IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 23 August 2023, accepted 11 September 2023, date of publication 18 September 2023,
date of current version 21 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3316211

RESEARCH ARTICLE

Multi-Criteria Path Finding Using Multi-Queues Based Bidirectional Search for Multiple Target Nodes in Networks

XIAOQING XU, XIAOJUN LIU, LIUYIHUI QIAN, NING ZHANG, JUAN WU, AND HONG TANG
Research Institute of China Telecom Company Ltd., Guangzhou 510630, China
Corresponding author: Hong Tang (tangh@chinatelecom.cn)

ABSTRACT The convergence of networking and cloud computing leads to an increasing number of cloud based services and applications, which require to be processed by network nodes with enough resources. There are multiple available nodes in the network and network operators need to find paths to forward the traffic to these nodes for processing. In addition, services and application may have diversified requirements on the paths in terms of bandwidth, delay, jitter, etc. Therefore, network operators need to determine the optimal paths considering multiple criteria. However, single-criterion based shortest path algorithms are often used to compute paths from a source node to one target node, which may result in uneven traffic distribution and low resource utilization. Furthermore, the paths obtained are only optimal to this criterion and may not satisfy the Service Level Agreement (SLA) requirements. Therefore, in this paper, we propose a multi-criteria path finding method to obtain all the Pareto-optimal paths from a source node to multiple target nodes (destinations). We extend an existing effective multi-criteria routing algorithm (ParetoBFS), and integrate a multi-queues based bidirectional search mechanism. Additionally, we use a topology sampling technique to accelerate path computation. We evaluate the performance of these path finding methods using various topologies with multiple criteria including bottleneck and additive types. Experimental results demonstrate that our approach can reduce the running time by $54\% - 89\%$ in different topologies compared to the ParetoBFS algorithm. By employing topology sampling, we further improve our algorithm's speed while still finding most of the Pareto-optimal paths.

INDEX TERMS Bidirectional search, multi-criteria, path finding, path selection, routing.

I. INTRODUCTION

Network technologies and cloud computing have been developing fast, driving cloud-network convergence to be a trend. As a result, many cloud based services (e.g., immersive VR/AR, live broadcasting, and interactive cloud games) have emerged and they require to be processed by appropriate nodes with resources in the network. At the same time, these services require multiple SLAs guarantees on the path they traverse by. For example, a cloud VR service requires paths with large bandwidth and low latency, while also needing access to a network node (e.g., a cloud data center, or an edge cloud node) with sufficient resources to complete associated processes such as encoding and rendering. Since there are multiple nodes able to provide processing resources, it is necessary for operators to consider the appropriate nodes to process the services in order to improve network resource utilization. On the other hand, when considering multiple criteria in path finding, the multi-criteria shortest paths are Pareto-optimal paths, which means no other paths are better than they in all the criteria. With multi-criteria optimal path finding, network operators can provide the services with higher SLA guarantee or offer they with more flexibility to achieve trade-off between multiple criteria.

The associate editor coordinating the review of this manuscript and approving it for publication was Alessandro Floris.

However, many traditional routing protocols (e.g., OSPF [1], RIP [2]) only use single-criterion shortest path algorithms (e.g., Dijkstra algorithm, Bellman-Ford algorithm) which may not be able to find a path fulfilling the SLA requirements. For example, if a path is shortest regarding the latency for a service from a source node to a target node, its bandwidth may not be able to satisfy the requirement. To consider multiple criteria comprehensively without preference and then to obtain multi-criteria optimal paths, all Pareto-optimal paths needs to be found. A path is Pareto-optimal if there is no other path that is better in all metrics.

Some multi-criteria path selection or finding algorithms and Quality of Service (QoS) routing methods have been proposed [3], [4], [5]. However, most of them either address additive criteria only or combine multiple metrics into a single composite one, e.g. weigh sum method. In addition, they usually consider one target node and neglect one to many scenarios. Chen et al. [6] proposed ParetoBFS, an approach based on BFS that enumerates all paths while applying Pareto optimality constraints to prune the traversal tree during path finding. As a result, it can identify all Pareto-optimal paths for a demand with specified source and destination. Chen et al. demonstrated that ParetoBFS offers numerous advantages over traditional multi-criteria path finding methods, such as the ability to tackle bottleneck-type criteria and faster execution speed. However, it primarily considers a single target (destination) node, i.e., it focuses on finding Pareto-optimal paths from a source node to target node without accounting for scenarios involving multiple target nodes. While it is possible to introduce certain modification to account for scenarios involving multiple target nodes, this may lead to a reduction in efficiency and an increase in computational time. In a network with multiple resource nodes, a service may be served by any of the available nodes; hence, in this scenario the service could have multiple target nodes. For a service, it is also possible to establish multiple connections between the source node and multiple target nodes. Additionally, there may be multiple services or users originating from the same node who require connections to different target nodes. Therefore, when multiple criteria exist in the topology, to find out optimal paths, it is necessary to compute the Pareto-optimal paths regarding these target nodes. By considering multiple target nodes rather than one, it becomes possible to select better paths or avoid choosing suboptimal paths. Moreover, it enables achieving resource load balancing across multiple target nodes. Since the ParetoBFS works well for multi-criteria path finding between a source node and a target node, one simple solution would be invoking the ParetoBFS multiple times to obtain all the Pareto-optimal paths between the source node and each target node and then determine the final Pareto-optimal paths among these found paths. However, simply applying this naive approach tends to be time-consuming. The reason is that it does not consider potential optimizations during the path finding process when there are multiple target nodes. In the scenario with multiple target nodes, some paths from a source node to these nodes might be pruned in advance during the path finding process, eliminating the need for further exploration.

On the other hand, a bidirectional search mechanism can be used to improve the speed of path finding method [7]. However, only additive criteria and single target node are considered in this method. The stopping condition for bidirectional search in multi-criteria scenarios differs from that in traditional single-criterion scenarios. Even forward search and backward search encounter at the intermediate state, it is insufficient to halt the path finding process in multi-criteria contexts. This is because in multi-criteria scenarios, it is still possible to find other Pareto-optimal paths when two search processes meet.

In this paper, we consider multi-criteria path finding for a service which can be processed by one of the multiple target nodes with resources in the network. We extend the previous BFS-based method ParetoBFS and use a bidirectional search to find out the full Pareto-optimal path set for the considered service to multiple target nodes. When using bidirectional search, we extend the stopping condition idea proposed by Demeyer et al. [7] We use multiple criteria queues to determine whether there is a need for further path finding, taking into account the optimal criteria values of the paths formed by forward and backward searches. Additionally, we use topology sampling technique with k shortest method to obtain the sub-topology and further increase the algorithm speed without comprising too much path quality.

Our contributions are listed as follows:

- We formulate the multi-criteria path finding problem for multiple target nodes scenario, wherein a given demand can be served by multiple nodes, necessitating the identification of Pareto-optimal paths from a specific source node to these targets.
- We propose a multi-criteria path finding method based on the ParetoBFS algorithm and bidirectional search mechanism, capable of handling the problem with multiple target nodes.
- We employ a topology sampling method to further accelerate our algorithm. By computing the k shortest paths for each criterion from the source to multiple targets, we construct a sub-topology consisting of the edges included in these paths. Subsequently, our multi-criteria Pareto-optimal path finding method is then performed using this sub-topology.

The remainder of the paper is organized as follows: Section II discusses related works on multi-criteria path finding in networks. Section III describes the modeling of the multi-criteria path finding scenario involving a network with multi-criteria links and multiple target nodes. Section IV introduces the Bidirectional Multi-Targets Pareto BFS algorithm (BMT-ParetoBFS) and a topology sampling method for algorithm acceleration. Section V presents experimental results and discussions. Section VI concludes the paper.

II. RELATED WORKS

Multi-criteria path selection or finding is important in many areas, e.g., transportation and communications. In transportation area, Kurbanov et al. [8] proposed a one to many version of Pareto path set finding method, which is similar to our considered multi-targets scenario. However, the criteria used are different, and the path computation in their work is based on the multicriteria label-setting algorithm, which can only deal with additive-type criteria. With the network technology development, multi-criteria path finding plays a crucial role in network operation and management. As services and applications increasingly demand more stringent and diversified QoS requirements, single-metric based shortest path algorithms become inadequate. QoS routing [9], [10], [11] has been extensively researched, but most approaches focus on dealing with additive types of criteria such as cost and delay, and bottleneck-type criteria such as bandwidth are treated as constraints rather than optimization objectives. It is proven that multi-criteria path finding accounting for more than two criteria constitutes an NP-hard problem [12], [13]. However, for medium-sized real-world topologies, it is possible to obtain Pareto-optimal paths in reasonable time [6]. Chen et al. proposed ParetoBFS [6], an efficient BFS-based algorithm with pruning technique to achieve the Pareto-optimal path set for a demand given a source node and a target node. ParetoBFS presents advances compared to those traditional multi-criteria routing algorithms but focuses on single-target scenario. In reality, there are multiple nodes in the network available to provide resources or perform relevant processing operations for services and applications. Therefore, multi-targets and multi-criteria path finding is needed in order to provide users or services better paths. We extend the ParetoBFS by introducing bidirectional search with multi-queues to accommodate the multi-targets scenario and use the original ParetoBFS as a baseline for comparing the performance of the proposed algorithm.

On the other hand, path finding is also important in traffic engineering (TE). In TE algorithms of B4 [14] and SWAN [15], k shortest path algorithm is used to compute paths form a data center to the other data centers in their wide area networks. Also, in other TE schemes [16], [17], [18], [19], k shortest path algorithm is used to find paths for given source-destination pairs. These TE algorithms concern the network flows optimization without considering whether the paths can satisfy the SLA requirements of the services. Also, they do not consider the scenario where a source node can be routed to multiple target nodes. Our work can be a complementary to TE, using our proposed method to obtain Pareto-optimal paths for some demands rather than using k shortest path. Therefore, better paths can be provided to TE. With differentiating the path selection in TE, it will better guarantee SLA of related services while optimizing the overall objectives of network flows.

The aforementioned path finding methods focus on the network operators' perspective. Recently, a user-centric path selection method has been proposed [20]. In this method, both cost and throughput of data transfer in cloud networks are taken into account to identify suitable paths that balance performance and cost. However, this method only considers two criteria and a single destination. Our path finding approach, can be provided to users to help determine optimal paths to multiple destinations while achieving a trade-off between various criteria, facilitating the user to select appropriate paths to connect to target nodes.

III. PROBLEM FORMULATION

A communication network with cloud resources, which can be abstracted as an undirected graph G, with V nodes and E links, where among V nodes there is N nodes with resources, e.g., GPU, CPU, and storage. Assume that there are multiple criteria for each edge, e.g., delay, cost, and bandwidth. Thus, for each edge $\{e(u, v)|u, v \in V\} \in E$, there exists an associated edge criteria $w(u, v) = (w_1, w_2, \cdots, w_K)$, where K represents the number of criteria. Basically, there exists two types of criterion, additive-type (e.g., delay) and bottleneck-type (e.g., bandwidth). For additive-type criterion, a path's criterion is calculated as the sum of criterion of each link in the path. For bottleneck-type criterion, a path's criterion is determined by the minimum criterion value of the links in the path. In some studies, the bottleneck-type criterion is alternatively referred to as the concave type criterion. Multiplicative criteria (e.g. link reliability) may be classified as another type of criterion, however, they can be transformed into additive-type criteria by using a logarithm transformation. In a network with multiple resource nodes, a service may be severed by one of these nodes. Therefore, for a service flow originated at a source node and need to reach one of the target nodes, in order to find optimal paths in the multi-criteria scenario, we need to find Pareto optimal paths. A path p from a source node s to a target node t is constituted by a sequence of edges that connects a sequence of vertices $(s, v1, v2, \cdots, t)$. Since multiple criteria exist within the links, a path can be associated with a path criteria vector $w(p) = \{w(p1), w(p2), \cdots, w(pK)\}$, where $w(pi) = \sum e_{u,v \in p} w_i(u, v)$ if $w_i(u, v)$ belongs to the additive-type criterion, where $w(pi) = \min(w_i(u, v), \forall e_{u,v \in p})$ if $w_i(u, v)$ belongs to the bottleneck-type criterion.

Multiple criteria of links result in multiple criteria of paths. Since a path contains multiple criteria, it is required to use Pareto-optimality to compare paths.That is, a path is Pareto-optimal if it is not be dominated by any other paths. Here, "dominate" indicates that a path's criteria are superior to those of another path when considering all criteria for comparison. To elaborate, a path p dominates another path q if and only if $w_i(p) \succeq w_i(q), \forall i \in \{1, 2, \cdots, K\}$, where the notation $\succeq$ represents optimal or equal, and strict inequality should holds at least once. Thus, a path p is Pareto-optimal if and only if it is not dominated by any path q. All not dominated paths constitutes the Pareto-optimal path set.

In this paper, the objective is to identify the Pareto-optimal path set from a source node to multiple target nodes within a given multi-criteria graph G. This differs from

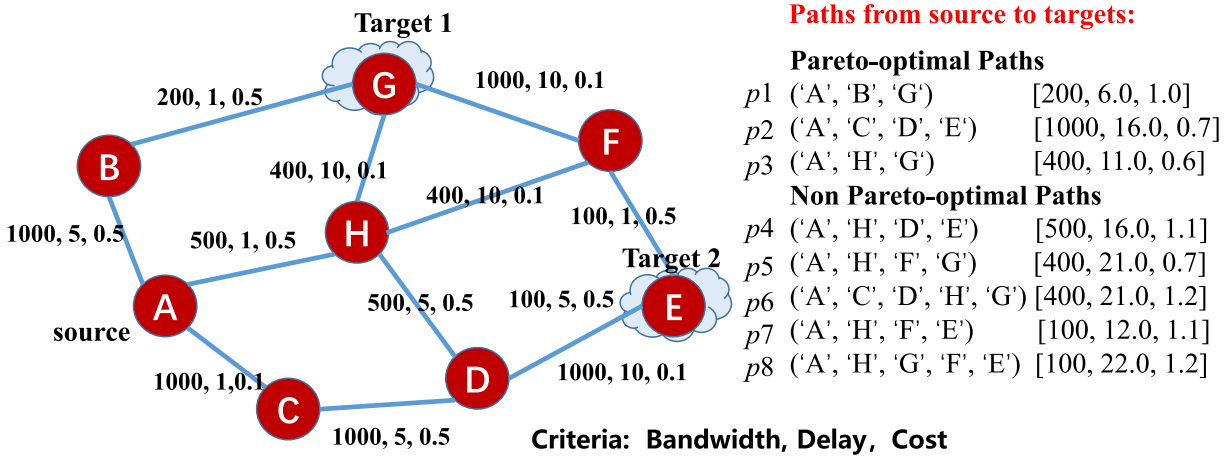

FIGURE 1. Example of Pareto-optimal path finding from the source node A to target nodes (G and E).

the conventional Pareto-optimal path-finding problem, which considers only a single target node.

To better illustrate this problem, an example is provided in Fig. 1. The edges of the graph are labeled with three metrics, bandwidth ($w1$), delay ($w2$), cost ($w3$). Node G and node E are the nodes with resources in the graph. A demand f originating from source node A must be routed to either node G or node E in order to access resources. There are 8 paths $(p1, p2, \cdots, p8)$ from node A to node G or node E. Among those Pareto-optimal paths, no paths are strictly more optimal than the others. For instance, $p1$ exhibits lower delay and cost compared to $p2$, while $p2$ possesses a larger bandwidth. However, any non Pareto-optimal path is dominated by at least one of the Pareto-optimal paths. As an illustration, $p3$ is more optimal than $p5$ since $w_2(p3) < w_2(p5)$ and $w_3(p3) < w_3(p5)$ while $w_1(p3) = w_1(p5)$. Ultimately, we identify the Pareto-optimal paths $p1 - p3$, taking multiple targets into consideration. Note that here is different from previous work, where only one target is take into account. In our case, the final optimal paths need to be determined by paths of multiple target nodes. For instance, if only Target 2 exists, $p7$ would be a Pareto-optimal path, as it is not dominated by $p2$. However, when two targets are present, $p7$ becomes dominated by both $p1$ and $p3$, thus rendering it a non-Pareto-optimal path.

In the illustrated example involving two destinations G and E, if ParetoBFS is applied, the Pareto-optimal path set from A to G: $p1 : (`A',` B',` G')$; $p3 : (`A',` H',` G')$ and that from A to E: $p2 : (`A',` C',` D',` E')$; $p7 : (`A',` H',` F',` E')$ need to be computed separately. Once these Pareto-optimal paths are obtained during individual path finding processes, they are utilized for comparison with paths awaiting expansion, determining whether pruning operations are performed on the latter. Ultimately, the two Pareto-optimal path sets are merged and non-Pareto-optimal paths are filtered out, leading to the ultimate Pareto-optimal path set for both destinations: $p1 : (`A',` B',` G')$; $p2 : (`A',` C',` D',` E')$; $p3 : (`A',` H',` G')$.

In contrast, our approach employs a multi-queues based bidirectional path finding strategy, simultaneously conducting path finding from source node A towards target nodes G and E (as depicted in Fig. 2). This bidirectional strategy accelerates the path finding process. Additionally, we utilize an overall Pareto-optimal path set for both G and E instead of separate sets to facilitate path pruning, which might lead to the early identification that $p7$ is not Pareto-optimal. Consequently, this path is discarded in advance, eliminating the need to compare it with other paths during the path finding process, thus reducing comparative operations and saving time.

There may exist many Pareto-optimal paths for multiple target nodes. Network operators can further consider each service's SLA requirements, each target node's resource status and overall network state, and use traffic engineering techniques with these paths as one of the input to achieve holistic network level optimization such as minimizing maximal link utilization and maximizing throughput, and guarantee the SLAs of services.

IV. PROPOSED METHOD

In this section, we first illustrate the original Pareto-optimal path finding algorithm ParetoBFS, which is based on BFS and pruning technique to effectively traverse all the paths between the source and the target. ParetoBFS is proven to have many advantages compared to traditional multi-criteria path computation algorithm, such as handling both bottleneck-type and additive-type criterion, and identifying all Pareto-optimal paths between the source and target. Then we extend the path finding scenario to accommodate multiple target nodes and proposed a new algorithm based on ParetoBFS that incorporates bidirectional search to accelerate the path finding process. In the multi-criteria path finding problem, the stopping condition for the bidirectional search is different from that in the single criterion scenario. In the single

criterion case, when forward search and backward search reach the same node during path finding, then the whole path find process can be considered complete. However, due to multi-criteria nature of paths, it remains possible to discover other Pareto-optimal paths even forward search and backward search converge at the same node. We use a similar stopping condition presented in [7], which can guarantee that there is no need to further explore the remaining unvisited paths. For the stopping condition for bidirectional search, we design a multi-queues method to determine the optimal value of the potential paths from source node to target nodes extending from the remaining unexplored paths.

A. ORIGINAL PARETOBFS ALGORITHM

The original ParetoBFS is present in [6]. Here we briefly introduce the basic process. The ParetoBFS optimal path finding algorithm employs the BFS traversal concept, using a first-in-first-out (FIFO) queue to enqueue and dequeue paths while continuously expanding paths from the source node to the target node. The execution process of the algorithm is based on path pruning, path extension, and Pareto-optimal path sets updating for each node. Path pruning uses two conditions to check if a path is Pareto-optimal for Pareto-optimal path set of the current node (the tail node of the path) and target node: (i) Whether this path is dominated by the Pareto-optimal path set of the current node and (ii) Whether this path is dominated by the Pareto-optimal path set of the target node. If this path is not dominated in these two conditions, consider it as temporarily Pareto-optimal, add it to the current node's Pareto-optimal path set, and extend it with its connected nodes to form new paths to be enqueued; otherwise, prune the path (remove it from the queue and do not expand further). By enqueueing and dequeueing paths, expand paths, assess optimality, and prune, traversing all paths from the source to the target. When there are no more paths in the queue, the algorithm stops, and the Pareto-optimal path set of the target node is found. Note that,Pareto-optimal path set of the target node contains all Pareto-optimal paths from the source node to the target node.

B. BIDIRECTIONAL MULTIPLE TARGETS-PARETOBFS ALGORITHM

The original ParetoBFS is primarily designed to handle one target node. However, in networks, multiple nodes are capable of serving the demand originated at a source node. To obtain Pareto-optimal paths concerning all target nodes, we could execute ParetoBFS N times, where N is equal to the number of target nodes. After calculation of each target node's Pareto-optimal path set, the elimination of dominated paths in these sets is also needed in order to identify the final Pareto-optimal paths for all target nodes. Therefore, in the scenario of multiple target nodes, using ParetoBFS would be time-consuming and less efficient. Thus, we extend the ParetoBFS and incorporate bidirectional search to find all

Algorithm 1 Bidirectional Multi Targets ParetoBFS

Input: G, Source, Targets
Output: *target_pareto_set*;

```
target_pareto_set ← ∅;
for v ∈ G(V) do
    forward_pareto_set[v] ← ∅;
    for target ∈ targets do
        multi_backward_pareto_set[target][v] ← ∅;
    end for
end for
forward_path_queue ← ∅
Multi_backward_path_queue ← ∅
forward_pareto_set[v].add(path(source))
forward_path_queue.push(path(source))
for target ∈ targets do
    multi_backward_pareto_set[target][v].add(path(target))
    multi_backward_path_queue[target].push(path(target))
end for
while optimal{forward_path_queue} + optimal{multi_backward_path_queue} is Pareto-optimal for target_path_set do
    next_search ← get_next_search();
    if next_search = 0 then
        ForwardSearching(G, targets);
    else if next_search > 0 then
        Backward_Searching(G, source, targets[nextsearch]);
    else
        Break
    end if
end while
return target_pareto_set
```

Pareto-optimal paths from a source node to multiple target nodes. Our approach is termed as Bidirectional Multiple Targets-ParetoBFS algorithm (BMT-ParetoBFS).

The complete BMT-ParetoBFS algorithm is depicted in Algorithm 1, which requires the invocation of Algorithm 2, 3, 4, and 5. Both Algorithm 2 and Algorithm 3 are extensions of ParetoBFS that account for multiple target nodes; these correspond to forward search and backward search, respectively. The main idea is to execute forward search and backward search in sequence to extend and find the Pareto-optimal paths regarding multiple target nodes.

First, we introduce algorithm 1, the main path finding process. The whole pseudo code is shown in algorithm 1, where the input is the graph (with multi-criteria links), the source node and multiple target nodes, while the output is *target_pareto_set*, the Pareto-optimal path set which contains all Pareto-optimal paths from the source node to these target nodes. Some temporary paths need to be stored for extension and comparison during path finding. *forward_pareto_set* stores the Pareto-optimal path sets of each node, and these sets contains Pareto-optimal paths from the source node to the corresponding node during forward search. *Multi_backward_pareto_sets* contains multiple Pareto-optimal path sets of each node for each backward search, which contains Pareto-optimal paths from

a target node to corresponding nodes. *Forward_path_queue* is a queue to store the temporary path waiting to be explored in forward search while *backward_path_queues* are queues performing same function in backward searches. *Multi_backward_path_queues* is comprised of these backward path queues of which the amount is equal to the number of the target nodes. Note that *target_pareto_set*, *forward_pareto_set*, *multi_backward_pareto_sets*, *forward_path_queue*, and *multi_backward_path_queues* initially are set as empty (line 1-9 in algorithm 1). Then the source node is pushed into *forward_path_queue* and each target nodes are pushed into each *backward_path_queue*.Also, source node and target nodes are added into *forward_pareto_set* and *multi_backward_pareto_set*, respectively. These operations are to finish the related initialization. Then the main progress are executed. When the stopping condition is not met, the forward and backward search are executed alternately in a specific order using a get next search function. This function identifies the subsequent search to execute by sequentially and iteratively generating numbers between 0 and $N-1$. The number 0 is corresponding to the forward search and number between 1 and $N-1$ is corresponding to backward search from different target nodes. If a specific search has been completed or halted, then it will be bypassed. Supposed there are N target nodes, first a forward search (algorithm 2) is executed and following the N times backward search (algorithm 3). For N times backward search, first, a backward search is from the first target node, sequentially, another backward search is from the second target node, and so on. Each search extends paths by enqueueing, dequeueing, and pruning paths. During the path finding, each search updates the relevant nodes' Pareto-optimal path set using *pareto_add* function in Algorithm 4 and combine path from other searches using *combine_path* function in Algorithm 4. Some of the searches may be stopped in advance when the corresponding path queue is empty. When the stopping conditions are met or there are no more paths to explore in each path queue, the algorithm terminates, resulting in the final Pareto-optimal path set regarding all target nodes.

Algorithm 2 describes the specific process of forward search. First, the *forward_path_queue* dequeues the path from the path queue by pop operation and the last node ($s1$) of the path is determined (dequeue process). Then the path is checked whether it is Pareto-optimal to *target_pareto_set* and to *forward_pareto_set*[$s1$]. If not, it means that there exists at least a better path in *target_pareto_set* or *forward_pareto_set*[$s1$], leading to no need to further expand the current path, because extension cannot make a suboptimal path optimal. If so, then we find out all the connected nodes ($s2$) of $s1$ according to the topology information, and extend the path with a new edge ($s1, s2$). The *out_edges* means finding all the connected edges of $s1$ and *dest_node* means identifying the end node of the edge. Then new path can be formed by appending the new edge to the original path. The number of connected next nodes determines the number of new paths created. Therefore, for each new path we need to repeat the following operations (The loop in line 4 in Algorithm 2). Note that before adding the new edge, we check whether $s2$ is in the path in order to prevent path loops (line 6 in Algorithm 2). If the original path contains $s2$, it then will form a loop when adding the new edge, thus this new edge will not be considered. Once a new path is formed, we again check its Pareto optimality to *target_pareto_set* and *forward_pareto_set*[$s2$]. The reason is the same as before: extending suboptimal paths is unnecessary, as they will not be transformed into optimal ones in the future. After Pareto-optimality check, the new path is temporally Pareto optimal and if $s2$ is one of the targets, we add the new path in to *target_pareto_set*[$s2$]. Otherwise, we add the new path in to *forward_pareto_set*[$s2$] using *pareto_add* function in Algorithm 4. Meanwhile, we enqueue the new path into the *forward_path_queue*. Then we loop all targets to see whether the backward search from each target executed by algorithm 4 has reached $s2$. If so, then we can combine the new path and the path from the target to $s2$ to get a complete path from the source node to one target node, using *combine_path* function

Algorithm 2 Forward Search

```
1:  path ← forward_path_queue.pop();
2:  s1 ← path.end();
3:  if path is Pareto-optimal for target_pareto_set and
    path ∈ forward_pareto_set[s1] then
4:      for edge ∈ s1.out_edges() do
5:          s2 ← edge.dest_node()
6:          if s2 ∉ path then
7:              new_path ← path.append(edge)
8:              if new_path is Pareto-optimal to
    target_pareto_set and forward_pareto_set[s2] then
9:                  if s2 not in targets then
10:                     forward_pareto_set[s2] ←
    pareto_add(forward_pareto_set[s2], new_path)
11:                     forward_path_queue.push(new_path)
12:                     for target ∈ targets do
13:                         if
    multi_backward_pareto_sets[target][s2] ≠ ∅ then
14:                             target_pareto_set ←
    combine_path(target_pareto_set,
    multi_backward_path_queue[target][s2], new_path)
15:                         end if
16:                     end for
17:                 else
18:                     target_pareto_set ←
    pareto_add(target_pareto_set, new_path)
19:                 end if
20:             end if
21:         end if
22:     end for
23: end if
24: return target_pareto_set
```

Algorithm 3 Backward Search

```
backward_pareto_set ← multi_backward_pareto_sets[target]
backward_path_queue ← multi_backward_path_queues[target]
path ← backward_path_queue.pop()
s1 ← path.end()
if path is Pareto-optimal for target_pareto_set and path ∈ backward_pareto_set[s1] then
    for edge ∈ s1.out_edges() do
        s2 ← edge.dest_node()
        if s2 ∉ path then
            new_path ← path.append(edge)
            if new_path is Pareto-optimal to target_pareto_set and backward_pareto_set[s2] then
                if s2 ≠ source then
                    backward_pareto_set[s2] ← pareto_add(backward_pareto_set[s2], new_path)
                    backward_path_queue.push(new_path)
                    if forward_pareto_set[s2] ≠ ∅ then
                        target_pareto_set ← combine_path(target_pareto_set, forward_pareto_set[s2], new_path)
                    end if
                else
                    target_pareto_set ← pareto_add(target_pareto_set, new_path.reverse())
                end if
            end if
        end if
    end for
end if
```

Algorithm 4 *pareto_add*

```
if new_path ∈ pareto_set then
    return pareto_set
end if
result_pareto_set ← ∅
for path in pareto_set do
    if path dominates new_path then return pareto_set
    else if new_path does not dominate path then
        result_pareto_set.append(path)
    end if
end for
result_pareto_set.append(new_path)
return result_pareto_set
```

Algorithm 5 *combine_path*

```
new_path_set ← ∅
for another_temp_path ∈ temp_pareto_set do
    new_path_set.add (combine(temp_path, another_temp_path))
end for
result_pareto_set ← target_pareto_set
for new_path in new_path_set do
    result_pareto_set ← pareto_add(result_pareto_set, path)
end for
return result_pareto_set
```

in Algorithm 5. Then this full new path from source node to one target node is added into *target_pareto_set* if it is Pareto optimal to this path set. Here, a forward search has been completed, and the Pareto-optimal path sets of the relevant nodes has been updated.

The Algorithm 3 describes the backward search of optimal paths form target nodes to the source node, sharing the similar logic of forward search in Algorithm 2, which also use a path queue to dequeue paths and enqueue new extending paths, combine new paths and the paths formed by forward searching to obtain full new paths and updates the *multi_backward_pareto_sets* and *target_pareto_set*.

Pareto_add function in Algorithm 4 is used to add new path into related Pareto path set and remove the existing path dominated by the new path. *Combine_path* function in Algorithm 5 is used to combine the path formed by forward and backward search. The path found by forward (or backward) search from the source node (or target nodes) to the intermediate node are combined with the found Pareto-optimal paths in backward (or forward) Pareto-optimal set of this node to form a full new path from the source node to one target node. It may also remove some exiting paths in the *target_pareto_set* if these paths are dominated by the new path.

Note that in the multi-criteria path finding scenario, stopping condition for bidirectional search is not when two search process reach at the same node. It requires to calculate the potential optimal path value of forward search and backward search regarding all the criteria. Then the values of a virtual optimal path can be obtained by combine the optimal values of remaining paths in forward search queue and backward search queues. If such a virtual optimal path is dominated by any paths in *target_pareto_set*, then it is no need to explore the remaining paths.

In order to effectively determine whether the stopping condition is met (the line 16 in Algorithm 1), we design a structure of multiple queues to achieve the potential optimal path value of the remaining paths in the path queues. The structure is shown in Fig. 2, for each search there are a path queue (FIFO) and K monotonic queues (K is the number of criteria) to obtain the potential optimal value of the remaining paths in this path queue. When a path is enqueued, it is added to the end of the FIFO path queue. At the same time, the K criteria of this path are added to the corresponding monotonic queues. When a path is dequeued from the path queue, its corresponding criteria elements are removed from the monotonic queues. The enqueue and dequeue processes of a monotonic queue differ from those of a FIFO queue and the monotonic queues can facilitate to obtain the optimal values for each criteria of the paths in the FIFO path queue.

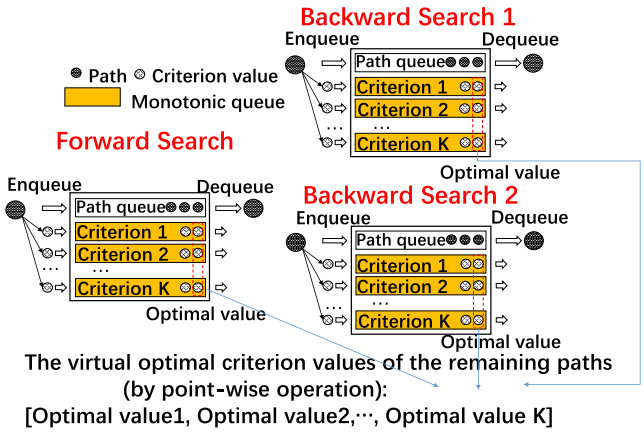

FIGURE 2. The illustration of path queues and K monotonic queues in forward and backward search to obtain the virtual optimal values for remaining unexplored paths.

In a monotonic queue, elements are arranged monotonically from the tail to the head, while in a FIFO queue, they are organized according to their enqueue order. Monotonicity is achieved in a monotonic queue as follows: Assume that, in the queue, the tail element is the maximum, the head element is the minimum, and the minimum value represents the optimal value. When a new element is enqueued, compare it with the tail element; if the new element is larger, then add it into the monotonic queue as the new tail element; otherwise, remove the tail element, and continue removing elements until the new element is larger, then the new element becomes the tail element. Consequently, the optimal criterion values of each monotonic queue can be fast obtained from their head elements. By integrating these optimal values, one can derive the potential optimal values of unexplored paths from source node to target nodes. Since these potential value are inferred from the remaining paths in path queues, we assume there is a virtual path with these potential optimal values. Specifically, the path values of the virtual optimal path are constructed by point-wise operation on the optimal values of forward search queue and backward search queues. For additive criteria, the optimal value is the sum of optimal value of forward search and backward search queues. For bottleneck-type criteria, the optimal value is determined by the minimum value of forward search and backward search queues. Therefore, combing the FIFO path queues and their corresponding K monotonic queues of forward search and backward searches, the optimal values of the virtual optimal paths from source node to target nodes are obtained. Then the virtual optimal path is used to compare to the paths in *target_pareoto_set* and determine whether the algorithm 1 can be stopped. Once the virtual optimal path is dominated by one of the Pareto-optimal paths of *target_pareoto_set*, there is no need to continue the path searching, and all Pareto-optimal paths from source node to target nodes are obtained.

Multi-criteria path finding has been proven to be an NP hard problem [12], [13]. The complexity of original ParetoBFS is $O(nmkp^*)$ [6], where n, m are the amount of nodes and links in the topology, respectively, and p^* is the amount of Pareto-optimal paths from the source node to the target nodes. Therefore, for multiple target nodes scenario, the complexity of invoking N time ParetoBFS would be $O(Nnmkp^*)$ and now p^* is determined by the largest Pareto-optimal path set of these target nodes. Since the forward search and backward searches of BMT-ParetoBFS are extensions of ParetoBFS, in the worst case, BMT-ParetoBFS will have the same complexity as simply using N times ParetoBFS. However, in BMT-ParetoBFS, some non Pareto-optimal paths may be eliminated in advance during path finding or some Pareto-optimal paths can be obtained faster by combine the paths found by forward search and backward search. Therefore, BMT-ParetoBFS can reduce running time compared to PareotBFS in the scenario of multiple target nodes.

C. BMT-PARETOBFS WITH SAMPLING ON THE TOPOLOGY

BMT-ParetoBFS can find all Pareto-optimal paths from a source node to multiple target nodes. With the increase in the number of criteria or the expansion of the topology, the running time of BMT-ParetoBFS also increases. As a result, even faster than the ParetoBFS, it still becomes somehow too slow to find all the Pareto-optimal paths when the topology is large or with many criteria. In [6], some sampling techniques are proposed to sample some Pareto-optimal paths to compare with the new path during path finding rather than using the whole Pareto-optimal path set. Here, we consider another type of sampling technique, constructing a sub-topology from the original topology with k-shortest algorithm for each criterion, instead of sampling the Pareto-optimal path set. The sub-topology construction is based on the observation that during a Pareto-optimal path fining it may not need to traverse all the intermediate nodes. Or in other words, it has a good chance to find out most of Pareto-optimal paths on the sub-topology constructed from original topology. First, from the source node to each target node, we use k shortest path algorithm to find out k shortest path regarding each criterion. Therefore, for N target nodes and K criteria, we can obtain $k * K * N$ paths. These paths may have duplicate edges and from these path a sub-topology can be formed. Comparing to the original topology, the sub-topology is significant smaller, leading to a running speed improvement when using BMT-ParetoBFS.

Supposed that BMT-ParetoBFS finds the Pareto-optimal path set $P = \{p_1, \cdots, p_m\}$ in the original topology and the Pareto-optimal set path $Q = \{q_1, \cdots, q_n\}$ in the sub-topology. We use three metric, Running Time Ratio (RT), Path Count Ratio (PC) and Path Quality (PQ) that are defined in [6] to compare these two path sets. Their definitions are explained in Table 1. T_1 is running time of BMT-ParetoBFS performed on the sub-topology while T_2 is that performed on the entire topology. n and m are the size of path set Q and P, respectively. RT and PC can be easily obtained according to the definitions. The PQ is more complicated and the following process describes how to calculate the PQ: (i) normalize the path criterion value of each path for each

TABLE 1. The explanation of RT, PC, and PQ.

Running Time Ratio (RT)	$\frac{T_1}{T_2}$
Path Count Ratio (PC)	$\frac{n}{m}$
Path Quality (PQ)	Average K-dimensional Euclidean distance between P's and Q's criteria vector sets $w(Q) = \{w(q_1), \cdots, w(q_n)\}$ and $w(P) = \{w(p1), \cdots, w(p_m)\}$

criterion, (ii) for every path in Q, calculate the distances of all criteria between it and each path in P, then select the minimum distance as the path distance for this path in Q and the most approximate path from P (iii) average these n path distances to obtain PQ. RT indicates acceleration efficiency by introducing the sampling method, PC indicates the extent to which the complete Pareto path set is found using the sampling method, and PQ indicates the difference between optimal paths found with and without sampling method.

D. THE APPLICATION OF THE PROPOSED METHOD

We have described the algorithm aspect to achieve Pareto-optimal paths in the previous sections. Here we describe how to how to apply the proposed method in production level networks. First we need a SDN controller and some network telemetry mechanism to collect the overall network status information, such as delay, jitter, and available bandwidth. Cost criterion of links is determined by actual link construction cost and operation cost, and may not be able to be collected from the link status directly. Therefore, it may require some prior information from the network operators or from other related network operation and management systems. We also need to select a few nodes as candidate target nodes. With all the criteria of network links are collected and target nodes are determined, the Pareto-optimal path from a source node to multiple target nodes can be calculated with the proposed method. After path calculation, the Pareto-optimal paths can be obtained. The network operators can further consider the whole network status and the SLA requirements of the services and then select the most appropriate paths accordingly from the Pareto-optimal path set. Then the selected paths can be installed by the SDN controller on the related routers. Since production level networks are dynamic changing, it is required to periodically to inform the SDN controller with the newest network status information, then re-execute path computation and path installation to deal with the network dynamics.

V. EXPERIMENTS AND DISCUSSIONS

A. EXPERIMENTAL SETUP AND DATASET

We use the real-world backbone network topology dataset: Rocketfuel [21], which is measured by University of Washington. In the experiment, we use 5 topologies, the size of which ranging from 79 nodes to 353 nodes. The details of these topology are shown in Table 2. Note that since some nodes and links are isolated in the original topologies, we only use the largest connected components of these topologies. As some of original topology data only contain edges and nodes without any criteria such as bandwidth or latency, we randomly generate values for the criteria using a uniform distribution. In the experiments, the values of the criteria are established as follows: For the bandwidth criterion, the range was set between 100 and 2500, with a granularity of 10; For the delay criterion, the range is set between 1 and 35, with a granularity of 1; For the cost criterion, the range is set between 0.1 and 1, with a granularity of 0.1; For all other criteria utilized in the experiment, the range is set between 0.1 and 1, with a granularity of 0.1. In order to reduce the impact of specific source and target node configurations, decrease random errors, and improve the reliability of our experiments, we implement 100 tests for each topology. In each test, one node is randomly selected as the source, while 5 or 10 nodes are randomly designated as target nodes, respectively. The final algorithm execution time is determined by averaging the running time of these 100 tests. We use Python to implement the algorithms on a laptop which has a CPU (AMD Ryzen 7 4800H).

To verify the effectiveness of our method, we compare it with the original ParetoBFS algorithm, which has been proven to be effective in finding Pareto-optimal paths in a multi-criteria topology. Since both methods can find out the entire Pareto-optimal path set, here we only compare the running time. Note that compared to the original ParetoBFS algorithm, our algorithm incorporates a bidirectional search mechanism. It performs searches from both the source node and multiple target nodes. For each search, we introduce multiple monotonic queues to obtain potential optimal values for the remaining paths in the path queue. By merging the potential path optimal values from both forward and backward searches, it is possible to reach the termination condition and complete the search earlier. Additionally, we maintain an overall Pareto-optimal path set for the multiple target nodes, which allows us to eliminate non-Pareto-optimal paths associated with these target nodes in advance. This helps reduce some unnecessary path comparison operations.

In addition, we use a topology sampling method to accelerate the BMT-ParetoBFS and we also compare our method with the k-shortest path algorithm [22]. The k-shortest path algorithm is single criteria-based, and we use it to find the first k shortest paths for each criterion. In a K criteria topology, $k * N * K$ paths are obtained when N targets are considered. Besides comparing the running time in this scenario, we also calculate difference between the obtained path sets in the sub-topology and the full Pareto-optimal path set in the original topology.

B. PARETOBFS AND BMT-PARETOBFS COMPARISON

In this section, we present the experimental results of the BMT-ParetoBFS. Original ParetoBFS primarily deal with one target node and thus to deal with multi-targets scenario we need to invoke it multiple times accordingly. Running

TABLE 2. The parameters of the used topologies.

Topology AS(Autonomous System) number	Nodes	Links
AS 3967	79	147
AS 1755	87	161
AS 1755-r0	175	382
AS 1221	318	763
AS 3967-r1	353	820

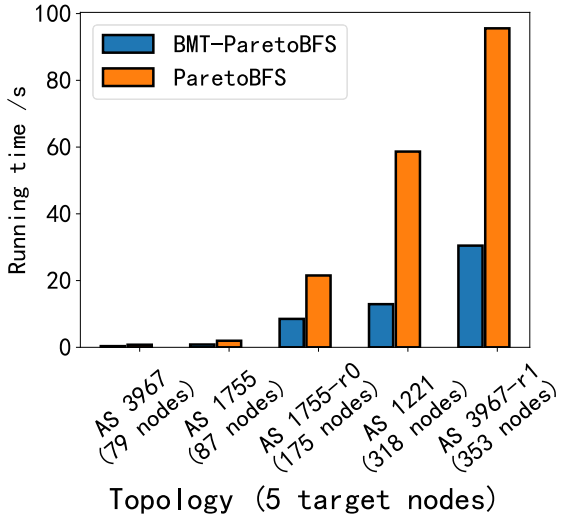

FIGURE 3. The running time of BMT-ParetoBFS and ParetoBFS using with different topologies (5 target nodes).

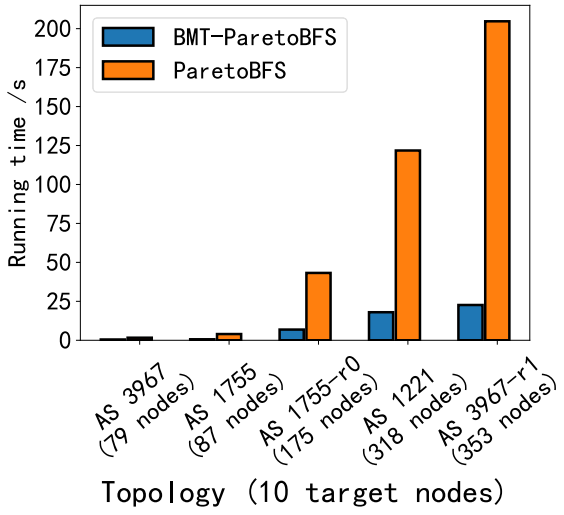

FIGURE 4. The running time of BMT-ParetoBFS and ParetoBFS using with different topologies (10 target nodes).

time here is recorded only about the execution time of core functions without considering the execution time of some input and output procedure. First, we present the running time of the ParetoBFS and BMT-ParetoBFS on topologies with 5 target nodes and 10 target nodes, shown in Fig. 3 and Fig. 4, respectively. In this experiment, we consider 3 criteria, which corresponds to bandwidth, latency and cost. It can be seen from the Fig. 3 and Fig. 4, for all the topologies,

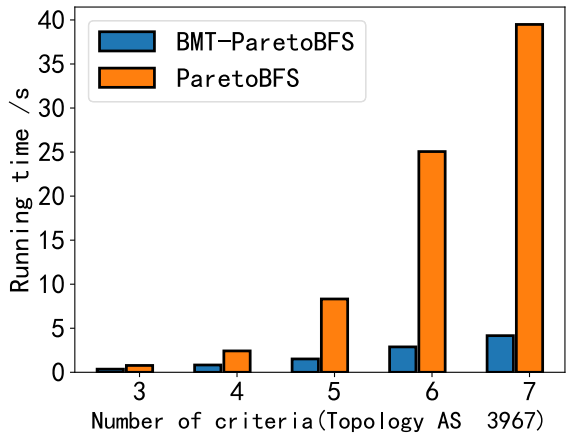

FIGURE 5. The running time of BMT-ParetoBFS and ParetoBFS with different criteria (topology AS 3967 with 79 nodes, 5 target nodes).

BMT-ParetoBFS outperforms ParetoBFS in terms of the running time. As the topology size increases, our algorithm exhibits lower increase in time consumption compared to the ParetoBFS algorithm. In these topologies, the ParetoBFS algorithm's running time is approximately 2 to 9 times longer than that of our algorithm. Therefore, our approach reduces running time by $54\% - 89\%$ in these cases. Our algorithm is faster than invoking ParetoBFS multiple times because it employs a bidirectional search method, which may expedite the finding of Pareto optimal paths. Moreover, we concurrently take into account the Pareto-optimal paths of multiple target nodes. Throughout the search process, we discard paths that are Pareto optimal solely for a single target node but not for all target nodes, thus avoiding comparisons with them in subsequent comparison processes and further accelerating the path finding. Consequently, as the number of target nodes increases, the running time of BMT-ParetoBFS may not necessarily increase; however, the running time of ParetoBFS will consistently increase. This is due to the requirement of invoking the ParetoBFS algorithm once for every individual target node.

To investigate the influence of the number of criteria, we fix a topology with 79 nodes and compare the running time of the two algorithms under different criterion conditions by increasing the number of criteria. The result is shown in Fig. 5. From Fig. 5, it can be observed that the running time of both algorithms increases as the number of criteria grows. This is because a higher number of criteria produces more Pareto-optimal paths, leading to longer running time. Besides, these paths need to be compared with other paths during the path finding, resulting in increased execution time. However, the running time of ParetoBFS increases more rapidly than that of the BMT-ParetoBFS algorithm. When the number of criteria increases from 3 to 7, the ParetoBFS's running time is about 2 to 9 times greater than the BMT-Pareto's running time, a conclusion similar to that drawn from Fig. 3 and Fig. 4. The reason is also the same, our method can combine forward search and backward search

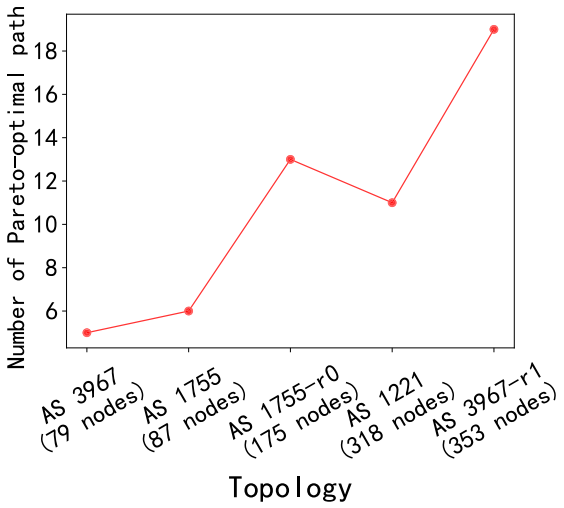

FIGURE 6. The number of optimal paths found on different Topologies (3 criteria, 5 target nodes).

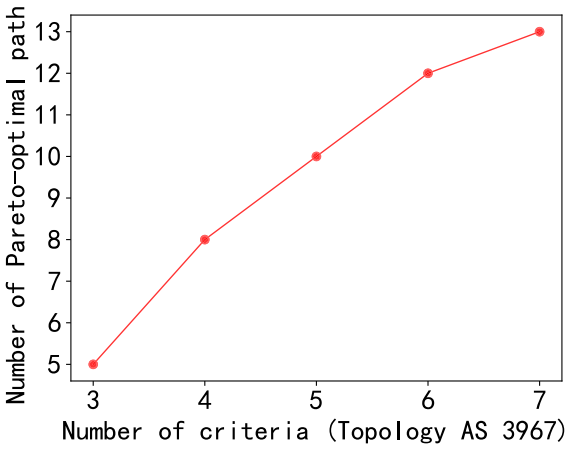

FIGURE 7. The number of optimal paths found on Topology AS 3967 (79 nodes, 5 target nodes) with different number of criteria.

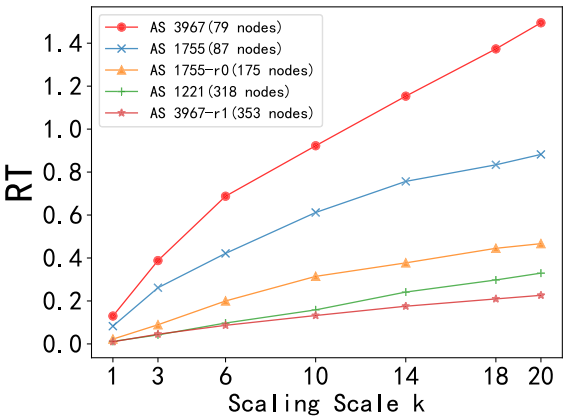

FIGURE 8. Running Time Ratio (RT) on 5 topologies with different sampling scale k.

to obtain paths faster and discard some non-optimal paths earlier.

We also investigate the number of Pareto-optimal paths in different scenarios. In the case of 3 criteria and 5 target nodes, basically, as the topology grows, the number of Pareto-optimal paths increases, as shown in Fig. 6. However, there is a special case. Although topology AS 1221 (318 nodes, 763 links) is larger than topology AS 1775-r0 (175 nodes, 382 links), it ends up having fewer Pareto-optimal paths, which may be related to the specific structure of the topology. On the other hand, we select topology AS 3967 specified with 5 target nodes and study the changes in the number of Pareto-optimal paths with the increase in the number of criteria. As can be seen from Fig. 7, as the number of criteria increases, the number of Pareto-optimal paths also increases, which consequently leads to an increase in the algorithm running time, as shown in Fig. 5.

The multi-criteria path finding problem is NP-hard. As the scale of the problem increases, the computational effort required for path finding does not increase linearly but exponentially, therefore simply allocating more computing resources may not be an efficient approach. Hence, developing a more efficient multi-criteria path finding algorithm holds significant value. Our algorithm can compute Pareto-optimal paths for multiple target nodes faster under the same computing resource constraints. In other words, our algorithm enables us to provide path finding for a larger number of users within the given computing resources.

C. BMT-PARETOBFS ACCELERATION WITH SAMPLING

To investigate the acceleration effect, we test different topologies, and run BMT-ParetoBFS on these whole topologies and their sampled topologies (sub-topologies), respectively. First, we compare the BMT-ParetoBFS with and without sampling on different topologies and calculate the three metrics in Table 2. By constructing sub-topologies through spatial sampling on the original topology, our algorithm can achieve effective acceleration, with the speed improvement being related to the spatial sampling scale (i.e., the value of k in k-shortest path algorithm). Meanwhile, the spatial sampling scale also affects the final path quality. As shown in Fig. 8, Fig. 9, and Fig. 10, as the spatial sampling scale increases, the algorithm acceleration efficiency decreases (higher RT) while the path quality improves (higher PC and lower PQ). Only a special case exists in Fig. 10, for the curve of topology AS 1755, the PQ value at $k = 1$ is smaller than other values at different k. It may be attributed to that in this case the algorithm finds less Pareto-optimal paths and less non Pareto-optimal paths and thus achieve a high average PQ. However, its corresponding PC in Fig. 9 is increasing with sampling scale k, which means more Pareto-optimal paths are found, therefore, overall path quality still can be considered as improving.

Spatial topology sampling can reduce the scale of the original topology, so even if the number of criteria in the topology increases, the sub-topology remains relatively

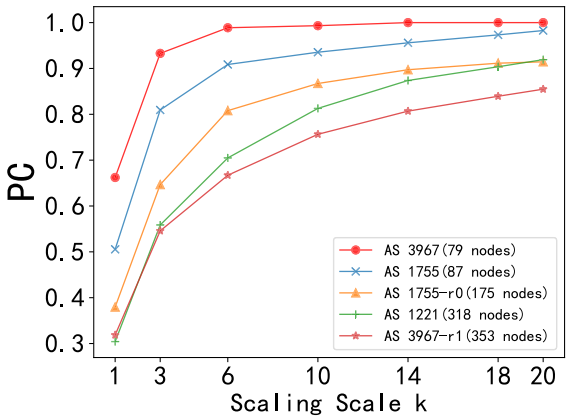

FIGURE 9. Path Count Ratio (PC) on 5 topologies with different sampling scale k.

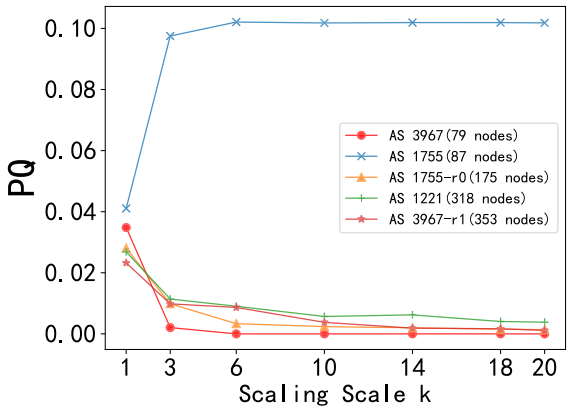

FIGURE 10. Path Quality (PQ) on 5 topologies with different sampling scale k.

small, resulting in less running time compared to that when applying the BMT-ParetoBFS algorithm on the original topology. However, it should be noted that for small topologies, an overly large spatial sampling scale may actually result in a slower overall algorithm speed. This is because the algorithm itself computes relatively quickly for small topologies; at this point, introducing the spatial sampling to construct sub-topologies adds extra running time. In this case, the algorithm's acceleration on the sub-topology does not effectively exceed the additional time, leading to no improvement or even a decrease in overall performance. It is demonstrated in the curve for topology AS 3967 in Fig. 8, when k is at 14, 18, and 20, the corresponding RT is larger than 1, which means with spatial sampling the algorithm becomes slower. In the experiments, the value of k ranges from 1 to 20. For most topologies, when the number of Pareto-optimal paths is approximately 90% of the complete Pareto-optimal path set, the algorithm only needs 30%-40% of the original running time, resulting in an acceleration of approximately 2.5 - 3.3 times. For instance, for the topology AS 1221, when PC is about 0.9 in Fig. 9, the corresponding RT is about 0.3 in Fig. 8. Hence, by employing topology sampling, a trade-off can be achieved between running time and path quality. That is, in the case of increasing criteria numbers, to ensure path quality, it is necessary to correspondingly increase topology sampling, i.e., increase the k value used in constructing the sub-topology with the k-shortest path algorithm.

On the other hand, since we adopt the k-shortest path method to construct sub-topologies, we obtain the k-shortest paths in this process at first. Therefore, we compare our proposed algorithm with the direct k-shortest path algorithm and further demonstrate the effectiveness of our approach. Similarly, we use different topologies and values of k, applying both BMT-ParetoBFS algorithm combined with spatial topology sampling and the direct k-shortest path algorithm for path finding. We then compare their execution time and path quality, and the results are shown in Table 3.

As can be seen from Table 3, the time consumed by the k-shortest path algorithm is significantly less than that of our approach (since our method requires not only the k-shortest path computation but also the Pareto-optimal path calculation). However, our method yields better path quality. Basically, when our approach can obtain about more than 90% of the complete Pareto-optimal paths, the k-shortest path algorithm in most cases can only achieve less than 80% of that. At the same time, the distance between the paths obtained by the k-shortest path algorithm and the paths of the complete Pareto-optimal path set is much larger than that of the paths calculated by our algorithm. This indicates that the paths found by our algorithm with spatial sampling are closer to the full Pareto-optimal paths in the original topology while the paths found by k-shortest path algorithm is significantly different from those paths.

Through these experiments, we demonstrate that the advantages of our method to the k-shortest path algorithm. By constructing sub-topologies using the k-shortest path method and applying the BMT-ParetoBFS algorithm, we can effectively reduce the algorithm running time while finding most of the Pareto-optimal paths.

VI. CONCLUSION

We investigate the multi-criteria optimal path finding problem, which aims to find the Pareto-optimal paths from a given source node to multiple target nodes in a network with multiple criteria. We extend the ParetoBFS algorithm, introduce a multi-queues based bidirectional search method, and propose a multi-criteria path finding algorithm (BMT-ParetoBFS). A large number of experiments are conducted in the publicly real backbone dataset. Compared to the original ParetoBFS algorithm, our algorithm significantly reduces the running time and thus obtains the entire Pareto-optimal path set in a shorter time.

Furthermore, we accelerate BMT-PareotoBFS with spatial topology sampling. We propose using the k-shortest path algorithm to obtain the k-shortest paths from source node to

TABLE 3. Comparison BMT-ParetoBFS algorithm with sampling and *k* shortest paths algorithm on different topologies.

Topology	Algorithm	Target node	Sampling Scale *k*	RT	PC	PQ
AS 3967	BMT-ParetoBFS	5	3	0.38695	0.932886	0.002056
	k shortest path	5	3	0.114263	0.841163	2.126037
AS 1755	BMT-ParetoBFS	5	6	0.426552	0.908661	0.102091
	k shortest path	5	6	0.086066	0.779528	1.692489
AS 1755-r0	BMT-ParetoBFS	5	15	0.412979	0.903226	0.001484
	k shortest path	5	15	0.042172	0.728432	1.314443
AS 1221	BMT-ParetoBFS	10	15	0.25642	0.907463	0.001327
	k shortest path	10	15	0.040445	0.344279	2.899825
AS 3967-r1	BMT-ParetoBFS	10	15	0.373805	0.933948	0.000661
	k shortest path	10	15	0.059211	0.735023	2.137804

target nodes for each criterion, constructing sub-topologies with these single-criterion shortest paths, and performing BMT-ParetoBFS on the sub-topologies. This method further reduces the running time and is possible to balance the sampling degree to ensure that the calculated path set contains most paths of the complete Pareto-optimal path set obtained without topology sampling.

In this paper, we do not consider the differences in target nodes and only consider the multi-criteria for the links. Future research will take into account the impact of different types of target nodes. By considering target nodes with various levels of resource, we can better utilize network resources. Also, if the service characteristics are considered, we could further select the more appropriate paths from the Pareto-optimal path set.

As the development of 5G and cloud computing technologies progresses, the integration between cloud and network deepens. With the growing number of services that can be served by multiple resource nodes in networks, it is crucial to identify comprehensive optimal paths for these services to reach their target nodes. Consequently, our work holds substantial significance for network operators and cloud service providers. By identifying all Pareto-optimal paths, we can better ensure compliance with SLA requirements of services and improve resource utilization, ultimately benefiting both service providers and users.

REFERENCES

[1] R. Coltun, D. Ferguson, J. Moy, and A. Lindem, *OSPF for IPv6*, document RFC 5340, 2008.

[2] G. Malkin and R. Minnear, *RIPng for IPv6*, document RFC2080, 1997.

[3] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman, "Polynomial time approximation algorithms for multi-constrained QoS routing," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 656–669, Jun. 2008.

[4] R. G. Garroppo, S. Giordano, and L. Tavanti, "A survey on multi-constrained optimal path computation: Exact and approximate algorithms," *Comput. Netw.*, vol. 54, no. 17, pp. 3081–3107, Dec. 2010.

[5] Z. Li and J. J. Garcia-Luna-Aceves, "A distributed approach for multi-constrained path selection and routing optimization," in *Proc. 3rd Int. Conf. Quality Service Heterogeneous Wired/Wireless Netw. (QShine)*, 2006, p. 36.

[6] X. Chen, H. Cai, and T. Wolf, "Multi-criteria routing in networks with path choices," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, Nov. 2015, pp. 334–344.

[7] S. Demeyer, J. Goedgebeur, P. Audenaert, M. Pickavet, and P. Demeester, "Speeding up Martins' algorithm for multiple objective shortest path problems," *4OR*, vol. 11, no. 4, pp. 323–348, Dec. 2013.

[8] T. Kurbanov, M. Cuchý, and J. Vokrínek, "Fast one-to-many multicriteria shortest path search," *IEEE Trans. Intell. Transp. Syst.*, early access, Jun. 12, 2023, doi: 10.1109/TITS.2023.3282069.

[9] N. Varyani, Z.-L. Zhang, and D. Dai, "QROUTE: An efficient quality of service (QoS) routing scheme for software-defined overlay networks," *IEEE Access*, vol. 8, pp. 104109–104126, 2020.

[10] N. Saha, S. Bera, and S. Misra, "Sway: Traffic-aware QoS routing in software-defined IoT," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 1, pp. 390–401, Jan. 2021.

[11] C.-L. Hu, C.-Y. Hsu, and W.-M. Sung, "FitPath: QoS-based path selection with fittingness measure in integrated edge computing and software-defined networks," *IEEE Access*, vol. 10, pp. 45576–45593, 2022.

[12] M. Müller-Hannemann and K. Weihe, "On the cardinality of the Pareto set in bicriteria shortest path problems," *Ann. Oper. Res.*, vol. 147, no. 1, pp. 269–286, Oct. 2006.

[13] P. Hansen, "Bicriterion path problems," in *Proc. 3rd Conf., Multiple Criteria Decis. Making Theory Appl.* Hagen/Königswinter, West Germany: Springer, 1980, pp. 109–127.

[14] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Aug. 2013.

[15] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, Aug. 2013, pp. 15–26.

[16] Y. Wang, J. Zheng, L. Tan, and C. Tian, "Joint optimization on bandwidth allocation and route selection in QoE-aware traffic engineering," *IEEE Access*, vol. 7, pp. 3314–3319, 2019.

[17] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. Kleinberg, P. Lapukhov, C. L. Lim, and R. Soulé, "Semi-oblivious traffic engineering: The road not taken," in *Proc. 15th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2018, pp. 157–170.

[18] J. Bogle, N. Bhatia, M. Ghobadi, I. Menache, N. Bjørner, A. Valadarsky, and M. Schapira, "TEAVAR: Striking the right utilization-availability balance in WAN traffic engineering," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 29–43.

[19] J. Zhang, M. Ye, Z. Guo, C.-Y. Yen, and H. J. Chao, "CFR-RL: Traffic engineering with reinforcement learning in SDN," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2249–2259, Oct. 2020.

[20] P. Jain, S. Kumar, S. Wooders, S. G. Patil, J. E. Gonzalez, and I. Stoica, "Skyplane: Optimizing transfer cost and throughput using cloud-aware overlays," in *Proc. 20th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2023, pp. 1375–1389.

[21] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *Proc. 2nd ACM SIGCOMM Workshop Internet Meas. (IMW)*, 2002, pp. 231–236.

[22] J. Y. Yen, "Finding the K shortest loopless paths in a network," *Manage. Sci.*, vol. 17, no. 11, pp. 712–716, Jul. 1971.

XIAOQING XU received the B.S. degree in optical information science and technology and the Ph.D. degree in optics from Sun Yat-sen University (SYSU), Guangzhou, China, in 2013 and 2018, respectively. He is currently a Researcher with Research Institute of China Telecom Company Ltd., China. His main research interests include network design, network optimization, and AI in networks.

XIAOJUN LIU received the M.S. degree from the Huazhong University of Science and Technology, Wuhan, China. He is currently a Researcher with Research Institute of China Telecom Company Ltd. His research interests include network intelligence, XR, cloud computing, and big data analysis.

LIUYIHUI QIAN received the B.S. and M.S. degrees in optical engineering from the Huazhong University of Science and Technology, Wuhan, China. He is currently a Researcher with Research Institute of China Telecom Company Ltd. His research interests include virtual reality, image processing, machine learning, and network optimization.

NING ZHANG received the B.S. and M.S. degrees from the School of Intelligent Systems Engineering, Sun Yat-sen University (SYSU), Guangzhou, China, in 2020 and 2023, respectively. He is currently a Researcher with Research Institute of China Telecom Company Ltd., China. His primary research interests include network design, network optimization, incentive mechanism, and federated learning.

JUAN WU received the M.S. degree in electrical engineering from the South China University of Technology, Guangzhou, China. She is currently a Researcher with Research Institute of China Telecom Company Ltd. Her research interests include network intelligence, XR, machine learning, video encoding, and big data analysis.

HONG TANG received the B.S. degree from the Huazhong University of Science and Technology, Wuhan, China, in 1997, and the M.S. degree in optical engineering from Jinan University, Guangzhou, China, in 2003. He is currently the Chief Expert in IP networks with Research Institute of China Telecom Company Ltd. His main research interests include traffic planning in IP backbone networks, traffics scheduling in software-defined networks, and network function virtualization.