Silos and Lazy Shortest Paths on Ordered Directed Acyclic Graphs

Jad Soucar [1]

Abstract

Many dynamic programs can be interpreted as shortest path problems on ordered directed acyclic graphs (DAGs), where edge weights are optimal values of non-trivial optimization problems. In such cases using approximate lower-bounding weights can reduce computational cost. In this paper we introduce general formalisms to study these “lazy shortest path” problems where edge weight computation is delayed or avoided. Our primary contribution in this area is introducing the concept of a graph Silo, which captures the degree to which a graph permits paths that are nearly tied to the shortest path. We show that such formalisms are especially useful in bounding behavior of canonical lazy shortest path algorithms such as LazySP. We then use our formalisms to motivate the introduction of a new class of lazy shortest path algorithms that introduce lazy label correcting algorithms to the lazy shortest path literature. Empirically we find that on graphs with many near ties our proposed Pruning Shortest Path family of algorithms outperforms the LazySP framework.

1. Introduction

Dynamic programming problems of the form,

$$\begin{aligned} f(j) &= \min_{0\leq i<j\leq T} \{f(i) + w(i,j)\} \\ \text{s.t} \quad w(i,j) &= \min_{x\in\mathcal{X}_{ij}} g(x;i,j). \end{aligned} \tag{1}$$

can be found in a wide variety of fields ranging from change-point detection, time-series segmentation, optimal control with switching, and even recent applications to LLM sequence planning (Jackson et al., 2005b; Auger & Lawrence, 1989; Zheng & Ye, 2023; Yao et al., 2023). The canonical interpretation of recursion 1 is that of a minimum-cost path/shortest path on an ordered directed acyclic graph (ordered DAG) $G = (V, E, w)$. We define such an “ordered DAG” in definition 1.1. Such graphs can often be visualized as a fully connected DAG like that in figure 1.

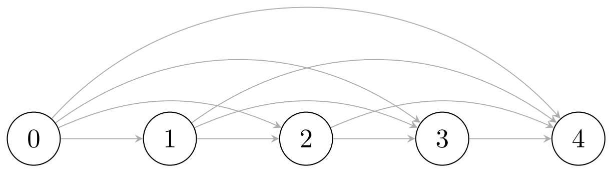


Figure 1. Fully connected forward time-ordered DAG.

Definition 1.1 (Ordered DAGs)**.** We let $\mathcal{D}(T) = \{D = (V,E,w) | V = \{0,1,\ldots,T\}, E = \{(i,j) : 0 \leq i < j \leq T\}, w(i,j) : E \to \mathbb{R}, T < \infty\}$ be the space of T node ordered DAGs. We refer to G as a T node *ordered DAG* if $G \in \mathcal{D}(T)$.

The computational challenge in finding the shortest path on a graph $G \in \mathcal{D}(T)$ often times depends on the cost of computing $w(i,j)$. In many classical stetting these edge weights admit closed form-expressions or exploit strict properties about w, which makes computing all $T(T+1)/2$ edge weights feasible. However in an increasing number of modern applications $w(i,j)$ is itself the optimal value of a non-trivial optimization or learning problem.

1.1. Examples of Ordered DAGs with Expensive Edge Evaluations

The problem structure we’ve outlined is broad enough to fit applications from many domains. To illustrate this point and motivate the importance of studying such problems we identify three examples from three different domains which fit the structure of problem 1. We begin with a classical example from change point detection.

Example 1.2 (Segment Neighborhood)**.** Segment Neighborhood, aims to find an optimal splitting scheme of a time-series by solving

$$\{\tau_0 < \tau_1 < \cdots < \tau_r\} = \operatorname*{argmin}_{r,\tau_0<\cdots<\tau_r} \sum_{i=1}^{r} \ell(\tau_{i-1},\tau_i) + r\gamma$$

[1]Daniel J. Epstein Department of Industrial & Systems Engineering, University of Southern California, Los Angeles, California U.S.A. Correspondence to: Jad Soucar <soucar@usc.edu>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

,where $\ell(i,j) = \min_{x\in\mathcal{X}_{\rangle|}} g(x;i,j)$ such that $\ell(i,i) = 0$ (Auger & Lawrence, 1989). Such a problem is easily recast to

$$f(j) = \min_{0\leq i<j\leq T}\{f(i) + \ell(i,j) + \gamma\mathbb{I}_{i\neq 0}\}, \quad f(0) = 0$$

by interpreting each edge as a segment, ,which matches problem 1. The optimal splitting regime $\{0 = \tau_0 < \tau_1 < \cdots < \tau_r = T\}$ is then recovered by identifying the nodes in the shortest path.

There are several other algorithm in change-point detection literature such as the more modern PELT algorithm which make use of the same dynamic programming based problem structure (Jackson et al., 2005a). Next we explore an example from on-off control, sometimes referred to as "Bang-Bang" control.

Example 1.3 (On–Off Control with Resets)**.** Consider a dynamical system with binary control input $u_t \in \{0,1\}$ and state dynamics $x_{t+1} = F(x_t, u_t)$. Assume that whenever the control switches at time τ_{i-1}, the system state is reset to a known state $x_{\text{reset}}(\tau_{i-1})$. The objective is to choose switching times $\{0 = \tau_0 < \tau_1 < \cdots < \tau_r = T\}$ that minimize the total cost

$$\sum_{i=1}^{r} \ell(\tau_{i-1}, \tau_i) + r\gamma(x_{\tau_{i-1}}, \tau_{i-1})$$

where $\ell(i,j)$ denotes the cost incurred by holding the control fixed over the interval $(i,j]$ starting from the reset state at time i. In particular,

$$\ell(i,j) = \min_{m\in\{0,1\}} \left\{ \sum_{t=i}^{j-1} c(x_t^{(i,m)}, m, t) + \phi(x_j^{(i,m)}) \right\},$$

where the state trajectory satisfies $x_i = x_{\text{reset}}(i)$ and $x_{t+1}^{(i,m)} = F(x_t^{(i,m)}, m)$ is the state at $t+1$ given a control choice $m \in \{0,1\}$. Defining $f(j)$ as the minimum achievable cost up to time j, the problem admits the dynamic programming recursion

$$f(j) = \min_{0\leq i<j\leq T}\{f(i) + \ell(i,j) + \gamma(x_{\tau_{i-1}}, \tau_{i-1})\mathbb{I}_{i\neq 0}\},$$

which matches the form of problem 1. The optimal switching schedule is recovered by identifying the nodes along the shortest path from 0 to T.

On-off control with resets has been a popular modeling choice for problems involving regenerative dynamics, including machine maintenance, inventory systems with replenishment, and more. Finally we identify a simple but computationally difficult example from robotic planning.

Example 1.4 (Simulation-Based Motion Planning under Uncertain Obstacles)**.** Consider a robot that must navigate from a point in space A to B given uncertain/probabilistic obstacles appearing on that path. Assume that the robot has access to $T-2$ way-point/check-points along the way. The robot must then choose whether to use a given checkpoint and how to arrive at each check point. The problem reduces to a shortest path problem on a ordered DAG, where $\{A, 1, \ldots, T-2, B\}$ are nodes, and an each edge has a cost

$$w(i,j) = \min_{x\in\mathcal{X}_{\rangle|}(i,j)} \mathbb{E}_\omega[L(x;i,j) \ + \ \lambda\,\mathbb{I}_{\text{collision}}]$$

where $\lambda > 0$ is an obstacle collision penalty and $L(x;i,j)$ is the cost of executing a path x from checkpoint i to j.

Then the globally optimal plan over checkpoints satisfies the ordered DAG recursion

$$f(j) = \min_{0\leq i<j\leq T}\{f(i) + \min_x L(x;i,j) + \lambda\mathbb{I}_{\text{collision}}\}.$$

In cases, such as that outlined above, the evaluation of w, can be a costly process in and of itself, which may require computationally expensive iterative solvers or even Monte-Carlo simulations whose costs dominate the overall runtime and makes $\mathcal{O}(T^2)$ edge weight evaluations unreasonable. This means that applying traditional shortest path algorithms like Dijkstra, Bellman-Ford and A*, where computing many or all true-edge weights is required, is infeasible.

1.2. Previous Work & Literature Gap

The recent trend toward solving shortest path problems on graphs with expensive-to-evaluate edge weights has motivated the use of delayed or reduced cost edge weight evaluations, in which the edge weights are lower bounded by a simpler to compute quantity $w_{lb}(e) \leq w(e)$ and the true edge weights are only computed when necessary.

Much of this work has come out of the field of robotic planning, where Dellin et.al developed the LazySP algorithm class (Dellin & Srinivasa, 2016). LazySP is a general algorithmic framework that repeatedly computes shortest paths on a graph with lower bounded edge weights then selects edges to update to the true edge weight. In recent years this approach has been expanded to include different edge selectors or incorporate tools from optimal control to improve the algorithm's efficiency, in terms of the number of true edge evaluations required, and show equivalence to other lazy shortest path algorithms (Mandalika et al., 2018). For example it was found that depending on the selector, LazySP can be shown to be equivalent to Lazy A* and weighted A* (Dellin & Srinivasa, 2016). Other works, aiming to characterize the behavior of LazySP, have bounded the number of true edge evaluations for probabilistic edge weight function $w(i,j)$ (Haghtalab et al., 2018). While the LazySP framework most explicitly formalizes the idea of using lower bound edge weights, other algorithms D* or D*

Lite also handle the idea of graphs that evolve over time but lack explicit relationships with lower bounding strategies (Stentz, 1995).

We observe that a major gap in prior literature, is the lack of algorithmic and theoretical differentiation between general graphs and ordered DAGs of the form induced by recursion 1 with lazy edge weights. Such a differentiation is useful given the prevalence of ordered DAGs, as explored in 1.1, and the fact that many ordered DAGs admit specialized properties such as topological dynamic programming, prefix dominance, and structured label correcting algorithmic schemes. Due to the focus on algorithmic development for general graphs, lazy shortest-path literature is largely dominated by algorithmic approaches that resemble label-setting methods where a candidate path is identified and true-edge evaluations are performed along the path until optimality can be certified (Dellin & Srinivasa, 2016). On the other hand, label-correcting type algorithms for shortest path problems with lazy edge weights are largely absent from lazy shortest path literature.

To address this gap, we first introduce a DAG specific analytical framework for studying lazy graph representations and lazy shortest path problems. Specifically we show that the behavior of the canonical approach, LazySP, can be characterized by geometric objects we call Silos, which capture the degree to which a graph permits near ties among candidate paths. We show that Silos largely govern the algorithmic efficiency of LazySP shortest path schemes. Building on this characterization we introduce a Pruning Shortest Path family of algorithms based on label correcting mechanisms. Empirically we find that our proposed class of algorithms can be more efficient then LazySP type algorithms in graphs that alow for near ties (ill-defined Silos) whereas LazySP dominates in regimes where near ties are rare (well-defined silos).

2. Formalisms for Lazy Ordered DAGs

In this section we formalize a theoretical framework through which we can analyze general lazy shortest path algorithms on ordered DAGs. We begin by formalizing the concept of a "lazy graph" and some useful path wise characterizations of a graph. We build up to the concept of a "graph Silo" which functionally captures the degree to which a graph allows for paths that are nearly tied for the position of shortest path.

Definition 2.1 (ϵ- Lazy Graphs)**.** Given a graph $G = (V, E, w) \in \mathcal{D}(T)$. A corresponding graph $G^{\epsilon}_{lb} = (V, E, w^{\epsilon}_{lb})$ is referred to as a *"ϵ lazy graph of G"* if for all $e \in E$ we have that $0 \leq w(e) - w_{lb}(e) \leq \epsilon$. We refer to $\mathcal{D}_{lb}(G; \epsilon)$ to be the set of ϵ lazy graphs of a graph $G \in \mathcal{D}(T)$.

Definition 2.2 (Lazy Graph Couples)**.** Given a graph $G \in \mathcal{D}(T)$ we define

$$C(G; \epsilon) = \{(G, G^{\epsilon}_{lb}) | G^{\epsilon}_{lb} \in \mathcal{D}_{lb}(G; \epsilon)$$

as the set of all ϵ *lazy graph couples of* G

Definition 2.3 (Path-wise Quantities)**.** Let $G \in \mathcal{D}(T)$.

1. (Feasible Paths): The set of feasible paths from node 0 to k is defined as

$$P_{0\to k}(G) = \{(0 = v_0 < v_1 < \cdots < v_m = k) : v_i \in V\}$$

2. (Path Cardinality): Each path $\pi \in P_{i\to k}$ travels on edges $E(\pi) \subset P_{0\to k}(G)$. We denote π itself as an ordered set of nodes with cardinality $|\pi|$.

3. (Path Costs): Each path on a graph G has a cost $c(\pi; G)$ where

$$c(\pi; G) = \sum_{q=1}^{|\pi|} w(q-1, q)$$

Definition 2.4 (γ-Optimal Region)**.** Given a graph $G \in \mathcal{D}(T)$ and path $\pi \in P_{0\to k}(G)$. We define a γ-optimal region centered around π as

$$N_{\gamma}(\pi; G) = \{\pi' \in P_{0\to k}(G) : c(\pi'; G) - c(\pi; G) \leq \gamma\}$$

Definition 2.5 ((γ, μ)-Silo)**.** Given a graph $G \in \mathcal{D}(T)$ and path $\pi \in P_{0\to k}(G)$. We define a (δ, μ)-Silo centered around π as

$$\begin{aligned} S_{(\gamma,\mu)}(\pi; G) = \{\pi' \in N_{\gamma}(\pi; G) : \forall \hat{\pi} \notin N_{\gamma}(\pi; G), \\ c(\hat{\pi}; G) - c(\pi'; G) \geq \mu\} \end{aligned}$$

The concept of a (γ, μ)-Silo centered at π is best conceptualized by representing each path as a point in a cost space, where the space is imbued with a distance pseudometric $d_{S_{(\gamma,\mu)}(\pi;G)}(\pi') = |c(\pi'; G) - c(\pi; G)|$ for any paths $\pi', \pi \in P_{0\to k}(G)$. A (δ, μ)-Silo centered around π is then defined as all paths that are γ-close to $c(\pi; G)$ and μ-far from every path not in the silo. This conceptualization is illustration in 2. A useful quantity to measure of (γ, μ)-Silos is the Silo volume $\text{Vol}(S)$, which quantifies the total number of unique edges contained in a Silo.

Definition 2.6 (Edge-Volume)**.** Given a graph $G \in \mathcal{D}(T)$, a path $\pi \in P_{0\to k}(G)$. Then edge volume is the total number of unique edges in a set of paths $S \subseteq P_{0\to k}(G)$

$$\text{Vol}(S) = \Big|\bigcup_{\pi \in S} E(\pi)\Big|$$

We note that it will become useful to consider shortest paths π^* that are "separated" in cost space from all other paths. We refer to such a path is δ-robust.

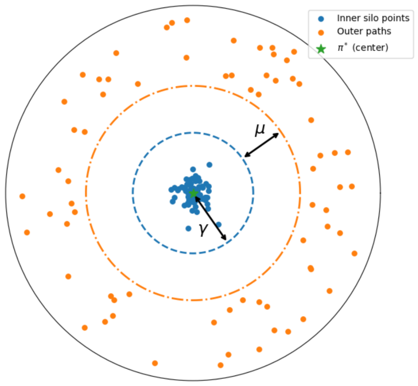


Figure 2. Illustration of (γ, μ)-Silos

Definition 2.7 (δ-Robust)**.** A graph G is δ-robust if the shortest path satisfies any one of the following properties, each of which is equivalent .

1. π^* is the unique shortest path on G, whith cost $c(\pi; G) - c(\pi^*; G) > \delta > 0$ for all other paths $\pi \in P_{0\to T}(G)$
2. $N_\delta(\pi^*; G) = \{\pi^*\}$
3. The graph G permits a $S_{0,\delta}(\pi^*; G)$ Silo centered at π^*

While these graph characterizations may seem arbitrary at first, we show that they allow us to prove a general result regarding shortest paths on lazy graph couples.

Lemma 2.8 (Sufficient Conditions for Shortest Path Equivalence)**.** *Given a ϵ lazy graph couple $(G, G^\epsilon_{lb}) \in C(G; \epsilon)$ such that G is δ-robust with shortest path π^* and $\delta \geq \epsilon T$. Then the shortest path on G^ϵ_{lb} is π^*.*

On its own Lemma 2.8 tells us that for any ϵ lazy graph couple $(G, G^\epsilon_{lb}) \in C(G; \epsilon)$ where G is δ-robust the shortest path problem on G^ϵ_{lb} is equivalent to that on G, which means that in practice, one could apply Dijkstras or Bellman-Ford algorithms directly on to G^ϵ_{lb} to retrieve the shortest path on G.

2.1. Application to LazySP

Next we demonstrate the practicality of our theoretical framework to solve an open problem for the family of LazySP algorithms. Namely, what properties of the ϵ lazy graph couple (G, G^ϵ_{lb}) are sufficient to achieve best case behavior using the LazySP family of algorithms. We begin by formalizing the LazySP family as described by Dellin et. al (Dellin & Srinivasa, 2016) within our analytical framework, along with the canonical metric to evaluate LazySP algorithmic performance.

Definition 2.9 (LazySP Family of Algorithms)**.** Given a ϵ-lazy graph couple $(G, G^\epsilon_{lb}) \in C(G; \epsilon)$, where $w(i, j)$ is hidden from the algorithm. The LazySP family of algorithms assumes access to a function $\text{ShortestPath}(G)$ which returns the shortest path on a graph G, a function $\text{Selector}(G, \pi)$ which selects one or more edges from a path π, and a function $\text{Eval}(i, j)$ which returns the true edge weight $w(i, j)$. Using these three oracle functions the family of LazySP algorithms is defined as algorithm 1. We refer to the families of LazySP algorithms as $\mathcal{LSP}$, with each algorithm $A \in \mathcal{LSP}$ being uniquely defined by the $\text{Selector}(G, \pi)$ used.

Algorithm 1 LazySP

```
1: Instantiate $G^0_{work} = (V, E, w_{work})$ with $w_{work}(i,j) = w_{lb}(i,j)$ as the working graph. Set $\nu = 0$.
2: loop
3:   $\pi \leftarrow \text{ShortestPath}(G^\nu_{work})$
4:   if $\forall (i,j) \in \pi,\quad w^\nu_{work}(i,j) = w(i,j)$ then
       return $\pi$
5:   end if
6:   $E_{selected} \leftarrow \text{Selector}(G^\nu_{work}, \pi)$
7:   for $(i,j) \in E_{selected}$ do
8:     if $w^\nu_{work}(i,j) \neq w(i,j))$ then
         $w^\nu_{work}(i,j) \leftarrow \text{Eval}(i,j) = w(i,j)$
9:     end if
10:    $\nu = \nu + 1$
11:  end for
12: end loop
```

To rank each algorithm in $A \in \mathcal{LSP}$ for any ϵ lazy graph couple (G, G^ϵ_{lb}) we define the commonly used metric of Total Evaluations, which we will use as the primary efficiency metric from this point forward. Studying total evaluations is particularly useful for graphs with expensive to evaluate edge weights, since our primary goal is to reduce the number or expensive true edge weight evaluations.

Definition 2.10 (Total Evaluations)**.** We define *total evaluations* for any $A \in \mathcal{LSP}$ as the the total number of times the oracle $\text{Eval}(i, j)$ is called before termination.

Note that the trivial upper bound for any algorithm $A \in \mathcal{LSP}$ is simply all edges present in a graph, $|E|$. Within the setting of ordered DAGs $|E|$ is at most $T(T+1)/2$. However using our framework we prove a general statement regarding when we can bound the worst case total evaluations of an algorithm $A \in \mathcal{LSP}$.

Theorem 2.11 (Siloed Worst Case Behavior)**.** *Given an ϵ lazy graph couple (G, G^ϵ_{lb}) where the shortest path on G is π^* and the shortest path on G^ϵ_{lb} is π^*_{lb}. Assuming (1) there exists $S_{(\gamma,\mu)}(\pi^*; G)$, a (γ, μ)-Silo centered around π^* on graph G, and (2) that $\mu + \gamma - \epsilon L > 0$ where $L = \max_{\pi \notin S_{(\gamma,\mu)}(\pi^*;G)} |\pi|$. Then for all $A \in \mathcal{LSP}$ where* $\text{Selector}(G, \pi) \subseteq \pi$ *(i.e no out-of-path evaluations) the worst case total evaluations will be* $\text{Vol}(S_\gamma(\pi^*; G))$.

In other words if a silo exists around π^* we can achieve a tighter bound on the total evaluations. The proof of Theorem

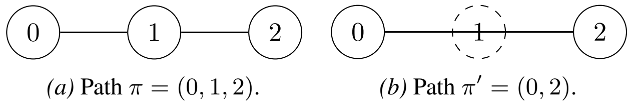


Figure 3. Two time-ordered DAG paths that are *path-wise close*: π' is obtained from π by a single node deletion (equivalently, π from π' by one insertion). This illustrates near-tie alternatives that lie within small edit distance of an optimal path, corresponding to small silo volume.

2.11 is instructive in that it provides us with the following invariant property of LazySP. Namely that if a candidate path selected is contained within a Silo then all future paths selected will remain inside the Silo.

Corollary 2.12 (Silo Containment)**.** *Given an ϵ lazy graph couple (G, G^{ϵ}_{lb}) where the shortest path on G is π^* and an algorithm $A \in \mathcal{LSP}$. Assuming (1) there exists $S_{(\gamma,\mu)}(\pi^*; G)$, a (γ, μ)-Silo centered around π^* on graph G, and (2) that $\mu + \gamma - \epsilon L > 0$ where $L = \max_{\pi \notin S_{(\gamma,\mu)}(\pi^*;G)} |\pi|$. Then if a path π^{ν} is selected at some iteration ν of A by* ShortestPath *then all paths π^r for $r \geq \nu$ will remain in $S_{(\gamma,\mu)}(\pi^*; G)$.*

This containment invariant yields two core insights. First that LazySP's behavior depends largely on the guess of an initial path. Second that LazySP's total evaluation behavior, is poorly bounded when the there are many distinct paths with costs near the optimal cost. However we observe that even if there are many near ties LazySP can still perform well if most near ties are within a small edit distance of the optimal path. Specifically if a most near optimal paths require few node insertions or deletions then the Silo volume remains small and LazySP remains well behaved. Such paths that are near ties in terms of cost and yet remain path-wise "close" to the optimal path can be illustrated as figure 3.

A special case of theorem 2.11 is when a the graph permits a $S_{0,\delta}(\pi^*; G)$ silo centered at π^* in which case the shortest path π^* is referred to as δ-robust. In this case any algorithm $A \in \mathcal{LSP}$ is guaranteed to have only $|\pi^*|$ total evaluations, which is the best case behavior of A.

Proposition 2.13 (Sufficient Conditions for Best Case Performance)**.** *Given an ϵ lazy graph couple (G, G^{ϵ}_{lb}) where the shortest path on G is π^*. Assuming that (1) G^{ϵ}_{lb} and G are δ-robust and (2) $\delta > \epsilon T$. Then for all $A \in \mathcal{LSP}$ where* $\text{Selector}(G, \pi) \subseteq \pi$ *(i.e no out-of-path evaluations) the total evaluations will be $|\pi^*|$.*

We note, that the assumptions of proposition 2.13 or theorem 2.11 could be difficult to satisfy. However they may be defensible on a K-node subgraph $G_{(0,K)}$ of $G \in \mathcal{D}(T)$ where where $V_{(0,K)} = \{0, \dots, K\}$ and $E_K = \{(i, j) \in E : 0 \leq i < j \leq K\}$. In the case where the sufficient conditions are satisfied on a K-node subgraph one could, if the graph is prefix dominant, reduce the problem into two stages. Where the first stage solves the shortest path problem on $G^{\epsilon}_{lb,(0,K)}$, and second applies LazySP to the ϵ lazy couple $(G^{\epsilon}_{lb,(\tau_r,T)}, G_{(\tau_r,T)})$ where $\pi^*_{(0,K)} = \{0, \tau_1, \dots, \tau_r, K\}$.

Definition 2.14 (Prefix Dominance)**.** Given a graph $G \in \mathcal{D}(T)$ with optimal path π^*. Let $G_{(0,K)}$ be a K-node subgraph of G such that $V_K = \{0, \dots, K\}$, $E_K = \{(i, j) \in E : j \leq k\}$, and $\pi^{*,(0,K)}$ is the shortest path on $G_{(0,K)}$. We refer to the G is *prefix dominated*, if for all $0 < K \leq T$ we have that $\pi^{*,(0,K)} = \{0, \tau_1, \dots, \tau_r, K\}$ has prefix $\{0, \dots, \tau_r\}$ which is also the r prefix of π^*.

Corollary 2.15 (Subgraph Dominance)**.** *Given an ϵ lazy graph couple (G, G^{ϵ}_{lb}) where the shortest path on G is π^*. Assuming that (1) the $(0, K)$-subgraphs $G^{\epsilon}_{lb,(0,K)}$ and $G_{(0,K)}$ are δ-robust, (2) $\delta > \epsilon T$ and (3)G is prefix dominant. Then for all $A \in \mathcal{LSP}$ where* $\text{Selector}(G, \pi) \subseteq \pi$ *(i.e no out-of-path evaluations), $A \in \mathcal{LSP}$ can be augmented to reduce the total evaluations to a worst case of $|\pi^*_{(\tau_r)}| + (T - \tau_r)(T - \tau_r - 1)/2$. Where $\pi^*_{(0,K)} = \{0, \tau_1, \dots, \tau_r, K\}$ is the shortest path on $G_{(0,K)}$.*

Weconclude with the observation that the formalisms and properties we've developed to describe lazy ordered DAGs are useful in proving bounds on existing LazySP-type algorithms for ordered DAGs. We also observe that LazySP's total evaluation behavior is largely dominated by geometric properties in both the cost space of DAG along with the path-level geometry (in terms of edit distance).

2.2. Correspondence Theorems

Properties required to reap the benefits of results like Lemma 2.8 and proposition 2.13, such as δ-robustness, δ-degeneracy, and ϵ-lazy graphs, may be difficult to verify in a brute force fashion. To that end we note that concepts introduced are deeply related to well understood properties about the edge weight function

$$w(i, j) = \min_{x \in \mathcal{X}_{\rangle|}} g(x; i, j). \tag{2}$$

To study such relationships we first introduce a few definitions, before moving on to correspondence between structured $w(i, j)$ optimization problems and the previously defined quantities.

Definition 2.16 (m-paths)**.** Given a graph $G \in \mathcal{D}(T)$ a path π_m is considered an m-path if $\pi_m \in P_{0 \to T}(G)$ and $|E(\pi_m)| = m$. We denote the shortest m-path among m-paths as π^*_m.

Definition 2.17 (x Parameter Space)**.** Given $G \in \mathcal{D}(T)$ where $w(i, j) = \min_{x \in \mathcal{X}} g(x; i, j)$. Consider $\pi \in P_{0 \to k}(G)$. We define $x^*_{\pi} = (x^*_{e_1}, \dots, x^*_{e_m}) \in \mathcal{X}^m$ to be the vector of optimal $x^*_{e_i} = \arg\min_{x \in \mathcal{X}} g(x; e_i)$ for each edge $(i, j) \in E(\pi)$.

Proposition 2.18 (Splicing Regimes δ-Robustness)**.** *Given $G \in \mathcal{D}(T)$ such that $\mathcal{X} \subseteq \mathbb{R}^{T\times n_1 \times \cdots \times n_q}$ and $w(i,j) = \min_{x\in\mathcal{X}} g(x^{[i:j]})$, where $x^{[i:j]}$ is the vector splice of x with all indices outside $[i,j]$ set to 0. Assume that (1) g is α-strongly convex in x and (2) that $\|x^*_\pi - x^*_{\pi^*}\| > \delta$ then G is $(\alpha/2)\delta^2$-robust.*

Proposition 2.18 is especially useful for data-based problems where the aim of $w(i,j)$ is to fit parameters to a certain subset of data. Such structures arise frequently in change point detection literature, where the problem must find parameters $x \in \mathbb{R}^{T\times \prod n_i}$ for various splices of data.

Proposition 2.19 (Convexity in Edge Length δ-Robustness)**.** *Given $G \in \mathcal{D}(T)$ where the weight function $w(i,j)$ is structured as $w(i,j) = \min_{x\in\mathcal{X}} f(l) + h(x,l) + \lambda$ where $l = j - i$. Assume that $f(l)$ is α-discrete strongly convex in l and h is jointly convex in x and l. Then π^*_m is α-robust among the other m-paths*

Proposition 2.19, on the other hand, is useful for regimes where the edge weights represent the cost of traveling true distances which grow with the length of each edge. Additionally many settings require weaker conditions then full δ-robustness and instead require a δ-robustness over paths of a specific length. Such settings arise in robotic planning where the cost of executing a strategy on a longer edge is always more expensive then executing a strategy on a shorter edge and the robot is expected to reach exactly m-nodes on its path. We've shown that the concepts of δ-Robustness which are central to Lemma 2.8 and theorem 2.11 can be related to traditional concepts from convex optimization. Next we show that path level convexity can help us bound the Volume a Silo. This particularly useful in actualizing the bounds from theorem 2.11.

Proposition 2.20 (Volume Bounds)**.** *Given a $G \in \mathcal{D}(T)$ where the cost function satisfies*

$$c(\pi; G) - c(\pi^*; G) \geq \frac{\alpha}{2}\|\phi(\pi) - \phi(\pi^*)\|_2^2$$

for an injective function $\phi(\pi) : P_{0\to T}(G) \to \mathbb{Z}^d_+$ for some $d > 0$. Then for a silo $S_{\gamma,\mu}(\pi^*; G)$ we have that

$$\mathrm{Vol}(S_{(\gamma,\mu)}(\pi^*; G)) \leq \min\left\{\frac{T(T+1)}{2}, \sum_{i=1}^{\left\lceil\left(2\left\lfloor\sqrt{\frac{2\gamma}{\alpha}}\right\rfloor+1\right)^d\right\rceil} \left\lfloor\frac{T}{i}\right\rfloor\right\}$$

Unlike the other properties covered above, ϵ-lazy graph constructions are trivially related to ϵ-approximate minimizer, which can be derived from many algorithms like gradient descent method, proximal gradient methods, and many more (**?**). One could also imagine that a lower bound weight function $w_{lb}(i,j)$ can be obtained by finding the ϵ-approximate maximizer of the dual problem of problem 2.

3. Pruning Shortest Path Family of Algorithms

Two major computational challenges of the LazySP is that it requires the user pre-compute all lower-bounded weights and it requires repeated shortest path computations. While this may by feasible in the case where the graph T is small for $G \in \mathcal{D}(T)$, or $w_{lb}(i,j)$ is trivial to compute, we note that this is not always the case. The shortest path computations and the computation of $w_{lb}(i,j)$ may also carry computational cost such as when the lower-bound weight function is computed via an abridged iterative process or a smaller monte-Carlo simulation then what would be required to compute the true edge weight $w(i,j)$. In-fact we can derive the following conditions under which LazySP is a viable competitor to simply computing the shortest path on G.

Proposition 3.1 (Run-Time Winner)**.** *Let (G, G^ϵ_{lb}) be an ϵ-lazy graph couple. Consider an algorithm $A \in \mathcal{LSP}$. Let $\mathcal{T}_{lb}$ be the maximum time it takes to compute the lower bound function $w_{lb}(i,j)$ and let $\mathcal{T}$ be the maximum time it takes to compute the true weight function $w(i,j)$ for any (i,j), where we assume that $\mathcal{T}_{lb} \leq \mathcal{T}$. Also Let $\mathcal{T}_{sp}(G)$ be the cost of computing the shortest path on a graph G and $\mathcal{V}$ be the number of iterations of A before termination. Then algorithm A is guaranteed to have smaller runtime if*

$$\mathcal{T}_{sp} \leq \frac{1}{\mathcal{V}-1}\Big[|E|(\mathcal{T} - \mathcal{T}_{lb}) + \mathcal{T}|N_\gamma(\pi^*; G)|\Big]$$

In cases where T is sufficiently large the need to iteratively compute shortest paths of the working graph G^ν_{work} at every iteration ν may dominate the algorithms runtime for large graphs. Additionally LazySP was built for general graphs, and does not exploit any structural properties of the ordered DAGs such as the permiting efficient label-correcting mechanisms. To address this these two problem we introduce a new class of lazy shortest path solvers that do not require iterative shortest path computations and that generalize existing lazy shortest path algorithms to incorporate both label correcting and label setting algorithms on order DAGs.

Definition 3.2 (Pruning SP Family of Algorithms)**.** Given a ϵ-lazy graph couple $(G, G^\epsilon_{lb}) \in C(G;\epsilon)$, where $w(i,j)$ is hidden from the algorithm. The Pruning SP family of algorithms, which we will refer to as $\mathcal{PSP}$. Each algorithm in the family maintains. A working graph G^ν_{work}, a vector $F^\nu \in \mathbb{R}^{T+1}$ where each element F^ν_j represents the current smallest cost for a path $P_{0\to j}(G)$ and a predecessor list pred^ν such that for all j, $\mathrm{pred}[j] = i$ if $F^\nu_j = F^\nu_i + w^\nu_{work}(i,j)$. Each algorithm also assumes access to three oracles. First a function $\mathrm{Eval}(i,j)$ which returns the true edge weight $w(i,j)$. Second a function $\mathrm{Selector}(S)$ which selects edges from a set of edges $S \subseteq E$. And third, an update function $\mathrm{Update}(F^\nu, E^\nu_{\text{selected}}, \mathrm{pred}^\nu)$ which returns a vector $F^{\nu+1}$ and $\mathrm{pred}^{\nu+1}$. The oracle functions must satisfy four properties.

1. Update returns $F^{\nu+1}$ such that all $(i,j) \in E^{\nu}_{\text{selected}}$ satisfy
$$F^{\nu+1}_j = \min\{F^{\nu}_j, F^{\nu+1}_i + w^{\nu+1}_{work}(i,j)\}$$
2. Update returns $\mathrm{pred}^{\nu+1}[j] = i$ if $F^{\nu+1}_j = F^{\nu+1}_i + w^{\nu+1}_{work}(i,j)$ for all $(i,j) \in E^{\nu}_{\text{selected}}$
3. Every iteration before termination must be effective, meaning, that $(w^{\nu+1}_{work}, F^{\nu+1}) \neq (w^{\nu}_{work}, F^{\nu})$.

Each algorithm $A \in \mathcal{PSP}$ returns the optimal cost as $F^{\bar{\nu}}_T$ of the terminal iteration $\bar{\nu}$ and the optimal path is retrieved by backtracking on the predecessors list. The algorithmic framework is summarized in algorithm 2

Algorithm 2 Pruning SP

```
1: Instantiate $G^0_{work} = (V, E, w_{work})$ with $w^0_{work}(i,j) = w_{lb}(i,j)$ as the working graph. Set $\nu = 0$
2: Choose an arbitrary path $\pi^0 \in P_{0\to T}(G)$ or set $F^{\nu}_j = \infty$ for all $j \leq T$
3: Set $\mathrm{pred}[j] = i$ for edge $(i,j) \in E(\pi)$.
4: Warm Start: If $\pi^0$ chosen, for $j = 0, \ldots, T$, let $s(j,\pi) = \max_{\tau_i \leq j} \tau_i$ set $w_{work}(e) = \mathrm{Eval}(e)$ for all $e \in E(\pi)$, and set
   $$F^0_j \leftarrow \sum_{q=1}^{s(j,\pi)} w(\tau_{i-1}, \tau_i) \quad + w(s(j), j)$$
   , and set $\mathrm{pred}^{\nu}[\tau_{i+1}] = \tau_i$ for all $i = 1, \ldots, s(j,\pi)$
5: loop
6:     Let $E^{\nu}_{\text{violated}} = \{(i,j) \in E : F^{\nu}_j > F^{\nu}_i + w^{\nu}_{work}(i,j)\}$
7:     if $E^{\nu}_{\text{violated}} = \phi$ then
8:         terminate
9:     end if
10:    $E^{\nu}_{\text{selected}} = \mathrm{Selector}(E^{\nu}_{\text{violated}})$
11:    For every $(i,j) \in E^{\nu}_{\text{selected}}$ set $w^{\nu+1}_{work}(i,j) = \mathrm{Eval}(i,j)$ if $w^{\nu}_{work}(i,j) \neq w(i,j)$
12:    Update $F^{\nu+1} \leftarrow \mathrm{Update}(F^{\nu}, E^{\nu}_{\text{selected}})$
13:    $\nu = \nu + 1$
14: end loop
15: return $c(\pi^*; G) = F^{\nu}_T$ and $\pi^* = \mathrm{BackTrack}(\mathrm{pred}^{\nu})$
```

We first show that such an algorithmic class does in fact yield the correct shortest path. The proof itself is very similar to other label-correcting correctness proofs.

Theorem 3.3 (Correctness of $A \in \mathcal{PSP}$)**.** *Given a finite ordered DAG $G \in \mathcal{D}(T)$. Any algorithm $A \in \mathcal{PSP}$ will terminate in finite steps and yield the true shortest path π^* and $c(\pi^*; G)$.*

3.1. $\mathcal{PSP}$ Instantiations

We note that each algorithm $A \in \mathcal{PSP}$ has behavior, in terms of total evaluations and max iterations, that is uniquely determined by its selector and update function. That that end we describe three different concrete instantiations of the $\mathcal{PSP}$ family of algorithms, one of which has label setting flavor while the others are label correcting algorithms.

Example 3.4 (Incremental Topological Sort)**.** The Incremental Topological Sort variant requires the following (Selector,Update) pair.
$$\mathrm{Selector}_{ITS}(E^{\nu}_{\text{violated}}) = \{(i,j) \in E^{\nu}_{\text{violated}} : j = 1, \ldots, \nu\}$$
and Update is defined as function 3, which is called on $j = \nu$ during the ν^{th} iteration.

Algorithm 3 Incremental Topological Sort $\mathrm{Update}(F^{\nu}, \mathrm{pred}^{\nu}, j)$

```
1: Input: $F^{\nu}, \mathrm{pred}^{\nu}$
2: $F^{\nu+1} \leftarrow F^{\nu}$,    $\mathrm{pred}^{\nu+1} \leftarrow \mathrm{pred}^{\nu}$
3: for each $i = 0, \ldots, j-1$ do
4:     if $F^{\nu+1}_j > F^{\nu+1}_i + w^{\nu+1}_{\text{work}}(i,j)$ then
5:         $F^{\nu+1}_j \leftarrow F^{\nu+1}_i + w^{\nu+1}_{\text{work}}(i,j)$
6:         $\mathrm{pred}^{\nu+1}[j] \leftarrow i$
7:     end if
8: end for
9: return $(F^{\nu+1}, \mathrm{pred}^{\nu+1})$
```

We note that Incremental Topological Sort variation is a member of the $\mathcal{PSP}$ class of algorithms and as such will return the true shortest path in a finite number of iterations. In fact we can show that the algorithm will terminate in $\nu = T$ iterations and so calling $\mathrm{Update}(F^{\nu}, \mathrm{pred}^{\nu}, j = \nu)$ is well defined at any point in the algorithm.

Proposition 3.5 (Iteration Complexity of Incremental Topological Sort)**.** *Incremental Topological Sort as defined in example 3.4 will terminate in $\nu = T$ iterations.*

Example 3.6 (First Edge & Greedy Edge Label Correcting)**.** The First Arc & Greedy Arc Label Corrector variations are defined as the following (Selector,Update) pairs. For the first arc label corrector the selector returns a single edge
$$\mathrm{Selector}_{FLC}(E^{\nu}_{\text{violated}}) = \min_{j \leq T}\{(i,j) \in E^{\nu}_{\text{violated}}\}$$
, whereas the greedy edge selector is defined as
$$\begin{aligned}&\mathrm{Selector}_{GLC}(E^{\nu}_{\text{violated}}) = \\ &\arg\max\{F^{\nu}_j - (F^{\nu}_i + w(i,j)) : (i,j) \in E^{\nu}_{\text{violated}}\}\end{aligned}$$
Note that both selectors return a single edge. and make use of the update function 4 which is called on the single edge $(i,j) \in E^{\nu}_{\text{selected}}$.

The first and greedy arc variations both are members of the $\mathcal{PSP}$ family of algorithms. Interestingly, however, they exhibit very different iteration complexity. This is primarily due to the fact that unlike the greedy arc variation, the first arc label correcting variation can guarantee that once an edge (i,j) exits the violated set it can never renter.

Proposition 3.7 (First Arc Label Correcting Iteration Complexity)**.** *The First Arc Label Correcting Algorithm will terminate in at worst $|E| = T(T+1)/2$ iterations.*

Algorithm 4 Label Correcting $\mathrm{Update}(F^{\nu}, \mathrm{pred}^{\nu}, (i,j))$

```
1: $F^{\nu+1} \leftarrow F^{\nu}, \quad \mathrm{pred}^{\nu+1} \leftarrow \mathrm{pred}^{\nu}$
2: for each $t = j+1, \ldots, T$ do
3:   if $F_t^{\nu+1} > F_j^{\nu+1} + w_{\text{work}}^{\nu+1}(j,t)$ then
4:     $F_t^{\nu+1} \leftarrow F_j^{\nu+1} + w_{\text{work}}^{\nu+1}(j,t)$
5:     $\mathrm{pred}^{\nu+1}[t] \leftarrow j$
6:   end if
7: end for
8: return $(F^{\nu+1}, \mathrm{pred}^{\nu+1})$
```

Proposition 3.8 (Greedy Arc Label Correcting Iteration Complexity)**.** *The Greedy Arc Label Correcting Algorithm will terminate in at worst* $(T^3 - T)/6$ *iterations.*

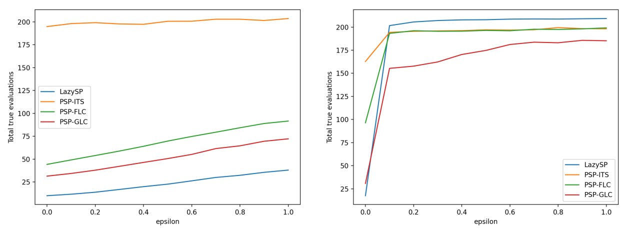
Through our analysis of the three concrete instantiations of the $\mathcal{PSP}$ family of algorithms, we've also come across an interesting phenomenon. Namely that some algorithms have a no-re violation property, where once an edge (i,j) has been removed for $E^{\nu}_{\text{violated}}$ then it $(i,j) \notin E^{r}_{\text{violated}}$ for all $r > \nu$. Such a property can be reduced to one core assumption which is present in both the first arc label corrector and incremental topological sort variations. Whether or not the algorithms preserve the invariant that if an edge (i,j) is violated and fixed then no edge (t,i) will be in the violated set for all future iterations. Such a property is direct result of ones choice of the $\mathrm{Selector}$ and Update functions. Such a distinction is also what separates label-correcting from label-setting type algorithms in $\mathcal{PSP}$. We conclude our analysis by noting that each algorithm in $\mathcal{PSP}$, which acts on a ϵ-lazy graph couple (G, G^{ϵ}_{lb}), can be warm-started with any path computed using the true edge weights $w(i,j)$.

We've shown that our algorithm class can accommodate both label-setting and label-correcting type methods, while also incorporating lazy evaluations similar to the LazySP family of algorithms.

4. Numerical Examples

Throughout our analysis we've made use of the object we call a Silo, and have found that well-defined Silos (i.e Silos with small Volumes) will likely exhibit a small number of true evaluations. We also discussed the PruningSP class as a label-correcting competitor to LazySP. In this section we study LazySP and PurningSP in two graph regimes: Graphs that permit Silos with large and small volumes.

To test the algorithms in both regimes we generate two classes of 100 fully connected ordered DAGs $G \in \mathcal{D}(T)$ with random weights and unique shortest path. The first class of graphs will be constructed to have many paths that are nearly tied for the position of shortest path (ill-defined Silos) and the second class will have few near ties. We also warm start all PSP algorithms and LazySP with a random path. In the case of LazySP where there is no native



(a) Well-defined silo (few near-optimal paths) *(b)* Ill-defined silo (many near-optimal paths)

Figure 4. Average total evaluations across 100 random DAGs in $\mathcal{D}(T)$ as a function of ϵ. We compare LazySP with Forward Selector (LazySP), Iterative TopSort (PSP-ITS), First Edge Label Corrector (PSP-FLC), and Greedy Edge Label Corrector (PSP-GLC).

capability to use a warm start solution we simply instantiate w^0_{work} with the same true weights $w(i,j)$ that the pruning shortest path algorithms receive.

We find that within the well defined Silo-Regime, as predicted by theorem 2.11, the LazySP class of algorithms perform beats pruning shortest path algorithms in terms of total evaluations.

However we find that in the case where the graphs have many nearly tied paths (ill-defined Silos), the Pruning Shortest path algorithms outperform LazySP. In particular the greedy label corrector is particularly promising. Additionally the pruning shortest path algorithms do not require iterative computation of the shortest path which also reduces the total runtime of the algorithms in comparison to LazySP.

All in all our numerical experiments support our earlier theoretical work, and indicate that the new class of $\mathcal{PSP}$ algorithms have the capacity to beat LazySP for graphs that do not permit well-defined silos.

5. Conclusion

In conclusion, we identified that lazy shortest paths algorithms specialized for ordered DAGs and that made use of label-correcting mechanisms were missing from the lazy shortest path literature. To close this gap we introduced a class of formalisms to formally study lazy shortest paths problems on ordered DAGs and proposed a lazy shortest path family of algorithms that bridged the gap between label setting and label correcting algorithms. We showed that empirically our family of algorithms outperforms the canonical lazy shortest path algorithm, LazySP, on graphs with ill-defined Silos.

6. Impact Statement

Our work primarily presents advances in the domain of lazy shortest path problems. Our proposed algorithms are applicable to a wide variety of fields. As such there are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Auger, I. E. and Lawrence, C. E. Algorithms for the optimal identification of segment neighborhoods. *Bulletin of Mathematical Biology*, 51(1):39–54, 1989. doi: 10.1007/BF02458835.

Dellin, C. M. and Srinivasa, S. S. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 459–467, 2016.

Haghtalab, N., Mackenzie, S., Procaccia, A. D., Salzman, O., and Srinivasa, S. S. The provable virtue of laziness in motion planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 106–113, 2018.

Jackson, B., Scargle, J., Barnes, D., Arabhi, S., Alt, A., Gioumousis, P., Gwin, E., San, P., Tan, L., and Tsai, T. T. An algorithm for optimal partitioning of data on an interval. *IEEE Signal Processing Letters*, 12(2): 105–108, February 2005a. ISSN 1070-9908. doi: 10.1109/lsp.2001.838216. URL http://dx.doi.org/10.1109/LSP.2001.838216.

Jackson, B., Scargle, J. D., Barnes, D., Arabhi, S., Alt, A., Gioumousis, P., Gwin, E., Sangtrakulcharoen, P., and Tan, L. An algorithm for optimal partitioning of data on an interval. *IEEE Signal Processing Letters*, 12(2):105–108, 2005b. doi: 10.1109/LSP.2001.838216.

Mandalika, U., Gammell, J. D., Srinivasa, S. S., and Pavone, M. Lazy receding-horizon a*: Trading planning quality for time in optimal motion planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

Stentz, A. The focussed d* algorithm for real-time replanning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pp. 1652–1659, August 1995. URL https://robotics.caltech.edu/~jwb/courses/ME132/handouts/Dstar_ijcai95.pdf.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Zheng, W. and Ye, H. Identification of switched linear system based on dynamic programming. In *2023 CAA Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS)*, pp. 1–5, 2023. doi: 10.1109/SAFEPROCESS58597.2023.10295779.

A. Proofs

A.1. Lemma 2.8: Sufficient Conditions for Shortest Path Equivalence

Proof. Because G^{ϵ}_{lb} is a ϵ-lazy graph, by definition of an ϵ-lazy graph couple, we know that for any path $\pi \in P_{0 \to T}(G)$ each edge weight can differ by at most ϵ and as a result

$$0 \leq c(\pi; G) - c(\pi; G^{\epsilon}_{lb}) \leq \epsilon|\pi| \leq \epsilon T$$

. Using the fact that G is δ-robust with shortest path π^* and the assumption that $\delta > \epsilon T$ we get that for all paths $\pi \in P_{0 \to T}(G)/\{\pi^*\}$

$$c(\pi; G) - c(\pi^*; G) > \delta > \epsilon T$$

Putting the two inequalities together, we get that for all $\pi \in P_{0 \to T}(G)$,

$$c(\pi; G^{\epsilon}_{lb}) > c(\pi; G) - \epsilon T > c(\pi^*; G^{\epsilon}_{lb})$$

. We conclude that because $c(\pi^*; G^{\epsilon}_{lb}) < c(\pi; G^{\epsilon}_{lb})$, π^* is also the shortest path of G^{ϵ}_{lb} □

A.2. Theorem 2.11, Corollary 2.12: Siloed Worst Case Behavior, Silo Containment

Proof. Assuming that every path π^{ν} selected during each iteration ν of an algorithm $A \in \mathcal{LSP}$ satisfies $\pi^{\nu} \in S_{(\gamma,\mu)}(\pi^*; G)$ then the worst case behavior, in terms of total evaluations will be to evaluate each unique edge of the paths in $S_{(\gamma,\mu)}(\pi^*; G)$. This implies that the worst case total evaluations is $\mathrm{Vol}(S_{(\gamma,\mu)}(\pi^*; G))$. What is left to show is this assumption is true. We proceed by induction. To show the base case at $\nu = 0$, we take an arbitrary $\pi \notin S_{(\gamma,\mu)}(\mu^*; G)$. By definition such paths must satisfy

$$c(\pi; G) \geq c(\pi^*; G) + \gamma + \mu$$

. Using the fact that $G^{0}_{work} = G^{\epsilon}_{lb}$ is a ϵ-lazy graph we have that

$$c(\pi; G^{0}_{work}) \geq c(\pi; G) - \epsilon|\pi|$$

. Using the fact that $|\pi| \leq L$ for $\pi \in S_{(\gamma,\mu)}(\pi^*; G)$ along with the above two inequalities we recover that

$$c(\pi; G^{0}_{work}) \geq c(\pi; G) - \epsilon L \geq c(\pi^*; G) + \gamma + \mu - \epsilon L$$

. Since $\gamma + \mu - \epsilon T > 0$ by assumption, and the fact that the cost of a path on the working graph is always less than the cost of the same path on the true graph, we get that

$$c(\pi; G^{0}_{work}) > c(\pi^*; G) \geq c(\pi^*; G^{0}_{work})$$

for all $\pi \notin S_{(\gamma,\mu)}(\pi^*; G)$. Since $\pi^* \in S_{(\gamma,\mu)}(\pi^*; G)$ by definition we conclude that all out-of-silo paths have cost greater than at least one path inside the silo and so the shortest path π^0 of G^{0}_{work} will be a member of $S_{(\gamma,\mu)}(\pi^*; G)$. Next we move on to the inductive step. We assume that the shortest path π^{ν} of G^{ν}_{work} is a member of $S_{(\gamma,\mu)}(\pi^*; G)$ and our aim is to show that the shortest path $\pi^{\nu+1}$ of the updated working graph $G^{\nu+1}_{work}$ is also a member of $S_{(\gamma,\mu)}(\pi^*; G)$. To show this we observe that at iteration ν at most L edges of the current path π^{ν} are selected and evaluated, which results in the updated working graph $G^{\nu+1}_{work}$. Note that $G^{\nu+1}_{work}$ is also an ϵ-lazy graph since all edge weights are still at most ϵ away from the true edge weights. Which means that we can proceed exactly as in the base case and conclude that the shortest path $\pi^{\nu+1}$ of $G^{\nu+1}_{work}$ is also a member of the silo $S_{(\gamma,\mu)}(\pi^*; G)$. □

A.3. Proposition 2.13: Sufficient Conditions for Best Case Performance

Proof. Since G is δ-robust with $\delta > \epsilon T$ Lemma 2.8 applies and we conclude that

$$\mathrm{ShortestPath}(G^{\epsilon}_{lb}) = \mathrm{ShortestPath}(G) = \pi^*$$

. Then what is left to show is that for all iterations ν until termination $\mathrm{ShortestPath}(G^{\nu}_{work})$ returns π^*. We proceed by induction by maintaining the hypothesis that G^{ν}_{work} is δ-robust with a shortest path of π^* for all iterations ν. At the first iteration ($\nu = 0$) $\mathrm{ShortestPath}(G^{0}_{work})$ will return π^* since $G^{0}_{work} = G^{\epsilon}_{lb}$. Additionally G^{0}_{work} is δ-robust by assumption on G^{ϵ}_{lb}. Next we move on to the inductive step where we assume that G^{ν}_{work} is δ-robust with shortest path π^*. Our aim is to show that $G^{\nu+1}_{work}$ is δ-robust with shortest path π^*. At iteration ν by the inductive hypothesis $\text{ShortestPath}(G^{\nu}_{work}) = \pi^*$. Next Selector will select $1 \leq k \leq |\pi^*| - \nu \leq |\pi^*| - 1$ edges from the optimal path π^* to evaluate. Assume that each of the k edges in $E_{selected}$ is evaluated, and the corresponding weights are updated in $G^{\nu+1}_{work}$. Note that G^{ν}_{work} is an ϵ lazy graph since all edge weights still satisfy $0 \leq w(i,j) - w^{\nu}_{work}(i,j) \leq \epsilon$. So each of the k updated edge weights will increase by at most ϵ. As a result any path π on the updated $G^{\nu+1}_{work}$ including π^* will have cost that increased by at most $\epsilon|\pi^*|$ and will satisfy

$$\begin{aligned} c(\pi; G^{\nu}_{work}) + \delta &> c(\pi; G^{\nu}_{work}) + \epsilon|\pi^*| \\ &\geq c(\pi; G^{\nu+1}_{work}) > c(\pi; G^{\nu}_{work}) \end{aligned}$$

,since by assumption $\delta > \epsilon|\pi^*|$. Because G^{ν}_{work} is a δ-robust graph by the inductive hypothesis, we know that any path $\pi \neq \pi^*$ on G^{ν}_{work} will have cost such that

$$c(\pi; G^{\nu}_{work}) > c(\pi^*; G^{\nu}_{work}) + \delta$$

. Putting the two inequalities together, we get that any path $\pi \neq \pi^*$ on the the recently updated G^{1}_{work} will satisfy

$$\begin{aligned} c(\pi; G^{\nu+1}_{work}) > c(\pi; G^{\nu}_{work}) &> c(\pi^*; G^{\nu}_{work}) + \delta \\ &> c(\pi^*; G^{\nu+1}_{work}) \end{aligned}$$

.Therefor at the next iteration $G^{\nu+1}_{work}$ will remain δ-robust with an optimal path of π^* and as a result $\text{ShortestPath}(G^{\nu+1}_{work})$ will reselect π^*.

This concludes the induction. So we conclude that because π^* is iteratively reselected and no repeat true evaluations are allowed for an algorithm $A \in \mathcal{LSP}$, then there will only be $|\pi^*|$ total evaluations before termination, since any algorithm $A \in \mathcal{LSP}$ terminates once the path returned by $\text{ShortestPath}(G^{\nu}_{work})$ has all edges evaluated. □

A.4. Corollary 2.15: Subgraph Dominance

Proof. Assume that for all $A \in \mathcal{LSP}$, A is augmented to first find the shortest path $\pi^{*(0,K)} = \{0, \tau_1, \ldots, \tau_r, K\}$ on the subgraph $G^{\epsilon}_{lb,(0,K)}$ then solve the shortest path problem on $G_{(\tau_r,T)}$. By Lemma 2.8, the shortest path $\pi^{*(0,K)}_{lb}$ of $G^{\epsilon}_{lb,(0,K)}$ will equal the shortest path $\pi^{*(0,K)}$ on $G_{(0,K)}$ which requires at most $|\pi^{*(0,K)}|$ true evaluations of $w(i,j)$ by proposition 2.13. Then by prefix dominance (assumption 3) the shortest path π^* on the full graph G must contain the prefix $\{0, \tau_1, \ldots,\ \tau_r\}$. The algorithm $A \in \mathcal{LSP}$ can then be called on the ϵ-lazy subgraph couple $(G_{(\tau_r,T)}, G^{\epsilon}_{lb,(\tau_r,T)})$ to $\pi^*_{(\tau_r,T)}$ which will require in the worst case the true evaluation of all $(T - \tau_r)(T - \tau_r - 1)/2$ edges in $G_{(\tau_r,T)}$. By the prefix dominance of G the shortest path π^* is the concatenation of $\{0, \tau_1, \ldots, \tau_r\}$ and $\pi^*_{\tau_r,T}$. So we conclude that the worst case total evaluations behavior of the augmented $A \in \mathcal{LSP}$ is $|\pi^*_{(\tau_r)}| + (T - \tau_r)(T - \tau_r - 1)/2$ □

A.5. Proposition 2.18: Splicing Regimes δ-Robustness

Proof. Under the structure of $w(i,j)$ the problem of finding the shortest path reduces to solving

$$\min_{\pi \in P_{0 \to T}(G)} \left(\min_{x \in \mathcal{X}} \sum_{(i,j) \in E(\pi)} g(x^{[i:j]}) \right)$$

Since g is α-strongly convex in x and the mapping from $x \to x^{[i:j]}$ is linear,$g(x^{[i:j]})$ remains α-strongly convex. Therefore the sum $F(\pi, x) = \sum_{(i,j) \in E(\pi)} g(x^{[i:j]})$ is also α-strongly convex in x. Recall from definition 2.17 that x^*_{π} is the optimal parameters for a given path π. Then strong convexity of F in x implies that for all paths π

$$F(\pi, x) - c(\pi; G) \geq \frac{\alpha}{2} \|x - x^*_{\pi}\|^2_F, \quad \forall x \in \mathcal{X}$$

,where we use the fact that $F(\pi, x^*_{\pi}) = c(\pi; G)$. Let π^* be the shortest path of G, then plugging $x^*_{\pi^*}$ in to the above inequality yields

$$F(\pi, x^*_{\pi^*}) - c(\pi^*; G) \geq \frac{\alpha}{2} \|x^*_{\pi^*} - x^*_{\pi}\|^2_F$$

By the fact the $c(\pi; G) = \min_x F(\pi, x)$ we get that

$$c(\pi; G) - c(\pi^*; G) \geq F(\pi, x^*_{\pi^*}) - c(\pi^*; G) \geq \frac{\alpha}{2}\|x^*_{\pi^*} - x^*_{\pi}\|_F^2$$

So by assumption (2) we conclude that G is $(\alpha/2)\delta^2$-robust. □

A.6. Proposition 2.19: Convexity in Edge Length δ-Robustness

Proof. Since $h(x, l)$ is convex l for every fixed x we know by Danskins Theorem that

$$k(l) = \min_{x\in\mathcal{X}} h(x, l) = -\min_{x\in\mathcal{X}} -h(x, l) = \max_x h(x, l)$$

is also convex in l. As a result $w(i, j) = f(l) + k(l) + \lambda$ is strongly convex in l. Let π_m be an m-path of G. So the sum of edge weights

$$c(\pi_m; G) = m\lambda + \sum_{(i,j)\in E(\pi_m)} f(j - i) + k(j - i)$$

is α discrete strongly convex in $l_{\pi_m} = (l_1, \dots, l_m) \in \{z \in \mathbb{Z}^m_+ : \sum_{i=1}^m z_i = T\} = \mathcal{L}_m$. Therefore

$$c(\pi_m; G) - c(\pi^*_m) \geq \frac{\alpha}{2}\|l_{\pi_m} - l_{\pi^*_m}\|_2^2$$

Note that for any two distinct m-paths we have that $\|l_{\pi_m} - l_{\pi'_m}\|_2^2 \geq 2$, therefore

$$c(\pi_m) - c(\pi^*_m) \geq \alpha\mathbb{I}_{\pi_m \neq \pi^*_m}$$

, where $\mathbb{I}$ is the binary indicator function. □

A.7. Proposition 2.20: Volume Bounds

Proof. Denote the shortest path on G as π^* and $\pi \in S_{(\gamma,\mu)}(\pi^*; G)$. By the assumption on c and the definition of the silo $S_{(\gamma,\mu)}(\pi : G)$ we get that

$$\gamma \geq c(\pi; G) - c(\pi^*; G) \geq \frac{\alpha}{2}\|\phi(\pi) - \phi(\pi')\|_2^2.$$

Which implies that

$$\|\phi(\pi) - \phi(\pi^*)\|_2^2 \leq \frac{2\gamma}{\alpha} \implies \|\phi(\pi) - \phi(\pi^*)\|_\infty \leq \sqrt{\frac{2\gamma}{\alpha}}$$

Using the fact the ϕ is injective with integer lattice domain we that

$$|S_{(\gamma,\mu)}(\pi^*; G)| \leq \Big(2\Big\lfloor\sqrt{\frac{2\gamma}{\alpha}}\Big\rfloor + 1\Big)^d.$$

Note that if there are K distinct paths in the silo, then at worst there is 1 path with T edges, $\lfloor T/2\rfloor$ paths with $T - 1$ edges, and so on. So we know that there are at most $\sum_{i=1}^K \lfloor T/i\rfloor$ unique edges contained in the K paths within the silo. So in the worst case where each path in the silo has distinct edges we get that

$$\mathrm{Vol}(S_{(\gamma,\mu)}(\pi^*; G)) \leq \min\Big\{\frac{T(T+1)}{2}, \sum_{i=1}^{|S_{(\gamma,\mu)}(\pi^*;G)|} \Big\lfloor\frac{T}{i}\Big\rfloor\Big\}$$

□

A.8. Proposition 3.1: Runtime Winner

Proof. By proposition 2.11, an algorithm $A \in \mathcal{LSP}$ will be contained (siloed) to a silo $S_{(\gamma,\mu)}(\pi^*; G)$ and as such will only require the computation of at worst $\mathrm{Vol}(S_{(\gamma,\mu)}(\pi^*; G)) = \mathrm{Vol}(N_\gamma(\pi^*; G))$, where π^* is the shortest path of G. Therefore the total runtime cost of computing the shortest path π^* directly on G (LHS) will be smaller then the total runtime cost of the algorithm A (RHS) if

$$\mathcal{T}\frac{T(T+1)}{2} + \mathcal{T}_{sp} \leq \mathcal{T}_{lb}\frac{T(T+1)}{2} + \mathcal{T}|N_\gamma(\pi^*; G)| + \mathcal{V}\mathcal{T}_{sp}$$

Simplifying the expression yields

$$\mathcal{T}_{sp} \leq \frac{1}{\mathcal{V}-1}\Big[|E|(\mathcal{T} - \mathcal{T}_{lb}) + \mathcal{T}|N_\gamma(\pi^*; G)|\Big]$$

□

A.9. Theorem 3.3: Correctness of $A \in \mathcal{PSP}$

Proof. Recall that $w^\nu_{work}(i,j) \leq w(i,j)$ since $w^0_{work}(i,j) = w_{lb}(i,j) \leq w(i,j)$ and $w^\nu_{work}(i,j)$ is only updated via the $\mathrm{Eval}(i,j)$ function. Assume that for some ν, $E^\nu_{\text{violated}} = \phi$. Then

$$F^\nu_j \leq F^\nu_i + w^\nu_{work}(i,j) \leq F^\nu_i + w(i,j), \quad \forall (i,j) \in E$$

. Let $\pi \in P_{0\to T}(G)$ be an arbitrary path with nodes $\{0, \tau_1, \ldots, \tau_m, T\}$. Unwinding the recursion above over the path π yields

$$\begin{aligned} F^\nu_T &\leq F^\nu_{\tau_m} + w(\tau_m, T) \\ &\leq F^\nu_{\tau_{m-1}} + w(\tau_{m-1}, \tau_m) + w(\tau_m, T) \\ &\leq F^\nu_0 + \sum_{q=1}^{m} w(\tau_{q-1}, \tau_q) + w(\tau_m, T) \\ &= \sum_{q=1}^{m} w(\tau_{q-1}, \tau_q) + w(\tau_m, T) = c(\pi; G) \end{aligned}$$

Let π^* be the path retrieved via backtracking on pred^ν. Given line 10 and 11 of algorithm 2 and by assumption on the update function we know that every predecessor pair $\mathrm{pred}^\nu[j] = i$ if $F^\nu_j = F^\nu_i + w(i,j)$. This means that F^ν_T represents the cost of the concrete path π^*. So we conclude that $c(\pi^*; G) = F^\nu_T \leq c(\pi; G)$ and that π^* is the shortest path. What is left to show is the $E^\nu_{\text{violated}} = \phi$ for some finite ν. Assume that $(i,j) \in E^\nu_{\text{violated}}$, then by assumption on Update and after the evaluation step which sets $w^{\nu+1}_{work}(i,j) = w(i,j)$ we get that

$$F^{\nu+1}_j \leq F^{\nu+1}_i + w(i,j)$$

. Since the F^ν_j update assumption implies that $F^{\nu+1}_j$ can not increase for all remaining iterations, the only way that (i,j) can re-enter the violation set is if ,at some future iteration r, an edge $(s,i) \in E^r_{\text{selected}}$ such that $0 \leq s < i$. During the update on (s,i) we get that

$$F^{r+1}_i \leq F^{r+1}_s + w(s,i)$$

and that $F^{r+1}_i < F^{\nu+1}_i$ may be true, which could result in

$$F^{r+1}_j > F^{r+1}_i + w(i,j)$$

, which in turn would force (i,j) back into $E^{r+1}_{\text{violated}}$. Given that $G \in \mathcal{D}(T)$ we know that there are exactly i such edges and so (i,j) may re-enter the violations set at most i times. Therefor after at most

$$\sum_{i=0}^{T-1} i(T-i) = \frac{T^3 - T}{6}$$

iterations the violated set will become empty. □

A.10. Proposition 3.5: Iteration Complexity of Incremental Topological Sort

Proof. Recall that any algorithm $A \in \mathcal{PSP}$ terminates when $E^{\nu}_{\text{violated}} = \phi$, at which point the optimal path and cost are returned by prop 3.3. What is left to show is that at $\nu = T$ iterations $E^{\nu}_{\text{violated}} = \phi$. We proceed by induction on the invariant that at iteration j, $(s,t) \notin E^{\nu}_{\text{violated}}$ for all $(s,t) \in E$ such that $t \leq j$. At the base case of $\nu = 0$, this condition trivially holds. Next we prove the inductive step and assume that the invariant holds at $\nu = j$. At iteration $\nu = j+1$, Selector_{ITS} will select an either select an edge of the form $(i, j+1)$ or select no edges if there are no violated edges of the form $(i, j+1)$. In the case where no edges are selected the inductive step trivially holds. In the case where the and arc $(i, j+1)$ is selected then the update step will ensure that $F^{\nu}_{j+1} \leq F_i + w(i, j+1)$. Since F_{j+1} cannot decrease by definition the edge $(i, j+1)$ will be re violated only if F^r_i sufficiently decreases at some future iteration r such that $F^r_{j+1} > F^r_i + w(i, j+1)$. This can only happen if an edge (s,i) is selected at iteration r such that $F^r_i > F^r_s + w(s,i)$. However by construction of the selector an edge (s,i) can not be selected after the i^{th} iteration and so we conclude that the edge $(i, j+1)$ cannot be re violated. This concludes the induction. And so by iteration T, $(i,j) \notin E^{T}_{\text{violated}}$ for all edges $(i,j) \in E$ such that $j \leq T$, which implies that $E^{T}_{\text{violated}} = \phi$. □

A.11. Proposition 3.7:First Arc Label Correcting Iteration Complexity

Proof. By construction once an edge (i,j) we can conclude that no edge (s,i) for $s < i$ is violated. And so in the worst case each edge $(0,1), (0,2), \ldots, (0,T), (1,T), (2,T), \ldots, (T-1,T)$ is selected one after another. In which case $E^{|E|}_{\text{violated}} = \phi$, the algorithm terminates in $|E|$ iterations, and yields the true shortest path and cost by proposition 3.3. □

A.12. Proposition 3.8: Greedy Arc Label Correcting Iteration Complexity

Proof. Because the algorithm selects edges in non-increasing order its possible that if an edge (i,j) is selected an earlier edge (s,i) is still in the violated set. At which point selecting (s,i) at a later iteration has the potential to re-violated edge (i,j). Consider that F^r_i sufficiently decreases at some future iteration r such that $F^r_i > F^r_s + w(i,s)$. In which case at iteration r it may be true that $F^r_{j+1} > F^r_i + w(i, j+1)$. So in the worst case each edge (i,j) is reselected i times. Which means that the after, at worst,

$$\sum_{i=0}^{T-1} i(T-i) = \frac{T^3 - T}{6}$$

iterations, $E^{(T^3-T)/6}_{\text{violated}} = \phi$, and the algorithm will terminate with the correct shortest path/ smallest cost by proposition 3.3. □