$P_{26}=\{e_3\}$
$P_2=\{e_2,e_5\}$	$P_7=\{e_1,e_2\}$	$P_{12}=\{e_1,e_6\}$	$P_{17}=\{e_1,e_5\}$	$P_{22}=\{e_1,e_4\}$	$P_{27}=\{e_2,e_6\}$
$P_3=\{e_4,e_6\}$	$P_8=\{e_3,e_4\}$	$P_{13}=\{e_3,e_5\}$	$P_{18}=\{e_3,e_6\}$	$P_{23}=\{e_2,e_3\}$	$P_{28}=\{e_4,e_5\}$
$P_4=\{e_2,e_3,e_4\}$	$P_9=\{e_1,e_3,e_6\}$	$P_{14}=\{e_1,e_2,e_3\}$	$P_{19}=\{e_1,e_3,e_4\}$	$P_{24}=\{e_1,e_3,e_5\}$	$P_{29}=\{e_1,e_2,e_4\}$
$P_5=\{e_3,e_5,e_6\}$	$P_{10}=\{e_2,e_4,e_6\}$	$P_{15}=\{e_2,e_5,e_6\}$	$P_{20}=\{e_4,e_5,e_6\}$	$P_{25}=\{e_2,e_4,e_5\}$	$P_{30}=\{e_1,e_5,e_6\}$

Step 3: In any simple weighted graph G, the weight of any path is the sum of the weights on its edges and denoted by $w(P)$. Consider the simple graph in example 2. Suppose that $w(e_1) = 5$, $w(e_2) = 4$, $w(e_3) = 6$, $w(e_4) = 2$, $w(e_5) = 9$, $w(e_6) = 8$. Then the weights of all paths are shown in Table 2:

Table 2. The weights of all paths in table 1.

$v_1 - v_2$	$v_1 - v_3$	$v_1 - v_4$	$v_2 - v_3$	$v_2 - v_4$	$v_3 - v_4$
$w(P_1) = 5$	$w(P_6) = 9$	$w(P_{11}) = 2$	$w(P_{16}) = 4$	$w(P_{21}) = 8$	$w(P_{26}) = 6$
$w(P_2) = 13$	$w(P_7) = 9$	$w(P_{12}) = 13$	$w(P_{17}) = 14$	$w(P_{22}) = 7$	$w(P_{27}) = 12$
$w(P_3) = 10$	$w(P_8) = 8$	$w(P_{13}) = 15$	$w(P_{18}) = 14$	$w(P_{23}) = 10$	$w(P_{28}) = 11$
$w(P_4) = 12$	$w(P_9) = 19$	$w(P_{14}) = 15$	$w(P_{19}) = 13$	$w(P_{24}) = 20$	$w(P_{29}) = 11$
$w(P_5) = 23$	$w(P_{10}) = 14$	$w(P_{15}) = 21$	$w(P_{20}) = 19$	$w(P_{25}) = 15$	$w(P_{30}) = 22$

The paths $P_1, P_8, P_{11}, P_{16}, P_{22}$ and P_{26} represent the shortest paths between distinct vertices. In this way we can choose the minimum or maximum weight for the paths depending on the problem.

Step 4: By theorem 2 and step 3, the shortest path tree rooted at each vertex are:

1. The shortest paths from v_1 to the vertices v_2, v_3, and v_4 are $P_1 = \{e_1\}$, $P_8 = \{e_3, e_4\}$ and $P_{11} = \{e_4\}$ respectively. Then by theorem 2 the union $P_1 \cup P_8 \cup P_{11} = \{e_1, e_3, e_4\}$ is the shortest-path tree rooted at v_1.
2. The shortest paths from v_2 to the vertices v_1, v_3, and v_4 are $P_1 = \{e_1\}$, $P_{16} = \{e_2\}$ and $P_{22} = \{e_1, e_4\}$ respectively. Then by theorem 2 the union $P_1 \cup P_{16} \cup P_{22} = \{e_1, e_2, e_4\}$ is the shortest-path tree rooted at v_2.
3. The shortest paths from v_3 to the vertices v_1, v_2, and v_4 are $P_8 = \{e_3, e_4\}$, $P_{16} = \{e_2\}$ and $P_{26} = \{e_3\}$ respectively. Then by theorem 2 the union $P_8 \cup P_{16} \cup P_{26} = \{e_2, e_3, e_4\}$ is the shortest-path tree rooted at v_3.
4. The shortest paths from v_4 to the vertices v_1, v_2, and v_3 are $P_{11} = \{e_4\}$, $P_{22} = \{e_1, e_4\}$ and $P_{26} = \{e_3\}$ respectively. Then by theorem 2 the union $P_{11} \cup P_{22} \cup P_{26} = \{e_1, e_3, e_4\}$ is the shortest-path tree rooted at v_4.

When the edges topology for a given vertices of real network such as road network is found in terms of subsets, investigation is not only whether each subset is a path or not but it should be whether a real path in the original network or not.

2. Framework of the shortest route problem

The framework proposed for the problem is divided into two: offline and online as shown in Figure 3. The offline is only in the webserver and is supposed be finished before starting the online one. In the online, the webserver provides the shortest path service to clients. The session of the offline is expected so much longer than the online session because during the offline, in the webserver, large amount of data would be manipulated and created. It is expected that the webserver that can quickly and successfully finish the offline session, will provide the fastest and best solution ever to the online query.

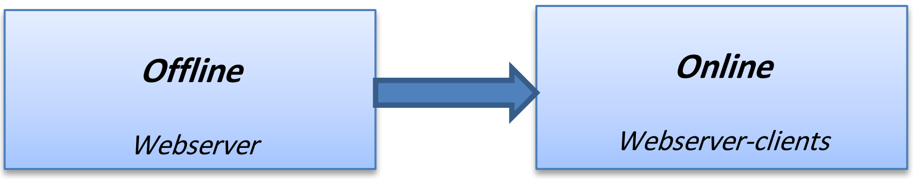

Figure 3 Two session framework for the shortest route problem.

In offline session as shown in Figure 4, the webserver inputs the data of the network: vertices and distances between them, and apply the steps shown above in the previous subsection. This is like the preprocessing to initiate new data that contains all vertices and edges. The data produced is used as an input for the processing the graph. From the processing of data files new data files are initiated to be used later for the online query.

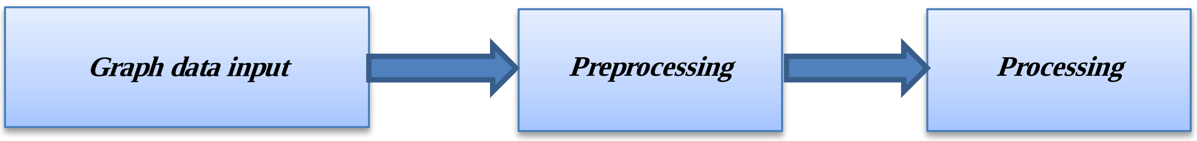

Figure 4 Offline session schemes in webserver for the shortest route problem.

:, has a query of a shortest path between two vertices, it sends the query represented by two vertices: starting and destination, to the webserver as shown in Figure 5. The webserver will access the data that is already made to read the path that has these two ends with shortest path (least weight) and send it to the client. When the client's device receives the shortest path in a particular code, a search in client's device data is performed to extract the edges that client should orderly follow to reach the destination. The two sides, webserver and client, are supposed to be capable to accomplish the solution of the problem on time. For the webserver side, some issues such as the memory, the speed and the users interconnect should be considered in the framework. How the webserver can perform the topology and deals with huge graphs with large data and construct larger data are discussed and evaluated.

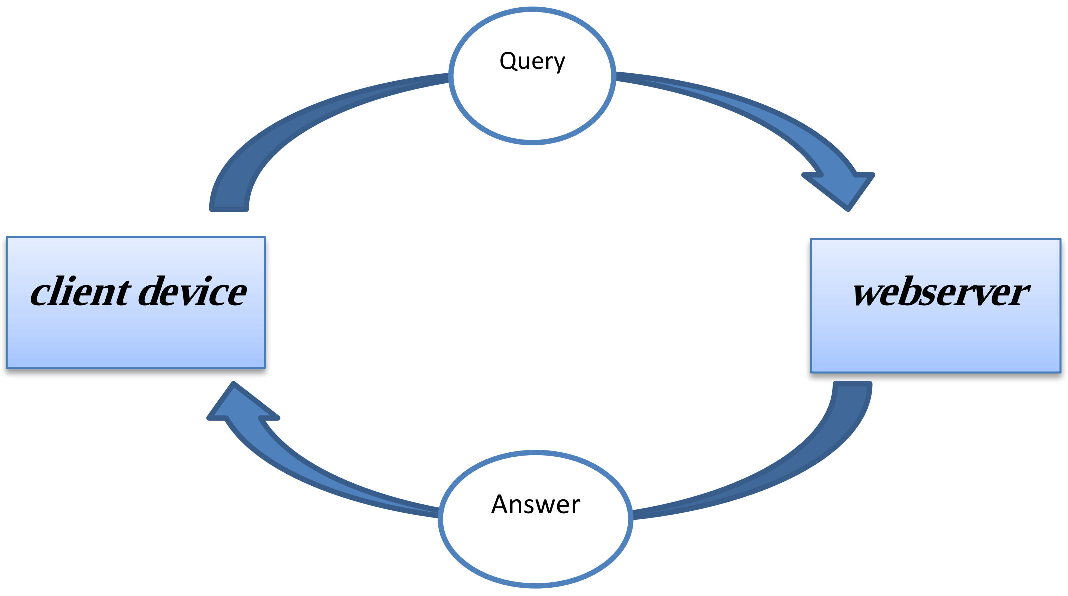

Figure 5 Online session scheme between server and the webserver for the shortest route problem (see the text for more details).

3. Webserver side

The webserver has the responsibility to provide quick service to the clients. The performance of the webserver computers is higher than the clients'. These computers have very fast CPUs and very large RAM however the large graphs may have millions of vertices and the database initiated in the webserver computer are so much larger than the input data of the graphs. The fast implementation of the computer is one of the properties that support this framework because webserver presents online response during obtaining the shortest path from the database for a query. Furthermore, the big rate of users' queries received by the webserver is one of the issues that should be considered. In this scheme, every point, listed below, has issues that should be considered and discussed in more details,

A. Input of the graph data:
B. Preprocessing of the input data
C. Processing of graph paths
D. Implementation of data access
E. Webserver-users connection.

It is important to know that the input of the graph data, preprocessing the input data and the processing of data initiation are previously done off-line before the webserver receives any query. Hence, the implementation of data access and the webserver-users connection are the online processes for the client query.

A. Input of the graph data: The input of the graph data is not a big issue as a size of memory and a speed of implementation when it is compared to the next two stages: the preprocessing and the processing. The two stages totally depend on it, so, any change in input leads to so many changes in the next two. Such changes are related to long and short-term update due to initiation of new nodes or edges and deletion, interdiction or some constraints on roads. In the present work we conducted a general framework that solves the shortest problem for a very large graph, which finds the shortest path in the offline webserver computers. Updates and forbidden roads can be included in different schemes and it is out of the scope of the current work. In general, the size of the graph is the issue that we consider here. The roads exit where people populate. All maps of countries are divided into provinces that are divided into cities. A few high ways connect the provinces and more between the cities. The vertices inside the cities are most dense. This view leads to consider the huge graph as made up from clustered graphs, and then each cluster could be considered a graph of a province or city road network. Hence, the clusters can be considered as big vertices with highway between them in a high-scale network. Therefore the high-scale road network is smaller than the graph of the city network. Within this view, the shortest route between two vertices on different cities can be considered as three shortest paths: inside the first city (from a vertex to another on its border), first city to the second city (within the high-scale network) and from border of the second city to the vertex inside it. From this view, the size of the graph could be considered as only a cluster and the biggest cluster is the road network of a city. The size of graph that should be considered for studying the graph problems of road network should not be more than that of a city. Therefore, we may not need to consider a graph made up of 30 million vertices for North America but a graph of only a city [10]. Kim et al. (2014) chose two real road networks, Oldenburg (6015 nodes and 7035 edges) in Germany, and Singapore (11414 nodes and 15641 edges) to test their proposed algorithm [22]. The largest network chosen to test an algorithm of the shortest path problem was made from 2500 nodes with a ratio edges/nodes=15 [23]. A 11640 nodes random graph was generated to represent a large

Table 3 Real examples of graph for some cities in USA.

Name	Description	# nodes	# arcs
FLA	Florida	1,070,376	2,712,798
COL	Colorado	435,666	1,057,066
BAY	San Francisco Bay Area	321,270	800,172
NY	New York City	264,346	733,846

graph [24]. In USA some states have so much more nodes and edges as shown in Table 3 (taken from [25]). Florida has the largest number of nodes, about one million, and edges, 3 million. The size of the files that contain vertices, distances of the arcs of Florida was about 50 MB after downloaded and decompressed.

B. Preprocessing of the input data

In this stage the input data of the graph is read and maybe manipulated to be ready for the next stage processing. The manipulations could be different for different processes based on different algorithm. In the present work and for a large graph, such as that of 3 million edges, the input data is performed and rewritten in terms of edge number, the weight and vertices. In other words, it is to provide the data in a suitable format for the processing stage. The file of data obtained is a necessary reference for the webserver and the client for linking the edges to their vertices. However the size of the data is not a serious problem because of the capability of the webserver computers, the data could be divided into a certain number of files according to a purpose related to speed up the processing or to parallel manipulations by more than processers. Müller et al. (2006) used Message Passing Interface technique of implementation to handle huge input graphs that can even handle graphs as large as the road map graph of Western Europe [26]. In this implementation, the run can be on parallel architectures, e.g., on a massively parallel shared-memory machine.

C. Processing the graph paths

In processing, all edges with their weights (lengths) are input to apply steps 1,2 and 3 to construct all real possible paths. Each path is a record that contains identification code, weight, and edges. Such data will be replicated because so many paths have common edges. In the topology of edges, so many subsets are produced but not all of them are paths, and it builds the subsets according to the combination, for instance, it extracts all possible subsets made from two different elements chosen from a set of n elements. For Florida road network of 3 million edges, the subsets of edges that are produced by the edges topology are more than the real paths. Using the formula 2^n, the number of these subsets can be calculated as follows

For $\mathrm{n} = 3{,}000{,}000$

$$\mathrm{No.\,of\ subsets} = 2^{3000000} \approx 8\,x\,10^{600}$$

This huge number cannot be used in computers and represents a floating point and if we assume that each subset is a few bytes then there is no a memory has this capacity. If we think about the implementation and assume that the CPU finishes one task with a speed equals to its frequency about 3 MHz (three million operations per second), it takes about 10^{594} sec (~10^{585} centuries). The difference between one and three millions has no matter with the big power. Sedgewick (2003) mentioned that either the computers are developed to have speed of 200,000 times faster, the factorial-time algorithm applied to one million vertices takes centruies to fully manupulate the graph [27]. Since we don't concern about the algorithm of online processing but with offline processing, finding a method reduces the computations is our target. It has been clear that applying the topology for a huge number of edges is impossible but we discuss how this will be reduced by modifying the edge topology to less number. The real paths needed by the clients are not made from 3 million edges, hundred thousand, thousand or hundred edges. The paths made from a few edges can be calculated using the formula for calculating the number of different combination from choosing r edges out of n edges $C_r^n = n!/r!\,(n-r)!$.

No. of two-edges subsets out of one million $\approx 5\ x\ 10^{11}$

No. of ten-edges subsets out of one million $\approx\ 10^{54}$

So, the number of subsets increases when the number of elements of the subset increases. Practically real paths are made from a small number of edges, therefore only subsets with small number of element are important. If we assume that the maximum number of edges in real paths is 25, the total subsets

- The sum of subsets with edges; 2-25 $\approx\ 10^{124}$

- The sum of subsets with edges; 2-50 $\approx\ 10^{235}$

In comparison with 10^{600}, the reduction in the number of subsets is so much when the paths with limited number of edges are considered. But the number of subsets is still too large to be manipulated in computers either those in webserver. The different combinations used in the topology to choose any r elements from n elements is randomly occurs with considering whether the edges are connected or not. This is why the number of two-edges subsets $\approx 5\ x\ 10^{11}$ was too large. There is no doubt that an edge in a district is not connected (adjacent) with another edge in another district located a way or at other side from the city. The vertices are usually connected at maximum to four vertices. In other words, an edge is connected at maximum to 6 edges as shown in the Figure 3. In the topology, for each edge (1000,000 -1), subsets are constructed while there are only 6 edges, so for our example of one million edges,

The no. of paths made from two adjacent edges $\approx\ 10^6 \times 6$

When the path, made from two adjacent edges, is considered to be extended to any possible next edges from its two ends, there are also only 6 edges (see Figure 3), so

The no. of paths made from three adjacent edges $\approx\ 10^6 \times 6 \times 6 = 10^6 \times 6^2$

By similar way, when every path of two edges are considered to be connected to one edge from its tips,

The no. of paths made from four $\approx 10^6 \times 6^2 \times 6 \approx = 10^6 \times 6^3$

In general, the number of paths N made from r edges in a graph made from n edges can be written as

$$N(r) = n\,6^{r-1} \quad ,for\ r \leq n \tag{1}$$

If we consider all paths that have r = 1, 2, 3, 4,…, and k edges, the total number of paths is the sum of paths given in Eq. (1),

$$S_k = \sum_{i=1}^{k} n\,6^{i-1} \ \ ,\ \ for\ k \leq n \tag{2}$$

This is a geometric series and the sum can be written as

$$S_k = \frac{n\,(1-6^k)}{1-6} \ \ ,\ \ for\ k \leq n \tag{3}$$

Now, calculating the total possible paths made from edges from 2 to 25in one million edges graph is

$$S_{25} = \frac{10^6\,(1-6^{25})}{1-6} \approx 10^{24}$$

And,

$$S_{50} = \frac{10^6\,(1-6^{50})}{1-6} \approx 10^{44}$$

In summary, from the 10^{600} subsets only about 10^{44} are valid. This reduction is so big and indicates the possibility of dealing with the large graphs with offline processing. When a graph of a few thousand edges such as Oldenburg (7035 edges) in Germany is considered, the sum $S_{25} \approx 10^{21}$ and $S_{50} \approx 10^{41}$ which are not so much different from the graph of million edges. Equation 3 can be written for paths made from 25 and 50 edges as follows:

$$S_{25} = n\,10^{19} \tag{4}$$

$$S_{50} = n\,10^{38} \tag{5}$$

The last two equations show that the large number is mainly due to the long paths. If there is a method or technique (algorithm) that can obtain the long paths from short path, the number of the path will reduced so much. In addition, if we consider the edge a in Figure 1 again and assume that it is surrounded by 5 edges instead of 6, 10^{19} paths in Eq.(4) and 10^{38} paths in Eq.(5) will be subtracted from the total sum of paths. Therefore, the ratio of the edges to the vertices has a large influence on the determination of the exact number of paths and consequently reduces remarkably the number of paths.

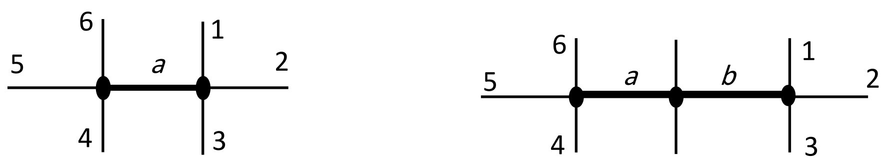

Figure 3 There are 6 adjacent edges around the edge ***a*** (on left) and there are also only 6 edges adjacent to the two ends of the path made from the two edges ***a*** and ***b*** (on right) when it is assumed that the maximum edges connected to a vertex is four.

The number of the paths obtained above is still so much larger than the real however it can be reduced more in future research work. Thus, a graph of Oldenburg city (7035 edges) with the sum $S_{25} \approx 10^{21}$ is considered as a primary evaluation for the current proposal. The processing creates the data that contains all paths. Other sets of data can be created for vertex-vertex, all pairs or single-source paths to be ready for any query. The way the data are saved or divided into files is subjected to many schemes depending on the webserver computers performance and the services offered. Hence, the data manipulation in the processing can be viewed in three points; the capacity of storage, capacity of data access

(RAM) and the duration of the implementations. The three points are the main challenges that face the offline processing of graph data.

The capacity of storage in the webserver is not a big issue since the computers already have a very large size of hard drive disks and the data can be distributed over more than one computer. So many servers in the globally provide services that deal with data so much larger than the data for 10^{21} paths.

The duration of implementation is one of the issues that always needed to be shorter. Although the implementation is offline and has no effect on the duration of the client service, the large data and the large duration of processing are still important because of the vast amount of data and the update. For the data such as that of 10^{21} paths, the computer that have a CPU of about 8 GHz, it can roughly finish only about 10^9 calculations in second. In one year (10^7 second), it can finish 10^{16} paths which are still so slow and the webserver with such a speed cannot finish the calculation of obtaining the paths in many years either the real path is less than this number. For such an issue even the parallel computers can shorten the duration of this process. Here we suggest that supercomputers can do this job then the data obtain is saved in the webserver computers. These computer, such as the "Jaguar Cray XT5 at Oak Ridge National Laboratory and "The BlueGene/P Intrepid system at Argonne National Laboratory, had over 1×10^5 cores and reached petaflop nominal speeds [28–30]. The petascale computers that can perform about 10^{15} calculation per second can finish this task in one year. The exascale supercomputer [30–35] that can perform about 10^{18} operations per second can finish this task in a few days.

The data yielded from the processing can be arranged in a way so that updating the input data don't need to redo the processing again and construct new data. The deleted and added edges can be individually manipulated and saved in additional data files that can be permanently or temporarily used.

The data access capacity is the most important issue that relates to manipulate very large data. However increasing the data access is always compatible to the speed of CPU to execute and finish a task in-time, the access memory of the webserver can be increased even the time taken to finish the processing is long because this is performed offline. Lately, Citrix offered a server, called Xenserver, has the ability to access to RAM memory with 128GB [36,37]. Such a server with fast with this ability can access with large memory but slower.

D. Implementation of data access

This implementation is different from that shown above for the offline processing. It is online implementation in which the webserver accessing the data of the paths identified by the starting and the destination vertices. As mentioned before such data of paths are so large and can be partitioned into different parts to be shared by different computers. In Xenserver, offered by Citrix, the speed of accessing 128 GB RAM is encouraging to let the server performs such a task.

E. Webserver-users connection

Since the webserver provides the user the service of the query, it is expected that the large number of users that instantly connect the server may cause delay in the service. This issue is actually solved by using the virtual servers that can be accessed through the interconnect networks. In the interconnected networks of cloud computing, it allows share information between the computer resources, which are already accessed by a group of user by the Internet. The computers are altogether hosted virtually in a centralized repository [38].

4. Conclusion

The framework and the topology, proposed for the shortest path problem, have been evaluated. The steps needed to apply the edges topology successfully finds the realistic shortest path but for small graph. The large graph data still has many challenges such as the storage, the data access RAM and the duration of implementation but the proposed framework that consists of the offline webserver and the online webserver-client, totally diminishes these challenges and partially that part of the processing the graph data to obtain the real paths. Hence, use of a supercomputer was suggested as temporary solution till a progress of future researches will find the less realistic number of paths or a method of computation that can overcome the too large data. The framework within the edges topology, compatible with the future cloud computing and supercomputers, presents best and fastest solution of the shortest path ever.

References

[1]E. W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numer. Math. 269 (1959).

[2]G. B. Dantzig, *On the Shortest Route Through a Network*, Manage. Sci. **6**, 187 (1960).

[3]M. Dell'Amico, M. Iori, and D. Pretolani, *Shortest Paths in Piecewise Continuous Time-Dependent Networks*, Oper. Res. Lett. **36**, 688 (2008).

[4]A. Kandel, H. Bunke, and M. Last, *Applied Graph Theory in Computer Vision and Pattern Recognition* (2007).

[5]D. R. Tobergte and S. Curtis, *Introduction to GraphTheory*, Vol. 53 (2013).

[6]S. Saha Ray, *Graph Theory with Algorithms and Its Applications*, Vol. 53 (2013).

[7]A. M. Frieze and G. R. Grimmett, *The Shortest-Path Problem for Graphs with Random Arc-Lengths*, Discret. Appl. Math. **10**, 57 (1985).

[8]A. V Goldberg and C. Harrelson, *Computing the Shortest Path : A * Search Meets Graph Theory*, Science (80-.). 156 (2005).

[9]B. Ding, J. X. Yu, and L. Qin, *Finding Time-Dependent Shortest Paths over Large Graphs*, Proc. 11th Int. Conf. Extending Database Technol. Adv. Database Technol. - EDBT '08 205 (2008).

[10] A. V. Goldberg and R. . Werneck, *Computing Point-to-Point Shortest Paths from External Memory*, ALENEX'2005, Proc. 7th Work. Algorithm Eng. Exp. 26 (2005).

[11] H. Ananthanarayanan and R. Ordóñez, *A Fast Converging Optimal Technique Applied to Path Planning of Hyper-Redundant Manipulators*, Mech. Mach. Theory **118**, 231 (2017).

[12]C. K. Poon, C. J. Zhu, and K.-Y. Lam, *Energy-Efficient Air-Indices for Shortest Path and Distance Queries on Road Networks*, Inf. Syst. **71**, 182 (2017).

[13]D. B. Johnson, *Efficient Algorithms for Shortest Paths in Sparse Networks*, J. ACM **24**, 1 (1977).

[14]S. Broumi, D. Nagarajan, A. Bakali, M. Talea, F. Smarandache, and M. Lathamaheswari, *The Shortest Path Problem in Interval Valued Trapezoidal and Triangular Neutrosophic Environment*, Complex Intell. Syst. **5**, 391 (2019).

[15]N. Shanmugasundaram, K. Sushita, S. P. Kumar, and E. N. Ganesh, *Genetic Algorithm-Based Road Network Design for Optimising the Vehicle Travel Distance*, Int. J. Veh. Inf. Commun. Syst. **4**, 344 (2019).

[16]Z. Zhang and M. Li, *Finding Paths With Least Expected Time in Stochastic Time-Varying Networks Considering Uncertainty of Prediction Information*, IEEE Trans. Intell. Transp. Syst. 1 (2023).

[17]M. Gmira, M. Gendreau, A. Lodi, and J.-Y. Potvin, *Tabu Search for the Time-Dependent Vehicle Routing Problem with Time Windows on a Road Network*, Eur. J. Oper. Res. **288**, 129 (2021).

[18] N. Phaphoom, X. Wang, and P. Abrahamsson, *Foundations and Technological Landscape of Cloud Computing*, ISRN Softw. Eng. **2013**, 1 (2013).

[19] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications* (1976).

[20] A. Kilicman and K. Abdulkalek, *Topological Spaces Associated with Edges Set of Graphs*, Electron. J. Graph Theory Appl. (under review) (n.d.).

[21] K. A. Abdu and A. Kilicman, *Topologies on the Edges Set of Directed Graphs*, **12**, 71 (2018).

[22] J. Kim, W. S. Han, J. Oh, S. Kim, and H. Yu, *Processing Time-Dependent Shortest Path Queries without Pre-Computed Speed Information on Road Networks*, Inf. Sci. (Ny). **255**, 135 (2014).

[23] L. Di Puglia Pugliese and F. Guerriero, *Shortest Path Problem with Forbidden Paths: The Elementary Version*, Eur. J. Oper. Res. **227**, 254 (2013).

[24] S. Wøhlk and G. Laporte, *Computational Comparison of Several Greedy Algorithms for the Minimum Cost Perfect Matching Problem on Large Graphs*, Comput. Oper. Res. **87**, 107 (2017).

[25] DIMACS, *Nineth DIMACS Implementation Challenge-Shortest Paths*, http://www.dis.uniroma1.it/challenge9/download.shtml.

[26] K. Müller, D. Delling, M. Holzer, F. Schulz, and D. Wagner, Design and Implementation of an Efficient Hierarchical Speed-up Technique for Computation of Exact Shortest Paths in Graphs, University of Karlsruhe (TH), 2006.

[27] B. R. Sedgewick, *Algorithms in Java, Third Edition, Part 5: Graph Algorithms* (2003).

[28] R. Schulz, B. Lindner, L. Petridis, and J. C. Smith, *Scaling of Multimillion-Atom Biological Molecular Dynamics Simulation on a Petascale Supercomputer*, J. Chem. Theory Comput. **5**, 2798 (2009).

[29] W. Jiang, Y. Luo, L. Maragliano, and B. Roux, *Calculation of Free Energy Landscape in Multi-Dimensions with Hamiltonian-Exchange Umbrella Sampling on Petascale Supercomputer*, J. Chem. Theory Comput. **8**, 4672 (2012).

[30] V. Voevodin and V. Voevodin, *Efficiency of Exascale Supercomputer Centers and Supercomputing Education*, in *International Conference on Supercomputing* (Springer, 2015), pp. 14–23.

[31] X. Yang, Z. Wang, J. Xue, and Y. Zhou, *The Reliability Wall for Exascale Supercomputing*, IEEE Trans. Comput. **61**, 767 (2012).

[32] J. Hsu, *When Will We Have an Exascale Supercomputer ? An Ultrasonic Scalpel for Brain Surgery*, Spectrum, IEEE (2015).

[33] A. Geist, B. Gropp, S. Kale, B. Kramer, M. Snir, F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, *Toward Exascale Resilience*, Int. J. High Perform. Comput. Appl. **23**, 374 (2009).

[34] D. Schneider, *The Exascale Era Is Upon Us: The Frontier Supercomputer May Be the First to Reach 1,000,000,000,000,000,000 Operations per Second*, IEEE Spectr. **59**, 34 (2022).

[35] C. Chang, V. L. Deringer, K. S. Katti, V. Van Speybroeck, and C. M. Wolverton, *Simulations in the Era of Exascale Computing*, Nat. Rev. Mater. **8**, 309 (2023).

[36] A. Kishimoto, A. Fukunaga, and A. Botea, *Scalable, Parallel Best-First Search for Optimal Sequential Planning*, In ICAPS-09 201 (2009).

[37] P. Efstathopoulos and F. Guo, *Rethinking Deduplication Scalability.*, in *HotStorage* (2010).

[38] H. A. Ismail and M. Riasetiawan, *CPU and Memory Performance Analysis on Dynamic and Dedicated Resource Allocation Using XenServer in Data Center Environment*, in *Proceedings - 2016 2nd International Conference on Science and Technology-Computer, ICST 2016* (Yogyakarta, Indonesia, 2017), pp. 17–22.