Poster Abstract: Always Heading for the Peak: Learning to Route with Domain Knowledge

Jing Chen, Zili Meng, Mingwei Xu*
Institute for Network Science and Cyberspace, Tsinghua University
* Department of Computer Science and Technology, Tsinghua University
Beijing National Research Center for Information Science and Technology (BNRist)
j-chen16@tsinghua.org.cn, zilim@ieee.org, xumw@tsinghua.edu.cn

I. Introduction

Routing optimization is a well-known yet complicated research topic for decades. On one hand, routing algorithms usually have complex inputs (e.g., structural topology), outputs (e.g., sequential paths for different demands), and optimization goals (e.g., link utilization). On the other hand, due to the rapid change of demands, routing algorithms must be fast enough to make online decisions. Existing methods usually make a trade-off between performance and efficiency. Offline algorithms [1] tend to cast the problem into a Linear Program (LP), offering near-optimal solutions yet taking up to tens of seconds, which are too heavyweight for rapidly changing networked systems. Online algorithms, such as shortest path (*SP*) and weighted shortest path (*WSP*), are broadly used, but their performance is far from optimal due to the ignorance of network status. Conventional heuristic algorithms fail to efficiently capture the dynamics inside varying network demands, which compromises the performance, as shown in Table I.

Recent advances in the machine learning community enable us to achieve near-optimal performance and millisecond-level decision latency at the same time, addressing the trade-off above [2, 3]. There are also preliminary research efforts trying to apply machine learning techniques into routing optimizations [4, 5]. However, existing approaches rarely address the challenge of how to make routing decisions *under constraints*. For example, one common constraint in the routing domain is avoiding loops. We take a learning-based routing algorithm [4], which optimizes the flow split ratios, as an example. As shown in Figure 1(a), since the optimizations at different nodes are independent of each other, there may exist inconsistency among their results and therefore form routing loops. For example, when forwarding towards `dst`, assume the flow split ratio at `s1` to `s2` is 0.8 and at `s2` to `s1` is 0.1. In this case, 8% of the flows originated from `s1` will be sent back to `s1`. There may be more routing loops involving multiple links when the topology goes more complex.

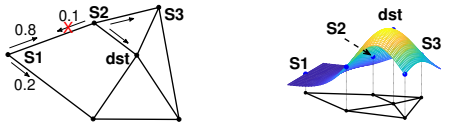
The ignorance of constraints attributes to the insufficiency of the design of existing learning-based routing algorithms. To fully utilize the ability of DNNs, network operators tend to let the DNNs learn everything themselves. This may work when the topology is small enough (several nodes) and the optimization goals are indirect and differentiable (e.g., predicting traffic demands) [4, 5]. However, when scaled to larger topologies, existing learning-based routing algorithms could not provide satisfying performance. For example, using the learning-based routing algorithm in [4] (denoted as *LR*), in a 24-node topology, 53% flows cannot arrive at the destination within 64 hops due to routing loops (Section III). Moreover, manually detecting such routing loops requires considerable human efforts. Therefore, existing algorithms either compromise by predicting intermediate variables (e.g., traffic demands [4], link latency [5]) and let network operators make decisions subsequently, or do not scale well to real-world topologies [4].

This poster is supported by the National Key R&D Program of China (2017YFB0801701) and the National Science Foundation of China (61625203, 61832013, 61872426). Mingwei Xu is the corresponding author.

TABLE I: Comparison of different routing methods

Methods	*Diverse*	*Loop-free*	*Demand-aware*	*Capacity-aware*
SP		✓		
WSP		✓		✓
LR	✓		✓	✓
ALT	✓	✓	✓	✓

*SP: shortest path; WSP: shortest path weighted by capacity; LR: learning-based routing in [4]; ALT: our approach.



(a) Topology. (b) Altitude illustration.

Fig. 1: According to the altitudes decided by the neural network, s2 will not forward any packets to s1 whose altitude is lower.

To optimize the network performance directly in real-world scenarios, our solution integrates domain knowledge into the design of learning methods. Our key observation is that routing is similar to climbing hills. Climbers may either directly climb up along the gradient of the hill or going along the winding hill road in a zig-zag way, but rarely go in the opposite direction. Similarly, *good routing policies will roughly route flows towards the destination*. As shown in Fig. 1(a), flows at `s2` may go by the direct link or by `s3` to reach `dst`, but may not go by `s1` and make a long detour in the network.

Based on this observation, we introduce a new decision variable for each node and each destination, denoted as *altitude*, inspired by [6] to mimic the hill-climbing process. In this case, different destinations construct their respective hills for each node. As shown in Fig. 1(b), flows can only be routed from a lower-altitude node to a higher-altitude node, therefore loops are strictly avoided. In this formulation, routing decisions can be efficiently expressed into fixed-length vectors and easily trained to satisfy routing constraints.

We evaluate our altitude-based algorithm with simulations

on two real-world topologies. Results show that compared with heuristics, our algorithm reduces maximum-link-utilization by up to 29%. Compared with *LR*, we avoid routing loops and at least 24% of more flows arrive at destinations within 64 hops.

II. Design

As mentioned above, the key insight behind our algorithm is that routing paths should not go back and forth or circle in the network. To avoid loops, shortest path algorithm simply restricts the flows on the single paths calculated by distances. But in this way, critical links are likely to be over-loaded. As an alternative, we offer altitudes to prevent flows from going towards opposite directions. They help avoid routing loops as well as be supportive of traffic diversion.

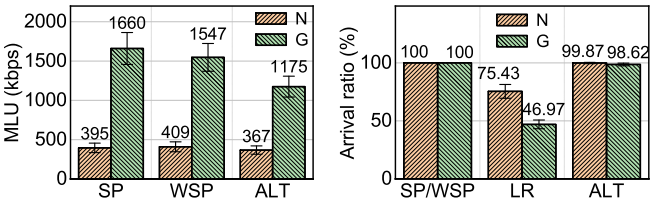
In another word, We expect intelligent loop-free decisions. For example, in Figure 1(a), assume that `s2` and `s3` have equal distances to `dst`. However, if the link between `s2` and `dst` is congested, we want clever decisions that lower the altitude of `s2` to make less flows go by `s2` (e.g. flows at `s3` could go by the direct link to `dst`) in this case. Therefore we leave altitude decisions up to the neural network, hoping that the network topology, traffic demands and link capacities can be considered together. However, we encounter the following challenges during design:

Challenge #1: Nondifferentiable objective. A straightforward challenge is that the optimization objective in the routing problem is *nondifferentiable*. For example, the calculation of link utilization or link hop of all flows needs to query the *index* of the output of the model (e.g., flow split ratio). Without formulating the optimization objective as an explicit differentiable expression, optimization methods with direct backpropagation are not applicable.

Inspired by recent advances in reinforcement learning (RL), we could train an agent to make routing decisions by interacting with the environment. The agent learns a policy for *action*-making in current *state* under the guidance of the *reward* from the environment. In the routing case, states are current traffic demand matrices and link capacities. Actions are *altitudes* (introduced above) and the flow split ratios. Rewards could be defined based on the requirements of network operators (e.g., path length, link utilization, etc.).

Challenge #2: Slow convergence. Another challenge is how to make the algorithm converge efficiently. Existing initialization methods (e.g., Gaussian initialization) confuse the agent in the first steps. Since flows are only allowed to route to a node whose altitude is higher than the current node, building a path from the source to the destination is hardly possible when multiple links are involved and the altitudes are randomly selected. Therefore, the agent cannot receive rewards for a long time and has a great chance to diverge.

Our solution is to initialize the agent with a *baseline*. The algorithm only needs to optimize the difference from the baseline. Specifically, we introduce a naive altitude based on the shortest path algorithm. The agent will only optimize the difference between its altitudes and the altitudes from the *SP*. In this way, training an agent on topologies with tens of nodes can converge within a few hours in our experiments.



(a) MLU (b) Flow arrival ratio

Fig. 2: Simulation results on the NSF and GEANT2 topology (denoted as N and G). Error bars span $\pm$ std.

III. Preliminary Evaluation

We evaluate our algorithm with two real-world topologies [5] : the 14-node NSF topology and the 24-node GEANT2 topology. Flow demands are uniformly sampled between $0.1C_{\text{avg}}$ and $0.5C_{\text{avg}}$, where C_{avg} is the average of network link capacities. We use a 3-layer perceptron to represent the policy network. We set up our optimization goal as minimizing the maximum-link-utilization (MLU) in the network.

We measure the MLU in different algorithms and present the results in Figure 2(a). Our approach (denoted as *ALT*) decreases MLU by 6.9% in NSF and 24.0% in GEANT2 comparing with *SP* and *WSP*. The performance improvement in GEANT2 topology is more significant since our approach scales well to larger topologies while heuristics do not.

We further measure the flow arrival ratios of different algorithms and plot the results in Figure 2(b). We set Time-to-Live (TTL) as 64 so that more flows (even with routing loops) can reach the destination. Flow arrival ratios of *ALT* are not 100% because an intermediate node may have a higher altitude than all of its neighbors. Results demonstrate that invalid altitude decisions are rarely made by *ALT* after training whereas *LR* suffers from severe problems of routing loops.

Finally, we measure the computation time of our approach. The offline-trained RL agent can make a routing decision within 0.6 ms for both topologies, which is acceptable since the period of route changes is much longer than 1 ms.

IV. Conclusions and Future Work

We propose an improved RL-based routing algorithm by making altitude decisions to follow the constraints in routing. Our approach shows significant improvements against existing ones. Real-world implementation of learning-based routing algorithms may require further considerations, e.g. the size of routing tables, which are left as future work.

References

[1] C. Zhang, Y. Liu, W. Gong, J. Kurose *et al.*, "On optimal routing with multiple traffic matrices," in *Proc. IEEE INFOCOM*, 2005.

[2] Z. Meng, J. Chen, Y. Guo, C. Sun *et al.*, "Pitree: Practical implementation of abr algorithms using decision trees," in *Proc. ACM Multimedia*, 2019.

[3] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM SIGCOMM*, 2019.

[4] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proc. ACM HotNets*, 2017.

[5] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in sdn," in *Proc. ACM SOSR*, 2019.

[6] N. Oba, H. Kobayashi, and T. Nakamura, "An adaptive network routing method by electrical-circuit modeling," in *Proc. IEEE INFOCOM*, 1993.