A Simple and General Operational Framework to Deploy Optimal Routes with Source Routing

Quentin Bramas, Jean-Romain Luttringer, Pascal Mérindol

A Simple and General Operational Framework to Deploy Optimal Routes with Source Routing

Quentin Bramas, Jean-Romain Luttringer, Pascal Mérindol

***Abstract*—Source Routing, currently facilitated by Segment Routing (SR), enables precise control of forwarding paths by specifying detours (or *segments*) to deviate IP packets along routes with advanced properties beyond typical shortest IGP paths. Computing the desired optimal segment lists, known as *encoding*, leads to interesting challenges as the number of detours is tightly constrained for hardware performance. Existing solutions either lack generality, correctness, optimality, or practical computing efficiency – in particular for sparse realistic networks. In this paper, we address all such challenges with GOFOR-SR. Our framework extends usual path computation algorithms to inherently look at optimal and feasible segment lists, streamlining the deployment of TE-compliant paths. By integrating encoding within the path computation itself and modifying the distance comparison method, GOFOR allows algorithms with various optimization objectives to efficiently compute optimal segment lists. Despite the loss of substructure optimality induced by SR, GOFOR proves particularly efficient, inducing only a linear overhead at worst. It also offers different strategies and path diversity options for intricate TE-aware load-balancing. We formally prove the correctness and optimality of GOFOR, implement our framework for various practical use-cases, and demonstrate its performance and benefits on both real and challenging topologies.**

***Index Terms*—Segment Routing, Path Computation, Operator Networks, Traffic Engineering, Optimization**

I. Introduction

IP networks typically employ a best-effort, hop-by-hop routing paradigm. Packets follow paths minimizing the IGP cost, an additive metric considering link bandwidth, delay, or some operator design intent. While this approach offers some level of performance, it lacks more elaborated forwarding guarantees.

Despite the scalability advantages of considering uni-dimensional paths, some use cases require more robust or intricate routes. For instance, specific paths are necessary for circumventing failures (e.g., as computed by TI-LFA [1]). Premium real-time flows may require specific paths ensuring guarantees both in bandwidth and latency (e.g., as computed by solving the Delay Constrained Least Cost problem, or DCLC [2]).

Such paths may be deployed over best-effort ones through *Source routing*. Source routing enables the source (*e.g.,* an edge provider router) to enforce paths with intricate properties. Packets are encapsulated to convey forwarding instructions to downstream routers, ensuring adherence to the desired path instead of best-effort routes. These instructions typically do not specify the path in its entirety, but encode it *loosely* as a list of mandatory checkpoints that the packet must go through (following the shortest IGP paths between each checkpoint by default).

All authors are within the ICube laboratory, University of Strasbourg, France.

Despite its benefits, the (loose) source-routing paradigm encountered deployment challenges due to cumbersome troubleshooting and significant protocol complexity in existing control planes (e.g., RSVP-TE). Consequently, large-scale adoption of fine-grained Traffic Engineering (TE) through source routing was infrequent [3].

More recently, Segment Routing [4] (SR) introduced a scalable and lightweight implementation of the loose source-routing paradigm. SR sparked the interest from both academia and the industry. As of now, the majority of network operators deploy (or plan to deploy) SR for various use-cases, including Traffic-Engineering [5].

SR allows the source to prepend *segments* to each packet. These segments typically represent nodes or links within the network, serving as mandatory *detours* from the standard IGP paths. There are two main types of segments:

- An adjacency segment designates a specific link that the packet should traverse.
- A node segment designates a particular node that the packet should pass through. Packets are forwarded to the node along the best path(s) on the IGP topology. SR supports multi-topologies [4], allowing for example to follow least-delay sub-paths rather than IGP ones.

Various types of segments can be combined to encode any desired path. To deploy optimal paths with respect to the Traffic Engineering (TE) objective, it thus becomes necessary to compute the appropriate *segment lists* and not only the paths themselves.

The computation of segment lists introduces an extra challenge: respecting the Maximum Segment Depth (MSD). This limit varies based on hardware capabilities, ranging from as few as 3 to up to 10 segments at line-rate [6]. Minimizing or at least controlling the number of segments when computing segment lists thus becomes critical.

In this paper we present our recipe to solve this problem in a generic and efficient manner. Given *any* objectives, our framework computes the optimal segment lists respecting the MSD limit. Before presenting in detail our achievements, we first illustrate the challenges raised by the problem we tackle.

Encoding optimal paths into segment lists is not enough*:* Translating a given optimal path into a minimal segment lists, known as the *path encoding problem*, leads to some algorithmic challenges. The encoding itself is arguably not the main one. However, encoding a (pre-computed) optimal path may lead to a segment list exceeding the MSD limit,

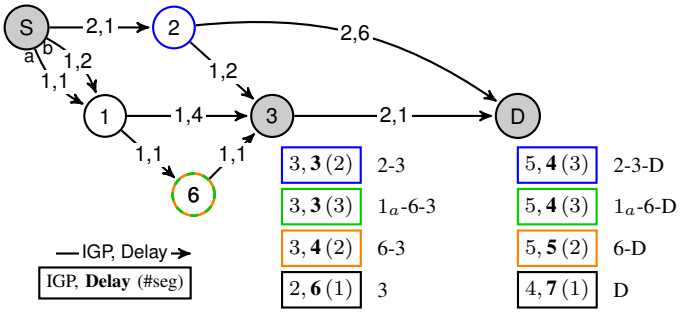

Figure 1. A Least-Delay use case highlighting the limit of an *a posteriori* encoding scheme and the loss of isotonicity. The orange path may be pruned at node 3 despite it becoming optimal if MSD is set to 2 on S.

as optimizing the number of segments may not align with optimizing the TE objective. This implies that path encoding must be performed *during* the path computation, in order to account for the number of segments when extending paths.

This phenomenon is familiar within the context of computing multi-metric constrained paths. Considering the number of segments as an additional metric, it is well-known that, to find solutions satisfying a constraint on a metric, all *non-dominated* paths must be extended for optimality correctness[1]. However, it works only if the metrics are isotone (*i.e.,* an optimal path is composed of optimal subpaths), but **this fundamental property does not hold when considering source routing**. This issue being at the core of our contribution, we precisely illustrate it in the following paragraph.

Illustration of the challenges: MSD constraint & Isotonicity*:* We now show why ignoring some dominated distances may lead to incorrect (possibly infeasible, *i.e.,* non-deployable) solutions.

We illustrate this challenge motivating the use of GOFOR-SR on a basic use-case (Fig. 1). The objective is to minimize the delay (called *Least-Delay* in this paper). We restrict our analysis to adjacency and IGP node segments. Fig. 1 shows a multi-valuated graph, where each link exhibits both an IGP cost and a delay. Possible distances to reach nodes (3) and (D) are shown below said nodes, exhibiting the IGP cost, the delay (and the required number of segments). The associated segment lists can be seen on the right side of their respective distances.

The first observation is that reaching (D) with a delay at most 4 requires at least 3 segments, but if MSD is set to 2 at (S), the orange path has to be chosen. Pre-computing the blue or the green optimal paths (delay-wise) and applying an encoding a posteriori would result in segment lists exceeding a tight MSD constraint. The orange segment list, while being dominated by the blue segment list at node (3), becomes the best option with at most 2 segments to reach (D).

In general, distances being suboptimal at an intermediate node are not supposed to be extended further (as the latter should not possibly lead to optimal distances towards downstream nodes). This fundamental property, called *isotonicity* or subpath optimality, is *essential* to bound the worst-case complexity of the path computation (as only a manageable number of distances are extended).

In Fig. 1, when exploring the graph, this standard behavior would thus dictate to only extend the blue distance from node (3) onward. Indeed, the orange distance is dominated as it has a higher delay while not offering better options than the blue distance on other metrics. The green distance has an optimal delay, but also require more segments than its blue counterpart.

The surprising effect occurs when extending these intermediate distances by the edge (3)–(D): the blue segment list now requires an additional segment to encode the desired distance while it is not the case for the two others. Starting from the current detour (2), adding a single node segment to (D) does not encode the distance exhibited by the path (S)–(2)–(3)–(D), as flows will follow the link (2)–(D), which exhibits a higher delay. Thus, while encoding the distances of the path (S)–(2)–(3) required 2 segments, encoding the distances of the path (S)–(2)–(3)–(D) requires one more segment.

On their sides, the green and orange segment lists do not evolve in the same fashion: they do not require more segments to encode the additional edge. Let us consider the green path: while encoding the distances of the path (S)–(1)–(6)–(3) already required 3 segments, encoding the extended path (S)–(1)–(6)–(3)–(D) just requires to modify the last node segment ((3)) by a node segment targeting (D).

The fact that some (but not all) segment lists require an additional segment when extended implies that at least some dominated segment lists *should* be extended, as it is the case for the orange one in the example. Moreover, if one aims at retrieving *all* optimal segment lists (*e.g.,* to perform TE-aware load balancing), the green segment list should also be extended, as it becomes equivalent to the blue one after the extension by the edge (3)–(D).

This observation bears a drastic consequence: it seems that to properly *wrap* SR around existing algorithms (*i.e.,* augmenting them to compute minimal and optimal segment lists), *all* paths should be extended to ensure that the optimal solution is found. This challenge leads to the main research question of this paper:

> **Research Question**
>
> **How can we extend path computation methods** (with additive metrics and isotone properties), **to correctly, optimally and *efficiently* wrap SR around them**?

Compared to the related works, we manage to answer this research question while remaining efficient and avoiding the use of heavy data-structures and graph transformation.

Main Achievements*:* Despite the practical interest of this research question, there exists a clear gap in the literature for generic means to augment path computation algorithms with the ability to consider SR both properly and efficiently.

Our answer is **GOFOR**-SR, a ***G**eneral **O**perational **F**ramework for **O**ptimal **R**outes* with SR. GOFOR is a simple

[1]In a multi-criteria context, a path is *optimal* (or *non-dominated*) if it is better on at least one metric than any other path towards the same node. Intuitively, a dominated path, being worse on all metrics, cannot become optimal later on and may be pruned from the exploration.

and generic framework that enables path computation algorithms to efficiently compute optimal segment lists. Despite the complexity of the problem, GOFOR allows their retrieval with linear overhead at worst. As the first generic and optimal solution for generic segment lists computation, GOFOR not only paves the way for easy deployments of SR in various use cases, but also for intricate TE-aware load balancing.

Our contributions are as follows:

1) The formalization of a new SR model for the problem(s) we tackle. We introduce the notions of path encoding, **SR wrapping (constrained or lexicographical), multi-topology segment lists, and path diversity strategies** in Sec. II;
2) A loose, ECMP-friendly, encoding scheme able to compute minimal segment lists possessing a given set of properties. This encoding is the first to handle multi-topologies, and is performed with a linear time overhead (wrt. #nodes) during path exploration (Sec. III-A);
3) **An extended dominance function in order to re-ensure an isotone relation** (Sec. III-B). We define a new, SR-aware dominance to correctly compare segment lists. This relation can be implemented within any traditional shortest path algorithm along with our encoding scheme to ensure that the optimal deployable segment lists are found for only a linear overhead (wrt. #nodes) (Sec. IV).
4) A C implementation of GOFOR and three modules showing how our framework can be used to properly wrap SR around algorithms computing (i) Delay Constrained Least Cost paths, (ii) Least Delay paths, and (iii) Best IGP paths avoiding a link failure. We experimentally show that GOFOR-SR allows reaching better performance than concurrent approaches (Sec. V). The code of our implementation is available online [7] and a python notebook is provided to ease the reproducibility of our experiments [7].

II. The SR-Wrapped Problem in a Formal Nutshell

A. Notations & Models

Let $G(E, V, w)$ be a directed graph with V the set of vertices, E the set of edges, and $w : \mathbb{N}_+^k$ as the weight function mapping each edge to its weight vector $(w_1(e), \ldots, w_k(e))$. Observe that initial metrics are indexed from 1 to k; metric with index 1 denotes the IGP distance – while, as formally defined later, metric with index 0 will represent the number of segments. Other components may be any desired additive metrics. For a path p, $p[i]$ denotes the i-th traversed node of p and $p[i:j]$ denotes the subpath of p from $p[i]$ to $p[j]$.

Segments*:* A segment $S = Seg(S^{type}, S^{src}, S^{dst})$ represents a set of paths from a source S^{src} to a destination S^{dst}. The set depends also on its type S^{type}, defining the kind of segment in use (either directly an adjacence or a subpath satisfying a distance regarding a given metric). If no path exists from S^{src} to S^{dst} with the property associated with S^{type}, we say that the segment is empty. Let S be a segment and $e = (S^{dst}, v)$ an edge. We say that S can be extended by e if the segment $Seg(S^{type}, S^{src}, v)$ is not empty and contains a path (of type S^{type}) from S^{src} to v passing through e. We denote by $S \circ e = Seg(S^{type}, S^{src}, v)$ this extension of S by e, otherwise we write $S \circ e = \emptyset$.

A segment satisfies two properties: (a) if p is a path in S, then its sub-segments are non-empty: $\forall i < j, (S^{type}, p[i], p[j]) \neq \emptyset$; and (b) if a path p is in a segment S and $S \circ e$ is not empty, then $p \circ e$ is in $S \circ e$.

The most common types of segments are *adjacency*, denoted Adj, and *node*, denoted $Node$. If $S^{type} = Adj$, S simply represents the edge (S^{src}, S^{dst}) if it exists, and is empty otherwise. By construction, an adjacency segment cannot be extended, so it verifies the two properties above.

If $S^{type} = Node_i$, S represents the set of paths from S^{src} to S^{dst} minimizing the i-th metric w_i. In this context, $Node$ stands for "all the best paths towards the destination S^{dst}". Node segments satisfy the two properties above because sub-paths of shortest paths are also shortest paths.

In this paper, and in particular in our examples, we mainly consider segments with types Adj and $Node_1$ (IGP based node segments as they are the only ones provided by default with SR), but our recipe works for any type of segments and their combination. GOFOR can also support IPv6 segments through the $NodeAdj_i$ segment type [2]. A $NodeAdj_i$ segment between u and v refers either (i) to the $(Node_i, u, v)$ segment if it is not empty, or (ii) to all the paths obtained by concatenated a path in $(Node_i, u, v')$ with the edge (v', v), for an arbitrary neighbor v' of v. If no such neighbor exists, then the segment is empty. In the first case, the segment satisfies its two required properties because it is a node segment. In the second case, the segment is not extendable[3], so it also satisfies the two properties (a and b). The set of types of segments (*i.e.,* $Node_i$, Adj and $NodeAdj_i$) is denoted $SegTypes$.

Distances*:* We assume all the metrics are additive. Therefore, the distance $d(p)$ of a path p is simply the sum of the weights of its edges, *i.e.,* for each metric i we have $d_i(p) = \sum_{e \in p} w_i(e)$.

As a segment S represents a set of paths, its distance $d(S)$ is defined as the maximum distance among all paths in S *for each metric*. In other words, $d_i(S) = \max_{p \in S} d_i(p)$. Note that $d(S)$ may not correspond to the distance of a specific path in S, since the maximum value for each metric might not be attained by the same path. Furthermore, observe that all the paths in a segment $Seg(Node_i, u, v)$ have the same distance for the metric i, since, by definition, they all minimize this metric between u to v. Finally, we denote Γ the size of the Pareto Front (*i.e.,* the number of non-dominated paths) induced by the set of the k metrics (at least with indexes 0 and 1).

Segment Lists*:* A segment list L is defined as a finite sequence of non-empty segments $L = (S_1, S_2, \ldots, S_l)$. In a segment list, each segment S_i starts at the end of the previous segment: more formally, for all $1 < i \leq l$, $S_{i-1}^{dst} = S_i^{src}$.

The distance $d(L)$ of a segment list L is the sum of the distances of its segments, *i.e.,* for each metric: $d_i(L) =$

[2]In SR-MPLS, guiding a packet through a link (u, v) involves two segments: first, a node segment to direct the packet to u, then followed by the relevant adjacency segment (whose local significance varies depending on the interpreting node). With SRv6, accomplishing this task requires just a single adjacency segment as the local ports of each node are globally broadcasted.

[3]Otherwise it implies that the corresponding node segment is non-empty and thus contradicts the fact that the second case is considered.

$\sum_{j=1}^{l} d_i(S_j)$. Moreover, we consider the number of segments as the 0-th metric, so that $d_0(L) = l$ (d_0 not being defined for paths). Note that $d_0(L)$ must be lower than MSD if the number of segments is constrained. A segment list distance $d(.)$ is a vector in $\mathbb{N}_+^{k+1}$.

We say that a segment list L can be edge-extended by an edge e if e can be concatenated to the last segment of L *i.e.,* $S_l \circ e$ is not empty. In this case the resulting segment list is denoted $L \circ e = (S_1, S_2, \ldots, S_l \circ e)$. Two segment lists L_1 and L_2 can be concatenated if the destination of the last segment in L_1 matches the source of the first segment in L_2. This concatenation is denoted $L_1 \oplus L_2$.

Two Encoding Paradigms*:* Encoding a path p consists in finding a list of detours encoding paths exhibiting properties of which p is a representative. We define two types of encoding.

Definition 1 (Strict Encoding)**.** *We say a segment list L strictly encodes a path p if L is a partition of p. In other words, each segment in L contains only one subpath of p and L has the same source and destination as p.*

Definition 2 (Loose Encoding)**.** *We say that a segment list L loosely encodes a path p if each segment in L contains a subpath of p, L has the same source and destination as p, and $d(L) = d(p)$ (in this equality we ignore the number of segments, since it is not defined for paths).*

Strict encoding is useful when considering use-cases such as monitoring or services chaining, in which the structure of the path must be enforced. While this type of encoding is commonly found in the literature [8], [9], loose encoding is better suited for most usual use-cases as it enables shorter segment lists benefiting from multiple load balanced paths (ECMP) having all bounded guarantees.

On the contrary to the strict paradigm, a packet that is source-routed through the loose encoding of a path p may not follow p (or just partially) but its effective route will have the same (or better) distance(s). For example, when relying on the loose paradigm to encode a least-delay path p (typically without using node segments of this type), this encoding ensures that any path in the resulting segment list L will possess the same delay as p, or a better one. Note that p remains within the set of paths encoded by L.

This paradigm is the most suited when the distance(s) matter more than the structure of the path itself, which is often the case in practice, and enables to mitigate $d_0(L)$. Note that topological properties (*e.g.,* avoiding a failed component) can also be enforced, and many other usecases can be enforced too (*e.g.,* applying a given instruction on a given node such as a firewall – or just going through an arbitrary node).

The SR Graph*:* SR-aware routers need network-wide knowledge of available segments to build paths. As segments encode shortest paths, computing all possible segments requires running an All-Pair-Shortest-Path algorithm.

The resulting segment database can be organized as a graph, commonly referred to as the SR Graph, where an edge (u, v) represents either a node segment $(Node_1, u, v)$ or an adjacency segment (Adj, u, v) (the edge weights being set as $d(S)$ within the graph) [2], [10].

Associated algorithms usually explore the SR Graph directly, exploiting the fact that paths computed on this graph are, in fact and by design, segment lists. This eliminates the need for a loose conversion algorithm to convert paths to segment lists, and allows them to terminate exploration once paths exceed MSD hops.

However, inherently if originally connected, the SR Graph is fully-meshed and so has a quadratic number of edges ($|E| = |V|^2$), significantly denser than the underlying initial network graph (where $|E|$ is generally similar to $|V|$ up to a multiplicative scalar factor). Exploring the SR Graph is costly and requires specialized algorithms not easily generalized to all use-cases.

GOFOR takes a different approach. It doesn't mandate the algorithms it extends to handle the complete density of an SR Graph. Instead, SR-wrapped algorithms can still explore the original (sparse) network graph. GOFOR uses the SR Graph indirectly as an efficient lookup table to facilitate the conversion of paths to segment lists. This approach enables our framework to be the most efficient in realistic cases as it leverages the sparsity of the initial network graph.

B. Problem Statement: How to SR-Wrap a given Path Computation Algorithm ?

Our main goal with GOFOR is to adapt an algorithm capable of solving a given (non-SR) problem to its SR-related version. We call this process *SR-wrapping* as it adds the SR metric (*i.e.,* the number of segments) and MSD constraint to the initial problem. Generally, a path computation problem is defined by a set of (optional) properties (*e.g.,* avoiding a link or a node), (optional) constraints that the solutions should verify (*e.g.,* keeping the delay under a given threshold), plus an objective function that the operator aims to optimize (*e.g.,* minimizing the delay).

Properties, Initial Constraint(s) and Objective(s)*:* We define the set of properties as a predicate $Properties$. Considering a graph G and a destination v, $Properties$ takes as input a path p and returns *true* if p has destination v and is a valid solution to the problem in G. For a segment S, $Properties(S)$ is true if the predicate is verified $\forall p \in S$. We assume $Properties$ to be isotonic: if $Properties(p)$ is true, then $Properties(p')$ is true for any subpath p' of p. The typical example we have in mind is: the sequence of edges in p does not include any failed components.

The initial constraints, if any, are defined as a vector $\mathcal{C}_{\backslash 0} = (c_1, \ldots, c_k)$, where each c_i sets an upper bound on the distance d_i that must be satisfied for all returned path p. Formally, we have $d(p) < \mathcal{C}_{\backslash 0}$. Only minimizing a given d_i, without constraining it, is equivalent to set $c_i = \infty$.

The objective function can be defined as a *comparison relation* $\leqslant_{\backslash 0}$, that applies to the distances, hence define the paths' distances ranking in the initial problem. The relation $\leqslant_{\backslash 0}$ could either represent a totally ordered relation with a given lexicographical ranking (*e.g.,* sorting paths according to their delay or cost, or both one after the other), or more intricate partially ordered relations such as Pareto-optimality (inducing a non-trivial Pareto front, *i.e.,* $\Gamma > 1$), in particular when considering multi-constrained problems.

Overall, $\leqslant_{\setminus 0}$ denotes the initial ordered relation (partial or total)[4] , while $<_{\setminus 0}$ denotes its associated strict counterpart.

Towards the SR-Wrapped Problem*:* From the inputs described above – covering most of the basic path computation problems – we will now define $\mathcal{P}$, as a problem consisting in finding the *optimal segment lists* compliant with the initial problem characteristics (properties, constraints and objectives).

Several steps are necessary to formulate such an SR-wrapped problem. While the set of $Properties$ does not need to be modified (as considering SR does not affect them), the constraints $\mathcal{C}_{\setminus 0}$ have to be extended to $\mathcal{C} = (c_0, c_1, \ldots, c_k)$, where $c_0 = MSD$ represents an (optional) upper bound on the number of segments. More importantly, the comparison relation, $\leqslant_{\setminus 0}$, also has to be modified to consider the number of segment as a new metric: the goal is now to compare the distances of segment lists (rather than paths').

Two Main Strategies to Extend $\leqslant_{\setminus 0}$*:* There exist several ways to wrap SR around $\mathcal{P}$, leading to different SR-wrapped variants of $\leqslant_{\setminus 0}$. The two most relevant strategies are to consider the number of segments either in a *lexicographical* or in a *constrained* fashion. With a *lexicographical* wrapped comparison, one aims at finding the minimal segment list(s) among the distance(s) optimizing the initial objective(s) (defined by $\leqslant_{\setminus 0}$, under constraints $\mathcal{C}_{\setminus 0}$ and verifying $Properties$). MSD is thus technically ignored, and d_0 is minimized as a secondary objective (meaning that the results may exceed MSD).

The *constrained* strategy aims to return a *deployable* segment list(s) L (*i.e.,* verifying $d_0(L) < c_0 = MSD$) whose underlying paths are optimal with respect to the initial problem. This method ensures that the returned segments lists are deployable, although the TE objective may have to be relaxed to find such a solution as a segment list that is not optimal with respect to the initial problem may become the only feasible solution for a subsequent destination.

Table 2 shows the formal definitions of the relations associated to these strategies. Several relations can be derived from a single strategy, depending on the chosen *path diversity option*.

Option	$\mathcal{R}$	Segment list x $\mathcal{R}$-dominates y, iff:
Constrained-SR		
oneBest	$\preccurlyeq$	$d(x) \leqslant_{\setminus 0} d(y) \wedge d_0(x) \leq d_0(y)$
allBest	$\prec$	$x \preccurlyeq y \wedge d(x) \neq d(y)$
all	$\nprec$	$d(x) <_{\setminus 0} d(y) \wedge d_0(x) \leq d_0(y)$
Lexicographic-SR		
oneBest	$\trianglelefteq$	$d(x) <_{\setminus 0} d(y) \vee \big(d(x) =_{\setminus 0} d(y) \wedge d_0(x) \leq d_0(y)\big)$
allBest	$\triangleleft$	$d(x) <_{\setminus 0} d(y) \vee \big(d(x) =_{\setminus 0} d(y) \wedge d_0(x) < d_0(y)\big)$
all	$\ntriangleleft$	$d(x) <_{\setminus 0} d(y)$

Figure 2. Definitions of the set of relations supported by GOFOR. Each corresponds to a given SR-wrapping strategy and path diversity option.

And Three Options for Path Diversity*:* GOFOR supports various path diversity options, in order to adapt to the operator's needs, *e.g.,* regarding load-balancing. Each of these options are applicable to both the lexicographical and the constrained strategies. We call the three diversity options *1best*, *allBest* and *all*. With respect to the initial problem and the chosen strategy; *1best* returns at least one optimal solution, *allBest* returns all optimal solutions, and *all* returns all solutions that are encodable with less than c_0 segments. The two last options allow the operator to perform source-routed load-balancing.

[4]More precisely, any order relation that contains the component-wise order *i.e.,* $\forall i, x_i \leq y_i \Rightarrow x \leqslant_{\setminus 0} y$. We assume monotone and isotone relations by construction (additive strictly increasing metrics because of the nature of w).

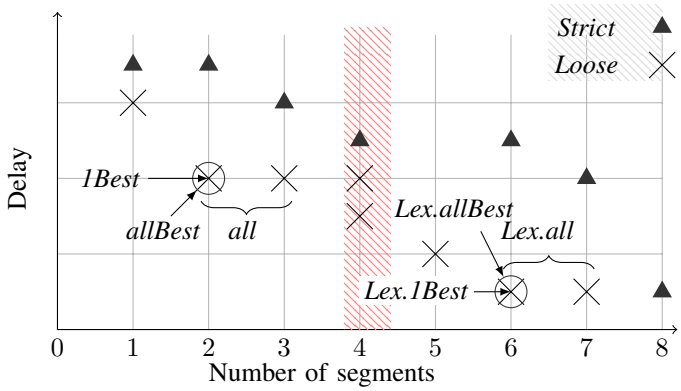

Figure 3. Optimal distances towards a given destination, depending on the SR-wrapping strategy (constrained or lexicographic), the path diversity option (1best, allBest or all) and the encoding scheme (loose or strict).

All the resulting relations are shown in Table 2. The desired form of the relation, noted $\mathcal{R}$, is to be chosen according to the operator's wishes and based on the initial relation $\leqslant_{\setminus 0}$. Solutions that are not optimal with respect to $\mathcal{R}$ are said to be $\mathcal{R}$-dominated.

Figure 3 illustrates graphically the solutions returned by each strategy and option. We consider the minimization of the delay as the initial objective (using IGP node segments). For a given (hypothetical) destination, each cross (resp. triangle) represents a non-$\mathcal{R}$-dominated, loose-encoded (resp. strict-encoded), segment lists. The y-axis shows the delay of the computed segment lists while the x-axis shows the required number of segments considering $MSD = 4$ (excluded). The *constrained* options return only segment lists that are below the MSD constraint. On the contrary, the *lex* strategies only focus on the best delays. With the *all* case, GOFOR returns either all optimal distances encodable in fewer than 4 segments (with the *constrained* strategy), or all optimal distances with respect to $\leqslant_{\setminus 0}$ when considering the *lex* strategy.

Each pair (strategy, option) corresponds to a different SR-wrapped dominance relation $\mathcal{R}$ (crafted by modifying $\leqslant_{\setminus 0}$), as defined in each row of Table 2. For instance, for the *allBest* option of the constrained strategy ($\prec$), GOFOR relies on the relation $\preccurlyeq$, that itself combines the relation $\leqslant_{\setminus 0}$ and the simple scalar comparison $\leq$, to rely on the number of segments and so discriminate equal distances with respect to $\leqslant_{\setminus 0}$.

From $Properties$, constraints $\mathcal{C}$, the SR-wrapping strategy and the diversity option leading to $\mathcal{R}$, we can now formally define the SR-wrapped problem for a given source.

Definition 3. *A problem* $\mathcal{P}(Properties, \mathcal{C}, \mathcal{R})$ *consists in finding,* $\forall v \in V$*, the minimal lists of segments towards* v*, under the constraints* $\mathcal{C}$*, verifying some properties* $Properties$*, and*

that are non-dominated with respect to a given dominance relation $\mathcal{R}$ (the SR wrapping of $\leqslant_{\setminus 0}$).

GOFOR solves $\mathcal{P}$, the SR-wrapped problem. More precisely, GOFOR is able to efficiently extend an algorithm designed for a given initial problem to solve $\mathcal{P}$ (with $\mathcal{R}$ and other parameters set to the operator needs among all options). The main challenges lie in the SR encoding and the extension of an extra subset of dominated distances, as introduced in Section I. Our framework not only provides correct and optimal solutions, it retrieves them with an efficient path exploration avoiding superfluous computations.

III. Two Main Ingredients: Encoding, Isotonocity

A. Encoding Distances & Properties: Paths and Segment Lists

To solve $\mathcal{P}$, it is necessary to compute the segment list(s) encoding the paths being explored during the initial exploration to properly take SR into consideration. In particular, a loose encoding scheme[5] is necessary to translate the distances explored in order to guide the search accordingly and consider MSD. Unlike existing loose encoding schemes, our algorithm is not only efficient (as it does not directly rely on an SR-graph), but also handles multiple topologies (node segments of distinct types in practice).

Greedily and Loosely Updating Segment Lists*:* Our encoding algorithm (Algorithm 1 with its subroutine 2) follows an efficient approach. Given a path p as input (the one currently explored), it incrementally translates p into a segment list L using a greedy strategy. Initially, all possible segments able to encode the first edge of p (which may be either $Node$ or Adj segments) are stored in `LastSeg` (Line 3 in Alg. 1). These segments are then extended to include more and more edges of p as long as it is possible (*i.e.,* the segment lists verify the desired properties and match, or are better than, the path's metrics). The extension process is outlined in Alg. 2. Segments that cannot be further extended correctly are removed from `LastSeg`. If no more segments can be extended to include properly the new edge e, a new segment is required. Among the remaining segments failing at e, one of them, say S, is added to the segment list L [6], and `LastSeg` is reset with all possible segments that can encode edge e (Line 8 in Alg. 2). While path p is used to guide the search, the segments lists may include other paths as well, but only if they verify the same properties and are equal or better than p with respect to its distances. While our approach encodes logical characteristics rather than structural ones, p is nevertheless part of the set of paths encoded by the resulting segment list.

Algorithm 1 returns a minimal encoding (Theorem 1). To ease the reading all proofs are given in the appendix, we here only provide lemmas to highlight their constructions. In particular, Lemma 2 states that loose encoding is *isotonic*, *i.e.,* when a segment loosely encodes a path, then a restriction of this segment loosely encodes a sub-path.

Algorithm 1: ENCODE($G, Properties, p$)

```
1  L := [ ]
2  e := first edge of p
3  LastSeg := {S = Seg(t, e) |
     e ∈ S ∧ d(S) = d(e) ∧ Properties(G, S), ∀t ∈ SegTypes}
4  for e ∈ p[1: ] do
5      L, LastSeg := EXTEND(L, LastSeg, e)
6  Let S be any segment in LastSeg
7  return L ⊕ S
```

Algorithm 2: EXTEND(L, `LastSeg`, e)

```
1  NewLastSeg := ∅
2  for S ∈ LastSeg do
3      if S ∘ e ≠ ∅ ∧ d(S ∘ e) = d(S) + d(e) ∧ Properties(S ∘ e)
         then
4          NewLastSeg := NewLastSeg ∪ {S ∘ e}
5  if NewLastSeg = ∅ then
6      Let S be any segment in LastSeg
7      L := L ⊕ S
8      LastSeg := {S = Seg(t, e) |
         e ∈ S ∧ d(S) = d(e) ∧ Properties(G, S), ∀t ∈ SegTypes}
9  else
10     LastSeg := NewLastSeg
11 return L, LastSeg
```

Lemma 1. ENCODE*(p) returns a loose encoding of p.*

Lemma 2. *Let $S = Seg(S^{type}, p[0], p[l])$ be a single segment that loosely encodes a path p of length l. Then, for any i, j in $[0, l]$, $i < j$, $Seg(S^{type}, p[i], p[j])$ loosely encodes $p[i : j]$.*

Theorem 1. ENCODE*(p) returns a loose encoding of p with a minimal number of segments.*

Our innovative encoding scheme seamlessly integrates with most path computation algorithms, making them aware of the number of segments needed for the paths they explore. By invoking these algorithms incrementally at each edge relaxation in practice, this method provides a highly efficient encoding scheme (no significant overhead as node segment information is retrieved in constant time). However, as detailed in Sec. I, maintaining the number of segments required during exploration is not sufficient: certain dominated distances must also be extended.

B. An Extended Relation to Regain the Isotonicity

As already showcased in the introduction, extending only the optimal segment lists (according to the chosen $\mathcal{R}$) is not enough to find a solution to the SR-wrapped problem $\mathcal{P}$. Indeed, the dominance function $\mathcal{R}$ is not isotonic anymore. Although optimal solutions are not $\mathcal{R}$-dominated (by definition), a prefix of an optimal solution *could* be $\mathcal{R}$-dominated, due to the peculiar way the number of segments evolves after an extension (as emphasized with the example of Fig. 1) [7].

[5]We focus on loose encoding as we are interested in distances and properties rather than structure. Strict encoding schemes are straight-forward to design, do not offer minimal segment lists and can be found in the literature [10].

[6]All segments lists can still be retrieved during the backtracking post-processing phase used to re-construct the desired set of segment lists (according to a given path diversity option).

[7]Interestingly, algorithms that directly explore the SR Graph do not require such a scheme. The transformation of the graph makes the SR metric isotonic and predictable by converting it into a simple hop count metric.

Extend the Dominance Relation $\mathcal{R}$ to $\overset{\Diamond}{\mathcal{R}}$: Our goal here is to define a new dominance relation, called *extended* dominance relation, in order to regain the isotonicity property and define which distances should be extended to prevent compromising either the optimality or the efficiency of our framework. One possible but costly solution would indeed be to treat all the lists as non-dominated. While returning optimal and correct solutions, this method would result in a huge overhead. Our extended dominance function is not only correct, but limits the induced overhead to $\times|V|$ at worst when updating distances (at each edge relaxation). Such an overhead is minimal as any weaker solution would miss valid solutions.

Our extension is formally defined in the following definition. Note that, for a segment S, $u \in S$ abusively means that there exists a path in S passing through u.

Definition 4. *Let $\mathcal{R}$ be a dominance relation over SR-distances. The extended relation $\overset{\Diamond}{\mathcal{R}}$ associated with $\mathcal{R}$ is defined over segment lists as follows. For any segment lists L and L', we have $L \overset{\Diamond}{\mathcal{R}} L'$,* i.e., *$L'$ is $\overset{\Diamond}{\mathcal{R}}$-dominated by L, if either*

(i) $d(L)\,\mathcal{R}\,d(L') \quad \wedge \quad L^{type}_{last} = L'^{type}_{last} \quad \wedge \quad L^{src}_{last} \in L'_{last}$
(ii) *or* $d(L)\,\mathcal{R}\,(d_0(L') - 1, d_{\setminus 0}(L'))$

where L_{last} denotes the last segment of L and L^{src}_{last} its source. Notation $d_{\setminus 0}$ refers to the exclusion of metric d_0 in the vector of considered metrics.

This extended relation applies directly to segment lists and not to distances. It is used to check whether a given relation could hold after the segment lists are extended, and ensure that only segment lists that are not currently $\mathcal{R}$-dominated or could potentially *become* non-dominated are extended. Informally, L' is $\overset{\Diamond}{\mathcal{R}}$-dominated by L if it is dominated by L regarding relation $\mathcal{R}$ and either (i) the source of the last segment of L is contained within the paths encoded by L'_{last} (also considering the same type of last segments), or (ii), L does have strictly fewer segments. Indeed, if segment list L' verifies case (i), one can show that if L'_{last} can be extended by an edge e, so can L, meaning that L will remain better than L' (see Lemma 3 and its proof).

This isotonic property of this extended relation is stated in Theorem 2 (for any relation $\mathcal{R}$ wrapped around $\leqslant_{\setminus 0}$ or $<_{\setminus 0}$), after exhibiting said Lemma. Again, the associated proofs are given in the appendix.

Lemma 3. *Let S and S' be two segments such that $S'^{src} \in S$, $S'^{dst} = S^{dst}$, and $S'^{type} = S^{type}$. We have that, if S can be extended by an edge e, then S' can also be extended by e.*

Theorem 2. *The extended dominance $\overset{\Diamond}{\mathcal{R}}$ is isotonic,* i.e., *the extension of an $\overset{\Diamond}{\mathcal{R}}$-dominated path remains $\mathcal{R}$-dominated, even after extensions.*

Thus far, with a peculiar attention to the computing complexity, we have shown how paths and their characteristics should be loosely encoded into segment lists (Sec. III-A), and which extra segments lists should be extended in order to guarantee optimality with respect to $\mathcal{R}$ (Sec. III-B). From these two ingredients, we now describe how GOFOR turns an initial algorithm solving a non-SR problem into its extended version, solving its related SR-wrapped problem $\mathcal{P}$.

IV. GOFOR-SR, an Efficient Recipe to solve $\mathcal{P}$

Given a shortest path algorithm $\mathcal{A}(\leqslant_{\setminus 0})$, looking for optimal paths with respect to a comparison relation $\leqslant_{\setminus 0}$ (modeling an arbitrary initial problem), we present here our recipe to create

$$\mathcal{A}(Loose|Strict, \overset{\Diamond}{\mathcal{R}})$$

That is an algorithm solving $\mathcal{P}$, the SR-wrapped version of the initial problem, with a flexible configuration: the operator can plug SR with the most suited encoding paradigm (loose or strict) and SR-wrapped relation $\mathcal{R}$ according to its needs.

A. Wrapping SR Around $\mathcal{A}(\leqslant_{\setminus 0})$

We assume that $\mathcal{A}(\leqslant_{\setminus 0})$ solves an additive routing problem – possibly already multi-dimensional, based on an isotonic relation $\leqslant_{\setminus 0}$, and overall aiming to optimize given distances, verify $Properties$ and respect some constraints $\mathcal{C}_{\setminus 0}$. GOFOR enhances $\mathcal{A}(\leqslant_{\setminus 0})$ to take SR into account and solve $\mathcal{P}(Properties, \mathcal{C}, \mathcal{R})$.

The crucial operation in all shortest path algorithms is to decide whether a path or rather its distance should be stored or not in the *Priority Queue* (denoted PQ) to be extended later on. For any relation R, we denote by $e \overset{R}{\hookrightarrow} \mathcal{E}$ the operation of storing the element e in the set $\mathcal{E}$ of all current optimal elements (with respect to the relation R). In more formal words, we have $e \overset{R}{\hookrightarrow} \mathcal{E} \Leftrightarrow \nexists e' \in \mathcal{E}$ such that $e' R\ e$.

Note that the operation carried by $\mathcal{A}(\leqslant_{\setminus 0})$ when extending a path p, and its distance, can be expressed as

$$d(p) \overset{\leqslant_{\setminus 0}}{\hookrightarrow} \mathcal{D}$$

This means that non-dominated distances with respect to $\leqslant_{\setminus 0}$ are stored in the priority-queue $\mathcal{D}$ to be extended further.

GOFOR replaces this operation in $\mathcal{A}(Loose|Strict, \overset{\Diamond}{\mathcal{R}})$ to

$$\textsc{Encode}(p) \overset{\overset{\Diamond}{\mathcal{R}}}{\hookrightarrow} \mathcal{D}_{\mathcal{L}},$$

The segment list (loosely or strictly) encoding p (as defined in Sec. III-A) is stored within the priority-queue $\mathcal{D}_{\mathcal{L}}$ to be extended further (the PQ, $\mathcal{D}_{\mathcal{L}}$, is now segment-list-based), if it is non-dominated with respect to the chosen extended relation $\overset{\Diamond}{\mathcal{R}}$ defined in Sections II and III-B.

B. Complexity of the Priority Queue Updates

Overall, our main computational challenge lies in the performance of the PQ with our extended dominance relation, *i.e.,* the $\overset{\overset{\Diamond}{\mathcal{R}}}{\hookrightarrow}$ operation. Since we consider a Dijkstra-like algorithm, the complexity overhead indeed comes from the management of the underlying PQ. In the worst case, the PQ contains an entire distance Pareto-front to all the destinations, *i.e.,* $n.\Gamma.c_0$ entries, where $n = |V|$. Observe that the PQ

does not contain each non-dominated segment lists but only non-dominated distances, whatever the path diversity option. Segment lists having the same distance are grouped into a single entry. Thus, the complexity of the `extract_min` operation (*i.e.,* retrieving the segment list to extend from the queue) is $O\left(n.\Gamma.c_0 \times \log(n.\Gamma.c_0)\right)$.

Once extracted, each distance (and their underlying segment lists) in the Pareto-front are extended to all adjacent neighbors. We extend $O(m.\Gamma.c_0)$ distances, where $m = |E|$. The complexity of this extension depends on the number of segments lists: as the behavior of the extension depends on the last segment of the segment list, each one has to be extended individually, even if they share the same distance. The complexity to perform all extensions is $O(m.\Gamma.c_0.r)$ where r is the maximum number of segment lists for a given distance. The number r is bounded by $n \times |SegTypes|$, but empirical evaluation suggests that in practice, $1 \leq r < 2$, resulting in a very limited overhead.

After extending a distance and its underlying segment lists to a neighboring node, we compare them to the existing distances towards this node, keeping only non-dominated distances. Comparing each newly extended distance to the existing ones has a complexity of $O(m.(\Gamma.c_0)^2)$. If the new distance already exists towards the considered node, we merge the associated segment lists (the newly extended ones and the existing ones) into a single distance entry.

In summary, the worst-case complexity of GOFOR is in

$$O\left(n.\Gamma.c_0 \times \log(n.\Gamma.c_0) + m.\Gamma.c_0.(\Gamma.c_0 + r)\right)$$

Note that with the *lex* strategy, the term c_0 can be ignored in the complexity as such a strategy does not plug the SR metric as a constraint in the problem. Thus, if the initial problem is not multi-dimensionnal in itself ($\Gamma = 1$), one can just ignore the term $\Gamma.c_0$ overall.

C. A Meta-DAG to Perform Source-Based Load Balancing

The last remaining challenge is to effectively utilize the set of computed solutions thanks to a condensed data structure. Most shortest path algorithms return a shortest path DAG, from which all optimal solutions can be easily reconstructed and used by routers.

In our case, representing the set of solutions through a *logical* (segment-lists level) DAG, with edges representing segments instead of physical edges, is also suitable for structuring the computed optimal segment lists. However, subtleties arise when considering multi-metric problems, where $\Gamma > 1$. In such cases, segment lists that are solutions for a node v, even if they have the same distance, can pass through an intermediary node u with different intermediary non-dominated distances. This introduces ambiguity when reconstructing segment lists. To address this, extra information must be stored within the segment-list DAG to ensure proper and coherent reconstruction of the segment lists.

Backtracking on a Logical DAG of Segment Lists*:* To effectively present the calculated solutions towards a node v, we introduce a "meta-DAG" (named as such because each edge represents a segment list that may encode multiple paths, and each node now possibly supports multiple distances). In this representation, each node u is replaced by at most Γ nodes $u_0, \ldots, u_{\Gamma-1}$, with each node corresponding to an occurrence of u in different solutions. In the worst case, u appears in every non-dominated solution. A directed link between u_d and $u'_{d'}$ represents a segment that is part of a solution, where d and d' are the intermediary distances of the solution at nodes u and u', respectively.

The meta-DAG serves as a tool to retrieve one or all segment lists from the set of solutions. In Figure 4, we provide an illustration of such a meta-DAG along with its initial network for a complex use-case, DCLC-SR (wrapping SR on DCLC). The objective is to retrieve all the best IGP paths, encoded with their best segment lists, from node (S) to node (D) while adhering to a delay constraint of 7 ms, for example. The DCLC problem itself generates multiple non-dominated optimal solutions with its two initial metrics.

In the meta-DAG on the left, we represent the node and adjacency segments used between nodes. Some nodes can be reached with distinct non-dominated distances (because DCLC is itself a multi-metric problem): for instance the physical node (4) has two corresponding nodes in the meta-DAG: $4_{(7,4)}$ and $4_{(8,3)}$. In fact, there exists several optimal segment lists passing through (4) to reach (D), *e.g.,* (S)–(3)–(1)–(4)–(D) and (S)–(4)–(5)–(D). This structure, thanks to the added information described, allow reconstructing the desired segment lists.

In practice, this meta-DAG can be built by backward induction from the destination (D) to the source (S), by listing, for all segments lists ending at u and with distance d, the source of their last segment (which become the predecessors of u_d in the meta-DAG).

In practice, this Meta-DAG could be used by the edge router, the source itself, to perform source-routed load-balancing across segment lists, *e.g.,* by choosing among optimal solutions through random walks in the Meta-DAG for each given flow. Such a feature allows to load-balance the traffic even if the latter is subject to complex traffic-engineering requirements (taking into account each node having multiple successors on the lists). Further studies on the possibilities offered by such advanced load-balancing methods, and practical investigations regarding its feasibility directly in the data-plane of real hardware/software, is left for future work.

V. SR-Wrapped TI-LFA, Least-Delay & DCLC

In this section, we use GOFOR to wrap SR around three problems and algorithms: TI-LFA (a Fast-ReRoute, FRR, use-case), Least-Delay (LD) and Delay-Constrained-Least-Cost (DCLC). We aim to demonstrate the genericity and performance of our framework. In particular, we focus on the evaluation of the efficiency of GOFOR on DCLC (the most computationally complex use-case). The source code of GOFOR and the experimental materials are available online [7].

A. Performance Analysis: Three Use-cases, a Single Recipe

Examples of General Settings for Defining $\mathcal{P}$*:* We consider three use-cases (DCLC, FRR, and LD) with the

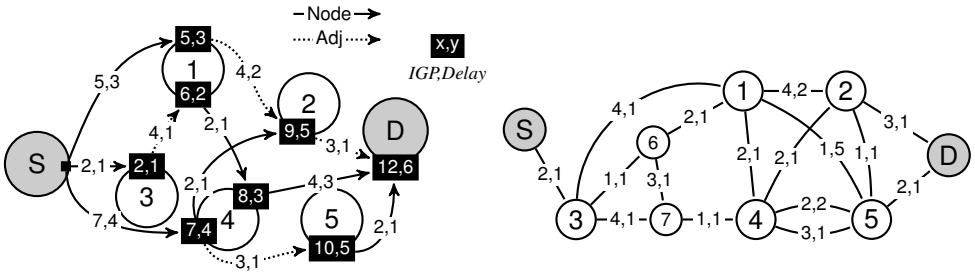

Figure 4. A raw network (at the right), and its resulting meta-DAG of *all* the segment lists from S towards node D having distance $(12, 6)$.

objective of transforming these problems into their corresponding SR-wrapped versions (DCLC-SR, FRR-SR, and LD-SR, respectively). As an example of strategy, let us first assume that the operator aims to ensure the deployability of the retrieved segments, indicating a preference for the *Constrained* alternative to deploy SR. Additionally, the operator expresses an interest in obtaining all optimal solutions, implying that the *allBest* diversity option should be used. Therefore, the operator should set the relation $\mathcal{R}$ to $\prec$ (refer to Table 2), and implement the associated extended relation $\overset{\Diamond}{\mathcal{R}}$ to compare the segments lists.

For the **DCLC-SR use-case**, where the initial goal is to find paths towards any destination with a delay bounded by c_{del} and minimizing the IGP distance, the related SR-wrapped problem can be formulated as: optimally encoding the DCLC paths requiring fewer than MSD segments. One would set $\mathcal{C} = (MSD, \infty, c_{del})$ and leave $Properties$ empty. With such settings, the desired $\prec$ is expressed as follows:

$$(nbSeg_1, cost_1, delay_1) \prec (nbSeg_2, cost_2, delay_2)$$
$$\Leftrightarrow \begin{cases} nbSeg_1 \leq nbSeg_2 \wedge cost_1 \leq cost_2 \wedge delay_1 \leq delay_2 \\ nbSeg_1 < nbSeg_2 \vee cost_1 < cost_2 \vee delay_1 < delay_2 \end{cases}$$

For the **FRR-SR use-case**, the delay metric should be ignored[8], and the IGP cost should be optimized. However, the predicate $Properties(S)$ should be considered false if a path in S uses the given failed link. The constraint should be set to $\mathcal{C} = (MSD, \infty)$.

For the **LD-SR use-case**, the IGP cost should be ignored, and no specific $Properties$ are required, as the only objective is minimizing the delay. Constraints should be set to $\mathcal{C} = (MSD, \infty, \infty)$.

In practice, the modifications required for path computation algorithms are relatively light, involving only the implementation of the encoding scheme and the choice of the path comparison function to the relation $\overset{\Diamond}{\mathcal{R}}$, as associated with the chosen $\mathcal{R}$ (and adjusted with the wrapping strategy and its options). While this section provides only a sample of possible use-cases, its purpose is to illustrate how GOFOR can transform almost any path computation problem into its SR-wrapped version, accommodating the operator's requirements in terms of optimization strategies and path diversity options.

[8]Mimicking a standard FRR use-case. GOFOR can also return DCLC-SR or LD-SR solutions avoiding a failed link.

We will now proceed to evaluate GOFOR on our three use-cases, considering several strategies and path diversity options.

A SAMCRA-Based Implementation for GOFOR*:* Recall that GOFOR, is a framework that transforms an existing algorithm to handle SR. Since we tackle use-cases encompassing several metrics, we used a generic multi-metric shortest path algorithm as a basis. We decide to rely on SAMCRA [11] for its flexible PQ implementation. Although SAMCRA can return DCLC paths, it does not support any encoding paradigm nor extended comparison relation. We thus modified it according to our framework. The resulting code for all use-cases is available online [7].

Lattices and Realistic Topologies*:* In the following, we start with an analysis highlighting the advantages of the loose encoding paradigm. We use specific graphs having two key properties: strong resilience and coarse valuation metrics that promote ECMP. These graphs are modified lattices with redundant links, created by doubling each link in a King's graph with a 0.3 probability. The metrics assigned to the graph weights are uniformly random, with five possible values ranging from 1 to 5. These topologies are not intended to replicate real-world scenarios but rather to emphasize the benefits (reduction in the number of segments) of the loose encoding in extreme cases. The second evaluated criteria is the computation time performance (on DCLC-SR). For this, we use both the synthetic lattices described, and realistic topologies. The latter are primarily extracted from the REPETITA framework [12].

Experiments were conducted on a MacBook Pro Laptop, equipped with an Apple M1 Pro Chip and 16GB of RAM.

B. Encoding Paradigms: a Look at the Number of Segments

We first study how well our loose encoding leverages ECMP compared to strict encoding.

Strict vs. Loose Encoding*:* We compared the length of segment lists of the two paradigms in two use-cases. Figures 5 and 6 show the differences from these two perspectives. The first perspective illustrates the resulting Pareto Fronts for a specific source-destination pair in our graph set, focusing on LD-SR (with $\prec$). Relying on loose encoding significantly reduces the overhead induced by pushing segments to the packet.

The second figure conveys the same message but for FRR-SR, using the *Lexicographical* strategy and *all* diversity options ($\lll$). The figure shows the number of solutions (avoid-

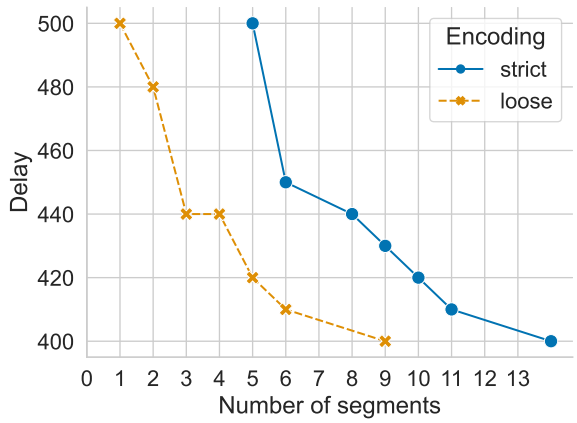

Figure 5. Comparison of the Pareto front solutions when performing loose encoding and strict encoding for LD-SR (with Cons.allBest).

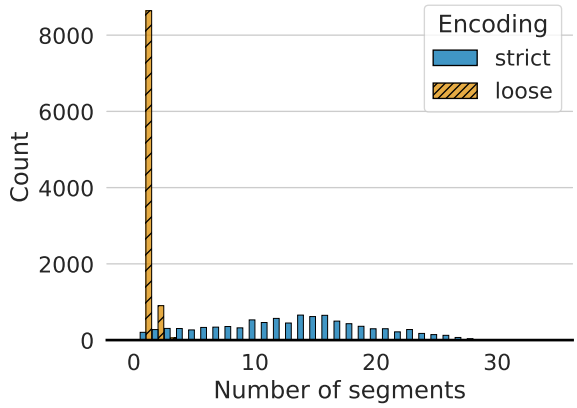

Figure 6. Distribution of the number of segments for the solutions found using loose encoding and strict encoding for FRR-SR (with Lex.all).

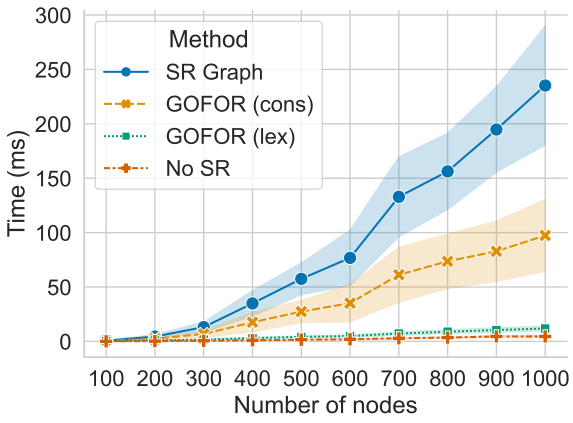

Figure 7. Comparison of the execution time overhead to solve DCLC-SR.

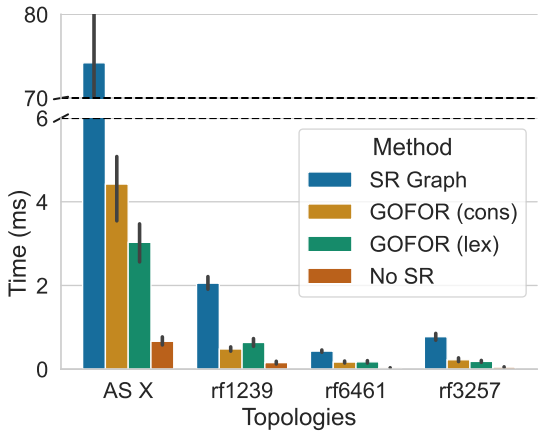

Figure 8. Execution times on realistic topologies of around 2000, 1000, 370 and 300 edges respectively.

ing the failed link) for each number of segments, summed for all the destinations.

As our graphs have symmetric valuation, only between 2 or 3 segments are required [1] to avoid a link failure (we include the final destination segment, and consider that each basic local LFA requires an adjacency segment). This limit is indeed satisfied by our loose encoding scheme, while strictly encoded solutions require much more segments to encode unique backup paths (as ECMP is frequent). The loose encoding paradigm is thus necessary to retrieve *all* feasible segments lists (*e.g.,* with MSD=10).

C. Experimental Complexity: GOFOR-SR is Lightweight

We now investigate how GOFOR performs compared to SR-graph-based frameworks with respect to the execution times of the transformed algorithm. We restrict here our study to DCLC-SR as it is the most challenging use case.

A Limited Computing Time Overhead*:* Figure 7 provides a comparison of the computing time required by GOFOR with a *Constrained* strategy ($\prec$); GOFOR with a *Lexicographical* strategy ($\lhd$); SAMCRA as baseline (without SR); and SAMCRA on top of the fully meshed SR-Graph given as input. We rely on the *allBest* diversity option in each of these setups. We consider lattices of variable sizes to plot the computing time evolution according to their dimensions. The figure shows the average and standard deviation among 100 runs for each size.

Finally, we perform the same experience on realistic topologies from the REPETITA framework, on ASes ranging in size from 2000 edges to only 300. The results are shown in Fig 8. Note that there is a break in the y-axis of the figure.

Both figures show a clear tendency: GOFOR *Lex.* has a negligible overhead (regarding the initial algorithm ignoring SR) and GOFOR *Cons.* outperforms the best existing methods relying directly on the SR-Graph. While the performance of *Cons.* compared to *Lex.* looks significant at first glance, it is worth recalling that *Cons.* fully adds the SR dimension to the initial problem (the MSD constraint is effective and cannot be ignored: $c_0 > 1$).

Furthermore, this result is particularly obvious with the tested lattices, intentionally designed to amplify such effects (as Γ becomes not negligible). However, on realistic topologies where problem instances are generally simpler, our SR-wrapped algorithm consistently solves this multi-metric problem in under 4ms. Experimental results on these realistic

topologies not only showcase the superior performance of GOFOR compared to other frameworks but also demonstrate the minimal overhead introduced by our framework. We experimentally observe that $r < 2$ on average on our realistic cases, while we also only have $r < 3$ with our lattices. We envision to theoretically investigate the distribution of r in future works, *e.g.,* with an average complexity analysis on random graphs.

VI. Related Work

SR has generated a lot of traction, leading to contributions from both the industry and academia [13]. It has been used to perform fine-grained monitoring [8], [14], increase network resiliency [15], [16] or perform traffic-engineering [17]. Rather than solving a specific routing use-case with SR, we proposed a general framework allowing to easily adapt existing (and future) path-computing algorithms to SR, streamlining the deployment of SR.

Some work aims to mitigate the limitations induced by MSD, making SR more scalable. For example, uSIDs carry several instructions in a single segment [18]. Similarly, binding segments (BSID) can be used to create a one-to-one mapping relationship between a segment and a segment list. These solutions can be used in conjunction with our schemes.

Several SR-related contributions indirectly address the path encoding problem, although it is not their primary focus. Some use generic optimization frameworks to combine segments and create compliant segment lists [19], [20]. Aubry proposed a dynamic programming approach, computing paths incrementally based on segment list size [10]. Another option consists in exposing segments into a SR-graph, treating segments as edges, and execute algorithms on this inflated fully-meshed graph while limiting the exploration dept to MSD [2], [21], or just encoding a specific input path [22]. While these techniques also support loose encoding, they suffer from performance drawbacks due to the density of the graph they explore. Exploring the SR-graph or using dynamic programming leads to n^2 operations while GOFOR mitigates this issue when the network graph is sparse, as it is often the case in practice. Other generic optimization frameworks generally need to heavily restrict MSD (often to 2 or 3) to reduce the exploration space.

Several works propose to strictly encode a specific path given as input to iteratively find the longest subpath encodable in a single segment [10], [9], [23], [24]. While such schemes follow the same greedy approach as ours, these schemes are meant to be used *a posteriori*, and thus neither leverage ECMP, nor ensure that the segment list found is deployable and/or minimal with respect to the operator needs. Finally, although the basic principles of GOFOR were mentioned in our previous work [2], the latter were neither complete nor were they proven, and were not generic nor evaluated. In this paper, we also add the support of distinct strategies, multi-topology and several path diversity models (enabling so fine-grained source-controlled load balancing) for various use-cases.

With GOFOR, we tackle the path encoding problem in a generic *and* optimal fashion, considering nearly all basic use-cases and offering many key operational features. By tying together the paths and the segment lists computation, GOFOR returns all the relevant solutions. GOFOR performs better than concurrent approaches as it leverages the sparsity, and overall characteristics, of real IP networks as shown in Sec. V.

VII. Conclusion

The conventional best-effort routing paradigm, while scalable, falls short of meeting all the requirements of IP networks. Certain use-cases demand deviations from basic shortest routes to navigate failures or consider multiple metrics as additional constraints and objectives. Segment Routing (SR) stands out as one of the most popular options for deploying flexible routes loosely guided from the source.

However, the introduction of SR adds an operational metric, the number of segments (or detours), and a new challenge, efficiently retrieving all optimal segment lists. Minimizing these detours is crucial to ensure line-rate speed packet processing. Existing encoding schemes, which convert paths to segment lists, often overlook this aspect. They either convert paths a posteriori and/or fail to leverage Equal-Cost Multi-Path (ECMP), resulting in inflated segment lists. Alternatively, they impose a quadratic overhead in the number of nodes by relying on a complete SR-graph.

In this paper, we introduce a novel approach that enhances existing algorithms to return optimal and deployable segment lists instead of paths. Our initial step involves the design of a *loose encoding* scheme, which seamlessly integrates into current path computation algorithms. This scheme computes minimal segment lists for the paths being explored, utilizing all available segment types and leveraging ECMP. We then address the transformation of the path comparison relation to ensure the discovery of the optimal segment list, despite the loss of isotonicity (*i.e.,* substructure optimality) induced by the SR metric. The newly proposed relation, able to accommodate various optimization strategies and path diversity options based on the operator's requirements, minimizes the computation overhead to a strict minimum. We show that algorithms extended by our resulting framework, GOFOR, remain particularly efficient compared to existing SR-aware approaches, even facing challenging multi-criteria routing problems.

As a promising future work, we aim to investigate whether our meta-DAG of optimal solutions could be fitted into the data-plane directly to perform efficient source-driven, TE-aware load-balancing across all segment lists.

References

[1] S. Litkowski, A. Bashandy, C. Filsfils, P. Francois, B. Decraene, and D. Voyer, "Topology Independent Fast Reroute using Segment Routing," Internet Engineering Task Force, Internet-Draft draft-ietf-rtgwg-segment-routing-ti-lfa-11, Jun. 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-rtgwg-segment-routing-ti-lfa/11/

[2] J.-R. Luttringer, T. Alfroy, P. Mérindol, Q. Bramas, F. Clad, and C. Pelsser, "Deploying near-optimal delay-constrained paths with segment routing in massive-scale networks," *Computer Networks*, vol. 212, p. 109015, 2022.

[3] C. Filsfils, K. Michielsen, and K. Talaulikar, *Segment Routing Part I*, 1st ed. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2017.

[4] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," RFC 8402, Jul. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8402

[5] R. Mota, "ACG Segment Routing Survey," ACG Research, Tech. Rep., 2023. [Online]. Available: https://www.ciena.com/insights/white-papers/acg-segment-routing-survey

[6] R. Guedrez, O. Dugeon, S. Lahoud, and G. Texier, "A new method for encoding mpls segment routing te paths," in *2017 8th International Conference on the Network of the Future (NOF)*, 2017, pp. 58–65.

[7] Anonymous, "GOFOR-SR: source code and experiments," Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7643270

[8] F. Aubry, D. Lebrun, S. Vissicchio, M. T. Khong, Y. Deville, and O. Bonaventure, "Scmon: Leveraging segment routing to improve network monitoring," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE Press, 2016, p. 1–9. [Online]. Available: https://doi.org/10.1109/INFOCOM.2016.7524410

[9] R. Guedrez, O. Dugeon, S. Lahoud, and G. Texier, "Label encoding algorithm for mpls segment routing," in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, 2016, pp. 113–117.

[10] F. Aubry, "Models and algorithms for network optimization with segment routing," Ph.D. dissertation, UCLouvain, 2020.

[11] P. Van Mieghem, H. De Neve, and F. Kuipers, "Hop-by-hop quality of service routing," *Computer Networks*, vol. 37, no. 3, pp. 407–423, 2001.

[12] S. Gay, P. Schaus, and S. Vissicchio, "Repetita: Repeatable experiments for performance evaluation of traffic-engineering algorithms," 2017.

[13] P. L. Ventre, S. Salsano, M. Polverini, A. Cianfrani, A. Abdelsalam, C. Filsfils, P. Camarillo, and F. Clad, "Segment routing: a comprehensive survey of research activities, standardization efforts and implementation results," 2019. [Online]. Available: https://arxiv.org/abs/1904.03471

[14] M. Xhonneux, F. Duchene, and O. Bonaventure, "Leveraging ebpf for programmable network functions with ipv6 segment routing," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 67–72. [Online]. Available: https://doi.org/10.1145/3281411.3281426

[15] F. Aubry, D. Lebrun, Y. Deville, and O. Bonaventure, "Traffic duplication through segmentable disjoint paths," in *2015 IFIP Networking Conference (IFIP Networking)*, 2015, pp. 1–9.

[16] F. Aubry, S. Vissicchio, O. Bonaventure, and Y. Deville, "Robustly disjoint paths with segment routing," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 204–216. [Online]. Available: https://doi.org/10.1145/3281411.3281424

[17] D. Wu and L. Cui, "A comprehensive survey on segment routing traffic engineering," *Digital Communications and Networks*, 2022.

[18] A. Tulumello, A. Mayer, M. Bonola, P. Lungaroni, C. Scarpitta, S. Salsano, A. Abdelsalam, P. Camarillo, D. Dukes, F. Clad, and C. Filsfils, "Micro sids: a solution for efficient representation of segment ids in srv6 networks," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022.

[19] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A declarative and expressive approach to control forwarding paths in carrier-grade networks," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 15–28, aug 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787495

[20] A. Brundiers, T. Schüller, and N. Aschenbruck, "Midpoint optimization for segment routing," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 1579–1588.

[21] A. Cianfrani, M. Listanti, and M. Polverini, "Translating traffic engineering outcome into segment routing paths: The encoding problem," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, pp. 245–250.

[22] F. Lazzeri, G. Bruno, J. Nijhof, A. Giorgetti, and P. Castoldi, "Efficient label encoding in segment-routing enabled optical networks," *2015 International Conference on Optical Network Design and Modeling (ONDM)*, pp. 34–38, 2015.

[23] L. Davoli, L. Veltri, P. L. Ventre, G. Siracusano, and S. Salsano, "Traffic engineering with segment routing: Sdn-based architectural design and open source implementation," in *2015 Fourth European Workshop on Software Defined Networks*, 2015, pp. 111–112.

[24] A. Giorgetti, P. Castoldi, F. Cugini, J. Nijhof, F. Lazzeri, and G. Bruno, "Path encoding in segment routing," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

APPENDIX

Lemma 1. ENCODE*(p) returns a loose encoding of p.*

Proof of Lemma 1. We prove by induction the following loop invariant: at the end of the l-th iteration, for any segment S in `LastSeg`, $L \oplus S$ is a loose encoding of $p[0:l]$.

Assume that at the beginning of the l-th iteration L is a loose encoding of $p[0:l']$, with $l' \in [0,l]$ and `LastSeg` is a set of segments that loosely encode $p[l':l]$.

In particular $d(L) = d(p[0:l'])$ and $d(S) = d(p[l':l])$.

In the function $Extend$, there are two cases:

1) If `NewLastSeg` $= \emptyset$ in Line 5, then the returned variable L is a loose encoding of $p[0:l]$ by assumption (L concatenated with a segment in `LastSeg`). Moreover, the returned value of `LastSeg` contains only segments that loosely encode $p[l:l+1]$.
2) Otherwise, by condition Line 3, each segment $S \circ e$ in `NewLastSeg` loosely encodes $p[l', l+1]$. Indeed, (i) $S \circ e$ contains $p[l', l+1]$ because S contains $p[l', l]$ by assumption and $S \circ e$ contains $e = p[l:l+1]$ (by definition); and (ii) it verifies $d(S \circ e) = d(p[l', l+1])$ because $d(S) = d(p[l', l])$ by assumption and from the condition we have

$$d(S \circ e) = d(p[l', l]) + d(p[l:l+1]) = d(p[l', l+1]).$$

□

Lemma 2. *Let $S = Seg(S^{type}, p[0], p[l])$ be a single segment that loosely encodes a path p of length l. Then, for any i, j in $[0,l]$, $i < j$, $Seg(S^{type}, p[i], p[j])$ loosely encodes $p[i:j]$.*

Proof of Lemma 2. Recall that we required that $p[i:j] \in Seg(S^{type}, p[i], p[j])$, so we have to show that $d(p[i:j]) = d(Seg(S^{type}, p[i], p[j]))$.

Let k be any metric. By definition of segment

$$\begin{aligned} d_k(Seg(S^{type}, p[0], p[i])) &\geq d_k(p[0:i]), \\ d_k(Seg(S^{type}, p[i], p[j])) &\geq d_k(p[i:j]), \\ d_k(Seg(S^{type}, p[j], p[l])) &\geq d_k(p[j:l]). \end{aligned}$$

So we have

$$\begin{aligned} & d_k(Seg(S^{type}, p[0], p[i])) \\ +& d_k(Seg(S^{type}, p[i], p[j])) \\ +& d_k(Seg(S^{type}, p[j], p[l])) \geq d_k(p) \end{aligned}$$

On the other side, the second property of segments implies that any path in $Seg(S^{type}, p[0], p[i])$ can be extended to a path in $Seg(S^{type}, p[0], p[j])$ (extended through the edges of $p[i:j]$) and to $Seg(S^{type}, p[0], p[l])$ (extended through the edges of $p[j:l]$). Thus by definition of the distance of a segment, the distance of $Seg(S^{type}, p[0], p[l])$ is at least the sum of distances of the partial segments *i.e.,* we have

$$\begin{aligned} d_k(S) \geq & d_k(Seg(S^{type}, p[0], p[i])) + \\ & d_k(Seg(S^{type}, p[i], p[j])) + \\ & d_k(Seg(S^{type}, p[j], p[l])) \end{aligned}$$

On the other side, we have $d_k(p) = d_k(S)$ by definition of loose encoding. So we obtain that the inequalities are in fact equalities, and we have

$$d(Seg(S^{type}, p[i], p[j])) = d(p[i:j])$$

So $Seg(S^{type}, p[i], p[j])$ loosely encodes $p[i:j]$

□

Theorem 1. ENCODE*(p) returns a loose encoding of p with a minimal number of segments.*

Proof of Theorem 1. From Lemma 1, we know that the returned segment list L is a loose encoding of p. We now prove that L has a minimal number of segments. Assume by contradiction that there exists a segment lists L' that is a loose encoding of p and that has a smaller number of segments than $L = \text{ENCODE}(p)$. Since L' has fewer segments, there must be a segment $S' \in L'$ that encodes a subpath $p[l'_1 : l'_2]$ of p such that a segment $S \in L$ encodes $p[l_1 : l_2]$ with $l_1 \geq l'_1$ and $l_2 < l'_2$.

Let $S' = Seg(S^{type}, p[l'_1], p[l'_2])$. By Lemma 2, we know that $Seg(S^{type}, p[l_1], p[l_2 + 1])$ loosely encodes $p[l_1 : l_2 + 1]$ (and it also verifies *Properties* as any sub-path does by assumption), which contradicts the fact that no segment that encodes $p[l_1 : l_2]$ can be extended by the edge $p[l_2 : l_2 + 1]$ (Line 3). □

Lemma 3. *Let S and S' be two segments such that $S'^{src} \in S$, $S'^{dst} = S^{dst}$, and $S'^{type} = S^{type}$. We have that, if S can be extended by an edge e, then S' can also be extended by e.*

Proof of Lemma 3. Consider a path $p \in S$ that passes through node S'^{src}. The Lemma follows from the two properties of segments. Indeed, if $S \circ e$ is not empty, then, by the second property, $p \circ e$ is in $S \circ e$. By the first property the sub path of $p \circ e$ starting from S'^{src} is in $S' \circ e$ so $S' \circ e$ is not empty. □

Theorem 2. *The extended dominance $\overset{\Diamond}{\mathcal{R}}$ is isotonic,* i.e., *the extension of an $\overset{\Diamond}{\mathcal{R}}$-dominated path remains $\mathcal{R}$-dominated, even after extensions.*

Proof of Theorem 2. For simplicity, we assume that $\mathcal{R}$ is the relation $\prec$.

Let L be a segment list that is $\overset{\Diamond}{\prec}$-dominated by a segment list L'. Let S, resp. S' be the last segment of L, resp. L'. Let us denote L^{+1}, resp. L'^{+1}, the segment list L, resp L', once extended by an additional edge (u, v).

By assumption, $L' \overset{\Diamond}{\prec} L$, which means, either
(i) $d(L') \prec (d_0(L) - 1, d_{\setminus 0}(L))$ or
(ii) $d(L') \prec d(L)$, $S^{type} = S'^{type}$, and $S'^{src} \in S$. In both cases, we have $L' \prec L$, hence we have $d(L') <_{\setminus 0} d(L)$.

We know that, after the extension, the distance of the segment list L increases by $(\delta_S, d_{\setminus 0}(u, v))$, where δ_S is 0 or 1 depending on whether S can be extended by (u, v) or not. So the $\prec$-dominance between L^{+1} and L'^{+1} depends only on δ_S and $\delta_{S'}$.

Case *(i)* and S cannot be extended by (u, v): Then, we have $\delta_{S'} \leq \delta_S = 1$ and (i), which implies that $d(L'^{+1}) \prec (d_0(L^{+1}) - 1, d_{\setminus 0}(L^{+1}))$, and in turn $L'^{+1} \overset{\Diamond}{\prec} L^{+1}$.

Case *(ii)* and S cannot be extended by (u, v): Then, either $\delta_{S'} = \delta_S - 1$ (S' can be extended) and (ii), which implies that $d(L'^{+1}) \prec (d_0(L^{+1}) - 1, d_{\setminus 0}(L^{+1}))$, and in turn $L'^{+1} \overset{\Diamond}{\prec} L^{+1}$.
or $\delta_{S'} = \delta_S = 1$ and (ii), which implies $d(L'^{+1}) \prec d(L^{+1})$. Also, the last segment of L^{+1} and L'^{+1} are equal, so $L'^{+1} \overset{\Diamond}{\prec} L^{+1}$.

Case *(i)* and S can be extended by (u, v): If S' can also be extended, then again, $\delta_{S'} = \delta_S = 0$ and (i), implies $L^{+1} \overset{\Diamond}{\prec} L'^{+1}$. Otherwise, if S' cannot be extended, (i) still implies that $d(L'^{+1}) \prec d(L^{+1})$, and since S' is not extended, the source of last segment of L^{+1} is node u and in included in the last segment of L'^{+1}, so $L'^{+1} \overset{\Diamond}{\prec} L^{+1}$.

Case *(ii)* and S can be extended by (u, v): Then, since $S'^{src} \in S$ and $S'^{type} = S^{type}$, S' can also be extended (by Lemma 3), so we still have $d(L'^{+1}) \prec d(L^{+1})$. Also, we still have $S'^{src}_{ext} \in S_{ext}$ and $S'^{type}_{ext} = S^{type}_{ext}$, where S_{ext} and S'_{ext} are the extension of S and S', respectively, so $L'^{+1} \overset{\Diamond}{\prec} L^{+1}$. □