Speeding Up Bellman Ford via Minimum Violation Permutations

Silvio Lattanzi [1] Ola Svensson [2] Sergei Vassilvitskii [1]

Abstract

The Bellman-Ford algorithm is a basic primitive for computing single source shortest paths in graphs with negative edge-weights. Its running time is governed by the order the algorithm examines vertices for iterative updates on the value of their shortest path. In this work we study this problem through the lens of 'Algorithms with Predictions,' and show how to leverage auxiliary information from similar instances to improve the running time. We do this by identifying the key problem of Minimum Violation Permutations, and give algorithms with strong approximation guarantees as well as formal lower bounds. We complement the theoretical analysis with an empirical evaluation, showing that this approach can lead to a significant speed up in practice.

1. Introduction

Traditionally, worst-case analysis has been the main tool for understanding algorithms' performance, as it gives insight into both the complexity and practicality of algorithms. There are however, numerous examples where worst-case analysis is overly pessimistic, as the "typical" instances seen by the algorithm are far from those generating worst-case bounds. A recent line of work attempts to rigorously analyze this setting by developing algorithms that get the best of both worlds — they are consistent, performing near-optimally on such 'typical' examples, but are also robust, doing not much worse than their classical counterparts on atypical instances.

Formally, these algorithms are parameterized by some "advice" that can be efficiently learned from past data. Examples in this area include efficient data structures (Kraska et al., 2018; Mitzenmacher, 2018), algorithms for online resource allocation (Lattanzi et al., 2020; Dinitz et al., 2021; Gollapudi & Panigrahi, 2019), online-learning (Bhaskara et al., 2020) and many others, see the overview by Mitzenmacher & Vassilvitskii (2021) for more details. In many of the settings learning the best advice for a sequence of examples is relatively easy, and the main challenge lies in using it. For instance, for scheduling (Lattanzi et al., 2020) and matching (Dinitz et al., 2021) the best advice is obtained by averaging the optimal solution on each instance.

In this work we give an example of the opposite problem. We consider a basic computer science primitive, finding single source shortest paths in a graph $G = (V, E)$ with negative weight edges. The Algorithms 101 solution for this problem is to use the Bellman-Ford algorithm, which repeatedly iterates through all of the vertices, updating the best estimate for the distance from the source every time. The algorithm terminates when no more updates are made. An unspecified parameter of the algorithm is the order in which the vertices should be processed. It is easy to see that there is an ordering (that of increasing distance from s) so that the algorithm terminates after a single pass through the data. On the other hand, in the worst case, one may require $|V|$ iterations. Since each pass through the data takes $O(|E|)$ time, the choice of the ordering is critical – it is the difference between a linear time $O(|E|)$ algorithm, and much higher running time of $O(|E||V|)$.

From a worst case standpoint, there is almost nothing that can be done. However suppose that you have access to graphs $G_1, G_2, \ldots, G_k$ that arise from related shortest path instances. A natural arising question is whether one can exploit this "statistical" knowledge by computing the best ordering for the collection of graphs, i.e. one that minimizes the total running time of Bellman-Ford on $G_1, G_2, \ldots, G_k$. As we will see, this problem will turn out to have connections to the Feedback Arc Set problem, and is computationally non-trivial. Nonetheless, we will design effective algorithms and show that this approach can beat traditional Bellman-Ford on real world data.

1.1. Our Contributions

Our main goal is to improve the running time of the Bellman-Ford algorithm in the setting when the examples are not worst case. To do so, we introduce a pre-processing step

[1]Google Research [2]EPFL, Switzerland. Correspondence to: Silvio Lattanzi <silviol@google.com>, Ola Svensson <ola.svensson@epfl.ch>, Sergei Vassilvitskii <sergeiv@google.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

to the Bellman-Ford algorithm that, given a collection of typical instances, selects a good order to examine the vertices. In doing so, we allow for a larger (yet still polynomial) pre-processing cost in order to have a smaller running time per instance.

- We introduce the Minimum Violation Permutation (MVP) problem that captures the problem of finding the best ordering to relax vertices in the algorithm (Section 2).
- We show how to approximate the MVP problem by first getting a fractional solution via Linear Programming, and then giving a rounding scheme (Section 3.1).
- We give another method that obtains a bi-criteria approximation, finding a collection of total orders, one of which is good for each path (Section 3.2).
- We complement our algorithmic results with strong theoretical lower bounds (Section 4).
- We bound the sample complexity and adapt our techniques to the stochastic variant where we wish to minimize the expected (and not worst-case) running time of a given distribution of instances (Section A).
- Finally, we give an empirical evaluation of this problem, and show that a learned ordering can be effective in speeding up the classical Bellman-Ford algorithm (Section 5).

1.2. Related Work

There are three main avenues of related work. The first is on using predictions to improve performance of algorithms. This line of work started with the seminal work by Kraska et al. (2018), which demonstrated improved empirical performance. The setup was then extended by Mitzenmacher (2018) (see also (Mahdian et al., 2007) for an early theoretical work) and modeled by Lykouris & Vassilvitskii (2021), we point to the survey by Mitzenmacher & Vassilvitskii (2021) for an overview of results. For the specific problem of improving running times, Dinitz et al. (2021) was the first to apply the algorithms with predictions framework to show how to warm start the Hungarian algorithms for matchings. Chen et al. (2022) built upon this work to generalize to warm starting multiple primal-dual based graph algorithms. Importantly, in both of those papers the "advice" is given in form of a set of dual solutions, whereas in our work we look for a permutation of the nodes.

One of the applications of the techniques in (Chen et al., 2022) is the single source shortest path problem with negative edge-weights. The proposed algorithm is significantly different and is not based on the Bellman-Ford algorithm. Instead, the authors use the idea that given an optimal dual solution, we can transform the instance into one in which there are no negative edge-weights and use one of the algorithms for such instances (e.g. Dijkstra's algorithm). This is similar to the algorithm by Bernstein et al. (2022) that guarantees a near-linear running time on any instance without advice. However, the latter algorithm is significantly more complex than the Bellman-Ford method.

Another fundamental difference between the two algorithms is the aforementioned advice/predictions that they receive. As a result, the algorithms are incomparable from a running time or experimental perspective. Indeed, both algorithms run in linear time when the predictions are perfect but as soon as there is an error in the prediction the running time of the algorithms diverges because they use different forms of advice and adjust their behavior differently. Finally, we also note that the Chen et al. (2022) error measure of the prediction is pseudo-polynomial (can depend on the value of the distances) or they lose a $\sqrt{n}$-factor compared to our polylogarithmic factor.

Next, as we will see the problem that is central to our analysis is that of Feedback Arc Set (FAS). Given a directed graph, the FAS problem asks for the minimum (weight) set of edges to be removed from the graph so that the remainder is acyclic. It is well known to be NP-hard, and hard to approximate in general graphs assuming the unique games conjecture (Guruswami et al., 2011; Svensson, 2013). For the special case of tournament graphs (where each pair of nodes has a directed edge between them), the problem admits a constant approximation. The best known theoretical bounds for weighted FAS are due to (Even et al., 1998), who give a $O(\log n \log\log n)$-approximation. However, the algorithms that achieve this approximation are relatively complex. On the other hand, given the prominence of the FAS problem, there has been empirical work on the problem, and relatively simple greedy methods have been shown to achieve good results (Simpson et al., 2016; Fox et al., 2010). We will use these methods for our empirical evaluation in Section 5.

Finally, previous authors have made the observation that the Bellman-Ford algorithm's running time is dependent on the order in which the vertices are processed. We highlight two works in this regard. The first, by Yen (1970) showed how to reduce the worst case number of iterations to $|V|/2$ by alternating between two orders in successive iterations. A further improvement was made by Bannister & Eppstein who proved that a random permutation leads to $|V|/3$ iterations in expectation on any instance.

2. Preliminaries

Let $G = (V, E)$ denote a weighted directed graph with $|V| = n$ nodes and $|E| = m$ edges. For an edge $e \in E$ we let w_e be the weight of e, and note that some weights may be negative, although the graph does not contain any negative weight cycles.

Given a designated node $s \in V$, the Bellman-Ford algorithm computes the shortest path from s to every node $v \in V$. The algorithm maintains an upper bound on the shortest distance, d_v from s to v. It begins by setting $d_v = \infty$ (except $d_s = 0$) and proceeds by iterating over all of the vertices, and *relaxing* all of the incoming edges, i.e. for a vertex v, setting $d_v = \min_{y:(y,v)\in E} d_y + w(y,v)$. It is elementary to show that if G does not have any negative weight cycles, the algorithm converges after at most $n-1$ iterations through the full vertex order.

Formal problem statement. While the number of iterations is $n-1$ in the worst case, we can be more precise about the number of iterations when considering a specific ordering of the vertices.

Given a graph G and a node $s \in V$, let $P_v = (s, x_1, x_2, \ldots, v)$ denote the shortest path from s to v. Let $<_{\text{tot}}$ be an ordering on V where for two vertices u, v, we have $u <_{\text{tot}} v$ if u appears earlier in the order. For a path P, let $\operatorname{dist}(<_{\text{tot}}, P)$ be the number of edges in P reversed by $<_{\text{tot}}$, i.e., $\operatorname{dist}(<_{\text{tot}}, P) = |\{(u,v) \in P \mid v <_{\text{tot}} u\}|$.

We start with the following easy to prove Lemma.

Lemma 2.1. *Given a graph G, the running time of the Bellman-Ford algorithm when using order $<_{tot}$ to process the vertices is*

$$O(m \cdot \max_{v\in V} \operatorname{dist}(<_{tot}, P_v)).$$

Proof. Consider a vertex $v \in V$ and the shortest path $P_v = (s, x_1, x_2, \ldots, x_{p-1}, v)$. Denote the edges of P_v that $<_{\text{tot}}$ reverses by (x_{i_j}, x_{i_j+1}) for $j = 1, 2, \ldots, \operatorname{dist}(<_{\text{tot}}, P_v)$, where we for notational simplicity let $x_0 = s$ and $x_p = v$. The statement follows by arguing that the shortest distance from s to v is calculated in at most $2\operatorname{dist}(<_{\text{tot}}, P_v)$ iterations (since each iteration of Bellman-Ford takes time $O(m)$). This follows from the following observation: in the first iteration, all distances from s to vertices $x_1, x_2, \ldots, x_{i_1}$ are correctly calculated as we relax the vertices in the order of P_v for these vertices. In the second iteration the correct distance to x_{i_1+1} is calculated (when we relax x_{i_1+1}), and then by the same argument as the first iteration, the correct distances to $x_{i_1} + 2, \ldots, x_{i_2}$ are calculated in the third iteration; and so on. □

The above Lemma formally points to the benefit of processing vertices in a good order. However the good order is precisely the one that we are trying to find using the Bellman-Ford algorithm, so without any additional information this knowledge is moot.

Suppose, however, that the graph G is fixed, but the weights on the edges are drawn from some unknown distribution. What is the best ordering that we should use?

We first define an abstract problem of Minimum Violation Permutation (MVP).

Definition 2.2. Given a vertex set V and a set of k paths $P_1, P_2, \ldots, P_k$, where each $P_i \subseteq V$, find a total order $<_{\text{tot}}$ on V that minimizes the objective

$$\max_{i\in[k]} \operatorname{dist}(<_{\text{tot}}, P_i).$$

In order to see the relationship between MVP and the shortest path problem, let $G = (E, V)$ and a set of weight functions $w^1, w^2, \ldots, w^k$. For each $i \in [k]$, we define P_v^i as the shortest path from s to v under w^i. Now consider the solution $<_{\text{tot}}$ to the MVP problem for the set of paths $\cup_{i\in[k]} \cup_{v\in V} P_v^i$. This is precisely the solution that reduces the worst case running time of Bellman-Ford on any of the graphs G. Specifically, the maximum number of iterations of the Bellman-Ford algorithm on any of the k instances is at most $\max_{i\in[k],v\in V} \operatorname{dist}(<_{\text{tot}}, P_v^i)$ which equals the value of the solution $<_{\text{tot}}$ to the associated MVP instance.

3. Nearly Tight LP-Based Algorithms

Our algorithms in this section use a very natural linear programming relaxation. Consider a MVP instance $V, P_1, \ldots, P_k$ and let E be the edges appearing in the paths. For each $e \in E$, we have a variable x_e with the intended meaning that x_e indicates whether the edge e is reversed in the total ordering.

$$\begin{array}{lll} \text{minimize} & T & \\ \text{subject to} & \sum_{e\in P_i} x_e \le T, & \text{for every path } P_i, \\ & \sum_{e\in C} x_e \ge 1, & \text{for every cycle } C \text{ in } (V,E), \end{array} \tag{1}$$

$$x_e \ge 0 \qquad \text{for every edge } e \in E. \tag{2}$$

The first set of constraints say that at most T edges should be reversed in each path; the second set of constraints is valid for any total ordering because any such ordering must reverse at least one edge in any cycle. We refer to this linear program as LP-MVP. We will also heavily rely on the following theorem proved in (Even et al., 1998), which in turn builds upon earlier work on the unweighted version (Seymour, 1995). Recall that $n = |V|$ denotes the number of vertices in the underlying graph (V, E).

Theorem 3.1. *Suppose x satisfies (1) and (2), and let $w : E \to \mathbb{R}_{\ge 1}$ be arbitrary edge-weights taking value* *at least one. There is a polynomial-time algorithm that returns a feedback arc set of cost at most $\alpha \cdot \tau$, where $\tau = \sum_{e\in E} x_e w(e)$ and $\alpha = O(\min\{\log\tau \cdot \log\log\tau, \log n \log\log n\})$.*

In the next subsection, we use this theorem to give nearly tight bounds on the integrality gap of LP-MVP. Recall that the integrality gap is defined as the largest ratio, over all instances of MVP, of the optimal integral value divided by the optimal value of the linear program relaxation LP-MVP. Then, in Section 3.2, we give a bi-criteria algorithm with improved guarantees. Finally, in Section 3.3 we state the implications of these algorithms on the Bellman-Ford algorithm.

3.1. Almost tight rounding of linear program

Our rounding algorithm of the linear program relies on a structural property regarding directed graphs with no short cycles. Structural questions in directed graphs are often much more challenging than the counterpart in undirected graphs, and many questions remain open in combinatorics. However, (Fox et al., 2010) gave a result that is very close to what we want to prove. Recall that the girth of a graph is the length of the shortest cycle. Informally, they proved that a directed graph with girth g has a feedback *arc* set of size at most $O(n^2/g^2)$. Moreover, this bound is tight. Here, we give a simple proof that upper bounds the feedback *vertex* set in directed graphs of large girth. Recall that, in the feedback vertex set problem, we are given a directed graph and the task is to find the minimum set of vertices to be removed so that the remaining graph is acyclic.

Lemma 3.2. *An n-vertex directed graph G with girth g has a feedback vertex set of size $O(\tau \cdot \log\tau \log\log\tau)$, where $\tau = n/g$. Moreover, such a feedback vertex set can be found in polynomial time.*

Proof. We obtain digraph H from G by making two copies of each vertex v, v_{in} and v_{out}, adding a new edge $(v_{\text{in}}, v_{\text{out}})$, and redirecting the edges going into v to v_{in} and those going out of v so that they go out of v_{out}. We will refer to the edges of type $(v_{\text{in}}, v_{\text{out}})$ as new edges and the other edges as original edges.

Now let x be fractional solution on H defined by letting x_e equal $1/g$ on the new edges and x equals 0 on the remaining original edges. Note that x satisfies constraints (1) and (2) since the graph has girth g. Letting the weight function w equal 1 on new edges and ∞ on original edges, we have $\sum_e x_e w(e) = n/g = \tau$. Moreover, Theorem 3.1 implies that we can in polynomial-time find a set F of only new edges (since an original edge e has $w(e) = \infty$) such that

$$|F| = O(\tau \cdot \log\tau \log\log\tau)\,.$$

The vertices in the original graph corresponding to edges in F is the desired feedback vertex set. □

Algorithm 1 LP Rounding Algorithm

Input: A fractional solution x to LP-MVP.
Output: A total order $<_{\text{tot}}$ of V.

```
1: Let $g = \sqrt{n\log n\log\log n}$ and $F = \{e \in E \mid x_e \geq 1/g\}$.
2: Use Lemma 3.2 to find $V'$ such that $|V'| = O(g)$ and graph $(V \setminus V', E \setminus F)$ is acyclic.
3: Output $<_{\text{tot}}$ obtained by topological sorting $(V\setminus V', E\setminus F)$ and appending the vertices $V'$ in an arbitrary order.
```

We remark that the above lemma is almost tight for $g = \sqrt{n}$. Indeed, if we consider a cycle of length $\sqrt{n}$ where every vertex is replaced by a "cloud" of $\sqrt{n}$ vertices and every adjacent cloud is a complete bipartite graph, then the smallest feedback vertex set equals $\sqrt{n}$. We conjecture that this is the right bound, i.e., that any directed graph of girth $\sqrt{n}$ has a feedback vertex set of size $O(\sqrt{n})$. We believe that this is an interesting problem in itself. Moreover, an affirmative solution to this conjecture would imply a tight rounding algorithm of the linear program LP-MVP using the techniques we describe next (by replacing $g = \sqrt{n\log n\log\log n}$ in Algorithm 1 by $g = \sqrt{n}$).

LP-rounding algorithm. Fix an instance $V, P_1, \ldots, P_k$ of MVP and let x be a fractional solution to LP-MVP. As before, we let E denote the edges that appear in the paths. The first step of our rounding algorithm (Algorithm 1) is now to identify the edge set $F = \{e \in E : x_e \geq 1/g\}$ where $g = \sqrt{n\log n\log\log n}$. Note that the feasibility of x implies that $(V, E\setminus F)$ has girth at least $g+1$. Indeed, if we consider a cycle C of length at most g, then $\sum_{e\in C} x_e \geq 1$ by the feasibility of x. This in turn implies that at least one of the edges in C is in F. This allows us to use Lemma 3.2 in the second step of the algorithm to find a feedback vertex set V' of $(V, E\setminus F)$ of size at most $O\left(\frac{n}{g}\log\frac{n}{g}\log\log\frac{n}{g}\right)$. By the selection of g this equals $O(g)$. Finally, Algorithm 1 outputs the total order $<_{\text{tot}}$ of V obtained by doing a topological sort of the acyclic graph $(V \setminus V', E\setminus F)$ and then appending the vertices V' in any order.

Algorithm 1 clearly runs in polynomial time. It remains to upper bound its approximation guarantee.

Lemma 3.3. *Algorithm 1 outputs a total order $<_{tot}$ such that*

$$\max_{i\in[p]} \operatorname{dist}(<_{tot}, P_i) = O(g\cdot T + g)\,,$$

where $g = \sqrt{n\log n\log\log n}$.

Proof. We prove that $\operatorname{dist}(<_{\text{tot}}, P_i) = O(g\cdot T + g)$ for any path P_i. If an edge $(u,v) \in P_i$ is reversed in $<_{\text{tot}}$, i.e., $v <_{\text{tot}} u$, at least one of the following holds:

1. $(u,v) \in F$,
2. $\{u,v\} \cap V' \neq \emptyset$.

Indeed all other edges of P_i are part of the graph $(V \setminus V', E \setminus F)$ and thus correctly ordered in $<_{\text{tot}}$ by the topological sort. We now bound the impact of F and V' on P_i separately. As F only includes edges of fractional value at least $1/g$,

$$\sum_{(u,v)\in P_i} \mathbf{1}\{(u,v) \in F\} \leq \sum_{(u,v)\in P_i} x_{(u,v)} \cdot g \leq T \cdot g\,.$$

Now, since P_i is a path, we have that each vertex in P_i takes part in at most 2 edges (exactly 2 if not an end point). Therefore,

$$\sum_{(u,v)\in P_i} \mathbf{1}\{\{u,v\} \cap V' \neq \emptyset\} \leq 2 \cdot |P_i \cap V'| \leq 2|V'| = O(g)\,.$$

Summing up the two bounds yields $\text{dist}(<_{\text{tot}}, P_i) = O(g \cdot T + g)$, which in turn implies the lemma. □

It is known that we can solve the linear program LP-MVP via the ellipsoid method since we can separate efficiently over the inequalities (it reduces to finding a negative cycle in a graph). Moreover, we may assume that the optimal solution has value at least 1. Indeed, otherwise the graph (V,E) is acyclic and we can easily find an optimal solution $<_{\text{tot}}$ of cost 0 by a simple topological sort. Our rounding algorithm with an optimal solution to LP-MVP therefore yields the following theorem:

Theorem 3.4. *There is a $O(\sqrt{n \log n \log\log n})$-approximation algorithm for the MVP problem.*

Lower bound on the integrality gap We complement the upper bound given by Theorem 3.4 with an almost matching lower bound on the integrality gap[1] of LP-MVP. Theorem 3.4 upper bounds this ratio by $O(\sqrt{n \log n \log\log n})$. We now show that it is lower bounded by $\sqrt{n}$.

The description of the integrality gap instance of MVP is as follows. Let $q = \sqrt{n}$. The set V of vertices consists of q groups of vertices. Group i has q vertices denoted by $v^{(i)}(1), v^{(i)}(2), \ldots, v^{(i)}(q)$. It is instructive to think that the groups are ordered, where the vertices of group i appear before those of group $i+1$. The MVP instance has $k = q^q$ paths: for every possible choice of $i_1, i_2, \ldots, i_q \in [q]$, there is a path

$$v^{(1)}(i_1) \to v^{(1)}(i_1+1) \to \cdots \to v^{(q)}(i_q) \to v^{(q)}(i_q+1)\,,$$

where we slightly abused notation and also identified $v^{(i)}(1)$ by $v^{(i)}(q+1)$.

Let E be the edges that appear in at least one of the paths in the constructed MVP instances. Define x by letting x_e have value $1/q$ for every $e \in E$ that go between vertices in the same group; the remaining edges receive x-value 0. It is easy to see that this definition of x satisfies (5) and (6). Indeed, the non-negativity (6) of x is immediate. To see that the cycle constraints (5) are satisfied, it is sufficient to observe that the only cycles in (V,E) are of the type

$$v^{(i)}(1) \to v^{(i)}(2) \to \cdots \to v^{(i)}(q) \to v^{(i)}(1)\,.$$

That is, they only contain edges between vertices of the same group (since all other edges in E go from a group of index i to the consecutive index $i+1$). Moreover, the length of the cycle is q and since x receives value $1/q$ on each of these edges the cycle constraints are satisfied. Finally, since each path P in the MVP instance contains exactly one edge per group, we have $\sum_{e\in P} x_e = 1$ and so the value of the LP solution x is 1.

We now complete the analysis of the integrality gap instance by showing that any integral solution to the MVP instance has value at least $\sqrt{n}$. Indeed, any total order $<_{\text{tot}}$ of V must violate at least one edge of the form $v^{(i)}(j) \to v^{(i)}(j+1)$, where for notational convenience we also identify $v^{(i)}(1)$ by $v^{(i)}(q+1)$. Select i_1 to be such an index corresponding to a reversed edge in the first group. Similarly, let $i_2, \ldots, i_q \in [q]$ be the indices of reversed edges in the remaining groups. Then the path defined for $i_1, i_2, \ldots, i_q \in [q]$ have at least $\sqrt{n}$ reversed edges and so $<_{\text{tot}}$ has value at least $\sqrt{n}$.

This completes the description and analysis of the integrality gap instance with a lower bound of $\sqrt{n}$. A final remark is that the number k of paths in the constructed instance of MVP is exponential in n. It is an interesting open problem whether it is always possible to find a total order $<_{\text{tot}}$ that is a $O(\text{polylog}(k,n))$ approximation. While this remains open, we give a bi-criteria algorithm with similar guarantees in the next subsection.

3.2. Polylogarithmic bi-criteria approximation

In this section, we give a bi-criteria algorithm that outputs a collection $\mathcal{O}$ of at most $\log_2(k)$ total orders with the guarantee: for any path P_i there is a total order $<_{\text{tot}} \in \mathcal{O}$ such that $\text{dist}(<_{\text{tot}}, P_i) = O(\log n \log\log n)$. Let

[1]We consider the interesting case when $T \geq 1$ as the problem becomes trivial when the optimal value is 0. Without this constraint, the linear program has a trivial integrality gap of n: take a cycle of length n where each edge belongs to a path by itself in the MVP instance. For this instance, $1/n$ is a feasible fractional solution of value $1/n$.

$\alpha = O(\log n \log\log n)$ be the approximation guarantee promised by Theorem 3.1.

Fix an MVP instance $V, P_1, \ldots, P_k$ and a solution x to LP-MVP of value T. As aforementioned, we can solve LP-MVP to optimality in polynomial time and so we may assume that T is at most the value of an optimal solution to the MVP instance. Algorithm 2 now initializes $\mathcal{P}$ to be the set of all paths. It then repeatedly proceeds as follows. In Step 3, it defines the edge-weights so that $w(e)$ equals the number of remaining paths in $\mathcal{P}$ that e belongs to. In Step 4, Algorithm 2 then uses Theorem 3.1 to find a $<_{\text{tot}}$ so that the total weight of reversed arcs (i.e., the total weight of the feedback arc set) is at most $\alpha \cdot \sum_e x_e w(e)$. It adds this order to the collection $\mathcal{O}$ and then remove all paths in $\mathcal{P}$ that have distance at most $2\alpha T$ to $<_{\text{tot}}$.

Algorithm 2 Bi-criteria Algorithm

```
Input: A fractional solution $x$ to LP-MVP of value $T$.
Output: A collection $\mathcal{O}$ of total orders of $V$.
1: Let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be the family of paths.
2: repeat
3:    Define edge-weights $w(e) = |\{P_i \in \mathcal{P} \mid e \in P_i\}|$.
4:    Apply Theorem 3.1 using $x$ and $w$ to obtain a total
      ordering $<_{\text{tot}}$ such that the total weight of reversed
      arcs is at most $\alpha \cdot \sum_e x_e w(e)$.
5:    Add $<_{\text{tot}}$ to $\mathcal{O}$.
6:    Remove $P_i \in \mathcal{P}$ from $\mathcal{P}$ if $\text{dist}(<_{\text{tot}}, P_i) \le 2\alpha T$.
7: until $\mathcal{P} \neq \emptyset$
```

From the description of the algorithm, it is clear that it only terminates when for every P_i there is a total ordering $<_{\text{tot}} \in \mathcal{O}$ such that $\text{dist}(<_{\text{tot}}, P_i) \le 2\alpha T$. This thus approximates the optimal cost within a $O(\log n \log\log n)$ factor if we start with an optimal solution x to the linear program. It remains to argue that the collection $\mathcal{O}$ is of cardinality at most $\log_2 k$.

Lemma 3.5. *Algorithm 2 outputs a collection $\mathcal{O}$ of total orders of cardinality at most $\log_2 k$.*

Proof. We prove the lemma by arguing that the cardinality of $\mathcal{P}$ is at least halved in each iteration.

The total ordering $<_{\text{tot}}$ returned by Theorem 3.1 is such that the total weight of the reversed arcs is at most $\alpha \cdot \sum_e x_e w(e)$. Now recall that $w(e)$ equals the number of paths in $\mathcal{P}$ containing e. Thus,

$$\alpha \cdot \sum_{e\in E} x_e w(e) = \alpha \cdot \sum_{e \in E} x_e \sum_{P_i \in \mathcal{P}} \mathbf{1}\{e \in P_i\}$$
$$= \alpha \cdot \sum_{P_i \in \mathcal{P}} \sum_{e \in P_i} x_e \le \alpha \cdot \sum_{P_i \in \mathcal{P}} T = \alpha \cdot |\mathcal{P}| T.$$

Similarly, if we let $F = \{(u,v) \in E : v <_{\text{tot}} u\}$ be the edges reversed by $<_{\text{tot}}$, we can rewrite the total weight of the reversed arcs as

$$\sum_{e \in F} w(e) = \sum_{e \in F} \sum_{P_i \in \mathcal{P}} \mathbf{1}\{e \in P_i\}$$
$$= \sum_{P_i \in \mathcal{P}} |P_i \cap F| = \sum_{P_i \in \mathcal{P}} \text{dist}(<_{\text{tot}}, P_i)\,.$$

Combining the above calculations gives $\sum_{P_i \in \mathcal{P}} \text{dist}(<_{\text{tot}}, P_i) \le \alpha \cdot |\mathcal{P}| T$. In other words, the average number of edge-reversals over the paths in $\mathcal{P}$ is αT. By Markov's inequality, we then have that at most half of the paths have more than $2\alpha T$ reversals. By Step 6 of Algorithm 2, this implies that $\mathcal{P}$ is halved in each iteration which yields the lemma. □

3.3. Implications on the running time of Bellman-Ford

In this section, we explain the implications of our LP-based algorithms on the running time of Bellman-Ford. Recall the setting outlined in the introduction: We are given a graph $G = (V, E)$, a source $s \in S$, and k weight functions $w^1, w^2, \ldots, w^k$ on the edges. It is instructive to think that these weight functions are drawn from a distribution of related weight functions. In particular, an improved running time will be possible by exploiting that the shortest path problem on the different weight functions are related. Formally this is captured as follows. For each $i \in [k]$, we define P_v^i as the shortest path from s to v under w^i. Now any ordering $<$ of V has a worst-case running time over the k instances that equals (see Lemma 2.1)

$$O\left(m \cdot \max_{i \in [k], v \in V} \text{dist}(<, P_v^i)\right),$$

where $m = |E|$. To find the ordering that minimizes the running time is thus exactly the MVP problem for the set of paths $\cup_{i\in[k]} \cup_{v \in V} P_v^i$. Plugging in Theorem 3.4 immediately yields the following speed-up of the the Bellman-Ford algorithm:

Corollary 3.6. *Given k shortest path instances $w^1, \ldots, w^k$ on a graph G, we can in polynomial-time find an ordering so that the running time of Bellman-Ford is at most $\tilde{O}(\sqrt{n})$ times the running time given by an optimal ordering.*

Here we use the $\tilde{O}(\cdot)$ notation to suppress logarithmic terms. The speed-up is obtained by limiting the number of iterations of Bellman-Ford and, for related instances, the above corollary leads to up to $\tilde{O}(1/\sqrt{n})$ fewer iterations compared to an implementation of Bellman-Ford that does not use a learned ordering.

While a $\tilde{O}(\sqrt{n})$ speed up is significant, we can achieve a much bigger reduction via our bi-criteria algorithm described in Section 3.2. This is obtained by using a clever idea that was already present in the early work of (Yen, 1970): to not keep the order fixed. While (Yen, 1970) alternated between two total orders to achieve an upper bound of $n/2$ iterations, we will alternate between $O(\log kn)$ orders and achieve the *optimal number of iterations up to logarithmic factors*. We remark that this implementation of Bellman-Ford will alternate between $O(\log kn)$ orders as the considered MVP instance contains at most kn paths, i.e., $\left|\cup_{i\in[k]} \cup_{v\in V} P_v^i\right| \leq kn$. More formally, given k shortest path instances $w^1, \ldots, w^k$ on a graph G, we run Algorithm 2 on the paths $\cup_{i\in[k]} \cup_{v\in V} P_v^i$ to obtain a family $\mathcal{O}$ of $O(\log kn)$ orders with the following guarantee:

$$\max_{i\in[k],v\in V} \min_{<\in\mathcal{O}} \operatorname{dist}(<, P_v^i) \leq O(\log n \log\log n)\mathrm{opt},$$

where opt is the value of $\max_{i\in[k],v\in V} \operatorname{dist}(<_{\mathrm{tot}}, P_v^i)$ for a total order $<_{\mathrm{tot}}$ that minimizes this expression. Now if we run Bellman-Ford where we alternate between these orders, then by the same argument as in the proof of Lemma 2.1 we have that the correct distance from s to v under weight function w^i is calculated in $O(\min_{<\in\mathcal{O}} \operatorname{dist}(<, P_v^i)) \leq O(\log n \log\log n)\mathrm{opt}$ alternations. We thus get the following corollary since one alternation takes time $O(m|\mathcal{O}|) = O(m \log kn)$.

Corollary 3.7. *Given k shortest path instances $w^1, \ldots, w^k$ on a graph G, we can in polynomial-time find $O(\log kn)$ orders so that the running time of Bellman-Ford that alternates between these orders is at most $O(\log(kn) \cdot \log n \log\log n)$ times the running time given by an optimal ordering.*

4. Hardness of Approximation

In this section we prove a strong hardness of approximation result assuming the so-called V Label Cover conjecture, introduced in (Brakensiek & Guruswami, 2021) in the similar spirit as the unique games conjecture. We remark that the definition of the V Label Cover conjecture is not needed to understand our reduction, and as it requires the introduction of a significant amount of additional notation, we refer the interested reader to Section 3.1 and Conjecture 3.1 of (Brakensiek & Guruswami, 2021) for the precise definitions.

Theorem 4.1. *Assuming the V Label Cover conjecture, it is hard to approximate the MVP problem within any constant factor. In particular, for any constant $c > 1$, it is hard to distinguish instances that have a solution of value 1 from those that have no solution of value less than c.*

The starting point of our hardness proof is a strong hardness result for coloring hypergraphs that was proved in (Brakensiek & Guruswami, 2021). Specifically, assuming the V Label Cover conjecture, they proved the following (see Theorem 1.2 in their paper). For every integer $d \geq 2$, given a d-uniform hypergraph G it is hard to distinguish between the following cases where $q = \lceil d + \sqrt{d} - 1/2 \rceil$:

- (Yes case:) There is a q-coloring of G such that the vertices in each hyperedge receive distinct colors.
- (No case:) In any q-coloring there is a monochromatic hyperedge.

Due to space constraints, we describe our reduction from the hypergraph coloring problem in Appendix B. The reduction allows us to translate the hardness of the hypergraph coloring problem to the MVP problem and we remark that any improvement in the hardness of the coloring, e.g. improved hardness factor or weakened assumption, would directly translate to an improved hardness result of the MVP problem.

5. Experiments

In this section we complement our theoretical analysis of the MVP problem along with the implications for speeding up the Bellman-Ford algorithm with an empirical evaluation, showing that substantial speed ups can be obtained.

We begin by describing the algorithms we use to compute a solution to MVP, then describe the synthetic and real world datasets along with the baselines we consider and finally describe the results.

5.1. Solving MVP in Practice

Recall that the MVP problem has a close connection the the Feedback Arc Set problem. While a rich theoretical literature exists for the latter, many of the theoretically optimal methods do not scale, and are outperformed by simple, near linear time heuristics.

Following the work of Simpson et al. (2016), we will adapt the *GreedyFas* method, originally due to Eades et al. (1993), which they showed to have the best performance. This algorithm greedily selects vertices to construct a linear arrangement of the vertices. The intuition behind the method is to move all nodes with low in-degree to the beginning of the list and those with low out-degree to the end. More specifically, the algorithm maintains two lists, s_1 and s_2, which are initialized to be empty. In every iteration, first all sources (nodes with in-degree 0) are removed and appended to s_1, and all sinks (nodes with out-degree 0) are removed and prepended to the list s_2. Finally, a node u, with the maximum difference between it's in-degree and out-degree is removed and appended to s_1. This loop repeats until all vertices are either in s_1 or s_2, at which point the two lists are concatenated together. We present the pseudocode in Algorithm 3.

We will use this linear time method since it is more efficient than the LP-based methods developed in the previous section, and as we will see it already leads to significant speed improvements.

Algorithm 3 GreedyFas(G)

```
1: Input: a directed graph $G$
2: Output: a permutation of the nodes in $G$
3: $s_1 \leftarrow \emptyset$, $s_2 \leftarrow \emptyset$.
4: while $G$ is not empty do
5:     while $G$ contains a source $u$ do
6:         $s_1 \leftarrow s_1 u$
7:         Remove $u$ from $G$
8:     end while
9:     while $G$ contains a sink $u$ do
10:        $s_2 \leftarrow u s_2$
11:        Remove $u$ from $G$
12:    end while
13:    Pick the vertex $u$ that minimize $d^-(u) - d^+(u)$
14:    $s_1 \leftarrow s_1 u$
15:    Remove $u$ from $G$
16: end while
17: Return $s_1 s_2$
```

5.2. Experimental Setup

In all of our experiments, we begin with four weighted and directed graphs G_1, G_2, G_3 and H. The graphs G_i will serve as a *training set* which we use to compute a solution to the MVP problem, and the graph H will represent the *test* where we evaluate the performance of this approach. As we will see, the more similar G_i and H are, the bigger the improvement we can obtain.

It remains to pick the starting node s. Observe, that if s has out-degree 0 then Bellman-Ford will terminate after one iteration no matter what ordering is used. To avoid such trivial cases, we begin by identifying a set of nodes, S in H that can reach at least half of the nodes in H via directed paths. We then select a node $s \in S$ uniformly at random. The choice of s induces shortest path trees in each of the training graphs, G_i, which then serve as input to the MVP problem. We then run Algorithm 3 to obtain a permutation on the nodes, and use that to evaluate the performance of Bellman-Ford in the graph H. Specifically, the input to Algorithm 3 is the graph formed by taking the union of the shortest path trees in G_1, G_2, and G_3.

Baselines We consider three kinds of baselines for evaluating our algorithms. First, we appeal to the work of Bannister & Eppstein and use a random ordering of the vertices. Recall that the authors proved that in this case the expected worst-case number of iterations is $|V|/2$. Interestingly we notice that in practice this heuristic works substantially better than what is predicted by the worst case bounds.

Second, for the real-world data which consists of temporal graphs (where each edge has an arrival time), we consider ordering nodes by the time they first appear in H. We call this ordering temporal ordering.

Third, we considering the ordering of used by the Bellman-Ford algorithm for the graphs G_1, G_2, and G_3. We notice that in practice the ordering on G_3 gets better performances and so we report only it in the experimental section. We call this ordering in this section BF-3 ordering.

Metrics The key metric we report is the number of iterations performed by Bellman-Ford under the different orders for vertex consideration. We remark that this metric is structural, and does not favor the efficiency of one implementation over another.

We will report the *relative improvement* in the number of iterations before convergence. Formally, let $BF(\pi)$ be the number of times that the Bellman-Ford algorithm scans the adjacency list of H before converging when using the ordering π. The relative improvement of an ordering π' over a baseline, π, is $100 \times \frac{BF(\pi) - BF(\pi')}{BF(\pi)}$.

Finally in all of our experiments we report the average improvement and the confidence interval over 100 runs of our experiment where in each iteration we pick a (possibly) different source.

5.3. Synthetic Data

We begin by studying the performance of our algorithm as a function of similarity between the training data (graphs G_1, G_2, G_3) and the test graph H. We generate a layered graph with $2,500$ layers of four nodes each. We add a fully connected bipartite graph between successive layers, and assign each edge a weight chosen uniformly at random from integers in the range $[-10, 10]$.

To artificially vary the similarity between graphs, in each experiment we randomly permute node identifiers in the last $x\%$ of the layers in each graph. As x increases from 0 to 100, the potential benefit of training data should fall substantially.

In Figure 1, we show the relative improvement as a function of x. As expected when $x = 0$, the improvement is near 100%, as the learned method finishes in one iteration. We see the gain decrease as the correlation between training and test data decreases. Notably, however, even when 90% of the layers are permuted, we still see a statistically significant improvement over the random baseline.

5.4. Real World Data

In this experiment we study the impact of our approach on real world data. In particular we consider four real-world temporal graphs from the Stanford Large Network Dataset

	Random Ordering	Temporal Ordering	BF-3 Ordering
CollegeMsg	$10.33\% \pm 3.07\%$	$6.07\% \pm 2.71\%$	$1.3\% \pm 1.34\%$
email-Eu-core	$24\% \pm 3.57\%$	$18.75\% \pm 3.94\%$	$3.17\% \pm 2.38\%$
MathOverflow	$21.27\% \pm 2.45\%$	$7.83\% \pm 2.58\%$	$0.85\% \pm 1.39\%$
AskUbuntu	$23.88\% \pm 2.61\%$	$7.15\% \pm 2.11\%$	$1.1\% \pm 1.29\%$

Table 1. Relative improvement of the MVPbased ordering compared to Random Ordering, Temporal Ordering and BF-3 Ordering on different real-world datasets.

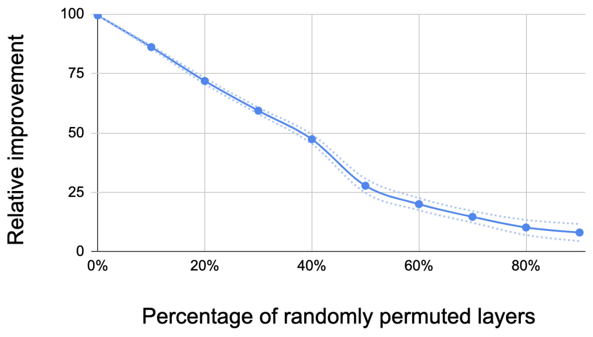

Figure 1. The relative improvement of the learned ordering compared to a random ordering as the correlation between G_1, G_2, G_3 and G_{test} decreases. The dotted line represents the confidence interval at 95% confidence level.

Collection[2]: CollegeMsg (Panzarasa et al., 2009), email-Eu-core (Paranjape et al., 2017), MathOverflow (Paranjape et al., 2017)and AskUbuntu (Paranjape et al., 2017). These datasets are diverse in their provenance and in their structure. The CollegeMsg has 1899 nodes and 20296 temporal edges, email-Eu-core has 986 nodes and 332334 temporal edges, MathOverflow has 24818 nodes and 506550 temporal edgesand AskUbuntu has 159316 nodes and 964437 temporal edges.

All of the datasets consist of a list of directed edges with timestamps. To generate the four graphs G_1, G_2, G_3 and H we order the edges temporally and then split into four contiguous groups with an equal number of edges.

In this experiment, in addition to the random order baseline, we consider two new baselines given by the temporal ordering in which nodes have been added to H and by the ordering in which the node have been explored by the Bellman-Ford algorithm in G_3. Intuitively, the last two ordering are likely to be a very good ordering for Bellman-Ford because it implicitly captures the temporal order of the graphs. We present the results in Table 1.

We observe that the learned ordering outperforms both baselines for all the datasets and in multiple cases with a significant margin. The fact that the relative improvement is lower for the temporal ordering and BF-3 ordering confirm our intuition that this ordering is stronger baseline than the random ordering for these problems.

[2]https://snap.stanford.edu/data/

6. Conclusion

In this work we showed how to find a good ordering of vertices to significantly improve the running time of the Bellman-Ford algorithm. We gave insights into the computational complexity of the problem and coupled it with experimental results showing the efficacy of relatively simple methods. Several interesting directions remain. First, the MVP problem can be seen as an instance of 1-center clustering under the particular dist functions between orderings. Fully resolving its complexity, as well as the natural extension to k-center are challenging open questions. The extension to k center can be seen as an instance of having multiple predictions, as introduced by Anand et al. (2022); Dinitz et al. (2022).

More broadly, this work continues a line of research showing how access to some historical data about problem instances can be rigorously analyzed to give additional improvement of algorithm performance. Extending this approach to other common algorithms is a fruitful avenue for future work.

References

Anand, K., Ge, R., Kumar, A., and Panigrahi, D. Online algorithms with multiple predictions. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 582–598. PMLR, 17–23 Jul 2022.

Bannister, M. J. and Eppstein, D. *Randomized Speedup of the Bellman–Ford Algorithm*, pp. 41–47. doi: 10.1137/1.9781611973020.6.

Bernstein, A., Nanongkai, D., and Wulff-Nilsen, C. Negative-weight single-source shortest paths in near-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO,* *USA, October 31 - November 3, 2022*, pp. 600–611. IEEE, 2022.

Bhaskara, A., Cutkosky, A., Kumar, R., and Purohit, M. Online learning with imperfect hints. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 822–831. PMLR, 2020. URL http://proceedings.mlr.press/v119/bhaskara20a.html.

Brakensiek, J. and Guruswami, V. The quest for strong inapproximability results with perfect completeness. *ACM Trans. Algorithms*, 17(3):27:1–27:35, 2021.

Chen, J. Y., Silwal, S., Vakilian, A., and Zhang, F. Faster fundamental graph algorithms via learned predictions. *CoRR*, abs/2204.12055, 2022. doi: 10.48550/arXiv.2204.12055. URL https://doi.org/10.48550/arXiv.2204.12055.

Dinitz, M., Im, S., Lavastida, T., Moseley, B., and Vassilvitskii, S. Faster matchings via learned duals. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 10393–10406, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/5616060fb8ae85d93f334e7267307664-Abstract.html.

Dinitz, M., Im, S., Lavastida, T., Moseley, B., and Vassilvitskii, S. Algorithms with prediction portfolios, 2022. URL https://arxiv.org/abs/2210.12438.

Eades, P., Lin, X., and Smyth, W. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323, 1993. doi: 10.1016/0020-0190(93)90079-O. URL https://researchrepository.murdoch.edu.au/id/eprint/27510/.

Even, G., Naor, J., Schieber, B., and Sudan, M. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998. doi: 10.1007/PL00009191. URL https://doi.org/10.1007/PL00009191.

Fox, J., Keevash, P., and Sudakov, B. Directed graphs without short cycles. *Comb. Probab. Comput.*, 19(2):285–301, 2010. doi: 10.1017/S0963548309990460. URL https://doi.org/10.1017/S0963548309990460.

Gollapudi, S. and Panigrahi, D. Online algorithms for rent-or-buy with expert advice. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2319–2327. PMLR, 2019. URL http://proceedings.mlr.press/v97/gollapudi19a.html.

Guruswami, V., Håstad, J., Manokaran, R., Raghavendra, P., and Charikar, M. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM J. Comput.*, 40(3):878–914, 2011.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pp. 489–504. ACM, 2018.

Lattanzi, S., Lavastida, T., Moseley, B., and Vassilvitskii, S. Online scheduling via learned weights. In Chawla, S. (ed.), *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pp. 1859–1877. SIAM, 2020. doi: 10.1137/1.9781611975994.114. URL https://doi.org/10.1137/1.9781611975994.114.

Lykouris, T. and Vassilvitskii, S. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1–24:25, 2021. doi: 10.1145/3447579. URL https://doi.org/10.1145/3447579.

Mahdian, M., Nazerzadeh, H., and Saberi, A. Allocating online advertisement space with unreliable estimates. In MacKie-Mason, J. K., Parkes, D. C., and Resnick, P. (eds.), *Proceedings 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11-15, 2007*, pp. 288–294. ACM, 2007.

Mitzenmacher, M. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, pp. 464–473, 2018.

Mitzenmacher, M. and Vassilvitskii, S. *Algorithms with Predictions*, pp. 646–662. Cambridge University Press, 2021. doi: 10.1017/9781108637435.037.

Panzarasa, P., Opsahl, T., and Carley, K. M. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, 2009.

Paranjape, A., Benson, A. R., and Leskovec, J. Motifs in temporal networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, pp. 601–610, 2017.

Seymour, P. D. Packing directed circuits fractionally. *Comb.*, 15(2):281–288, 1995.

Simpson, M., Srinivasan, V., and Thomo, A. Efficient computation of feedback arc set at web-scale. *Proceedings of the VLDB Endowment*, 10(3):133–144, 2016.

Svensson, O. Hardness of vertex deletion and project scheduling. *Theory Comput.*, 9:759–781, 2013.

Yen, J. Y. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27:526–530, 1970.

A. Stochastic Version

We bound the sample complexity and adapt our techniques to the stochastic version. In this setting, we are given a graph $G = (V, E)$, a source s, and a distribution D of weight functions on the edges. The goal is to find a permutation (or rather a small family of permutations) of the vertices so as to minimize the expected running time of the Bellman-Ford algorithm on the instance obtained by sampling the edge weights from D.

We use the following notation for a given weight function w on the edges of G. We let P_v^w denote a fixed shortest path from s to v under w. This is similar to the notation used in Section 3.3 and, as observed in that section, the running time of Bellman-Ford with ordering $<$ of V equals $O(m \cdot \max_{v\in V} \operatorname{dist}(<, P_v^w))$ and so the expected running time is $O(m)$ times

$$\mathbb{E}_{w\sim D}\left[\max_{v\in V} \operatorname{dist}(<, P_v^w)\right] .$$

We will achieve a near-optimal running time (up to polylogarithmic factors) by using the idea of alternating between a small (logarithmic-sized) family $\mathcal{O}$ of orders (as also done in Corollary 3.7). For such a family $\mathcal{O}$ of orderings, the Bellman-Ford algorithm that alternates between them, has a running time of $O(m \cdot |\mathcal{O}|)$ times

$$\mathbb{E}_{w\sim D}\left[\operatorname{Val}(\mathcal{O}, w)\right], \quad \text{where } \operatorname{Val}(\mathcal{O}, w) = \max_{v\in V} \min_{<\in\mathcal{O}} \operatorname{dist}(<, P_v^w)\,. \tag{3}$$

To see the claim, observe that the shortest path P_v^w from s to v is guaranteed to be calculated after the Bellman-Ford Algorithm has relaxed the vertices in the order of $\arg\min_{<\in\mathcal{O}} \operatorname{dist}(<, P_v^w)$ for $\min_{<\in\mathcal{O}} \operatorname{dist}(<, P_v^w)$ many iterations; and since we now alternate between $|\mathcal{O}|$ many orders, each iteration that goes through all orders takes time $m \cdot |\mathcal{O}|$.

Our problem is then to find a small family $\mathcal{O}$ so as to minimize (3). We first bound the sample complexity of this problem (Section A.1) and then obtain a good family of orderings (Section A.2).

A.1. Sample Complexity

As there is a finite number of permutations ($n!$ many), we get the following uniform convergence guarantees via a standard usage of concentration bounds and the union bound.

Lemma A.1. *Let $\mathcal{W}$ be the weight functions obtained by k independent samples of D. Fix $\varepsilon, \delta > 0$. If $k \geq n^4 \log n \log(1/\delta)/\varepsilon^2$, then with probability at least $1 - \delta$ the following holds*

$$(1-\varepsilon)\mathbb{E}_{w\sim D}[\operatorname{Val}(\mathcal{O}, w)] \leq \frac{1}{k}\sum_{w\in\mathcal{W}} \operatorname{Val}(\mathcal{O}, w) \leq (1+\varepsilon)\mathbb{E}_{w\sim D}[\operatorname{Val}(\mathcal{O}, w)]$$

for every family $\mathcal{O}$ of at most n permutations.

Proof. By definition, we have $0 \leq \operatorname{Val}(\mathcal{O}, w) \leq n-1$ for any $\mathcal{O}$ and w. Now if we fix any $\mathcal{O}$, $\sum_{w\in\mathcal{W}} \operatorname{Val}(\mathcal{O}, w)$ is a sum of k independent random variables taking values in $[0, n]$. Hence, Hoeffding's inequality implies

$$\Pr\left[\left|\frac{1}{k}\sum_{w\in\mathcal{W}} \operatorname{Val}(\mathcal{O}, w) - \mathbb{E}_{w\sim D}[\operatorname{Val}(\mathcal{O}, w)]\right| > \varepsilon\right] \leq 2\exp\left(-\frac{2k\varepsilon^2}{n^2}\right) .$$

There are at most $(n!) + (n!)^2 + \ldots + (n!)^n \leq n^{n^2}$ ways to select at most n permutations, i.e., possible values of $\mathcal{O}$. Thus, by the union bound, we get that the inequalities of the lemma are satisfied with probability at least

$$1 - n^{n^2} \cdot 2\exp\left(-\frac{2k\varepsilon^2}{n^2}\right)$$

which is greater than $1-\delta$ by the selection of k. □

A.2. Finding Good Orderings for Bellman-Ford

Equipped with Lemma A.1, if we form $\mathcal{W}$ by taking $k = \Theta(n^5)$ samples from D, then with overwhelming probability

$$\frac{1}{2}\mathbb{E}_{w\sim D}[\operatorname{Val}(\mathcal{O}, w)] \leq \frac{1}{k}\sum_{w\in\mathcal{W}} \operatorname{Val}(\mathcal{O}, w) \leq 2\mathbb{E}_{w\sim D}[\operatorname{Val}(\mathcal{O}, w)] \tag{4}$$

for every family $\mathcal{O}$ of at most n permutations. We now proceed to consider the computational problem of finding a set $\mathcal{O}$ so as to minimize $\frac{1}{k}\sum_{w\in\mathcal{W}} \mathrm{Val}(\mathcal{O}, w)$. Our approach is an adaptation of the bi-criteria rounding algorithm described in Section 3.2. We first adapt the linear program LP-MVP to the stochastic setting:

$$\begin{aligned}
\text{minimize} \quad & \sum_{w\in\mathcal{W}} T_w && \\
\text{subject to} \quad & \sum_{e\in P_v^w} x_e \le T_w, && \text{for every vertex } v\in V \text{ and weight function } w\in W,\\
& \sum_{e\in C} x_e \ge 1, && \text{for every cycle } C \text{ in } (V,E), \qquad (5)\\
& x_e \ge 0 && \text{for every edge } e\in E. \qquad (6)
\end{aligned}$$

As described in 3.1, the linear program relaxes the problem of finding a total ordering of the vertices V. Hence, assuming (4) holds, we have that the value of an optimal solution to the linear program is at most

$$2\mathbb{E}_{w\sim D}[\mathrm{Val}(<, w)]$$

for any permutation $<$ of the vertices. In other words, the value of the linear program is at most the value of an optimal permutation. We now adapt our bi-criteria algorithm (Algorithm 2) to this setting. Recall that $\alpha = O(\log n \log\log n)$ is the approximation guarantee promised by Theorem 3.1.

Algorithm 4 Stochastic Bi-criteria Algorithm

Input: A fractional solution x to the linear program of value $\sum_{w\in\mathcal{W}} T_w$.
Output: A collection $\mathcal{O}$ of total orders of V.

```
1: For $w \in \mathcal{W}$, let $\mathcal{P}_w = \{P_v^w \mid v \in V\}$ be the family of paths corresponding to weight function $w$.
2: repeat
3:     Define edge-weights $w(e) = \sum_{w\in\mathcal{W}} |\{P_v^w \in \mathcal{P}_w \mid e \in P_v^w\}|/T_w$.
4:     Apply Theorem 3.1 using $x$ and $w$ to obtain a total ordering $<_{\text{tot}}$ such that the total weight of reversed arcs is at most
       $\alpha \cdot \sum_e x_e w(e)$.
5:     Add $<_{\text{tot}}$ to $\mathcal{O}$.
6:     For $w \in \mathcal{W}$, remove $P_v^w \in \mathcal{P}_w$ from $\mathcal{P}_w$ if $\mathrm{dist}(<_{\text{tot}}, P_v^w) \le 2\alpha T_w$.
7: until $\mathcal{P} \ne \emptyset$
```

The analysis of the above algorithm is similar to that of Algorithm 2. Indeed, by definition, we have

$$\sum_{w\in\mathcal{W}} \mathrm{Val}(\mathcal{O}, w) \le 2\alpha \sum_{w\in\mathcal{W}} T_w$$

and we have the following version of Lemma 3.5:

Lemma A.2. *Algorithm 4 outputs a collection $\mathcal{O}$ of total orders of cardinality at most* $\log_2(kn)$.

Proof. Similarly to the proof of Lemma 3.5, we prove the statement by arguing that $\sum_{w\in\mathcal{W}} |\mathcal{P}_w|$ is at least halved in each iteration. The statement then follows since $\sum_{w\in\mathcal{W}} |\mathcal{P}_w|$ equals $|W|n = kn$ initially.

The total ordering $<_{\text{tot}}$ returned by Theorem 3.1 is such that the total weight of the reversed arcs is at most $\alpha\cdot\sum_e x_e w(e)$. By recalling the definition of $w(e)$ and using that x is a feasible solution to the linear program,

$$\begin{aligned}
\alpha\cdot\sum_{e\in E} x_e w(e) &= \alpha\cdot\sum_{e\in E} x_e \sum_{w\in\mathcal{W}} \sum_{P_v^w\in\mathcal{P}_w} \mathbf{1}\{e\in P_v^w\}/T_w\\
&= \alpha\cdot\sum_{w\in\mathcal{W}} \sum_{P_v^w\in\mathcal{P}_w} \sum_{e\in P_v^w} x_e/T_w \le \alpha\cdot\sum_{w\in\mathcal{W}} \sum_{P_v^w\in\mathcal{P}_w} 1 \le \alpha \sum_{w\in\mathcal{W}} |\mathcal{P}_w|\,.
\end{aligned}$$

Moreover, if we let $F = \{(u,v) \in E : v <_{\text{tot}} u\}$ be the edges reversed by $<_{\text{tot}}$, we can rewrite the total weight of the reversed arcs as

$$\begin{aligned}\sum_{e \in F} w(e) &= \sum_{e\in F} \sum_{w \in \mathcal{W}} \sum_{P_v^w \in \mathcal{P}_w} \mathbf{1}\{e \in P_v^w\}/T_w \\ &= \sum_{w \in \mathcal{W}} \sum_{P_v^w \in \mathcal{P}_w} |P_v^w \cap F|/T_w = \sum_{w \in \mathcal{W}} \sum_{P_v^w \in \mathcal{P}_w} \operatorname{dist}(<_{\text{tot}}, P_v^w)/T_w \,.\end{aligned}$$

Combining the above calculations gives

$$\sum_{w \in \mathcal{W}} \sum_{P_v^w \in \mathcal{P}_w} \operatorname{dist}(<_{\text{tot}}, P_v^w)/T_w \le \alpha \sum_{w \in \mathcal{W}} |\mathcal{P}_w| \,.$$

By Markov's inequality, we have that at most half of the paths P_v^w have $\operatorname{dist}(<_{\text{tot}}, P_v^w) > 2\alpha T_w$. By Step 6 of Algorithm 4, this implies that $\sum_{w \in \mathcal{W}} |\mathcal{P}_w|$ is halved in each iteration which yields the lemma. □

We summarize the implications on Bellman-Ford. Given a distribution D of weight functions w on a graph $G = (V, E)$ with source s, we proceed as follows:

1. Sample $k = \Theta(n^5)$ weight functions from D and let $\mathcal{W}$ be the obtained weights.
2. Use Algorithm 4 to obtain a family of $O(\log n)$ orderings $\mathcal{O}$ in polynomial-time with the guarantee
$$\sum_{w \in \mathcal{W}} \operatorname{Val}(\mathcal{O}, w) \le 2\alpha \sum_{w \in \mathcal{W}} T_w \,.$$
3. Run the Bellman-Ford Algorithm where we alternate between the orders in $\mathcal{O}$.

By Lemma A.1, we have with overwhelming probability (over the k samples) that

$$\mathbb{E}_{w\sim D} \operatorname{Val}(\mathcal{O}, w) \le 2\frac{1}{k} \sum_{w \in \mathcal{W}} \operatorname{Val}(\mathcal{O}, w) \le O(\alpha) \mathbb{E}_{w \sim D} \max_{v \in V} \operatorname{dist}(<_{\text{tot}}, P_v^w)$$

where $<_{\text{tot}}$ denotes an optimal ordering minimizing the expression of the right (i.e., the expected number of iterations of Bellman-Ford with an optimal permutation for D). Moreover, as a single alternation of all orders in $\mathcal{O}$ takes time $O(m|\mathcal{O}|) = O(m \log n)$ we have the following theorem by the same argument as for Corollary 3.7 (recall that $\alpha = O(\log n \log\log n)$).

Theorem A.3. *Given a distribution D of shortest path instances on a graph G and source s, we can in polynomial-time find $O(\log n)$ orders so that the expected running time of Bellman-Ford that alternates between these orders is at most $O(\log^n \log\log n)$ times the expected running time given by an optimal ordering for distribution D.*

B. Proof of Theorem 4.1

Recall that the starting point of our hardness proof is a strong hardness result for coloring hypergraphs that was proved by Brakensiek & Guruswami (2021). For convenience, we restate their result here. Assuming the V Label Cover conjecture, they proved (see Theorem 1.2 in their paper) the following. For every integer $d \ge 2$, given a d-uniform hypergraph G it is hard to distinguish between the following cases where $q = \lceil d + \sqrt{d} - 1/2 \rceil$:

- (Yes case:) There is a q-coloring of G such that the vertices in each hyperedge receive distinct colors.
- (No case:) In any q-coloring there is a monochromatic hyperedge.

We now describe our reduction from the hypergraph coloring problem. We then give the analysis and show that the above stated hardness of the coloring problem implies the theorem.

In our reduction we set d to equal the constant c in the theorem statement. Recall $q = \lceil d + \sqrt{d} - 1/2 \rceil$. Now starting with a d-uniform hypergraph $G = (V, E)$, we arbitrarily order the vertices V by $<$ and construct a MVP instance as follows:

- For each vertex $v \in V$, the MVP instance has q elements, referred to as $v(1), v(2), \ldots, v(q)$. For ease of notation we also refer to $v(1)$ as $v(q+1)$.
- For each hyperedge $e \in E$ with d vertices $v_1, v_2, \ldots v_d$ indexed so that $v_1 < v_2 < \cdots < v_d$ we construct q paths in our MVP instance. Specifically, for each color $i \in \{1, 2, \ldots, q\}$, the MVP instance has the path

$$\begin{aligned} &v_1(i) \to v_1(i+1) \to v_2(i) \to v_2(i+1) \to \cdots \\ &\cdots \to v_d(i) \to v_d(i+1)\,. \end{aligned}$$

This completes the description of the reduction. Observe that it clearly runs in polynomial time in the size of G since d and thus q is a constant. It remains to prove that it implies the stated hardness of the MVP problem. Specifically, we shall show that the constructed MVP instance has a solution of value 1 in the yes case whereas in the no-case any solution has value at least c. We start with the yes case, i.e., we assume that there is a coloring $\chi : V \to \{1, 2, \ldots, q\}$ of V such that the vertices in each hyperedge receive distinct colors. Using this coloring we define the total ordering $<_{\text{tot}}$ of the elements as follows. All elements corresponding to vertex u appear before those corresponding to vertex v in $<_{\text{tot}}$ if $u < v$. Further, the elements corresponding to vertex v are ordered as

$$v(\chi(v)+1) <_{\text{tot}} \cdots <_{\text{tot}} v(q) <_{\text{tot}} v(1) <_{\text{tot}} \cdots <_{\text{tot}} v(\chi(v))\,.$$

In words, the total order $<_{\text{tot}}$ satisfies $v(i) <_{\text{tot}} v(i+1)$ for all colors i except $\chi(v)$. Hence, by construction, any path P corresponding to hyperedge e and color i has $\mathrm{dist}(P, <_{\text{tot}})$ equal to the number of vertices in e with color i. This is at most 1 in the yes case and thus the MVP instance has a total order of value 1 in this case.

We proceed to prove that any solution has value at least d in the no case. Consider any total order $<_{\text{tot}}$ of the elements of the MVP instance and define a coloring $\chi : V \to \{1, \ldots, q\}$ by selecting an arbitrary color in

$$\{i \in \{1, \ldots, q\} \mid v(i+1) <_{\text{tot}} v(i)\}$$

for every vertex v. Note that the above set is empty as $<_{\text{tot}}$ cannot contain the cycle $v(1) \to \cdots \to v(q) \to v(q+1) = v(1)$. By the assumption of the no case, there must be a hyperedge e that is monochromatic with respect to χ. If we let i be the color that the vertices of e receives, then the path corresponding to e and color i has all edges reversed. This implies that $<_{\text{tot}}$ has value d.

We have thus proved that it is hard to distinguish whether the MVP instance has an optimal solution of value 1 or $d = c$ for any constant c. This completes the proof of the theorem.