Minimum-Cost Network Flow with Dual Predictions

Zhiyang Chen[1], **Hailong Yao**[2,3*], **Xia Yin**[1]

[1]Tsinghua University
[2]University of Science and Technology Beijing
[3]Key Laboratory of Advanced Materials and Devices for Post-Moore Chips, Ministry of Education of China

Abstract

Recent work has shown that machine-learned predictions can provably improve the performance of classic algorithms. In this work, we propose the first minimum-cost network flow algorithm augmented with a dual prediction. Our method is based on a classic minimum-cost flow algorithm, namely ε-relaxation. We provide time complexity bounds in terms of the infinity norm prediction error, which is both consistent and robust. We also prove sample complexity bounds for PAC-learning the prediction. We empirically validate our theoretical results on two applications of minimum-cost flow, i.e., traffic networks and chip escape routing, in which we learn a fixed prediction, and a feature-based neural network model to infer the prediction, respectively. Experimental results illustrate $12.74\times$ and $1.64\times$ average speedup on two applications.

Introduction

Minimum-cost network flow is an important computational problem in combinatorial optimization, with various applications in computer science and operations research, including design automation (Yan and Wong 2010), computer vision (Wang et al. 2019), and scheduling (Ahuja et al. 1995). Moreover, many network optimization problems, e.g., disjoint paths, maximum flow, minimum cut, matching, and assignment, are special cases of minimum-cost flow. Therefore, fast minimum-cost flow algorithms are essential for efficiently solving many practical optimization problems.

A bunch of minimum-cost flow algorithms have been developed throughout the last 40 years (Bertsekas 1998; Ahuja, Magnanti, and Orlin 1993), but the minimum-cost flow problem is far from being fully solved. On the one hand, although minimum-cost flow can be solved with polynomial time complexity, it is time-consuming in practice to solve a large-scale minimum-cost flow instance, e.g., a network with millions of nodes and edges. Academics are still actively exploring practically fast network flow solvers (Kara and Özturan 2022). On the other hand, the theoretical analysis of existing minimum-cost flow algorithms is not compatible with the algorithm behaviors in practice. Traditional worst-case time complexity analysis is often overly pessimistic, yielding loose complexity bounds. For instance, although the successive shortest path algorithm may theoretically require an exponential number of iterations to find the optimum (Zadeh 1973), it usually runs relatively faster on minimum-cost flow instances in practice.

Recent work has shown that leveraging machine-learned predictions can improve the worst-case performance of classic algorithms (Lindermayr and Megow 2024), namely *algorithms with predictions* or *learning-augmented algorithms*. It is a beyond-worst-case framework for algorithm design and analysis. In this framework, we assume the algorithm is given a black-box prediction which outputs information about the problem instance. We can leverage this prediction to improve the performance of the algorithm. In practice, such predictions are obtained by machine learning techniques and are hence imperfect. Therefore, we hope the algorithm is both *consistent* and *robust*: It works excellent if the prediction error is small and preserves the classic worst-case performance bound if the prediction is completely wrong.

Previous works study algorithms with predictions for many network optimization problems, including matching (Dinitz et al. 2021), maximum flow (Davies et al. 2023), and global minimum cut (Niaparast, Moseley, and Singh 2025). Therefore, it is natural to ask the question: *Can we use learning-based predictions to accelerate minimum-cost network flow algorithms?*

In this work, we propose the first minimum-cost flow algorithm with dual predictions, based on a classic algorithm, ε-relaxation (Bertsekas 1986). We show that a dual prediction can provably and practically accelerate ε-relaxation. One may wonder why we study ε-relaxation instead of other more popular minimum-cost algorithms, such as network simplex. We will discuss the advantages of ε-relaxation in preliminaries.

Our contributions

Theoretical results. On the theoretical side, we show that the complexity of the ε-relaxation algorithm for minimum-cost flow can be provably improved by a dual prediction.

For a network instance with n nodes and m edges, let p^* be an arbitrary optimal dual solution, and $\hat{p}$ be the predicted solution, we show that warm-starting ε-

*Corresponding author (E-mail: hailongyao@ustb.edu.cn).

relaxation with preprocessing solves minimum-cost flow in $O\left(\min\left\{n^3\|\hat{p}-p^*\|_\infty, n^4C\right\}\right)$ time, where C is the maximum of edge costs (Theorem 1). Combining this algorithm with a cost-scaling trick, we can improve this bound to $O\left(\min\left\{n^3\log\|\hat{p}-p^*\|_\infty, n^3\log(nC)\right\}\right)$ (Theorem 2). This bound reduces to the $O(n^3\log(nC))$ worst-case complexity of ε-relaxation if the prediction is completely erroneous, and efficiently reduces the running time if the prediction error is small. Therefore, our algorithm is both consistent and robust. Specially, for 0/1 flow instances[1], the time bound can be improved to $O\left(\min\left\{mn\log\|\hat{p}-p^*\|_\infty, mn\log(nC)\right\}\right)$.

We also show that we can PAC-learn the prediction from data with a small sample complexity, using the framework of data-driven algorithm design (Theorems 3 and 4).

Empirical results. On the empirical side, we evaluate our algorithm on two applications of minimum-cost flow: traffic networks and escape routing. For traffic networks, we can learn a fixed prediction if the network topology is fixed and the edge costs are random variables. For escape routing, we can learn a feature-based predictor using convolutional neural networks (CNN) based on labeled data of routing instances. Experimental results show that we achieve about $6.2 \sim 21.4\times$ acceleration on real-world traffic networks, and about $1.1 \sim 2.3\times$ acceleration for escape routing.

Related work

Minimum-cost flow algorithms. Network flow has been well studied. We refer readers to, e.g., Bertsekas (1998) and Ahuja, Magnanti, and Orlin (1993) for a comprehensive survey. The simplest minimum-cost flow algorithm is successive shortest path (SSP), which is an extension of Ford-Fulkerson for maximum flow. However, the complexity of SSP grows linearly in terms of the cost range. Subsequent works propose polynomial-time algorithms for minimum-cost flow, including capacity scaling (Edmonds and Karp 1972), cost scaling (Goldberg and Tarjan 1987), and cycle-canceling (Klein 1966). The ε-relaxation (Bertsekas 1986) is a kind of cost scaling algorithm.

The most popular and practically efficient algorithm is network simplex (NS), which is a variant of the simplex method specialized for network flow. Orlin (1997) shows that a special type of NS solves minimum-cost flow in polynomial time. However, the time complexity of the general version of NS implemented in practice remains unclear.

The state-of-the-art complexity is due to Brand et al. (2023), which presents an almost-linear time algorithm. However, this approach is purely theoretical.

Algorithms with predictions. Kraska et al. (2018) first propose incorporating machine-learned predictions into data structures. This area has then become popular in various fields. We refer readers to the website (Lindermayr and Megow 2024) for a full publication list.

For network optimization with predictions, Dinitz et al. (2021) propose a weighted bipartite matching algorithm with dual predictions. Chen et al. (2022) improve this result and extend it to other graph problems. Davies et al. (2023) propose a maximum flow algorithm with primal predictions.

Chen et al. (2022) obtains an $O(m^{3/2}n + (mn + m\log m)\|\hat{p}-p^*\|_0)$ algorithm for the special case, min-cost 0/1 flow. Their approach is based on a reduction from bipartite matching and is hence not practical. Their bound is not better than our $O(mn\log\|\hat{p}-p^*\|_\infty)$ bound for 0/1 flow. Also, their bound is based on 0-norm error, which is the number of indices such that the prediction is different from the optimal solution. In practice, it is hard for a learned model to predict exact values for each component. Therefore, our bound is more practical.

Sakaue and Oki (2022) studies discrete energy minimization for computer vision using tools from discrete convex analysis. The dual of min-cost flow can be seen as a special case of this problem. They provide an $O(mn^2\|\hat{p}-\hat{p}^*\|_\infty)$ algorithm, which is worse than our $O(n^3\log\|\hat{p}-\hat{p}^*\|_\infty)$ bound, both in terms of prediction error and graph size. We also emphasize that our min-cost flow algorithm is practical. Both Chen et al. (2022) and Sakaue and Oki (2022) study purely theoretical algorithms.

Data-driven algorithm design. Another line of research that combines algorithm design and machine learning techniques studies the sample complexity of learning an algorithm configuration from a class of parameterized algorithms. Gupta and Roughgarden (2017) propose the framework of data-driven algorithm design. Subsequent work provides sample complexity guarantees for a variety of parameterized algorithms (Blum, Dan, and Seddighin 2021; Balcan et al. 2021, 2022, 2024; Cheng et al. 2024; Chen, Yao, and Yin 2025). In this work, we will analyze the sample complexity of learning the dual prediction for ε-relaxation, following the philosophy of data-driven algorithm design.

Preliminaries

Linear minimum-cost network flow

Problem formulation. Let $G=(V,E)$ be a directed graph with $|V|=n$ nodes and $|E|=m$ edges. Each edge (v_i,v_j) is associated with a cost a_{ij} and a pair of capacity (b_{ij},c_{ij}) such that $0\le b_{ij}\le c_{ij}$. Each node v_i has a supply $s_i\in\mathbb{R}$. (If $s_i<0$, it represents v_i has a demand $-s_i$.) The *linear minimum-cost network flow problem* (MCF) is formulated as the following:

$$\begin{aligned}&\min_x \sum_{(v_i,v_j)\in E} a_{ij}x_{ij},\\ &\text{subj. to} \sum_{(v_i,v_j)\in E} x_{ij} - \sum_{(v_j,v_i)\in E} x_{ji} = s_i,\ \forall v_i\in V,\\ &\qquad b_{ij}\le x_{ij}\le c_{ij},\ \forall (v_i,v_j)\in E.\end{aligned}$$

We also make the following assumption:

Assumption 1. All capacities and supplies are integers so that the optimal solution is integral. Let $C=\max_{(v_i,v_j)\in E}|a_{ij}|$ be the maximum scale of edge costs.

Duality. The dual problem of MCF is

$$\max_p \sum_{(v_i,v_j)\in E} q_{ij}(p_i-p_j) + \sum_{v_i\in V} s_ip_i,$$

[1] A minimum-cost flow instance is 0/1 flow if the capacities satisfy $b_{ij}=0$ and $c_{ij}=1$ for each $(v_i,v_j)\in E$. A natural example is the minimum-cost bipartite matching.

where

$$q_{ij}(p_i - p_j) = \begin{cases} (a_{ij} + p_j - p_i)b_{ij} & \text{if } a_{ij} + p_j - p_i \geq 0, \\ (a_{ij} + p_j - p_i)c_{ij} & \text{if } a_{ij} + p_j - p_i < 0. \end{cases}$$

Therefore, a pair of primal solution x and dual solution p is optimal if x is feasible and (x, p) satisfies *complementary slackness* (CS), i.e., $\forall (v_i, v_j) \in E$,

$$\begin{aligned} x_{ij} < c_{ij} &\Rightarrow p_i - p_j - a_{ij} \leq 0, \\ x_{ij} > b_{ij} &\Rightarrow p_i - p_j - a_{ij} \geq 0. \end{aligned} \tag{1}$$

The ε-relaxation algorithm

The ε-relaxation (Bertsekas 1986) is a dual algorithm for the linear network flow problem. The algorithm considers a relaxed CS condition (1), namely ε-CS: For each edge (v_i, v_j), we have

$$\begin{aligned} x_{ij} < c_{ij} &\Rightarrow p_i - p_j - a_{ij} \leq \varepsilon, \\ x_{ij} > b_{ij} &\Rightarrow p_i - p_j - a_{ij} \geq -\varepsilon. \end{aligned} \tag{2}$$

We call $p_i - p_j - a_{ij}$ the *reduced cost* of edge (v_i, v_j). The algorithm maintains an ε-CS solution pair (x, p), and iteratively reduce the surplus

$$g_i = \sum_{(v_j, v_i) \in E} x_{ji} - \sum_{(v_i, v_j) \in E} x_{ij} + s_i$$

of each node v_i to zero to make x a feasible flow. The algorithmic detail of ε-relaxation is introduced in the appendix.

It can be shown that the algorithm outputs an optimal solution if ε is small enough (Bertsekas 1986). A pessimistic optimality condition is $\varepsilon < \frac{1}{n}$, but we can relax the bound if the graph satisfies specific structures. The time complexity of ε-relaxation is $O(n^3 \varepsilon^{-1} C)$, which is $O(n^4 C)$ by taking $\varepsilon = \frac{1}{n+1}$. Applying the classic cost-scaling trick improves this bound to $O(n^3 \log(nC))$.

It is advantageous to consider ε-relaxation with predictions, compared with other minimum-cost flow algorithms, such as network simplex:

- The ε-relaxation is a dual algorithm, making it easier to learn a prediction. For a network with n nodes and m edges, the prediction is required to predict only n (resp. $m = O(n^2)$) solution variables for a dual (resp. primal) algorithm.
- The ε-relaxation is easy to parallelize (Beraldi and Guerriero 1997). Many serial minimum-cost flow algorithms, including network simplex, are hard to massively parallelize. The iterations of nodes in ε-relaxation are decoupled, making it suitable for parallel acceleration (Lin et al. 2020).
- The ε-relaxation is mathematically easy to analyze. Although network simplex is state-of-the-art in practice, its theoretical time complexity remains unclear.

Dual Prediction for ε-Relaxation

In this section, we introduce and analyze the ε-relaxation with a dual prediction. We assume the algorithm is given a predicted dual solution $\hat{p}$ and use p^* to denote an arbitrary optimal dual solution. Proofs of technical lemmas are omitted to the appendix.

Notations. For a path H of $G = (V, E)$, it may contain both forward and backward edges of G, following the convention of network flow[2]. We use H^+ and H^- to denote the set of forward and backward edges of H, respectively. We also use $s(H)$ and $t(H)$ to denote the start and end nodes of H. We say a simple path is *unblocked with respect to* x, if $x_{ij} < c_{ij}$ for any $(v_i, v_j) \in H^+$ and $x_{ij} > b_{ij}$ for any $(v_i, v_j) \in H^-$. In other words, a path H is unblocked if we can send positive flow along H to x from $s(H)$ to $t(H)$ without violating the capacities.

Algorithm 1: Warm-start ε-relaxation.

Input: A graph $G = (V, E)$ and a dual prediction $\hat{p}$.
Output: The optimal flow of G.

1: Let $\hat{p}_{\min} \leftarrow \min_{1 \leq i \leq n} \hat{p}_i$;
2: Let $\hat{p} \leftarrow \min\{\hat{p} - \hat{p}_{\min}, (n-1)C\}$. (Clip the values of $\hat{p}$ to make $0 \leq \hat{p} \leq (n-1)C$.)
3: Run ε-relaxation with the initial dual solution $\hat{p}$;

Vanilla warm-start ε-relaxation

We first preprocess the predicted dual and run ε-relaxation with the initial dual solution $\hat{p}$. See Algorithm 1 for a description of our algorithm. Note that we have the following fact, since the dual cost only depends on $p_i - p_j$ for each pair of $(v_i, v_j) \in E$. Therefore, for simplicity, we first shift $\hat{p}$ to make $\min_i p_i = 0$.

Fact 1. *The dual problem of network flow is shift-invariant. If p^* is an optimal solution, $p^* + c$ for any constant c is also an optimal solution.*

We also have the following lemma. Therefore, after preprocessing, the prediction error of the initial dual solution will not increase.

Lemma 1. *There exists an optimal dual solution p^* such that $0 \leq p^* \leq (n-1)C$.*

Proof. By Fact 1, there exists an optimal solution such that $\min_i p_i^* = 0$. It suffices to show $p^* \leq (n-1)C$.

Proof by contradiction. Suppose for any optimal solution $p^* \geq 0$ with $p_s^* = 0$ for $v_s \in V$, there exists $v_t \in V$ such that $p_t^* > (n-1)C$. We can always find an $v_s - v_t$ cut that partitions V into two disjoint subsets, S and T, which satisfies $v_s \in S$, $v_t \in T$ and for any $v_i \in S$, $v_j \in T$, $|p_i^* - p_j^*| > C$. (Otherwise, there exists a path H from v_s to v_t, such that each edge $(v_i, v_j) \in H$ satisfies $|p_i^* - p_j^*| \leq C$, and thus $p_t^* \leq p_s^* + |H|C \leq (n-1)C$.) Since $p_s^* < p_t^*$, the dual values p^* of all nodes in S are smaller than those in T.

Since p^* is dual optimal, there exists a flow x^* such that (x^*, p^*) satisfies complementary slackness. For each forward or backward edge (v_i, v_j) of G in the $v_s - v_t$ cut, we have $p_j^* - p_i^* > C$. Therefore, we can decrease the dual values p^* of each node in T until a cut edge (v_i, v_j) satisfies

[2]An edge (u, v) is a forward (resp. backward) edge of $G = (V, E)$ if $(u, v) \in E$ (resp. $(v, u) \in E$).

$p_j^* - p_i^* = C$ without violating complementary slackness, and hence retain optimality. This is a contradiction. □

Theorem 1. *The time complexity of Algorithm 1 is* $O\left(\min\left\{n^3 + n^2\varepsilon^{-1}\|\hat{p} - p^*\|_\infty,\, n^3\varepsilon^{-1}C\right\}\right)$. *Specially, if the input is a 0/1 flow instance, the bound can be improved to* $O\left(\min\left\{mn + m\varepsilon^{-1}\|\hat{p} - p^*\|_\infty,\, mn\varepsilon^{-1}C\right\}\right)$.

Proof. By Lemma 1, after preprocessing, the infinity norm prediction error is at most $(n-1)C$, it suffices to prove the $O(n^3 + n^2\varepsilon^{-1}\|\hat{p} - p^*\|_\infty)$ bound (and $O(mn + m\varepsilon^{-1}\|\hat{p} - p^*\|_\infty)$ for 0/1 flow).

Following the original analysis of ε-relaxation, we first define a few notations. For any path H in G, we define the *reduced cost length* of H is the sum of each edge's reduced cost,

$$\begin{aligned} d_H(p) = \max\{0, &\sum_{(v_i,v_j)\in H^+} (p_i - p_j - a_{ij}) \\ &- \sum_{(v_i,v_j)\in H^-} (p_i - p_j - a_{ij})\} \\ = \max\{0,\, &p_{s(H)} - p_{t(H)} - \sum_{(v_i,v_j)\in H^+} a_{ij} + \sum_{(v_i,v_j)\in H^-} a_{ij}\}. \end{aligned}$$

For any dual solution p and primal solution x satisfying the flow conservation and capacity constraints (i.e., x is a feasible flow), we let

$$D(p, x) = \max_H\{d_H(p) \mid H \text{ is a simple unblocked path w.r.t. } x\}.$$

If no simple unblocked path exists, we simply let $D(p, x) = 0$. We let

$$\beta(p) = \min_x \{D(p, x) \mid x \text{ is a feasible flow}\}.$$

We can bound $\beta(\hat{p})$ with the infinity norm prediction error.

Lemma 2. *For warm-start dual solution* $\hat{p}$*, we have* $\beta(\hat{p}) \le 2\|\hat{p} - p^*\|_\infty$.

It is shown that the time complexity of ε-relaxation is upper bounded by $O(n^3 + n^2\beta(p_0)/\varepsilon)$ for an initial solution p_0. If the input is a 0/1 flow instance, the time complexity is $O(mn + m\beta(p_0)/\varepsilon)$. (See Proposition 4.1 of Chapter 5, Bertsekas and Tsitsiklis (1989)). Therefore, our bound follows directly from Lemma 2.

Now, it suffices to prove Lemma 2. We have

$$\begin{aligned} \beta(\hat{p}) &= \min_x D(\hat{p}, x) \\ &\le D(\hat{p}, x^*) \\ &= \max_H \{d_H(\hat{p}) \mid H \text{ is unblocked in } x^*\} \\ &= \max_H\{0,\, \hat{p}_{s(H)} - \hat{p}_{t(H)} - \sum_{(v_i,v_j)\in H^+} a_{ij} \\ &\qquad + \sum_{(v_i,v_j)\in H^-} a_{ij} \;\Big|\; H \text{ is unblocked in } x^*\}. \end{aligned} \tag{3}$$

Note that for any unblocked path H in the optimal flow x^*, we have

$$p^*_{s(H)} - p^*_{t(H)} - \sum_{(v_i,v_j)\in H^+} a_{ij} + \sum_{(v_i,v_j)\in H^-} a_{ij} \le 0.$$

This follows from the fact that the optimal solution (x^*, p^*) satisfies complementary slackness so that the reduced cost $p_i^* - p_j^* - a_{ij}$ of each edge (v_i, v_j) in H must be non-positive.

Therefore, for any dual solution p, we have

$$\begin{aligned} &p_{s(H)} - p_{t(H)} - \sum_{(v_i,v_j)\in H^+} a_{ij} + \sum_{(v_i,v_j)\in H^-} a_{ij} \\ &= \left(p^*_{s(H)} - p^*_{t(H)} - \sum_{(v_i,v_j)\in H^+} a_{ij} + \sum_{(v_i,v_j)\in H^-} a_{ij}\right) \\ &\qquad + (p_{s(H)} - p_{t(H)}) - (p^*_{s(H)} - p^*_{t(H)}) \\ &\le (p_{s(H)} - p_{t(H)}) - (p^*_{s(H)} - p^*_{t(H)}) \\ &\le \left|p_{s(H)} - p^*_{s(H)}\right| + \left|p_{t(H)} - p^*_{t(H)}\right| \\ &\le 2\|p - p^*\|_\infty. \end{aligned}$$

Plugging this bound into (3) yields the desired result $\beta(\hat{p}) \le 2\|\hat{p} - p^*\|_\infty$. □

We obtain a time bound $O(\min\left\{n^3\|\hat{p} - p^*\|_\infty,\, n^4C\right\})$, if we set $\varepsilon = \frac{1}{n+1}$ to ensure optimality.

Algorithm 2: Cost scaling for warm-start ε-relaxation.

Input: A graph $G = (V, E)$ and a dual prediction $\hat{p}$; The scaling parameter c.
Output: The optimal flow of G.

1: Run preprocessing as in Algorithm 1;
2: Let T be the maximal integer such that $c^T \le \|\hat{p} - p^*\|_\infty$.
3: Initialize the dual $p^{(T)} \leftarrow \lfloor (n+1)\hat{p}/c^T \rfloor$.
4: **for** each $t = T - 1, \ldots, 0$ **do**
5: Let $p' \leftarrow cp^{(t+1)}$.
6: Run ε-relaxation with the initial dual p', scaled edge costs $a^{(t)} = \lfloor (n+1)a/c^t \rfloor$, $\varepsilon = 1$, and get dual solution $p^{(t)}$.
7: **end for**
8: **return** the optimal flow for the dual $p^{(0)}$.

Cost-scaling for warm-start ε-relaxation

In practice, the vanilla version of the ε-relaxation algorithm is not scalable, since its time complexity grows linearly with respect to the cost scale C. Therefore, the vanilla algorithm is often combined with the ε-scaling trick to induce $O(n^3 \log(nC))$ time complexity. This trick is also applicable to ε-relaxation with dual predictions. The algorithm is described in Algorithm 2.

Note that ε-relaxation returns the optimal solution if $\varepsilon = \frac{1}{n+1}$. We can equivalently multiply the edge costs by $n + 1$ and run the algorithm with $\varepsilon = 1$. However, this does not reduce time complexity. In cost scaling, we solve a sequence of instances and gradually increase the scale of edge costs.

Let c be a scaling parameter (usually we choose $2 \le c \le 4$). Initially we multiply edge costs by roughly $(n+1)/\|\hat{p} - p^*\|_\infty$ and run ε-relaxation. After the algorithm converges, we multiply edge costs and dual solutions by c, and repeat this process until edge costs are in the scale of $(n+1)a$. The algorithm is obviously correct and we analyze the time complexity in the following.

Theorem 2. *The time complexity of Algorithm 2 is* $O\left(\min\left\{n^3 \log \|\hat{p} - p^*\|_\infty, n^3 \log(nC)\right\}\right)$. *Specially, for 0/1 flow instances, this bound can be improved to* $O\left(\min\left\{mn \log \|\hat{p} - p^*\|_\infty, mn \log(nC)\right\}\right)$.

Proof. Similar to Theorem 1, it suffices to prove the first terms of the minimum due to Lemma 1. Since Algorithm 2 will call ε-relaxation at most $\log \|\hat{p} - p^*\|_\infty$ times, we only need to bound the time complexity of each call.

Initially, $c^T = \Theta(\|\hat{p} - p^*\|_\infty)$. Therefore, the infinity norm prediction error of $p^{(T)} = \lfloor (n+1)\hat{p}/c^T \rfloor$ is $O(n)$. Since $\varepsilon = 1$, the time complexity of the first scaling iteration $t = T-1$ is $O(n^3)$ (and $O(mn)$ for 0/1 flow) by Theorem 1.

In subsequent iterations, for each $t = T-2, \dots, 0$, we have $p' = cp^{(t+1)}$ and $a^{(t)} - c \le ca^{(t+1)} \le a^{(t)} + c$. Since $p^{(t+1)}$ is the output of the last ε-relaxation step, $p^{(t+1)}$ satisfies 1-CS with respect to edge costs $a^{(t+1)}$. Therefore, we can simply verify that p' satisfies $2c$-CS with respect to $a^{(t)}$, which means for any path H, we have $d_H(p') \le 2c \cdot |H| = O(n)$, and thus $\beta(p') = O(n)$. The complexity of ε-relaxation for each step $t = T-2, \dots, 0$ is hence $O(n^3)$ (and $O(mn)$ for 0/1 flow). Summing up the time for each step yields the desired bound. □

Remark 1. Algorithm 2 requires the knowledge of $\|\hat{p} - p^*\|_\infty$. Theoretically, we can eliminate this dependence by guessing the value of T from 1 to $\lfloor \log(nC) \rfloor$. If calling ε-relaxation with $t = T-1$ does not return after $\Omega(n^3)$ time, we halt the algorithm and increase the value of T by 1. Otherwise, we continue the algorithm to compute the result. This does not affect the time complexity of the algorithm. In practice, we can directly tune the value of T empirically based on the learning error of the prediction.

Applications

In this section, we introduce applications of the proposed algorithm, and discuss how to learn the dual prediction.

Learning a fixed prediction

Consider the problem of solving minimum-cost flow on a traffic network: Every day, an amount of goods is manufactured at specific locations and to be transported to several warehouses. The topology of the network, the capacity of edges, and the supplies of each node are fixed, but the edge costs can be seen as random variables, depending on the dynamic traffic conditions of every day. Although the minimum-cost flow may vary, the dual optimal solution may not change drastically. It will be time-consuming if we compute the optimal solution from scratch every day. Therefore, we can learn the expected optimal dual solution from collected data, and apply our warm-start ε-relaxation algorithm.

Let $\mathcal{D}$ be a distribution over minimum-cost flow instances, with a fixed network but randomly perturbed edge costs. We use $a \sim \mathcal{D}$ to denote a sample of edge costs from $\mathcal{D}$. Let $a^{(1)}, a^{(2)}, \dots, a^{(k)} \sim \mathcal{D}$ be k identically independently distributed samples from $\mathcal{D}$. We are interested in learning a dual $\hat{p}$ based on the samples with low sample complexity.

Concretely, let $u(a, p) : \mathbb{Z}^m \times \mathbb{R}^n \to [0, H]$ denote the running time performance of warm-start ε-relaxation for edge costs a with the initial solution p, where H is an upper bound of the performance[3]. The generalization error of a learned dual $\hat{p}$ is defined as $\left|\mathbb{E}_{a\sim\mathcal{D}}[u(a,\hat{p})] - \frac{1}{k}\sum_{i=1}^{k} u(a^{(i)},\hat{p})\right|$. The sample complexity is the least number of samples such that the generalization error is at most ε with high probability.

We have the following PAC-learning result, indicating we can learn a fixed prediction with sample complexity $\tilde{O}\left(n/\varepsilon^2\right)$. Since it is a uniform convergence bound, it is independent of the learning algorithm.

Theorem 3. *Let $a^{(1)}, a^{(2)}, \dots, a^{(k)}$ be k i.i.d. samples from $\mathcal{D}$. If*

$$k = \Omega\left(\frac{H^2}{\varepsilon^2}\left(n\log(nC) + \log\left(\frac{1}{\delta}\right)\right)\right),$$

for any $\varepsilon > 0$ and $\delta \in (0,1)$, with probability at least $1-\delta$, for any $p \in \mathbb{R}^n$, we have

$$\left|\mathbb{E}_{a\sim\mathcal{D}}[u(a,\hat{p})] - \frac{1}{k}\sum_{i=1}^{k} u(a^{(i)},\hat{p})\right| \le \varepsilon.$$

The proof technique of this theorem is to prove an upper bound on the pseudo-dimension of the function class of u. The proof is omitted to the appendix due to the page limit.

Lemma 3. *Let $\mathcal{U} = \{u(\cdot, p) \mid p \in [0, (n-1)C]^n\}$, we have* $\mathrm{Pdim}(\mathcal{U}) = O(n \log(nC))$.

Remark 2. Theorem 3 is slightly different from sample complexity bounds in previous algorithm-with-prediction works (Dinitz et al. 2021; Chen et al. 2022; Davies et al. 2023). Previous works only consider the generalization error of the learned prediction, while we directly bound the error of the algorithm performance *end-to-end*, which is more general and practical. This follows the idea of *data-driven algorithm design* (see, e.g., Balcan (2020)).

Now we discuss how to learn the prediction. Let $a^{(1)}, a^{(2)}, \dots, a^{(k)}$ be k training samples, we first compute the optimal duals of these testcases. Since $u(a, \hat{p}) = O(n^3 \log \|\hat{p} - p^*\|_\infty)$, we can naturally learn the prediction by minimizing the surrogate loss, i.e.,

$$\min_{\hat{p}\in\mathbb{R}^n, \Delta\in\mathbb{R}^k} \frac{1}{k}\sum_{i=1}^{k} \log \|\hat{p} - p^{(i)} + \Delta_i\|_\infty,$$

[3]We assume H is a constant. This assumption is standard in previous learning-augmented algorithms (Dinitz et al. 2021; Chen et al. 2022; Davies et al. 2023), and is indeed reasonable. We can always normalize the running time since graph topology is fixed.

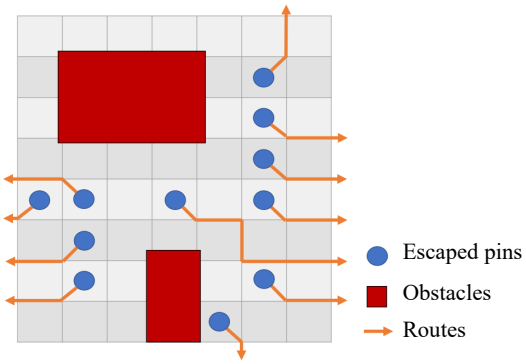

Figure 1: An example of unordered escape routing.

where $p^{(i)}$ is an optimal dual for $a^{(i)}$, Δ_i is the shift value by Fact 1. This problem can be converted into optimizing a convex function under linear constraints and solved by Frank-Wolfe:

$$\min_{\hat{p},\Delta,M} \frac{1}{k}\sum_{i=1}^{k}\log(M_i),$$
$$\text{subj. to } -M_i \le \hat{p}_j - p_j^{(i)} + \Delta_i \le M_i, \forall\, i,j,$$
$$M_i \ge 0, \forall\, i.$$

Learning a feature-based predictor

A more general way of obtaining the prediction is to learn a mapping from a problem instance to the dual solution using machine learning models, such as neural networks. In this work, we apply this method to the escape routing problem in electronic design automation.

Escape routing is an important design automation problem in the design of printed circuit boards (PCB) (Yan and Wong 2010). In this work, we focus on the unordered escape problem: Given a $w \times h$ pin array with specific pins required to escape, find a routing solution that routes these pins to the boundary of the board without intersection. The total routed length should be minimized. Part of the routing grids may be occupied by obstacles and cannot be routed. See Figure 1 for an illustrating example.

Unordered escape can be modeled as a minimum-cost flow problem. Given the grid structure of the routing graph, we can naturally predict the dual solution using a convolutional neural network model. The routing grid is modeled as a $(w+1)\times(h+1)$ grid graph, and edges with unit capacity and geometric distance cost between adjacent nodes are added. Each pin is modeled as a supply node, with edges to its adjacent grid points. An artificial demand node is added to represent the destination with edges from boundary nodes. Each routing path is viewed as a unit flow from the pin to the boundary.

We introduce how to learn a feature-based neural network model for escape routing. The input to the model is a 3-channel $(w+1)\times(h+1)$ feature map. The 3 features include a binary value denoting whether it is occupied by obstacles, the shortest path distance to the nearest pin, and the shortest path distance to the boundary. The output is a $(w+1)\times(h+1)$ matrix representing the predicted dual. The dual of the destination is fixed to be 0, and the dual of pins is the maximum dual of adjacent grid nodes. To normalize the inputs and outputs, we divide the input features and output duals by $\max(w,h)$ so that the range of values is between $[0,1]$. We use a fully convolutional neural network with a UNet-style architecture (Ronneberger, Fischer, and Brox 2015) as the prediction model. The model first transforms the input into 16-channel features using a 3×3 convolution kernel. The backbone consists of two downsampling and two upsampling blocks, each of which block contains two 3×3 convolution layers and two group normalization layers. Each pair of downsampling and upsampling layers with the same feature size is linked with residual connections. Finally, the model produces the output using a 3×3 convolution kernel. We directly use the average mean-square loss (a.k.a., l_2 norm error) to train the neural network, since we find that l_∞ norm error makes training very difficult.

We can similarly bound the sample complexity of the neural network model. Let $\mathcal{H}$ be a class of neural network functions that map the instance feature space to $[0,(n-1)C]^n$, the space of predicted duals. In other words, for a neural model $h\in\mathcal{H}$, it reads the feature of a minimum-cost flow instance and outputs an $\mathbb{R}^n$ vector, where the i-th component $h_i(\cdot)$ denote the initial dual of node v_i. We use $\mathcal{H}_i = \{h_i \mid h\in\mathcal{H}\}$ to denote the class of functions from $\mathcal{H}$ but limiting the output to be the i-th component. Let $d_{\mathsf{NN}} = \max_i \mathrm{Pdim}(\mathcal{H}_i)$ denote the pseudo-dimension of the neural network architecture. Again, we use $u(\Phi,p)$ to denote the running time of the algorithm with dual p on instance Φ, and $\mathcal{D}$ is a distribution of Φ.

Theorem 4. *Let $\Phi^{(1)},\Phi^{(2)},\dots,\Phi^{(k)}$ be features[4] of k i.i.d. samples from $\mathcal{D}$. If*

$$k = \Omega\left(\frac{H^2}{\varepsilon^2}\left(nd_{\mathsf{NN}}\log(nC)+\log\left(\frac{1}{\delta}\right)\right)\right),$$

for any $\varepsilon>0$ and $\delta\in(0,1)$, with probability at least $1-\delta$, for any $h\in\mathcal{H}$, we have

$$\left|\mathbb{E}_{\Phi\sim\mathcal{D}}[u(\Phi,h(\Phi))] - \frac{1}{k}\sum_{i=1}^{k}u(\Phi^{(i)},h(\Phi^{(i)}))\right| \le \varepsilon.$$

The proof of this theorem follows directly from Lemma 4.

Lemma 4. *Let $\mathcal{U} = \{u(\cdot,h(\cdot)) \mid h\in\mathcal{H}\}$, $\mathrm{Pdim}(\mathcal{U}) = O(nd_{\mathsf{NN}}\log(nC))$.*

Remark 3. The pseudo-dimension of neural networks d_{NN} for typical neural architectures has been widely studied. For example, Bartlett et al. (2019) gives an $O(WL\log W)$ upper bound for piecewise linear networks (e.g., networks with the ReLU activation) with W weights and L layers.

Experiments

We validate our theoretical results with empirical studies of two applications. We show that the ε-relaxation with predictions can significantly improve the running time, compared

[4]With a little abuse of notation, we use Φ to denote both a problem instance and its feature.

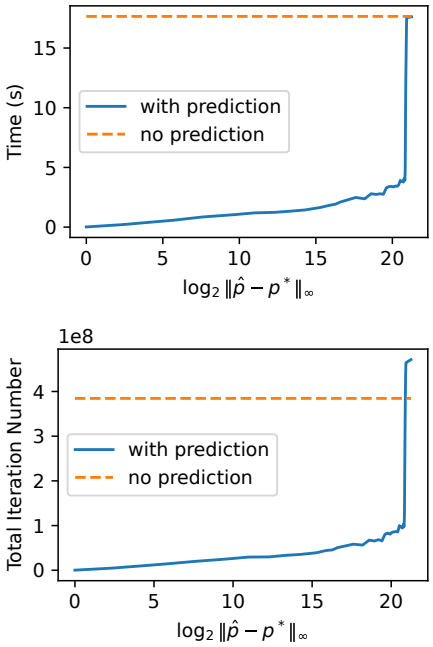

Figure 2: The running time and total number of iterations for the cost-scaling version of ε-relaxation on `flow_03_NH`.

with the algorithm without predictions. We also implement two min-cost algorithms, network simplex (NS), and successive shortest paths (SSP). Experimental settings are omitted to the appendix.

Synthesized predictions

We first study the effect of the dual prediction by synthesized instances. Given an instance, we first compute the optimal solution, and randomly perturb the solution to synthesize predictions of different errors. Figure 2 illustrates the result of the cost-scaling version of ε-relaxation. The efficiency of the algorithm grows with respect to the prediction error and is robust when the prediction is completely erroneous. The curves are not quite smooth due to the discreteness of rounding in Algorithm 2. The growth of running time is steady when the error is small, but increases sharply when the error is sufficiently large. We explain the reason in the appendix.

Road networks

We study the application of traffic networks. For each instance, we sample 10 instances by randomly perturbing the edge costs, compute their optimal solutions, and learn a fixed prediction. Then, we test the algorithm with this prediction on the original instance. We use the cost-scaling version of ε-relaxation for comparison. Table 1 illustrates the empirical results. We present results on large-scale instances with $>$ 200000 nodes. In the benchmark, there are 5 instances for each group. We report the average running time of each group. We achieve approximately $6.2 \sim 21.4\times$ speedup using the learned prediction. Note that the groups of `paths_*`

Group	n	ε-R (p.)	ε-R	NS	SSP
`paths_04_NV`	261155	**1.498**	24.247	19.293	11.684
`flow_04_NV`	261155	**18.214**	112.926	20.223	51.211
`paths_05_WI`	519157	**3.875**	58.839	61.41	73.493
`flow_05_WI`	519157	**30.548**	221.457	65.742	536.081
`paths_06_FL`	1048506	**5.193**	111.071	141.1	519.3
`flow_06_FL`	1048506	**45.946**	630.73	271.133	1769.58
`paths_07_TX`	2073870	**25.866**	323.846	694.118	2113.63
`flow_07_TX`	2073870	**238.41**	2247.67	1127.91	too long
avg.		**1.00**	12.74	12.08	38.19

Table 1: Running times (sec) of min-cost flow algorithms on road network instances. ε-R denotes ε-relaxation, and (p.) denotes "with prediction".

Case	$w \times h$	# pins	ε-R (p.)	ε-R	NS	SSP
1	296×277	710	**2.103**	3.302	11.749	11.546
2	366×418	963	**5.893**	10.035	63.578	54.41
3	500×500	1200	**17.929**	23.417	80.398	72.52
4	365×610	1188	**12.865**	29.815	135.689	122.328
5	635×572	1366	**45.24**	70.182	453.915	280.034
6	670×675	1432	**52.069**	94.842	687.878	385.966
7	805×917	1655	**88.99**	139.521	1572.02	1111.64
8	734×812	2019	**265.732**	293.837	1943.52	1065.56
9	1003×965	2001	**126.141**	245.585	2662.63	1616.98
10	1205×1374	1985	**630.487**	955.19	too long	2592.41
avg.			**1.00**	1.64	11.19	7.53

Table 2: Running times (sec) of min-cost flow algorithms on escape routing instances. "# pins" denotes the number of pins to escape. Other abbreviations follow Table 1.

are 0/1 flow instances, ε-relaxation hence perform better, even outperforming network simplex without the help of predictions.

Escape routing

We also study the application of escape routing. We use the cost-scaling version of ε-relaxation for comparison. Table 2 illustrates the empirical results on 10 instances in the test set. We achieve approximately $1.1 \sim 2.3\times$ speedup using the learned prediction. Note that the board size of the testcase varies from less than 300×300 to more than 1000×1000. We obtain the prediction using only a single neural network. This shows the generalization ability of the neural model across different problem sizes.

Conclusion

In this work, we show how to improve the ε-relaxation algorithm for min-cost flow with a machine-learned dual prediction. We prove strong time complexity bounds and study the sample complexity of learning the prediction. Experimental results illustrate considerable acceleration in two applications: traffic networks and escape routing.

Acknowledgments

This work is supported by the Major Research Plan of the National Natural Science Foundation of China (No. 92573107), and the Key Program of National Natural Science Foundation of China (No. 62034005).

References

Ahuja, R. K.; Magnanti, T. L.; and Orlin, J. B. 1993. *Network Flows: theory, algorithms, and applications*. Prentice Hall.

Ahuja, R. K.; Magnanti, T. L.; Orlin, J. B.; and Reddy, M. 1995. Applications of network optimization. *Handbooks in Operations Research and Management Science*, 7: 1–83.

Balcan, M.-F. 2020. Data-driven Algorithm Design. *arXiv preprint arXiv 2011.07177*.

Balcan, M.-F.; DeBlasio, D.; Dick, T.; Kingsford, C.; Sandholm, T.; and Vitercik, E. 2021. How Much Data is Sufficient to Learn High-Performing Algorithms? Generalization Guarantees for Data-Driven Algorithm Design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 919–932.

Balcan, M.-F.; Dick, T.; Sandholm, T.; and Vitercik, E. 2024. Learning to Branch: Generalization Guarantees and Limits of Data-Independent Discretization. *Journal of the ACM*, 71(2).

Balcan, M.-F.; Prasad, S.; Sandholm, T.; and Vitercik, E. 2022. Structural Analysis of Branch-and-Cut and the Learnability of Gomory Mixed Integer Cuts. In *Advances in Neural Information Processing Systems*, volume 35, 33890–33903.

Bartlett, P. L.; Harvey, N.; Liaw, C.; and Mehrabian, A. 2019. Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks. *The Journal of Machine Learning Research*, 20(1): 2285–2301.

Beraldi, P.; and Guerriero, F. 1997. A parallel asynchronous implementation of the ε-relaxation method for the linear minimum cost flow problem. *Parallel Computing*, 23(8): 1021–1044.

Bertsekas, D. 1986. Distributed relaxation methods for linear network flow problems. In *1986 25th IEEE Conference on Decision and Control*, 2101–2106.

Bertsekas, D. 1998. *Network optimization: continuous and discrete models*, volume 8. Athena Scientific.

Bertsekas, D.; and Tsitsiklis, J. 1989. *Parallel and Distributed Computation: Numerical Methods*.

Blum, A.; Dan, C.; and Seddighin, S. 2021. Learning Complexity of Simulated Annealing. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130, 1540–1548.

Brand, J. V. D.; Chen, L.; Kyng, R.; Liu, Y. P.; Peng, R.; Gutenberg, M. P.; Sachdeva, S.; and Sidford, A. 2023. A Deterministic Almost-Linear Time Algorithm for Minimum-Cost Flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, 503–514.

Chen, J.; Silwal, S.; Vakilian, A.; and Zhang, F. 2022. Faster Fundamental Graph Algorithms via Learned Predictions. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 3583–3602. PMLR.

Chen, Z.; Yao, H.; and Yin, X. 2025. Learning Configurations for Data-Driven Multi-Objective Optimization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, 9642–9663. PMLR.

Cheng, H.; Khalife, S.; Fiedorowicz, B.; and Basu, A. 2024. Sample Complexity of Algorithm Selection Using Neural Networks and Its Applications to Branch-and-Cut. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Davies, S.; Moseley, B.; Vassilvitskii, S.; and Wang, Y. 2023. Predictive flows for faster ford-fulkerson. In *Proceedings of the 40th International Conference on Machine Learning*.

Dinitz, M.; Im, S.; Lavastida, T.; Moseley, B.; and Vassilvitskii, S. 2021. Faster Matchings via Learned Duals. In *Advances in Neural Information Processing Systems*, volume 34, 10393–10406.

Edmonds, J.; and Karp, R. M. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *Journal of the ACM*, 19(2): 248–264.

Goldberg, A. V.; and Tarjan, R. E. 1987. Solving minimum-cost flow problems by successive approximation. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, 7–18.

Gupta, R.; and Roughgarden, T. 2017. A PAC Approach to Application-Specific Algorithm Selection. *SIAM Journal on Computing*, 46(3): 992–1017.

Kara, G.; and Özturan, C. 2022. Parallel network simplex algorithm for the minimum cost flow problem. *Concurrency and Computation: Practice and Experience*, 34(4): e6659.

Klein, M. J. 1966. A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems. *Management Science*, 14: 205–220.

Kraska, T.; Beutel, A.; Chi, E. H.; Dean, J.; and Polyzotis, N. 2018. The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data*, 489–504.

Lin, Y.; Li, W.; Gu, J.; Ren, H.; Khailany, B.; and Pan, D. Z. 2020. ABCDPlace: Accelerated Batch-Based Concurrent Detailed Placement on Multithreaded CPUs and GPUs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12): 5083–5096.

Lindermayr, A.; and Megow, N. 2024. Website for Algorithms with Predictions. Available at https://algorithms-with-predictions.github.io/.

Niaparast, H.; Moseley, B.; and Singh, K. 2025. Faster Global Minimum Cut with Predictions. In *Forty-second International Conference on Machine Learning*.

Orlin, J. B. 1997. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78: 109–129.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, 234–241.

Sakaue, S.; and Oki, T. 2022. Discrete-Convex-Analysis-Based Framework for Warm-Starting Algorithms with Predictions. In *Advances in Neural Information Processing Systems*, volume 35, 20988–21000.

Wang, C.; Wang, Y.; Wang, Y.; Wu, C.-T.; and Yu, G. 2019. muSSP: Efficient Min-cost Flow Algorithm for Multi-object Tracking. In *Advances in Neural Information Processing Systems*, 423–432.

Yan, T.; and Wong, M. D. F. 2010. Recent research development in PCB layout. In *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 398–403.

Zadeh, N. 1973. A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 5: 255–266.