On the Approximability of Time Disjoint Walks

Alexandre Bayen[1], Jesse Goodman[2(✉)], and Eugene Vinitsky[1]

[1] University of California, Berkeley, Berkeley, CA 94720, USA
{bayen,evinitsky}@berkeley.edu
[2] Cornell University, Ithaca, NY 14850, USA
jpmgoodman@cs.cornell.edu

Abstract. We introduce the combinatorial optimization problem Time Disjoint Walks. This problem takes as input a digraph G with positive integer arc lengths, and k pairs of vertices that each represent a *trip demand* from a source to a destination. The goal is to find a path and delay for each demand so that no two trips occupy the same vertex *at the same time*, and so that the sum of trip times is minimized. We show that even for DAGs with max degree $\Delta \leq 3$, Time Disjoint Walks is APX-hard. We also present a natural approximation algorithm, and provide a tight analysis. In particular, we prove that it achieves an approximation ratio of $\Theta(k/\log k)$ on bounded-degree DAGs, and $\Theta(k)$ on DAGs and bounded-degree digraphs.

Keywords: Hardness of approximation
Approximation algorithms · Disjoint Paths problem

1 Introduction

1.1 Related Work

Disjoint Paths is a classic problem in combinatorial optimization that asks: given an undirected graph G, and k pairs of vertices, do there exist vertex-disjoint paths that connect each pair? This problem captures the general notion of *connection without interference*, and has subsequently received much attention due to its applicability in areas like VLSI design [9,12] and communication networks [14,15].

These applications have motivated many variants of this basic problem. For example, one may choose the underlying graph to be undirected or directed, and the disjointness constraint to be over vertices or edges. As an optimization problem, one may consider the *maximum number of pairs* that can be connected with disjoint paths, the *minimum number of rounds* necessary to connect all pairs (where all paths in a round must be disjoint) [7], or the *shortest set of disjoint paths* to connect all pairs (if all pairs can, in fact, be disjointly connected) [8].

A few flavors of Disjoint Paths are tractable: for example, if k is fixed or G has bounded tree-width, then there exists a poly-time algorithm for finding

D. Kim et al. (Eds.): COCOA 2018, LNCS 11346, pp. 62–78, 2018.
https://doi.org/10.1007/978-3-030-04651-4_5

vertex-disjoint paths on undirected graphs [10,11]. Many interesting variants of Disjoint Paths are, however, extremely difficult. Indeed, finding vertex-disjoint paths on undirected graphs is one of Karp's NP-complete problems [6]. Furthermore, nearly-tight hardness results are known for finding the maximum set of edge-disjoint paths in a directed graph with m edges: there exists an $O(\sqrt{m})$-approximation algorithm [7], and it is NP-hard to approximate within a factor of $m^{1/2-\epsilon}$, for any $\epsilon > 0$ [5]. For detailed surveys on the complexity landscape of Disjoint Paths variants, see [7,8].

1.2 Contributions

Despite the great variety of Disjoint Paths problems that have been considered in the literature, it appears that little attention has been given to variants that relax the *disjointness* constraint, even though many natural applications do not always require paths to be completely disjoint. Consider, for example, the application of safely routing a collection of fully autonomous (and obedient) vehicles through an otherwise empty road network. In such a situation, we can certainly prevent collisions by routing all vehicles on disjoint paths. However, it is not difficult to see that if we have full control over the vehicles, using disjoint paths is rarely necessary (and, in fact, can be highly suboptimal).

Applications of this flavor motivate a new variant of Disjoint Paths, which roughly asks: given a graph G and k pairs of vertices that each represent a *trip demand*, how should we assign a delay and a path to each trip so that (1) trips are completed as quickly as possible, and (2) no two trips *collide* (i.e., occupy the same location at the same time). While there are problems in the literature (that do not wield the name "Disjoint Paths") that seemingly come close to capturing this goal, they exhibit some key differences. In particular, multicommodity flows over time [4,13] and job shop scheduling [3] seem, at first glance, very related to our problem. However, the former does not enforce unsplittable flows (as we require), and the latter does not capture the flexibility of scheduling job operations over any appropriate walk in a network.

As such, we are motivated to formalize and study this new variant of Disjoint Paths that relaxes the classical disjointness constraint to a "time disjointness" constraint. In particular, our contributions are threefold:

- We introduce a natural variant of Disjoint Paths, which we call *Time Disjoint Walks* (TDW). To the best of our knowledge, this is the first simple model that captures the notion of collision-free routing of *discrete objects* (i.e., instead of *flows*) over a shared network.
- We prove that Time Disjoint Walks is APX-hard, by providing an L-reduction from a variant of SAT. In fact, our reduction shows that this result holds even for directed acyclic graphs (DAGs) of max degree three ($\Delta \leq 3$).
- We describe an intuitive approximation algorithm for our problem, and provide a tight analysis: we show that it achieves an approximation ratio of $\Theta(k/\log k)$ on bounded-degree DAGs, and $\Theta(k)$ on DAGs and bounded-degree digraphs.

We formally introduce Time Disjoint Walks in Sect. 2. In Sect. 3 we provide some useful definitions regarding approximation. In Sect. 4 we prove our APX-hardness result. In Sect. 5 we describe our approximation algorithm, and provide bounds on its performance for the input classes mentioned above. In Sect. 6 we state our conclusions and present some open problems.

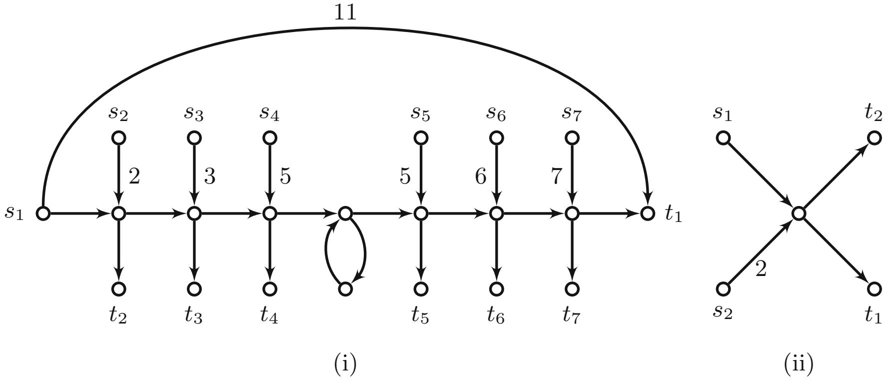

Fig. 1. (Unlabeled arcs have length 1): (i) A TDW instance with an optimal solution that contains cycles and intersecting walks, even though disjoint paths exist. (ii) A TDW instance with an obvious optimal solution, or a Shortest Disjoint Paths instance with no solution.

2 Time Disjoint Walks

We must first mention a few preliminaries: given $a, b \in \mathbb{Z}$, define $[a, b] := \{x \in \mathbb{Z} \mid a \leq x \leq b\}$, and for $b \in \mathbb{Z}$, we write $[b] := [1, b]$. Note that for $b < 1$, $[b] = \emptyset$. Given a directed graph (digraph) $G := (V, E)$, and $u, v \in V$, we define a *walk* W from u to v in G as a tuple $(w_1, w_2, \ldots, w_l)$ of vertices such that $w_1 = u, w_l = v$, and $(w_i, w_{i+1}) \in E$ for each $i \in [l-1]$. Note that a vertex can be repeated.

Given a digraph G with arc lengths $\lambda : E \to \mathbb{Z}_{\geq 1}$, and a walk $W = (w_1, w_2, \ldots, w_l)$ in G, we let $|W| := l$ denote the *cardinality* of the walk, and we define for every $j \in [l]$ the *length of the walk up to its* j^{th} *vertex* as

$$\lambda(W, j) := \sum_{i \in [j-1]} \lambda(w_i, w_{i+1}).$$

For convenience, we let $\lambda(W) := \lambda(W, l)$ denote the total length of the walk. Finally, given delays $d_1, d_2 \in \mathbb{Z}_{\geq 0}$ and walks W_1, W_2 in G, we say that (d_1, W_1) and (d_2, W_2) are *time disjoint* if, intuitively, a small object traversing W_1 at constant speed after waiting d_1 units of time does not collide/interfere with a small object traversing W_2 at the same speed after waiting d_2 units of time. We consider walks that have not departed, and walks that have already ended, to no longer exist on the network (and thereby not occupy any vertices). Formally, we have: for every $j_1 \in [|W_1|], j_2 \in [|W_2|]$ such that the j_1^{th} vertex of W_1 is equal to the j_2^{th} vertex of W_2,

$$d_1 + \lambda(W_1, j_1) \neq d_2 + \lambda(W_2, j_2).$$

We are now ready to formally define the problem examined in this paper:

Definition 1 (Time Disjoint Walks). *Let $G := (V, E)$ be a digraph, let $\lambda : E \to \mathbb{Z}_{\geq 1}$ define arc lengths, and let $\mathcal{T} := \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\} \subseteq V^2$ define a set of demands across unique vertices. For each $i \in [k]$, find a delay $d_i \in \mathbb{Z}_{\geq 0}$ and walk W_i from s_i to t_i such that the tuples in $\{(d_i, W_i) \mid i \in [k]\}$ are pairwise time disjoint, and $\sum_{i \in [k]}(d_i + \lambda(W_i))$ is minimized.*

We note that one can construct analogous problems by considering undirected graphs as input, edge lengths and delays that are real-valued, or a definition of *time disjoint* that requires large gaps between arrival times at common vertices (whereas the definition above simply requires a nonzero gap). Additionally, one may wish to consider a *min-max* objective instead of our *min-sum* objective.

We leave these variants to future work, noting that our primary goal in this paper is to study a basic flavor of this new combinatorial problem. Furthermore, our selection of this variant is well-motivated by our original application of routing a collection of identical autonomous vehicles over an empty road network (which, for the sake of this futuristic application, we may assume was built specifically for these vehicles). In particular, we may (1) model the road network as a directed graph, (2) assume that all routed vehicles traverse their walk at the same constant velocity, (3) measure road lengths as the time necessary to traverse it at that velocity, and (4) assume that road lengths are integer multiples of the time length of each vehicle. Additionally, we may motivate our *min-sum* objective by the desire to find a socially optimal solution.

Finally, we emphasize the novelty of our *time disjoint* constraint by comparing it to the standard *disjoint* constraint used in classical variants of Disjoint Paths. In particular, observe that if we modify the definition of Time Disjoint Walks to use the latter constraint instead of the former, we arrive at the (Min-Sum) Shortest Disjoint Paths problem [8]. However, this constraint makes all the difference: given an instance of Time Disjoint Walks, it is often the case that a solution under the standard *disjoint* constraint is suboptimal if examined under the *time disjoint* constraint. Indeed, the optimal solution under the latter constraint may even include paths that repeat vertices - hence the name Time Disjoint *Walks*; see (i) in Fig. (1). On the other hand, it is easy to construct an instance of Shortest Disjoint Paths that admits an obvious optimal solution under the time disjoint constraint, but does not yield any solution at all under the classical disjoint constraint; see (ii) in Fig. (1).

These observations strongly suggest that there is no simple reduction, in either direction, between Time Disjoint Walks and Disjoint Paths. Furthermore, using time-expanded networks [13] to reduce Time Disjoint Walks into Disjoint

Paths appears to offer little hope: such reductions will approximately square the size of the original graph, and many variants of Disjoint Paths are hard to approximate within $m^{1/2-\epsilon}$, for any $\epsilon > 0$ [5]. Thus, an approximation algorithm for Disjoint Paths, applied to a transformed Time Disjoint Walks instance, would likely fail to perform better than a trivial approximation algorithm for Time Disjoint Walks. These observations highlight the novelty of our problem and (in)approximability results.

3 Approximation Preliminaries

Given an optimization problem $\mathcal{P}$, we let $I_{\mathcal{P}}$ denote the instances of $\mathcal{P}$, $SOL_{\mathcal{P}}$ map each $x \in I_{\mathcal{P}}$ to a set of feasible solutions, and let $c_{\mathcal{P}}$ assign a real cost to each pair (x, y) where $x \in I_{\mathcal{P}}$ and $y \in SOL_{\mathcal{P}}(x)$. For $x \in I_{\mathcal{P}}$, we let $OPT_{\mathcal{P}}(x) := \min_{y^* \in SOL_{\mathcal{P}}(x)} c_{\mathcal{P}}(x, y^*)$ if $\mathcal{P}$ is a minimization problem, and $OPT_{\mathcal{P}}(x) := \max_{y^* \in SOL_{\mathcal{P}}(x)} c_{\mathcal{P}}(x, y^*)$ otherwise.

If $\mathcal{A}$ is a polynomial time algorithm with input $x \in I_{\mathcal{P}}$ and output $y \in SOL_{\mathcal{P}}(x)$, we say that $\mathcal{A}$ is a *ρ-approximation algorithm*, or *has approximation ratio ρ*, if $\mathcal{P}$ is a minimization problem and $c_{\mathcal{P}}(x, \mathcal{A}(x))/OPT_{\mathcal{P}}(x) \leq \rho$, or $\mathcal{P}$ is a maximization problem and $OPT_{\mathcal{P}}(x)/c_{\mathcal{P}}(x, \mathcal{A}(x)) \leq \rho$, for all $x \in I_{\mathcal{P}}$. Note that $\rho \geq 1$.

The class *APX* contains all optimization problems that admit a ρ-approximation algorithm, for *some* constant $\rho > 1$. An optimization problem is said to be *APX-hard* if every problem in APX can be reduced to it through an approximation-preserving reduction. One reduction of this type is the *L-reduction*:

Definition 2 (L-Reduction). *An L-reduction from an optimization problem $\mathcal{P}$ to an optimization problem $\mathcal{Q}$, denoted $\mathcal{P} \leq_L \mathcal{Q}$, is a tuple (f, g, α, β), where:*

- *For each $x \in I_{\mathcal{P}}$, $f(x) \in I_{\mathcal{Q}}$ and can be computed in polynomial time.*
- *For each $y \in SOL_{\mathcal{Q}}(f(x))$, $g(x, y) \in SOL_{\mathcal{P}}(x)$ and can be computed in polynomial time.*
- *α is a positive real constant such that for each $x \in I_{\mathcal{P}}$,*

$$OPT_{\mathcal{Q}}(f(x)) \leq \alpha \cdot OPT_{\mathcal{P}}(x).$$

- *β is a positive real constant such that for each $x \in I_{\mathcal{P}}, y \in SOL_{\mathcal{Q}}(f(x))$,*

$$\big|OPT_{\mathcal{P}}(x) - c_{\mathcal{P}}(x, g(x, y))\big| \leq \beta \cdot \big|OPT_{\mathcal{Q}}(f(x)) - c_{\mathcal{Q}}(f(x), y)\big|.$$

If a problem is APX-hard, it is NP-hard to ρ-approximate for some constant $\rho > 1$; thus, showing APX-hardness is strictly stronger than showing NP-hardness. To show APX-hardness, one can simply L-reduce from a known APX-hard problem. We refer the reader to [1] for a good reference on approximation.

4 Hardness of Approximation

To show the hardness of our problem, we show an L-reduction from MAX-E2SAT(3), which is known to be APX-hard [2]. We remind the reader of the definition, below, and then proceed with our proof.

Definition 3 (MAX-E2SAT(3)). *Let ϕ be a CNF formula in which (i) each clause contains exactly two literals on distinct variables, and (ii) each variable appears in at most three clauses. Find a truth assignment to the variables in ϕ that maximizes the number of satisfied clauses.*

Theorem 1. *Time Disjoint Walks is APX-hard, even for DAGs with $\Delta \leq 3$.*

Proof. We let $\mathcal{P}$:= MAX-E2SAT(3), $\mathcal{Q}$:= TDW with instances restricted to those containing DAGs with $\Delta \leq 3$, and show that $\mathcal{P} \leq_L \mathcal{Q}$. Below, we describe our L-reduction (f, g, α, β).

Description of f: Given an instance $\phi \in I_{\mathcal{P}}$ with n variables and m clauses, we let $X := \{x_1, \ldots, x_n\}$ refer to its variables and $\mathcal{C} := \{C_1, \ldots, C_m\}$ refer to its clauses. We let $L := \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$ refer to its literals. For convenience, we define $e : L \to X$ that extracts the variable from a given literal; i.e., $e(x_i) = e(\overline{x}_i) = x_i$. We label the literals in clause C_j as l_j^1, l_j^2. For each $l \in L$, we let $S_l := \{l_j^a \mid a \in [2], j \in [m], l_j^a = l\}$ capture all occurrences of literal l in ϕ. Finally, for each $l \in L$, we define an arbitrary bijection $\pi_l : S_l \to [|S_l|]$ to induce an ordering on S_l. We will let π_l^{-1} denote its inverse: i.e., $\pi_l^{-1}(1)$ is the first element in S_l in the order induced by π_l.

We may now describe f, which constructs an instance $(G, \lambda, \mathcal{T}) \in I_{\mathcal{Q}}$ from ϕ. We start with the construction of G (see Fig. (2)), which closely follows the standard proof of NP-hardness for Disjoint Paths: for each clause $C_j = (l_j^1 \vee l_j^2)$ in ϕ, we create a new *clause gadget* and add it to G. That is, for each clause C_j, we add the following vertex and arc set to our construction:

$$V_{C_j} := \{c_j^s, l_j^1, l_j^2, l_j^{1'}, l_j^{2'}, c_j^t\}$$
$$E_{C_j} := \{(c_j^s, l_j^1), (c_j^s, l_j^2), (l_j^1, l_j^{1'}), (l_j^2, l_j^{2'}), (l_j^{1'}, c_j^t), (l_j^{2'}, c_j^t)\}$$

Next, for each $x_i \in X$, we add an *interleaving variable gadget* as follows: first, we add two vertices x_i^s, x_i^t to $V(G)$. Then, we wish to create exactly two directed paths (walks), $W_{x_i}^+, W_{x_i}^-$, from x_i^s to x_i^t: we want $W_{x_i}^+$ to travel through all vertices corresponding to positive literals of x_i, and $W_{x_i}^-$ to travel through all vertices corresponding to negative literals of x_i. Formally, for each $l \in \{x_i, \overline{x}_i\}$, we create a path from x_i^s to x_i^t as follows. First, if $|S_l| = 0$, we add arc (x_i^s, x_i^t) to $E(G)$. Otherwise, we add arcs $(x_i^s, \pi_l^{-1}(1))$ and $((\pi_l^{-1}(|S_l|))', x_i^t)$ to $E(G)$, and then for each $j \in [|S_l| - 1]$, we add arc $((\pi_l^{-1}(j))', \pi_l^{-1}(j+1))$. Note that the prime symbols are merely labels, and are used in our construction to ensure that the max degree of G remains at most three. This completes our construction of G. We now define a set of $n + m$ demands, where each corresponds to a variable or a clause:

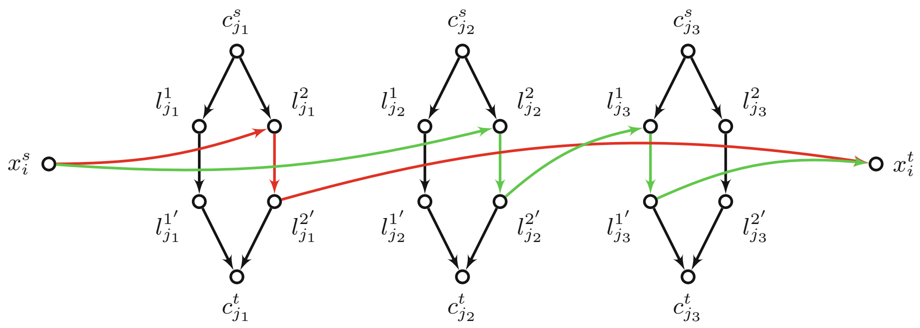

Fig. 2. An interleaving variable gadget (and its affiliated clause gadgets) corresponding to a variable with one negative occurrence (red path) and two positive occurrences (green path). (Color figure online)

$$\mathcal{T} := \{(x^s_i, x^t_i) \mid i \in [n]\} \cup \{(c^s_j, c^t_j) \mid j \in [m]\}$$

Finally, we must define arc lengths $\lambda : E \to \mathbb{Z}_{\geq 1}$. We will do this in a way that for each $j \in [m], a \in [2]$, we have $\lambda(W_{c^s_j}, l^a_j) = \lambda(W_{x^s_i}, l^a_j)$, where $W_{c^s_j}$ is the unique walk in G from c^s_j to l^a_j, and $W_{x^s_i}$ is the unique walk in G from $x^s_i = e(l^a_j)^s$ to l^a_j. Call this property $(*)$. To facilitate our analysis, we will also want every demand-satisfying path in G to have the same length.

Since ϕ is an instance of MAX-E2SAT(3), we know that for each $i \in [n]$, each of the two paths between x^s_i and x^t_i passes through at most 3 clause gadgets. Thus, by our construction, each such path includes at most 7 arcs, and any path from a variable x^s_i to some literal l^a_j with $x_i = e(l^a_j)$ can use at most 5 arcs. Thus, we can successfully force each demand-satisfying path in G to have length 7 while maintaining property $(*)$ by defining $\lambda : E \to \mathbb{Z}_{\geq 1}$ as follows, completing our construction of $(G, \lambda, \mathcal{T}) \in I_{\mathcal{Q}}$:

$$\lambda(u,v) := \begin{cases} 1, & \textbf{if } (u,v) = (l^a_j, l^{a'}_j), j \in [m], a \in [2];\ \textbf{or} \\ & \textbf{if } (u,v) = (x^s_i, l^a_j), i \in [n], j \in [m], a \in [2];\ \textbf{or} \\ & \textbf{if } (u,v) = (l^{a'}_h, l^b_j), h, j \in [m], a, b \in [2]; \\ 7, & \textbf{if } (u,v) = (x^s_i, x^t_i), i \in [n] \\ 7 - 2|S_l|, & \textbf{if } (u,v) = (l^{a'}_j, x^t_i), j \in [m], a \in [2], i \in [n], l^a_j = l \\ 2\pi_l(l^a_j) - 1, & \textbf{if } (u,v) = (c^s_j, l^a_j), j \in [m], a \in [2], l^a_j = l \\ 7 - 1 - \lambda(c^s_j, l^a_j), & \textbf{if } (u,v) = (l^{a'}_j, c^t_j), j \in [m], a \in [2] \end{cases}$$

Description of g: Given a solution $y \in SOL_{\mathcal{Q}}(f(\phi))$, we construct a solution $g(\phi, y) \in SOL_{\mathcal{P}}(\phi)$ through two consecutive transformations: z, followed by q. That is, we will define transformations z and q such that g is the composition $g(\phi, y) := q(\phi, z(y))$.

We define z to transform solution y into another solution $y' \in SOL_{\mathcal{Q}}(f(\phi))$ such that $c_{\mathcal{Q}}(f(\phi), y') \leq c_{\mathcal{Q}}(f(\phi), y)$ and such that y' assigns 0 delay to demands associated with interleaving variable gadgets. To accomplish this, recall that $y = \{(d_1, W_1), \ldots, (d_{n+m}, W_{n+m})\}$, by definition of $SOL_{\mathcal{Q}}$. Without loss of generality, we may assume tuples indexed with $[n]$ correspond to demands on interleaving variable gadgets, and tuples indexed with $[n+m] \setminus [n]$ correspond to demands on clause gadgets.

Now, while there exists some $i \in [n]$ such that $d_i > 0$ (and thus $d_i \geq 1$), we perform the following modification on y: first, we reset W_i to be the path traveling through at most one clause gadget - the positive or negative path *must* have this property, because each variable appears in ϕ at most three times, by definition of MAX-E2SAT(3). Now, reset d_i to 0. If W_i shares a vertex with another walk W_j, we know $j \in [n+m] \setminus [n]$, by construction of G. In this case, reset d_j to 1 if and only if d_j is currently 0. By construction of λ, the walks remain time disjoint and the cost of the solution does not increase.

In the second transformation, q, we transform modified solution y' into an assignment $(A : X \rightarrow \{T, F\}) \in SOL_{\mathcal{P}}(\phi)$ as follows: for each $i \in [n]$, set $A(x_i) = T$ if and only if W_{x_i}, the walk from x_i^s to x_i^t, takes the negative literal path.

Valid value for α: We will show that for $\alpha = 29$, $OPT_{\mathcal{Q}}(f(\phi)) \leq \alpha \cdot OPT_{\mathcal{P}}(\phi)$. To see this, we make two observations. First observation: if $A : X \rightarrow \{T, F\}$ is a truth assignment for ϕ, then we can construct a solution to $f(\phi)$ as follows: for each $i \in [n]$, connect demand (x_i^s, x_i^t) using the negative literal path if $A(x_i) = T$, and the positive literal path if $A(x_i) = F$. Either way, assign a delay of 0. Then, for each $j \in [m]$ where clause C_j is satisfied by assignment A, connect demand (c_j^s, c_j^t) using a walk that goes through a literal that evaluates to true under A. Assign a delay of 0 to this demand. For each clause C_j that isn't satisfied by A, select an arbitrary walk to complete the corresponding demand (c_j^s, c_j^t). Assign a delay of 1 to this demand. It is clear that this is a valid solution to $f(\phi)$. Furthermore, the cost of our solution is $7(n+m) + U(A, \phi)$, where $U(A, \phi)$ is the number of clauses in ϕ unsatisfied by A. Second observation: by linearity of expectation, if ϕ is an instance of MAX-E2SAT(3), then there must exist an assignment $A : X(\phi) \rightarrow \{T, F\}$ that satisfies at least 3/4 of the clauses.

We may now prove the desired inequality for $\alpha = 29$. From our first observation and the fact that $n \leq 2m$ (since each of the m clauses has 2 literals),

$$OPT_{\mathcal{Q}}(f(\phi)) \leq 7(n+m) + (m - OPT_{\mathcal{P}}(\phi)) \leq 22m - OPT_{\mathcal{P}}(\phi). \tag{1}$$

Now, by our second observation, we know $OPT_{\mathcal{P}}(\phi) \geq 3m/4$. Thus, we have:

$$OPT_{\mathcal{Q}}(f(\phi)) \leq 22 \cdot (4/3) \cdot OPT_{\mathcal{P}}(\phi) - OPT_{\mathcal{P}}(\phi) \leq 29 \cdot OPT_{\mathcal{P}}(\phi).$$

Valid value for β: We will show that for $\beta = 1$ and any $y \in SOL_{\mathcal{Q}}(f(\phi))$, $\big(OPT_{\mathcal{P}}(\phi) - c_{\mathcal{P}}(\phi, g(\phi, y))\big) \leq \beta \cdot \big(c_{\mathcal{Q}}(f(\phi), y) - OPT_{\mathcal{Q}}(f(\phi))\big)$, as required. As a first step, we recall that transformations z, q define g, and let γ denote the number of clause gadget demands assigned a delay of 0 by solution $z(y)$ to $f(\phi)$. We make the following crucial claim:

$$c_{\mathcal{P}}(\phi, g(\phi, y)) := c_{\mathcal{P}}(\phi, q(\phi, z(y))) \geq \gamma. \tag{2}$$

To see this, note the following: by construction, $z(y)$ is a valid solution to $f(\phi)$. Thus, if $z(y)$ assigns clause gadget demand (c_j^s, c_j^t) a delay $d_j = 0$ and walk W_j that passes through literal l, then l is a positive literal if and only if the walk selected for the interleaving variable gadget demand (x_i^s, x_i^t) (where $x_i = e(l)$) does not travel through the positive literals of x_i. By definition of q, this occurs if and only if $g(\phi, y)$ assigns *true* to x_i. Thus, a clause gadget demand given 0 delay by $z(y)$ corresponds to a clause in ϕ satisfied by $g(\phi, y)$, thus proving (2).

Next, by definition of γ and z, we have:

$$7(n+m) + (m-\gamma) \leq c_{\mathcal{Q}}(f(\phi), z(y)) \leq c_{\mathcal{Q}}(f(\phi), y). \tag{3}$$

Combining inequalities (2) and (3), we get:

$$c_{\mathcal{P}}(\phi, g(\phi, y)) \geq \gamma \geq 7n + 8m - c_{\mathcal{Q}}(f(\phi), y). \tag{4}$$

Finally, using the leftmost inequality in (1) along with inequality (4) gives us:

$$\begin{aligned} OPT_{\mathcal{P}}(\phi) - c_{\mathcal{P}}(\phi, g(\phi, y)) &\leq \big(7n + 8m - OPT_{\mathcal{Q}}(f(\phi))\big) - \big(7n + 8m - c_{\mathcal{Q}}(f(\phi), y)\big) \\ &= \beta \cdot \big(c_{\mathcal{Q}}(f(\phi), y) - OPT_{\mathcal{Q}}(f(\phi))\big), \end{aligned}$$

for $\beta = 1$, as desired. This completes the proof that (f, g, α, β) is a valid L-reduction, and subsequently that TDW on DAGs with $\Delta \leq 3$ is APX-hard. □

5 Approximation Algorithm

5.1 Algorithm

We present Algorithm 1, which approximates TDW by finding shortest paths to satisfy each demand, and then greedily assigning delays to each trip (with priority given to shorter trips). To simplify notation, *we assume that the inputted terminal pairs are ordered by nondecreasing shortest path length* (if not, we may simply sort the indices after finding the shortest demand-satisfying paths). The algorithm clearly runs in poly$(|V|, |E|, k)$ time, and the bad_delay variables ensure its correctness. Next, we briefly note the following easy bound:

Proposition 1. *Algorithm 1 has an approximation ratio of $O(k)$ on general digraphs.*

Proof. Let $x := (G, \lambda, \mathcal{T}) \in I_{TDW}$, and let $\mathcal{A}(x) \in SOL_{TDW}$ be the output of Algorithm 1 on x. First, we show by induction that for each $i \in [k]$,

$$d_i \leq 2 \sum_{h \in [i-1]} \lambda(W_h).$$

Algorithm 1. Shortest paths & greedy delays, with priority to shorter paths.

Input: $x := (G := (V, E), \lambda : E \to \mathbb{Z}_{\geq 1}, \mathcal{T} := \{(s_1, t_1), \ldots, (s_k, t_k)\}) \in I_{TDW}$
Output: $y \in SOL_{TDW}(x)$

$y \leftarrow \{\}$
$\triangleright$ Get shortest paths and dummy delays:
for $i \in [k]$ **do**
  $W_i \leftarrow \mathsf{Dijkstra}(G, \lambda, s_i, t_i)$
  $d_i \leftarrow 0$
  $y \leftarrow y \cup (d_i, W_i)$
end for
$\triangleright$ Greedily assign delays, with priority given to shorter paths:
for $i \in [k]$ **do**
  $\mathsf{bad\_delays}_i \leftarrow \{\}$
  for $h \in [i-1]$ **do**
    $\mathsf{bad\_delays}_{i,h} \leftarrow \{\}$
    for $v \in W_h \cap W_i$ **do**
      $\mathsf{bad\_delay} \leftarrow (d_h + \lambda(W_h, v) - \lambda(W_i, v))$
      $\mathsf{bad\_delays}_{i,h} \leftarrow \mathsf{bad\_delays}_{i,h} \cup \{\mathsf{bad\_delay}\}$
    end for
    $\mathsf{bad\_delays}_i \leftarrow \mathsf{bad\_delays}_i \cup \mathsf{bad\_delays}_{i,h}$
  end for
  $d_i \leftarrow \min(\mathbb{Z}_{\geq 0} \setminus \mathsf{bad\_delays}_i)$
end for
return y

For the base case $i = 1$, note that $\mathsf{bad\_delays}_1 = \emptyset$ and so $d_1 = 0$. For $i > 1$, first observe that by definition of $\mathsf{bad\_delay}$, we have $d_i \leq 1 + \max_{h \in [i-1]}(d_h + \lambda(W_h))$. Thus,

$$\begin{aligned} d_i &\leq 1 + \max_{h \in [i-1]} \left(2 \sum_{h' \in [h-1]} \lambda(W_{h'}) + \lambda(W_h) \right) && \text{(induction hypothesis)} \\ &\leq 1 + 2 \sum_{h' \in [i-2]} \lambda(W_{h'}) + \lambda(W_{i-1}) && \text{(pick } h = i-1\text{)} \\ &\leq 2 \sum_{h' \in [i-1]} \lambda(W_{h'}), && \text{(trips have length} \geq 1\text{)} \end{aligned}$$

completing the induction. Now, recallling that our algorithm uses the shortest paths to satisfy each demand, and that it assigns delays to shorter paths first, we can bound the approximation ratio as follows:

$$\begin{aligned} \rho \leq \frac{c_{TDW}(x, \mathcal{A}(x))}{OPT_{TDW}(x)} &\leq \frac{\sum_{i \in [k]}(d_i + \lambda(W_i))}{\sum_{i \in [k]} \lambda(W_i)} \leq 1 + \frac{2 \sum_{i \in [k]} \sum_{h \in [i-1]} \lambda(W_h)}{\sum_{i \in [k]} \lambda(W_i)} \\ &\leq 1 + \frac{2k \sum_{i \in [k]} \lambda(W_i)}{\sum_{i \in [k]} \lambda(W_i)} = O(k). \end{aligned}$$ $\square$

5.2 Analysis on Bounded-Degree DAGs

We now show that our algorithm is able to achieve a better approximation ratio on bounded-degree DAGs. In what follows, we call a directed graph a "$(2, l)$-in-tree" if it is a perfect binary tree of depth l, in which every arc points toward the root. Analogously, a "$(2, l)$-out-tree" is a perfect binary tree of depth l, in which every arc points away from the root.

Theorem 2. *Algorithm 1 achieves an approximation ratio of $\Theta(k/\log k)$ on bounded-degree DAGs.*

Proof. Upper bound: Let $x := (G, \lambda, \mathcal{T}) \in I_{TDW}$ such that G is a DAG. Let $\mathcal{A}(x) \in SOL_{TDW}$ be the output of Algorithm 1 on x. In what follows, we will justify the following string of inequalities that proves the upper bound:

$$\begin{aligned}\rho \leq \frac{c_{TDW}(x, \mathcal{A}(x))}{OPT_{TDW}(x)} &\leq_{(1)} \frac{\sum_{i\in[k]}\left(d_i + \lambda(W_i)\right)}{\sum_{i\in[k]}\lambda(W_i)} \\ &\leq_{(2)} 1 + \frac{d_{i^*}}{\lambda(W_{i^*})}, i^* := \max_{i\in[k]}\left(\frac{d_i}{\lambda(W_i)}\right) \\ &\leq_{(3)} 1 + O(1)\cdot\frac{d_{i^*}}{\log d_{i^*}} \\ &\leq_{(4)} 1 + O(1)\cdot\frac{k}{\log k} = O(k/\log k).\end{aligned}$$

Inequality (1) is clear, because our algorithm takes the shortest path to satisfy each demand. **Inequality (2)** follows (by induction) from the following general observation: given $d_1, d_2 \in \mathbb{Z}_{\geq 0}$ and $\lambda_1, \lambda_2 \in \mathbb{Z}_{\geq 1}$, observe $d_1/\lambda_1 \leq d_2/\lambda_2 \implies (d_1+d_2)/(\lambda_1+\lambda_2) \leq d_2/\lambda_2$, and thus $(d_1+d_2)/(\lambda_1+\lambda_2) \leq \max(d_1/\lambda_1, d_2/\lambda_2)$.

To show **inequality (3)**, we need two observations. We first observe that for each $i \in [k]$:

$$d_i \leq |\mathsf{bad\_delays}_i| \leq |\{h \in [i-1] \mid W_h \cap W_i \neq \emptyset\}| =: \mu_i$$

To see this, suppose for contradiction that there exists some $h \in [i-1]$ with $W_h \cap W_i \neq \emptyset$ and $|\mathsf{bad\_delays}_{i,h}| > 1$. Then, by definition of $\mathsf{bad\_delay}$, there exist vertices $u, v \in W_h \cap W_i$ and delays $\delta_u \neq \delta_v \in \mathbb{Z}_{\geq 0}$ such that:

$$\begin{aligned}\delta_u + \lambda(W_i, u) &= d_h + \lambda(W_h, u), \\ \delta_v + \lambda(W_i, v) &= d_h + \lambda(W_h, v), \\ \lambda(W_h, u) - \lambda(W_h, v) &= \lambda(W_i, u) - \lambda(W_i, v) + (\delta_u - \delta_v),\end{aligned}$$

where the last equality follows from the first two. But because $\delta_u \neq \delta_v$, this implies that the length of the path that W_h and W_i use to travel between u and v is not the same. Because G is a DAG, W_h and W_i must visit u and v in the same order, implying that one of these walks is not taking the shortest path from u to v, which contradicts the definition of the algorithm. Because $|\mathsf{bad\_delays}_{i,h}| = 0$ if $W_h \cap W_i = \emptyset$, we have $d_i \leq \mu_i$.

Next, we observe that:

$$\mu_i \leq \min(\Delta^{4\lambda(W_i)}, k).$$

Showing $\mu_i \leq k$ is trivial, by definition of μ_i and because $i \in [k]$. To show $\mu_i \leq \Delta^{4\lambda(W_i)}$, first note that in a digraph with max degree Δ, the number of paths that (i) have z arcs, (ii) start at distinct vertices, and (iii) all end at a common vertex, is upper bounded by Δ^z (this is easy to show by induction on z). Thus, the number of paths with $\leq z$ arcs, in addition to properties (ii) and (iii), is upper bounded by $\sum_{l=0}^{z} \Delta^l \leq \Delta^{z+1}$, for $\Delta > 1$. Call this lemma ($*$).

Now, note that for each $h \in [i-1]$ we may consider each W_h to terminate once it first hits a vertex in W_i (i.e., cut off all vertices that are hit afterwards) without changing the value of μ_i. Now, recall the following facts about our problem and algorithm: (I) each inputted demand has a unique source; (II) each edge in our digraph has length ≥ 1; (III) for all $h \in [i-1]$, $\lambda(W_h) \leq \lambda(W_i)$. Thus, by (III) and lemma ($*$), each vertex in W_i can be hit by at most $\Delta^{\lambda(W_i)+1}$ walks in $\{W_1, \ldots, W_{i-1}\}$. Furthermore, (II) tells us that the number of vertices in W_i is no more than $\lambda(W_i) + 1$. Thus, recalling that our problem statement ensures $\Delta > 1$, no demands have the same source and destination, and (II), we see that

$$\mu_i \leq (\lambda(W_i) + 1)\Delta^{\lambda(W_i)+1} \leq \Delta^{2(\lambda(W_i)+1)} \leq \Delta^{4\lambda(W_i)},$$

as desired. We now note that we may assume d_i is greater than any constant (otherwise, inequality (2) automatically proves a constant approximation ratio, completing the proof). Thus, from this and the above observations, we have $\log(d_i) \leq 4\lambda(W_i)\log(\Delta)$. This proves inequality (3), because our graph has bounded degree.

Inequality (4) is not difficult: as stated above, we will always have $d_i \leq k$, and we may always assume $d_i \geq 3$. Basic calculus shows the function $x/\log x$ increases over $x \geq 3$.

Lower bound: We show $\forall l \in \mathbb{N}_{\geq 1}, k := 2^l$, $\exists (G_k, \lambda_k, \mathcal{T}_k) \in I_{TDW}$ such that G_k is a bounded-degree DAG and Algorithm 1 achieves an approximation ratio of $\Omega(k/\log k)$. Construct G_k by taking a $(2,l)$-in-tree A_S and a $(2,l)$-out-tree A_T. Draw an arc from the root of the former to the root of the latter. Then, arbitrarily pair each leaf (source) in A_S with a unique leaf (destination) in A_T. For each such pair, draw an arc from source to destination (called a "bypass arc"), and add a demand to $\mathcal{T}_k$. Finally, define λ_k to assign length $1 + 2l$ to each "bypass" arc, and length 1 to all other arcs. We refer the reader to Fig. (3)(i).

We may assume our algorithm does not satisfy demands using the bypass arcs (as all demand-satisfying paths have length $2l+1$, and no tie-breaking scheme is specified). Thus, each demand-satisfying path uses the root of A_S, which incurs a total delay of $0+1+\ldots+(k-1) = \Omega(k^2)$ and total path length of $k \cdot (1+2l)$. Had the bypass arcs been used, no delay would have been required, and the total path length would have still been $k \cdot (1 + 2l)$. Thus, our algorithm achieves an approximation ratio of $(\Omega(k^2) + k \cdot (1 + 2l))/(k \cdot (1 + 2l)) = \Omega(k/\log k)$. □

5.3 Analysis on DAGs

We show that if we no longer require the graph family in Theorem (2) to have bounded degree, our algorithm loses its improved approximation ratio.

Theorem 3. *Algorithm 1 has an approximation ratio of $\Theta(k)$ on DAGs.*

Proof. By Proposition (1), it suffices to construct a family of TDW instances on DAGs, defined over all $k \in \mathbb{N}_{\geq 1}$, for which our algorithm achieves an approximation ratio of $\Omega(k)$. Construct G_k by fixing a "root" vertex and directly attaching $2k$ leaves. Orient half of these arcs towards the root, and half of the arcs away from the root. Call each vertex with out-degree 1 a *source*, and each vertex with in-degree 1 a *destination*. Then, arbitrarily pair each source with a unique destination. For each pair, add an arc from the source to the destination (called a

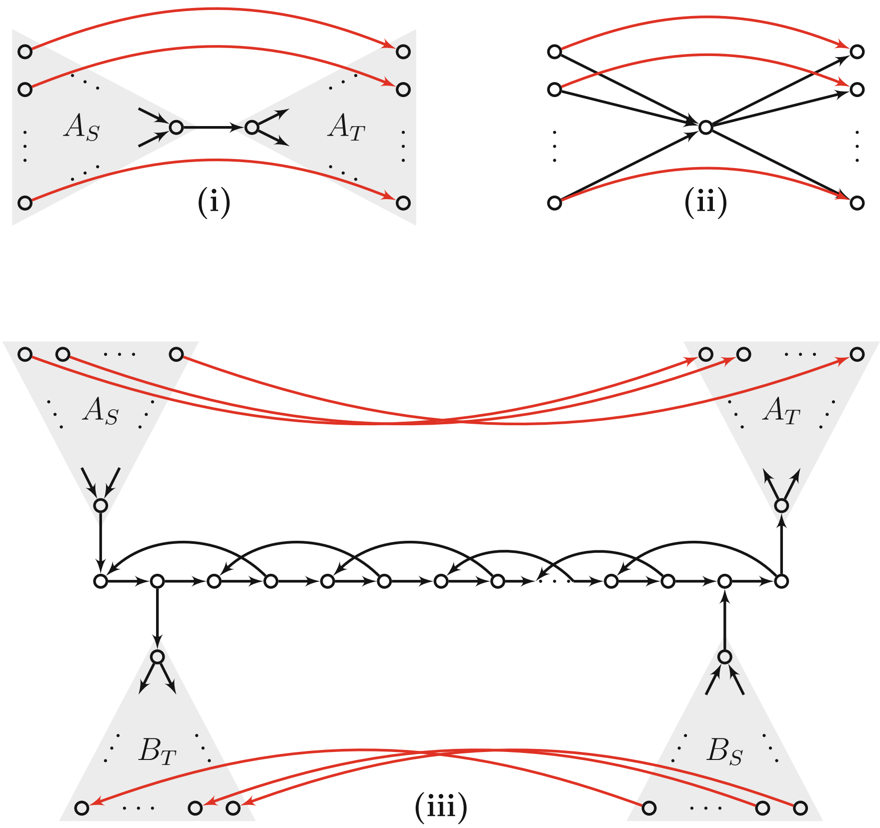

Fig. 3. (i): A bounded-degree DAG G_k upon which Algorithm 1 achieves an approximation ratio of $\Omega(k/\log k)$; (ii): A DAG G_k upon which Algorithm 1 achieves an approximation ratio of $\Omega(k)$; (iii): A bounded-degree digraph G_k upon which Algorithm 1 achieves an approximation ratio of $\Omega(k)$.

"bypass arc"), and add a demand to $\mathcal{T}_k$. Finally, let λ_k assign length 2 to each bypass arc, and length 1 to all other arcs. We refer the reader to Figure (3)(ii).

We may assume our algorithm does not satisfy demands using the bypass arcs (as all demand-satisfying paths have length 2, and no tie-breaking scheme is specified). Thus, each demand-satisfying path uses the root vertex, which incurs a total delay of $0 + 1 + \ldots + (k-1) = \Omega(k^2)$ and total path length of $2k$. Had the bypass arcs been used, no delay would have been required, and the total path length would have still been $2k$. Thus, our algorithm achieves an approximation ratio of $(\Omega(k^2) + 2k)/(2k) = \Omega(k)$. □

5.4 Analysis on Bounded-Degree Digraphs

In this section, we show that if we no longer require the graph family in Theorem (2) to be acyclic, our algorithm loses its improved approximation ratio.

Theorem 4. *Algorithm 1 has an approximation ratio of $\Theta(k)$ on bounded-degree digraphs.*

Proof. By Proposition (1), it suffices to construct a family of TDW instances on bounded-degree digraphs, defined over all $l \in \mathbb{N}_{\geq 2}$ with $\hat{k} := 2^l, k := 2\hat{k}$, for which our algorithm achieves an approximation ratio of $\Omega(k)$. Construct G_k by taking two $(2,l)$-in-trees A_S and B_S, and two $(2,l)$-out-trees A_T and B_T. Call their roots $r_{A_S}, r_{B_S}, r_{A_T}$, and r_{B_T}, respectively. Then, add a "central path" C consisting of vertices $\{c_1, c_2, \ldots, c_{\hat{k}}\}$, "forward" arcs $\{(c_i, c_{i+1}) \mid i \in [\hat{k}-1]\}$, and "backward" arcs $\{(c_j, c_{j-3}) \mid j \in [4, \hat{k}], j \bmod 2 = 0\}$. Attach the directed trees to the central path with arcs $\{(r_{A_S}, c_1), (r_{B_S}, c_{\hat{k}-1}), (c_{\hat{k}}, r_{A_T}), (c_2, r_{B_T})\}$. Next, pair each leaf (source) in A_S with an arbitrary, but unique, leaf (destination) in A_T. Do the same for B_S and B_T. For each such pair, add an arc from the source to destination (called a "bypass arc"), and add a demand to $\mathcal{T}_k$. Finally, let λ_k assign length $2\hat{k} + 2l - 1$ to each bypass arc, length $\hat{k} - 1$ to arcs $(r_{B_S}, c_{\hat{k}-1})$ and $(c_{\hat{k}}, r_{A_T})$, and length 1 to all other arcs. We refer the reader to Figure (3)(iii).

Observe that for each demand, there exist *two* shortest demand-satisfying paths, each of length $2\hat{k} + 2l - 1$. In particular, observe that a demand between leaves of A_S and A_T may be satisfied by a bypass arc, or by a path that travels from the source in A_S, towards the root of A_S, onto the central path vertex c_1, along all forward arcs of C, onto the root of A_T, and towards the destination in A_T. Similarly, a demand between leaves of B_S and B_T may be satisfied by a bypass arc, or by a path that travels from the source in B_S, towards the root of B_S, onto the central vertex $c_{\hat{k}-1}$, across C by alternating between forward and backward arcs (until arriving at c_2), onto the root of B_T, and towards the destination in B_T. We call the paths that do not use the bypass arcs the "meandering paths."

Because our algorithm specifies no tie-breaking scheme, we may assume that it satisfies demands using the meandering paths, and that it alternates between assigning delays to demands from A_S and assigning delays to demands from B_S every *four* iterations. In other words, out of the $2\hat{k}$ demands created above and fed as input to our algorithm, we may assume that those from A_S to A_T are labeled with indices $I_A := \{i \in [2\hat{k}] \mid \lfloor (i-1)/4 \rfloor \equiv 0 \pmod 2\}$, while those from B_S to B_T are labeled with $I_B := \{i \in [2\hat{k}] \mid \lfloor (i-1)/4 \rfloor \equiv 1 \pmod 2\}$.

To understand the suboptimality of this situation, we make several observations that help us determine the values our algorithm assigns to each d_i. First, note that for each $i \in [2\hat{k}]$, $z \in [\hat{k}]$, the length of walk W_i up to vertex c_z on the central path is:

$$\lambda_k(W_i, c_z) = \begin{cases} l + z, & \textbf{if } i \in I_A \\ l + 2\hat{k} - z - 2 \cdot (z \bmod 2), & \textbf{if } i \in I_B \end{cases}$$

Using this, we see that the details of Algorithm 1 give us the following relation, which is defined over $i \in [k], h \in [i-1]$:

$$\mathsf{bad\_delays}_{i,h} = \begin{cases} \{d_h\}, & \textbf{if } i, h \in I_A \textbf{ or} \\ & \textbf{if } i, h \in I_B \\ \{d_h - 2\hat{k} + 2z + 2 \cdot (z \bmod 2) \mid z \in [\hat{k}]\}, & \textbf{if } i \in I_B, h \in I_A \\ \{d_h + 2\hat{k} - 2z - 2 \cdot (z \bmod 2) \mid z \in [\hat{k}]\}, & \textbf{if } i \in I_A, h \in I_B \end{cases}$$

Because our algorithm defines $\mathsf{bad\_delays}_i := \bigcup_{h \in [i-1]} \mathsf{bad\_delays}_{i,h}$ and $d_i := \min(\mathbb{Z}_{\geq 0} \setminus \mathsf{bad\_delays}_i)$, observe that the above relation is in fact a *recurrence* relation. As such, after noting that $d_1 = 0$, it is straightforward to use the above relation to show by induction that for all $i \in [2\hat{k}]$,

$$d_i = i - 1 + \lfloor \frac{i-1}{8} \rfloor (2\hat{k} - 4).$$

Thus, our algorithm incurs a total delay of $\sum_{i \in [2\hat{k}]} (i - 1 + \lfloor (i-1)/8 \rfloor (2\hat{k} - 4)) = \Omega(\hat{k}^3) = \Omega(k^3)$ and total walk length of $2\hat{k} \cdot (2\hat{k} + 2l - 1)) = \Theta(\hat{k}^2) = \Theta(k^2)$. Had the algorithm opted to use the bypass arcs, no delay would have been required, and the total walk length would have been the same. Thus, our algorithm achieves an approximation ratio of $\Omega(k)$. □

6 Conclusions

In this paper, we introduce Time Disjoint Walks, a new variant of (shortest) Disjoint Paths that also seeks to connect k demands in a network, but relaxes the disjointness constraint by permitting vertices to be shared across multiple walks, as long as no two walks arrive at the same vertex at the same time. We show that Time Disjoint Walks is APX-hard, even for DAGs of max degree three. On the other hand, we provide a natural $\Theta(k/\log k)$-approximation algorithm for directed acyclic graphs of bounded degree. Interestingly, we also show that for general digraphs with just one of these two properties, the approximation ratio of our algorithm is bumped up to $\Theta(k)$.

An interesting future work is to tighten the gap between these inapproximability and approximability results for TDW on bounded-degree DAGs. We conjecture that our approximation algorithm is almost optimal, but that our hardness of approximation result can be strengthened to nearly match our algorithm's approximation ratio of $\Theta(k/\log k)$. This belief is based on the observation that TDW is a complex problem that involves both routing *and* scheduling, and many problems of the latter variety (of size n) are NP-hard to approximate within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$ [16]. One may also wish to explore similar complexity questions for the many variants of Time Disjoint Walks discussed in Sect. 2.

References

1. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer, Heidelberg (2012)
2. Berman, P., Karpinski, M.: On some tighter inapproximability results (extended abstract). In: Wiedermann, J., van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 200–209. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48523-6_17
3. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **45**(9), 1563–1581 (1966)
4. Groß, M., Skutella, M.: Maximum multicommodity flows over time without intermediate storage. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 539–550. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33090-2_47
5. Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., Yannakakis, M.: Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. J. Comput. Syst. Sci. **67**(3), 473–496 (2003)
6. Karp, R.M.: On the computational complexity of combinatorial problems. Networks **5**(1), 45–68 (1975)
7. Kleinberg, J.M.: Approximation algorithms for disjoint paths problems. Ph.D. thesis. Massachusetts Institute of Technology (1996)
8. Kobayashi, Y., Sommer, C.: On shortest disjoint paths in planar graphs. Discrete Optim. **7**(4), 234–245 (2010)
9. Lengauer, T.: Combinatorial Algorithms for Integrated Circuit Layout. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-322-92106-2
10. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. J. Combin. Theor. Ser. B **63**(1), 65–110 (1995)
11. Scheffler, P.: A practical linear time algorithm for disjoint paths in graphs with bounded tree-width. TU, Fachbereich 3 (1994)
12. Schrijver, A., Lovasz, L., Korte, B., Promel, H.J., Graham, R.: Paths, Flows, and VLSI-Layout. Springer, New York (1990)
13. Skutella, M.: An introduction to network flows over time. In: Cook, W., Lovász, L., Vygen, J. (eds.) Research Trends in Combinatorial Optimization, pp. 451–482. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-76796-1_21
14. Srinivas, A., Modiano, E.: Minimum energy disjoint path routing in wireless ad-hoc networks. In: Proceedings of the 9th Annual International Conference on Mobile Computing and Networking, pp. 122–133. ACM (2003)

15. Torrieri, D.: Algorithms for finding an optimal set of short disjoint paths in a communication network. IEEE Trans. Commun. **40**(11), 1698–1702 (1992)
16. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. In: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, pp. 681–690. ACM (2006)