EXPLOITING FAST ALGORITHMS FOR SOLVING NETWORKING RESOURCE ALLOCATION PROBLEMS

I. Scenario Description

We start to briefly introduce a network resource allocation problem by adopting some notations from [1]. We consider a general network model with some interconnected nodes. There are L uni-directional links indexed by l ($l = 1, 2, \ldots, L$) connecting the network nodes with link capacity c_l ($c_l \in \mathbb{R}^+, \forall l$). Define $\mathbf{c} = (c_1, c_2, \ldots, c_L)$ to be the link capacity vector of the overall network. The communication network is used to deliver data flows from source nodes to destination nodes to facilitate end-to-end data services. Specifically, we consider K flows in the network indexed by k ($k = 1, 2, \ldots, K$). For each flow k, there is a source-destination pair associated with it and we assume that there are P_k ($P_k \in \mathbb{Z}^+, \forall k$) available paths for this source-destination pair indexed by p_k ($p_k = 1, 2, \ldots, P_k, \forall k$). We use the following $L \times P_k$ routing matrix to represent the relationship between the links and the available paths of flow k:

$$\boldsymbol{R}_k = \begin{pmatrix} R^k_{1,1} & R^k_{1,2} & \cdots & R^k_{1,P_k} \\ R^k_{2,1} & R^k_{2,2} & \cdots & R^k_{2,P_k} \\ \vdots & \vdots & \ddots & \vdots \\ R^k_{L,1} & R^k_{L,2} & \cdots & R^k_{L,P_k} \end{pmatrix} \tag{1}$$

where $R^k_{l,p_k} \in \{0, 1\}$. When $R^k_{l,p_k} = 1$, it means that path p_k transverses link l for flow k and vice versa. We define the following $L \times P$ matrix (where we denote $P \triangleq \sum_{k=1}^{K} P_k$) to be the routing matrix of the overall network:

$$\mathbf{R} = (\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_K) \tag{2}$$

Let $\mathbf{x}_k = (x_{k,1}, x_{k,2}, \ldots, x_{k,P_k})$ be the rate allocation and path selection vector of flow k, where each element $x_{k,p}$ ($x_{k,p} \geq 0, \forall k, p$) measures the rate allocation at path p for flow k. $x_{k,p}$ also gives information about what paths to select for flow k. Specifically, the path selected by flow k are those paths with $x_{k,p_k} > 0$ for any p_k. Define $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K)$ to be the rate allocation and path selection vector of all flows.

Furthermore, we want the network load to be balanced across links so as to avoid network congestions and we consider to minimize the maximum (or worst-case) link utilization ratio to achieve load balancing. We formally formulate the network resource problem as follows

Problem 1.

$$\min_{\mathbf{x}} \quad \max_{l} \frac{\mathbf{R}[l]\mathbf{x}}{c_l} \tag{3a}$$

$$\textit{s.t.} \quad \mathbf{R}\mathbf{x} \leq \mathbf{c} \tag{3b}$$

$$\|\mathbf{x}_k\|_1 = d_k \tag{3c}$$

$$\mathbf{x} \geq \mathbf{0} \tag{3d}$$

where $\mathbf{R}[l]$ *is the* l*-th row of* $\mathbf{R}$*,* $d_k > 0$ *is the bandwidth demand for flow* k*.*

The nonlinear term $\max_l \frac{\mathbf{R}[l]\mathbf{x}}{c_l}$ in the objective can be equivalently transformed to linear objective and some constraints by introducing an auxiliary variable t. Specifically, we can replace $\max_l \frac{\mathbf{R}[l]\mathbf{x}}{c_l}$ with t and add linear constraints $\frac{\mathbf{R}[l]\mathbf{x}}{c_l} \leq t$ for all l and $t \geq 0$. The constraints $\|\mathbf{x}_k\|_1 = d_k$ and $\mathbf{x} \geq \mathbf{0}$ implies that $1^\top \mathbf{x}_k = d_k$ and $\mathbf{x} \geq \mathbf{0}$.

Based on the above formulation, we ask three questions to achieve the goal of efficiently solving it.

II. Q1: Comparison of Various Advanced Approaches

There are several ways to solve the network resource allocation problem. We are interested in developing algorithms using the following two approaches:

- applying the Dantzig-Wolfe decomposition to the above linear programming where the restricted master problem therein can solved using the open-source packages.
- applying the Lagrangian relaxation technique (see also page 12 in the following link) to solve the above linear programming where the primal problem therein can solved using the open-source packages.
  http://bicmr.pku.edu.cn/~wenzw/opt2015/lect-dual.pdf

It is suggested to use *Coin LP* [3] to solve the linear programming problems therein. After obtaining the two algorithms based on the above two approaches, we are interested in the implementation of the two algorithms because the actual performance heavily depends on how to implement the algorithms. Test dataset will be provided and comparisons of the performance and computation time are required. Interested students can refer to [4] for more details on similar approaches to solve multi-commodity flow problems.

III. Q2: Learn to Pivot

It is well-know that the performance of a simplex-based LP implementation depends on a well-designed pivot rule. There are many heuristics proposed in the past 50 years on how to choose the next neighboring vertex at each iteration and these vary in accuracy and computational cost.

In [2], the authors use deep value-based reinforcement learning to learn a pivot strategy that at each iteration chooses between two of the most popular pivot rules – Dantzig and steepest edge. However, the size of the problem in the experiment section is too small and the simulation results fail to convince the readers to adopt such an approach in the advanced LP solvers.

We are interested in the following two tasks in this project:

- implemented the following three pivot rules, the Dantzig rule, the steepest edge rule, and the AI-based pivot rule based on the framework in [2] and compare their performance based on the provided test cases.
- the new pivot rule in [2] is constrained in the action space where either Dantzig or steepest edge selected. The question is whether there is an opportunity to break this rule and develop a new framework to learn a totally new pivot rule.

IV. Q3: Learn to Crash

Study and implement the "Crash" technique in [7], and compare its performance with the default "Crash" in the simplex solver. It is quite interesting to test how well "Crash" in [7] performs on different data sets.

Study the most-obtuse-angle heuristics in [5], [6]. For a particular variable x in the standard LP program, implement the algorithm to compute the associated obtuse angle. Develop an heuristic for "Crash". Here we provide a simple example with a two-pass structure:

- Step 1: Compute the obtuse angle a_k for each variable x_k.
- Step 2: Classify the variables into different categories, and each category is associated with a pre-determined weight. For variable x_k, the weight is denoted by $w_k \in [0, 1]$.
- Step 3: Choose the set of variables with largest $w_k a_k$.

Standard machine learning techniques can be used to train the weights in the above heuristic based on data sets (various network configurations and associated optimal solutions). Compare the performance with that of default "Crash" in [5].

V. Dataset

Interested students can refer to the following website for generating problem instances: http://groups.di.unipi.it/optimize/Data/MMCF.html#LinMMCF

References

[1] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows. 1988.

[2] Anonymous authors. Deepsimplex: Reinforcement learning of pivot rules improves the efficienty of simplex algorithm in solving linear programming problems.

[3] CLP COIN. Coin-or linear programming solver, 2011.

[4] Weibin DAI, Jun ZHANG, and Xiaoqian SUN. On solving multi-commodity flow problems: An experimental evaluation. *Chinese Journal of Aeronautics*, 30(4):1481 – 1492, 2017.

[5] Nicholas IM Gould and John K Reid. New crash procedures for large systems of linear constraints. *Mathematical Programming*, 45(1-3):475–501, 1989.

[6] Wei Li and Haohao Li. On simplex method with most-obtuse-angle rule and cosine rule. *Appl. Math. Comput.*, 217(20):7867–7873, 2011.

[7] István Maros and Gautam Mitra. Strategies for creating advanced bases for large-scale linear programming problems. *INFORMS Journal on Computing*, 10(2):248–260, 1998.