An Efficient Binary-Search Based Heuristic for Extended Unsplittable Flow Problem

Erdal Akin  
Computer Science  
University of Texas at San Antonio  
San Antonio, Texas, USA  
Email: erdal.akin@utsa.edu

Turgay Korkmaz  
Computer Science  
University of Texas at San Antonio  
San Antonio, Texas, USA  
Email: korkmaz@cs.utsa.com

Abstract—Routing a given set of flows with bandwidth requirements is a fundamental problem, which has been formulated as Unsplittable Flow Problem (UFP). One of the key issues in this formulation is that the bandwidth requirement is fixed for each flow. In practice, however, many applications (e.g., video-on-demand, backup and replication services) would greatly benefit from getting more bandwidths beyond the fixed minimum bandwidth requirement. To achieve this, we extended UFP such that bandwidth requirements will be given in a range. The challenge is now how to find paths and dynamically determine the allocated bandwidths such that we can provide more bandwidth to each flow while making sure that each flow gets at least the requested minimum amount. Like UFP, the extended UFP problem is also NP-Hard. Therefore, we propose two heuristics and demonstrate their efficiency using simulation.

Index Terms—Bandwidth-constraint path selection; Multicommodity Flow; QoS; Cloud Computing; SDN

I. INTRODUCTION

One of the fundamental problems in computer networking is how to route a given set of flows through a network while satisfying the bandwidth requirements of these flows. This fundamental problem known as bandwidth-constrained path selection appears in many contexts such as IP traffic engineering and MPLS routing [1], [2], and more recently inter-cloud communications [3]. As a new paradigm, cloud computing allows users to access a shared pool of resources such as servers, storage, and applications from anywhere [4], [5]. However, to support large scale and more complicated services involving multiple resources from different cloud sites (e.g., content distribution and replication, fault tolerance), we need to figure out how to simultaneously connect multiple resources in different cloud sites through an underlying cloud network that can provide bandwidth guarantee as the main Quality-of-Service (QoS) parameter [3].

While being easy to be solved in the case of a single flow [6]–[8], the bandwidth-constrained path selection problem is known to be NP-hard in the case of multiple flows as it is related to the integer multi-commodity flow problem [9]–[12]. More specifically, it is known as the Unsplittable Flow Problem (UFP), which is formally defined as follows.

Definition 1: Unsplittable Flow Problem (UFP): Consider a network that is represented by a directed simple graph $G = (N, A)$, where $N$ is the set of nodes, $N = \{v_1, v_2, v_3, \ldots, v_n\}$ and $A$ is the set of links, $A = \{e_1, e_2, e_3, \ldots, e_m\}$. Each link $(i, j) \in A$ is associated with an available bandwidth parameter $bw(i, j) \geq 0$. A flow $f_k$ is a 3-tuple $(s_k, d_k, r_k)$ where $s_k$ is source node, $d_k$ is destination node and $r_k$ is the requested bandwidth demand of flow $f_k$. Each flow $f_k$ needs to be routed through a single path $p_k$ for which $bw(p_k) \geq \min\{bw(i, j) | (i, j) \in p_k\} \geq r_k$. The problem is to find paths that can maximize the total routed demand from given $K$ flows $F = \{f_1, f_2, f_3, \ldots, f_K\}$:

$$\max_{k=1}^{K} \sum r_k y_k$$

subject to

$$\sum_{k=1}^{K} r_k x_{ij}^k \leq bw(i, j) \text{ for each } (i, j) \in A. \quad (1)$$

$$\sum_{j: (i, j) \in A} r_k x_{ij}^k - \sum_{j: (j, i) \in A} r_k x_{ji}^k = \begin{cases} +r_k & \text{if } i = s_k \\ -r_k & \text{if } i = d_k \\ 0 & \text{otherwise} \end{cases} \quad \forall i, k \quad (2)$$

$$x_{ij}^k = \begin{cases} 0 & : x_{ij}^k \notin p_k \\ 1 & : x_{ij}^k \in p_k \end{cases} \quad (3)$$

$$y_k = \begin{cases} 1 & : \exists p_k \text{ s.t. } r_k \leq bw(p_k) \\ 0 & \text{Otherwise} \end{cases} \quad (4)$$

Since UFP is NP-hard [12], there is no polynomial algorithm to solve it unless $P=NP$. Accordingly, researchers have proposed various heuristics and approximation algorithms. We will review them in the next section and use one of them as part of our solutions when addressing an extended version of UFP.

One of the key issues in the above formulation is that the bandwidth demand $r_k$ is fixed for each flow. In practice, however, many applications (e.g., video-on-demand, backup and replication services) would greatly benefit from getting more bandwidths beyond the guaranteed minimum amounts [3], [13]. Finding such paths, albeit challenging, will improve user experiences and increase revenues for the network operators. Accordingly, we extend the above formulation such that the bandwidth requirement of each flow $f_k$ can be given in a range.
maximize the total routed demand for the given \( K \) allocated bandwidth \( UFP \). flow \( f \) that all flows can be routed when \( r = \frac{F}{bw} \) which \( u \) is maximum demand of flow \( f \). Again each flow \( f_k \) needs to be routed through a single path \( p_k \) for which \( bw(p_k) \geq r_k \). Assuming that all flows can be routed when \( r_k = l_k \) in the above UFP\(^1\), the problem here is to find a path \( p_k \) and and its allocated bandwidth \( r_k \) for each flow \( f_k \) so that we can maximize the total routed demand for the given \( K \) flows \( F = \{ f_1, f_2, f_3, ..., f_K \} \):

\[
\max \sum_{k=1}^{K} r_k
\]

subject to the same constraints (1), (2), and (3) in the above UFP formulation with the key difference that now \( r_k \)’s are decision variables that need to be dynamically determined under the following additional constraint:

\[
l_k \leq r_k \leq u_k \quad (5)
\]

Since \( r_k \)’s are now decision variables, our E-UFP problem turns out to be a quadratically constrained integer programming problem, which is known to be NP-hard \([15]\). Thus, no efficient algorithms exist to solve this problem, calling for efficient heuristics. To the best of our knowledge, we are the first to address the Extended Unsplittable Flow Problem (E-UFP).\(^2\)

In essence, we plan to use an existing efficient heuristic solution to UFP, namely Greedy Algorithm with Preemption (GAP) \([14]\), as the key building block in our solutions. So the key issue is how to set bandwidth requirements before calling the existing solution for UFP. For example, we can simply solve E-UFP in a brute force manner by first recursively generating all permutations of bandwidth requests ranging from \( l_k \) to \( u_k \) with the increments of 1. Then we can call GAP by setting \( r_k \)’s to the bandwidth values in each permutation and return the permutation that maximizes the objective function while finding a path for each flow. Clearly, this brute force approach can only work with a very small number of flows and small ranges as the number of permutations increases exponentially. Accordingly, we need new heuristics that can accomplish the similar performance of brute force in maximizing the objective function while significantly reducing the computation time.

With this in mind, we propose two heuristics, namely randomized and binary-search based algorithms that iteratively calls GAP while adjusting the bandwidth ranges depending on the paths returned in previous iterations. Using simulations we show that our binary-search based algorithm provides the best performance.

The rest of this paper is organized as follows. In Section II we present the related work. We describe our proposed heuristic algorithms in Section III. In Section IV we present our simulation setup and the results. Finally, we conclude this paper and discuss some possible future work in Section V.

II. RELATED WORKS

The main problem that we study is related to bandwidth-constrained path problems \([6], [7]\) and maximum multi-commodity flow problems \([9]\). Finding a path for a single flow is a simple problem which can be solved by existing bandwidth-constrained path selection algorithms such as Widest-Shortest Path algorithm and Shortest-Widest Path algorithm \([8], [20]\). Maximum multi-commodity flow problems \([9]\) are solved by dividing flows and sending over different paths. However, in our case, in order to avoid the problems related to re-ordering of the packets, each flow should be routed through a single path. This is known as Unsplittable Flow Problem (UFP), which has been initially represented in \([21]\).

To solve the UFP, researchers have offered various approximation algorithms, which often suffer from computational complexity while being hard to implement. In contrast, effective heuristics are also developed by the researchers. In this direction, Greedy Algorithm (GA) is the first heuristic introduced in \([21]\). GA is an iterative algorithm that relies on finding a shortest path for each request. Researchers tried to improve the performance of GA. Accordingly, Bounded Greedy Algorithm (BGA) is proposed in \([22]\). In essence, BGA tries to find a path for each flow as in GA by limiting hop counts with a bound \( L \). In addition to the hop count bound \( L \), they offered a new algorithm called \( cBGA \) which takes into account another bound to limit the overlapping link capacities of a path while searching a feasible path for each request. By generalizing the key ideas from these heuristics, the author in \([14]\) proposed a new heuristic, called Greedy Algorithm with Preemption (GAP). It basically tries to find better solutions by preempting an existing path and re-trying
the flows failed previously. Overall, this new approach gives the best performance in terms of maximizing the amount of sent data. Therefore, we will use it in our solutions. While we focus on the bandwidth-constrained path selection in this paper, there are several other related works considering how to provide effective communication between geographically distributed cloud sites by controlling traffic at application and network layers [23]–[25]. In [3], authors propose a mechanism to manage switches and network bandwidth by using Software Defined Network (SDN) principles. In [25], authors provide an integer linear programming formulations to reserve variable bandwidth capacity at different times and solve it using shortest path-based computation. In [26], authors consider multiple-layer network architecture and utilize different layers for transporting the bandwidth requests depending on their rate which can be classified as low, high, and highest.

III. PROPOSED HEURISTIC ALGORITHMS

Our general approach to solve E-UFP is to iteratively use an existing heuristic solution of UFP with a given set of bandwidth requirements. So the key issue here is how to select the bandwidth requirements from the given ranges and in which order to pass them to the existing heuristic solution of UFP. One simple approach will be to consider all possibilities in a brute force manner. Accordingly, we will first develop a brute force (BF) algorithm that can simply generate all permutations of bandwidth requirements from the given ranges and then call an existing solution of UFP with each permutation of bandwidth requirements. Unfortunately, BF algorithm can not be used in practice as the number of permutations increases exponentially. Therefore, there is a need for developing new efficient heuristics.

In response to that, we propose two heuristics. Our first heuristic (called Rand) is a randomization algorithm, where we randomly generate bandwidth requirements from the given ranges and call an existing solution of UFP. Depending on the returned paths, we adjust the ranges based on some heuristics that we will discuss. Then we randomly generate bandwidth requirements from the new ranges with the objective of obtaining better solutions in the next iteration. As the number of iterations increases, this algorithm is expected to return better solutions at the expense of increased computation time.

Our second heuristic (called BS) is a binary-search based algorithm, where we generate the bandwidth requirements by taking the mean of the corresponding ranges. Again depending on the returned paths, we adjust the ranges based on some heuristics that we will discuss. This search is repeated until the lower and upper bounds of all ranges become equal. As we show later, this algorithm provides the best performance.

As we discussed in the related work section, there are several existing heuristics and approximation solutions for UFP. In our heuristics, we will use the GAP algorithm as the existing solution of UFP because of its simplicity and better performance when compared to others [14].

We now present details of our solutions and their computational time analysis.

A. Brute Force (BF) Algorithm

The pseudo-code of brute force (BF) algorithm is given in Algorithm 1. It first calls Algorithm 2 which recursively generates all permutations of bandwidth requirements and stores them in Demands list. BF algorithm then sorts Demands list in decreasing order with respect to (w.r.t.) the sum of each demand. BF algorithm then calls GAP algorithm for each demand. Since the demands are sorted from maximum to minimum, BF algorithm stops when the GAP algorithm finds a solution that satisfies the bandwidth requirements in the given demand permutation for all flows. In the worse case, the running time of the BF algorithm would be $O((\max(u_k - l_k))^{|F|}GAP)$ where $|F|$ is number of flow, as it runs for all permutations whose number grows exponentially. Clearly, this algorithm can not be used in practice. Nevertheless, we run the BF algorithm for small number of flows and ranges so that we can show the effectiveness of our other heuristic solutions.

**Algorithm 1 Brute Force Algorithm**

| Input: | Graph $G(N, A), F = \{(s_k, d_k, l_k, u_k) : k = 1, 2, \ldots, K\}$ |
| Output: | $S_{success}$, flow set maximizes routed demand |
| 1: $Demands = \emptyset$ |
| 2: All Demand Permutations($F$, 1) |
| 3: Sort($Demands$) in descending order w.r.t. sum of each demand |
| 4: for $\forall D \in Demands$ do |
| 5: $S_{success} = GAP(G, F, D)$ |
| 6: if $|S_{success}| == |F|$ then |
| 7: return $S_{success}$ |
| 8: end if |
| 9: end for |

**Algorithm 2 All Demand Permutations**

| Input: | $F = \{(s_k, d_k, l_k, u_k) : k = 1, 2, \ldots, K\}, size$ |
| Output: | $Demands$, set of all possible demand requirements |
| 1: if $size == K$ then |
| 2: $Demands = Demands \cup [r_1, r_2, \ldots, r_K]$ |
| 3: else |
| 4: for $r_{size} = l_{size}$ to $u_{size}$ do |
| 5: All Demand Permutations($F$, $size + 1$) |
| 6: $r_{size} = r_{size} + 1$ |
| 7: end for |
| 8: end if |

B. Randomization (Rand) Algorithm

The pseudo-code of our randomized (Rand) algorithm is given in Algorithm 3. It is an iterative algorithm that randomly chooses current bandwidth requirement $r_k$ between lower bound ($l_k$) and upper bound ($u_k$) and then calls GAP algorithm. It repeats this process $\alpha$ times to increase the chance of finding a solution that maximize routed bandwidth
requirements. After each iteration, we actually adjust the bandwidth ranges to avoid the blind search. Specifically, if all flows are sent, then we set the lower bound \( l_k \) of all ranges to \( r_k \) (\( l_k = r_k \)). Otherwise, we keep the ranges the same. We will test the Rand algorithm with different values of \( \alpha \). The running time of the Rand algorithm is \( \mathcal{O}(\alpha \text{GAP}) \).

**Algorithm 3 Randomized (Rand) Algorithm**

**Input:** Graph \( G(N,A) \), Repeat value \( \alpha \), \( F = \{ (s_k,d_k,l_k,u_k) : k = 1, 2, \ldots, K \} \)

**Output:** \( S_{\text{success}} \), flow set maximizes routed demand

1: for \( \forall f_k \in F \) do \( r_k = l_k \) end for
2: \( S_{\text{success}} = \text{GAP}(G,F) \)
3: repeat = 0
4: while repeat \( \leq \alpha \) do
5: for \( \forall f_k \in F \) do \( r_k = \text{uniform}(l_k,u_k) \) end for
6: \( S_{\text{current}} = \text{GAP}(G,F) \)
7: if \( |S_{\text{current}}| = |F| \) then
8: \( \sum_{k=1}^{K} r_k \geq \sum_{k=1}^{K} |S_{\text{success}}| r_k \) then
9: \( S_{\text{success}} = S_{\text{current}} \)
10: end if
11: for \( \forall f_k \in F \) do \( l_k = r_k \) end for
12: repeat + +
13: end if
14: return \( S_{\text{success}} \)

**C. Binary-Search Based (BS) Algorithm**

The pseudo-code of our binary-search based (BS) algorithm is given in Algorithm 4. As in the binary search, it always takes the means of the ranges as the current bandwidth requests for all flows. It then calls \( \text{GAP} \) algorithm to find paths to send these flows. If it succeeds, then new lower bounds \( l_k \) of all flows are updated to \( r_k \). Otherwise, it changes the upper bound \( u_k \) of not-sent flows to \( r_k \) while keeping lower ranges the same. It then continues to search paths for flows with new bandwidth requirements \( r_k \). It terminates when all \( l_k \) and \( u_k \) become equal. The running time of the algorithm is \( \mathcal{O}(\log(\max(u_k - l_k))\text{GAP}) \).

**Algorithm 4 Binary-Search Based (BS) Algorithm**

**Input:** Graph \( G(N,A) \), \( F = \{ (s_k,d_k,l_k,u_k) : k = 1, 2, \ldots, K \} \)

**Output:** \( S_{\text{success}} \), flow set maximizes routed demand

1: for \( \forall f_k \in F \) do \( r_k = l_k \) end for
2: \( S_{\text{success}} = \text{GAP}(G,F) \)
3: for \( \forall f_k \in F \) do \( r_k = (l_k + u_k)/2 \) end for
4: LOOP:
5: \( S_{\text{current}} = \text{GAP}(G,F) \)
6: if \( |S_{\text{current}}| = |F| \) then
7: \( S_{\text{success}} = S_{\text{current}} \)
8: for \( \forall f_k \in F \) do \( l_k = r_k \) end for
9: for \( \forall f_k \in F \) do \( r_k = (l_k + u_k)/2 \) end for
10: else
11: for \( \forall f_k \in F \) do
12: if \( f_k \notin S_{\text{current}} \) then
13: \( u_k = r_k \)
14: \( r_k = (l_k + u_k)/2 \)
15: end if
16: end if
17: for \( \forall f_k \in F \) do
18: if \( l_k 
eq u_k \) then
19: Goto LOOP
20: end if
21: end if
22: end for
23: return \( S_{\text{success}} \)

**IV. EXPERIMENTAL RESULTS**

To evaluate our algorithm, we implemented a simulator in Java. We run our simulator on a computer which has Intel Core(TM) i7 – 4710HQ CPU @2.50 GHz with 12GB RAM. We compare our heuristics against brute force algorithm, which always performs he best in maximizing routed bandwidth requirements. However, since it suffers from its exponentially growing execution time, we only execute it with small number of flows. As the performance measure, we consider the total amount of sent demand and the execution time of each algorithm.

For our tests, we first use a realistic topology by adding new links to ANSNET in [27], as seen Figure 1. We also consider random graphs generated by BRITE [28] with different densities (\( M \)) and randomly assigned bandwidths. When generating graphs, link weights, and requested demands, we used common random numbers to minimize the variance and thus obtain a better confidence level on our comparison results. We run each algorithm 100 times and report their averages.

**Test 1**

Our first test is based on ANSNET topology [27]. Bandwidth capacity of each link is randomly generated from \( \text{uniform}(10,100) \). We consider \( K = 3 - 15 \) flows. For each flow set, we randomly generate source \( s_k \) and destination \( d_k \) from \( \text{uniform}(1,N) \) where \( N = 32 \) for this topology, and bandwidth requirement range \( (l_k,u_k) \) from \( \text{uniform}(1,5) \) and \( \text{uniform}(5,10) \), respectively. Algorithms try to maximize total routed \( r_k \) which can be any variable between \( l_k \) and \( u_k \).
large number of flows (optimum solution with better execution time especially for algorithm. In contrast to Rand algorithm, BS Algorithm approach es
for each link is randomly generated from uniform found in our technical paper [29]. We generated 100 inputs for Rand Algorithm, the execution time dramatically increases. We ran the same inputs for Rand Algorithm, the execution time grows linearly as α increases. Nevertheless, routed bandwidth demand is not increasing in parallel with execution time. So, we will run next experiments with α = 3 and 6 for Rand Algorithm. In contrast to Rand algorithm, BS Algorithm approaches optimum solution with better execution time especially for large number of flows (K ≥ 6).

Test 2

Our second test is based on random graphs generated by BRITE [28]. We have tried different densities (M) and observed the similar trends. Due to page limitations, we report the results with M = 3 and M = 5. But more results can be found in our technical paper [29]. We generated 100 random graphs with density M = 3 and M = 5. Bandwidth capacity for each link is randomly generated from uniform(10 – 100).

Test 3

We again consider K = 3 – 15 flows. For each flow set, we randomly generate source s_k and destination d_k from uniform(1, N), where N = 100, and bandwidth requirement range (l_k, u_k) from uniform(1, 10) and uniform(10, 20), respectively.

Figure 3a and Figure 3b show the performance results when M = 3. Again BS Algorithm runs effectively as much as BF Algorithm in maximizing total routed demand while significantly reducing execution time. Although Rand Algorithm with α = 3 is slightly better than BS Algorithm in terms of execution time, it underperforms in maximizing total routed demand. Assigning α to 6 does not increase performance of routing demand as much as it affects execution time negatively.

Figure 4a and Figure 4b show the performance results when M = 5. Clearly we observe the same trends as in previous tests. Again BS Algorithm has better performance than the others in maximizing total routed demand with reasonable execution time.

Test 3

Our third test is again based on ANSNET topology [27]. Same as first test, bandwidth capacity of each link is randomly generated from uniform(10, 100). We consider K = 3 – 15
flows. For each flow set, we randomly generate source $s_k$ and destination $d_k$ from $\text{uniform}(1, N)$ where $N = 32$ for this topology. In this test, different than first test, we determined bandwidth requirement range $(l_k, u_k)$ from $\text{uniform}(1, 10)$ and $\text{uniform}(10, 20)$, respectively.

As seen in the Figure 5a and Figure 5b, BF Algorithm gives optimum solution for maximizing bandwidth requirement. However, execution time of the algorithm is high even for $K = 3$ and it goes up exponentially when flow size and/or bandwidth demand range increases. Because of that we terminate the algorithm for next flows when execution time dramatically increases. We run the same inputs for Rand Algorithm with $\alpha = 3, 6$. It gives very close results to BF Algorithm in maximizing total routed demand with better execution time. BSA algorithm gives very close results with best execution time.

**Test 4**

Our fourth test is again based on random graphs generated by BRITE [28]. All variables are set with same values except bandwidth requirement range $(l_k, u_k)$ which is chosen from $\text{uniform}(1, 10)$ and $\text{uniform}(10, 30)$, respectively.

Figure 6a and Figure 6b show the performance results when $M = 3$. Because of high range of bandwidth requirements, execution time of BF algorithm is very high such that it did not fit the graph. Again BS Algorithm runs effectively as much as BF Algorithm in maximizing total routed demand while significantly reducing execution time. Although Rand Algorithm with $\alpha = 3, 6$ is slightly better than BS Algorithm in maximizing total routed demand, it suffers in terms of execution time.

Figure 7a and Figure 7b show the performance results when $M = 5$ and bandwidth requirement range $(l_k, u_k)$ which is chosen from $\text{uniform}(1, 30)$ and $\text{uniform}(30, 60)$, respectively. Clearly we observe the same trends as in previous tests. Again BS Algorithm has better performance than the others in maximizing total routed demand with reasonable execution time.

**V. CONCLUSIONS AND FUTURE WORKS**

We have extended the Unsplittable Flow Problem (UFP) such that bandwidth requirements will be given in a range. Since the extended UFP is still NP-hard, we have proposed two efficient heuristics using randomization and a binary-search based idea. We compared our heuristics against a brute-force algorithm. We observed that the binary-search based (BS) algorithm provides the best performance. However, there is still a room for improvement. So, in the future, we plan to integrate some of the ideas from Random algorithm into
BS algorithm. In addition to maximizing the routed demand, we will also consider priority and fairness when allocating bandwidth to each flow.

REFERENCES


Fig. 6: Performance results under topologies generated by [28] with density $M = 3$.

Fig. 7: Performance results under topologies generated by [28] with density $M = 5$.


