Deploying Near-Optimal Delay-Constrained Paths with Segment Routing in Massive-Scale Networks

Jean-Romain Luttringer, Thomas Alfroy, Pascal Mérindol, Quentin Bramas, François Clad, Cristel Pelsser

Version of Record: https://www.sciencedirect.com/science/article/pii/S1389128622001748
Manuscript_22aef5933b7eca19b8c97e6418d859c0

Deploying Near-Optimal Delay-Constrained Paths with Segment Routing in Massive-Scale Networks

Jean-Romain Luttringer[a,*], Thomas Alfroy[a], Pascal Mérindol[a], Quentin Bramas[a], François Clad[b], Cristel Pelsser[a]

[a] *ICube, University of Strasbourg, France*
[b] *Cisco Systems, California*

Abstract

With a growing demand for quasi-instantaneous communication services such as real-time video streaming, cloud gaming, and industry 4.0 applications, multi-constraint Traffic Engineering (TE) becomes increasingly important. While legacy TE management planes like MPLS have proven laborious to deploy, Segment Routing (SR) drastically eases the deployment of TE paths and is thus increasingly adopted by Internet Service Providers (ISP). There is now a clear need in computing and deploying Delay-Constrained Least-Cost paths (DCLC) with SR for real-time interactive services requiring both low delay and high bandwidth routes. However, most current DCLC solutions are not tailored for SR. They also often lack efficiency (particularly exact schemes) or guarantees (by relying on unbounded heuristics). Similarly to approximation schemes, we argue that the actual challenge is to design an algorithm providing both performances and strong guarantees. However, conversely to most of these schemes, we also consider operational constraints to provide a practical, high-performance implementation.

In this work, we extend and further evaluate our previous contribution, `BEST2COP`. `BEST2COP` leverages inherent limitations in the accuracy of delay measurements, accounts for the operational constraint added by SR, and provides guarantees and bounded computation time in all cases thanks to simple but efficient data structures and amortized procedures. We show that `BEST2COP` is faster than a state-of-the-art algorithm on both random *and* real networks of up to 1000 nodes. Relying on commodity hardware with a single thread, our algorithm retrieves all non-superfluous 3-dimensional routes in under 100ms in both cases. This execution time is further reduced using multiple threads, as the design of `BEST2COP` enables a significant speed-up thanks to a highly parallelizable core which also enables a balanced computing load between threads. Finally, we extend `BEST2COP` to deal with massive-scale ISP by leveraging the multi-area partitioning of these deployments. Thanks to our new topology generator specifically designed to model realistic patterns in such massive IP networks, we show that `BEST2COP` can solve DCLC-SR in approximately 1 second even for ISP having more than 100 000 routers.

Keywords: Traffic Engineering, Segment Routing, DCLC, CSP, Delay Constrained Least Cost, QoS Routing

1. Introduction

Latency is critical in modern networks for various applications. The constraints on the delay are indeed increasingly stringent. For example, in financial networks, vast amounts of money depend on the ability to receive information in real-time. Likewise, technologies such as 5G slicing, in addition to requiring significant bandwidth availability, demand strong end-to-end delay guarantees depending on the service they aim to provide, e.g., less than 15ms for low latency applications such as motion control for industry 4.0, VR, or video games [59]. For such interactive applications, the latency is as critical as the *IGP cost*.

The Interior Gateway Protocol cost is defined as an additive metric that usually reflects both the link's bandwidth and the operator's load distribution choices on the topology. Paths within an IGP are computed by minimizing this cost. Thus, although delay constraints are increasingly important, they should not be enforced to the detriment of the IGP cost. With minimal IGP distances, the traffic benefits from high-bandwidth links and follows the operator's intent in managing the network and its load. With bounded delays, the traffic can benefit from paths allowing for sufficient interactivity. It is thus relevant to minimize the IGP cost while enforcing an upper constraint on the latency. Computing such paths requires to solve DCLC, an NP-Hard problem standing for Delay Constrained Least Cost.

DCLC: a relevant issue. Although one may expect

*Corresponding author
Email addresses: `jr.luttringer@unistra.fr` (Jean-Romain Luttringer), `talfroy@unistra.fr` (Thomas Alfroy), `merindol@unistra.fr` (Pascal Mérindol), `bramas@unistra.fr` (Quentin Bramas), `fclad@cisco.com` (François Clad), `pelsser@unistra.fr` (Cristel Pelsser)

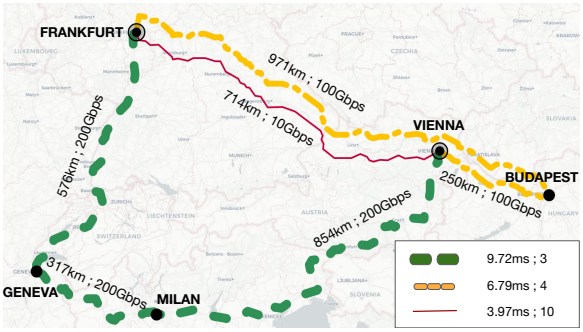


Figure 1: Practical relevance of DCLC in the GEANT network. IGP costs are deduced from the bandwidth of each link. Depending on their needs (in terms of delay and bandwidth), applications can opt for three non-comparable paths between Frankfurt and Vienna.

the two metrics to be strongly correlated in practice, there are various cases where the delay and the IGP cost may be drastically different. For example, the IGP cost may have been tuned arbitrarily by the operator. Heterogeneous infrastructures between countries or geographical constraints may also create this effect. This can be illustrated on real networks, as displayed by Fig. 1. This map is a sample of the GEANT transit network [56]. As fibers often follow major roads, we rely on real road distances to infer the propagation delay of each link while the bandwidth, and so the estimated IGP cost, matches the indications provided by GEANT. A green link has an IGP cost of 1 while the IGP cost is 2 and 10 respectively for the yellow and pink ones.

Note that the two metrics are not correlated, hence all three paths shown between Frankfurt and Vienna offer diverse interesting options. They are non-comparable (or *non-dominated*) paths and form the *Pareto-front* of the paths between the two cities. Either solely the delay matters and the direct link (in pink) should be preferred, or the ISP prefers to favor high capacity links, and the green path, minimizing the IGP cost, should be used. The yellow path, however, offers an interesting compromise. Out of all paths offering a latency well-below 10ms, it is the one minimizing the IGP cost. Thus, it allows to provide strict Service-Level Agreement (< 9ms), while considering the IGP cost. These kind of paths, retrieved by solving DCLC, provide more options by enabling tradeoffs between the two most important networking metrics. Applications such as videoconferences, for example, can then benefit both from real-time interactive voice exchange (delay) and high video quality (bandwidth). In addition, IGP costs are also tuned to represent the operational costs. Any deviation from the shortest IGP paths thus results in additional costs for the operator. For all these reasons, there exist a clear interest for algorithms able to solve and deploy DCLC paths [18]. However and so far, while this problem has received a lot of attention in the last decades from the network research community [30, 24], no technologies were available for an efficient deployment of such paths.

Segment Routing, MSD & DCLC's (large-scale) rebirth. Segment Routing (SR) is a vibrant technology gathering traction from router vendors, network operators and academic communities [53, 69]. Relying on a combination of strict and loose source routing, SR enables to deviate the traffic from the shortest IGP paths through a selected set of nodes and/or links by prepending routing instructions to the packet itself. Such deviations may for example allow to route traffic through a path with lower latency. These deviations are encoded in the form of *segments* within the packet itself. To prevent any packet forwarding degradation, the number of deviations (*i.e.,* instructions) one can encode is limited to MSD (Maximum Segment Depth), whose exact value depends on the hardware. While this technology is adequate to support a variety of services, operators mainly deploy SR in the hopes of performing fine-grained and ECMP[1]-friendly tactical Traffic-Engineering (TE) [1], due to its reduced overhead compared to RSVP-TE [20]. Our discussion with network vendors further revealed a clear desire from operators to efficiently compute DCLC paths deployable with Segment Routing [18]. Such a solution should thus not only encompass Segment Routing, but also fare well on large-sized networks of several thousand of nodes, as already observable in current SR deployments[53]. Indeed, while we showed in [50] that computing DCLC paths for Segment Routing (DCLC-SR) is possible in far less than a second on networks of up to 1000 nodes, scaling to ten or a hundred times more routers remains an open issue.

Challenges & Overview. In this paper, we extend our previous contribution, `BEST2COP` (Best Exact Segment Track for the 2-Constrained Optimal Paths) [50] to make DCLC-TE possible on (very) large scale networks. In order to achieve our goal, we solve several challenges: (i) provide near-exact algorithms with bounded error margin and strong guarantees, (ii) efficiently consider the number of segments to consider the packet manipulation overhead supported by routers, and (iii) scale with very large modern networks, despite the difficulty induced by considering three metrics (cost, delay, and number of segments)[2]. We now briefly explain the main design aspects allowing us to solve these challenges. They will later be explained more thoroughly in their dedicated sections.

Guarantees and practical concerns. DCLC is a well-known NP-Hard problem [72]. While there exist several ways to solve DCLC [30, 24], they usually do not consider the underlying deployment technologies and real-

[1]Equal Cost Multi-Path

[2]While difficult instances are unlikely to occur in practice, our algorithm can tackle even such corner cases. Thus, our proposal is not only efficient in practice but provably correct and efficient even for worst theoretical cases.

life deployment constraints, which often increases the complexity of the problem and limits the available data.

Solution. We keep such considerations at the core of our design. The nature of the paths we compute relies on a stable latency metric (the measured propagation delay) as recommended by [26, 27]. This delay is representative of the experienced delay in our case, as explained later on. Second, because of the inherent imprecision of these measurements and the arbitrary nature of the latency constraint, a small error margin regarding the delay of the computed path may be acceptable. `BEST2COP` may be easily tuned to remain within this acceptable error margin and thus offers strong delay guarantees. `BEST2COP` also leverages this acceptable error margin to compute DCLC paths efficiently, despite the NP-Hard nature of the problem. In addition, we consider the SR deployment technologies, despite the increase in complexity that the later brings to the DCLC problem.

Efficiently encompassing the Maximum Segment Depth constraint (MSD). An SR router can only prepend up to *MSD* routing instructions to a packet at line-rate, i.e., ≈ 10 with the best current hardware. Although we find in our new study, shown in Section 2.3 that this limit does not prevent from deploying most DCLC paths in practice, this constraint must still be taken into account. If ignored, the computed paths have no guarantees to be deployable, as they may exceed MSD.

Solution. To efficiently consider this new additive metric (the number of segments), we propose in [50] a new construct, the multi-metric SR Graph, which results from a transformation of the original graph. Through an adequate graph exploration of this structure, `BEST2COP` manipulates lists of segments natively, and paths requiring more than MSD segments are natively removed from the exploration space. While `BEST2COP` explores this structure natively, one may also explore the original graph and rely on this structure to efficiently convert explored paths to segment lists on the fly. The latter approach may be more appropriate when adapting existing schemes to fit SR constraint, as explained later.

Dealing with massive-scale networks efficiently. As aforementioned, current networks may be quite large, and growing. As such, modern computation schemes should maintain good performance even on large instances.

Solution. `BEST2COP-E` (`BEST2COP` Extended) leverages both multi-threaded architectures and the inherent structure of massive-scale networks to solve DCLC-SR in ≈ 1 second for $\approx 100\,000$ nodes. Indeed, by slightly modifying `BEST2COP` (as presented in [50]), the latter becomes able to greatly benefit from multi-threaded architectures. Furthermore, we present a new proposal, `BEST2COP-E`, which leverages the logical and physical area-partitioning usually observed in massive-scale networks. To evaluate our contribution, we create a topology generator, YARGG, able to construct massive-scale, multi-valuated, and multi-area topologies based on geographical data.

Summary and Contributions. By taking into consideration the operational deployment of constrained paths with SR, the current scale and structure of modern networks, as well as the practicality of the delay measures and constraints, **we designed BEST2COP-E (BEST2COP Extended), a simple efficient algorithm able to solve DCLC-SR in $\approx$ 1s in large networks of $\approx$100 000 nodes**.

The main achievements of our proposal follow the organization of this paper:

- In Section 2, we present SR in further details. Compared to [50], we discuss the context and works related to DCLC in further detail. We also add an evaluation regarding the relevance of SR for deploying DCLC paths;
- In Section 3, we formalize DCLC-SR and also its generalization (2COP). In particular, we describe the network characteristics we leverage and define the construct we use to encompass SR (in Sec. 3.1 and Sec. 3.2 respectively). In Sec. 3.3, we briefly summarize `BEST2COP` (initially introduced in [50] and detailed here in an appendix) and emphasize how it can easily benefit from multi-threaded architectures.

Besides the more detailed background and the evaluation of SR relevance, the main new contributions in this extension come in the last three sections:

- In Sec. 3.4, we extend `BEST2COP` to `BEST2COP-E`, to deal with massive scale networks relying on area partitioning (as with OSPF with a single metric), making it able to solve DCLC efficiently in graphs of $\approx 100\,000$ nodes;
- In Sec. 3.5, we formally define the guarantees brought by `BEST2COP` and its versatility to solve several TE optimization problems at once (*i.e.,* the sub-problems of 2COP), as well as its polynomial complexity;
- Finally, in Section 4, we present our large-scale topology generator, YARGG, and evaluate `BEST2COP-E` on the resulting topologies and compare our proposal to the relevant state-of-the-art path computation algorithm.

2. Back to the Future: DCLC vs SR

2.1. Segment Routing Background and Practical Usages

Segment Routing implements source routing by prepending packets with a stack of up to MSD *segments*. In a nutshell, segments are checkpoints the packet has to go through. There are two main types of segments:

- **Node segments.** A node segment v indicates that the packet should (first) be forwarded to v with ECMP (instead of its final IP destination). Flows are then load-balanced among the best IGP next hops for destination v.
- **Adjacency segments.** Adjacency segments indicate that the packet should be forwarded through a specific interface and its link.

Once computed, the stack of segments encoding the desired path is added to the packet. Routers forward packets according to the topmost segment, which is removed from the stack when the packet reaches the associated intermediate destination.

Adjacency segments may be globally advertised, and thus be used the same way as node segments, or they may only have a local scope and, as such, can only be interpreted by the router possessing said interface. In this case, the packet should first be guided to the corresponding router, by prepending the associated node segment. In the following, as a worst operational case, we consider the latter scenario, as it always requires the highest number of segments.

Segment Routing attracted a lot of interest from the research community. A table referencing most SR-related work can be found in Ventre et al. [69]. While some SR-TE works are related to tactical TE problems (like minimizing the maximum link utilization) taking indirectly into account some delay concerns [37, 35], most of the works related to SR do not focus on DCLC, but rather bandwidth optimization [7, 25, 9], network resiliency [22, 34], monitoring [48, 6], limiting energy consumption [10] or path encoding (the translation of path to segment lists) [31, 28]. Aubry [5] proposes a way to compute paths requiring less than MSD segments while optimizing an additive metric in polynomial time. The number of segments required is then evaluated. This work, however, considers only a single metric in addition to the operational constraints. The problem we tackle (*i.e.,* DCLC paths for Segment Routing) deals with two metrics (in addition to the operational constraints). This additional dimension drastically changes the problem, which then becomes NP-Hard. Some works use a construct similar to ours (presented in details in Sec. 3.2) in order to prevent the need to perform conversions from network paths to segment lists, [43] in particular. However, the authors of [43] do not pretend to solve DCLC and, as such, do not tune the structure the same way (*i.e.,* they do not remove dominated segments, as explained later on), and simply use their construct to sort paths lexicographically.

As aforementioned, while operators seem to mainly deploy SR to perform fine-grained TE, to the best of our knowledge, no DCLC variant exists for specifically tackling SR characteristics and constraints (except for our contribution). Using segments to steer particular flows allows however to deviate some TE traffic from the best IGP paths in order to achieve, for example, a lower latency (and by extension solve DCLC). A realistic example is shown on Fig. 1 where the node segment *Vienna*, as well as considering Vienna as the destination itself, would result in the packets following the best IGP path from Frankfurt to Vienna, i.e., the green dashed path. To use the direct link instead (in plain pink) and so minimize the delay between the two nodes of this example, the associated adjacency segment would have to be used as it enforces a single link path having a smaller delay than the best IGP one (including here two intermediary routers). Finally, the yellow path, offering a non dominated compromise between both metrics (and being the best option if considering a delay constraint of 8ms), requires the use of the node segment *Budapest* to force the traffic to deviate from its best IGP path in green. Before converting the paths to segment lists (and actually deploy them with SR), such non-dominated paths need first to be explored. Computing these paths while ensuring that the number of segments necessary to encode them remains under MSD is at least as difficult as solving the standard DCLC problem since an additional constraint now applies.

2.2. DCLC (Delay-Constrained, Least-Cost), a Well-known Difficult Problem having many Solutions?

DCLC belongs to the set of NP-Hard problems (as well as most related multi-constrained path problems). Intuitively, solely extending the least-cost path is not sufficient, as the latter may exceed the delay constraint. Thus, paths with greater cost but lower delays must be memorized and extended as well. These *non-dominated paths* form the *Pareto front* of the solution, whose size may grow exponentially with respect to the size of the graph. However, DCLC in particular, and related variants and extensions in general, does possess several interesting applications such as mapping specific flows to their appropriate paths (in terms of interactive quality). Thus, these problems have been extensively studied in the past decades. Many solutions have been proposed so far, as summarized in these surveys [42, 24, 30]: they range from to heuristics and approximations to exact algorithms, or even genetic approaches.

Heuristics. Because DCLC is NP-Hard, several polynomial-time heuristics have been designed to limit the worst-case computing time, but at the detriment of any guarantees. For example, [44] only returns the least-cost or least-delay path if one is feasible (*i.e.,* respect the constraints). More advanced proposals try to explore the delay and cost space simultaneously, by either combining in a distributed manner the least-cost and least-delay subpaths [63, 73, 46] or by aggregating both metrics into one in a more or less intricate manner.

Aggregating metrics in a linear fashion [38, 4] preserves the *subpaths optimality principle* (isotonicity of best single-metric paths) and therefore allows to use standard shortest paths algorithms. However, it leads to a loss of relevant information regarding the quality and feasibility of the com-

puted paths [72], in particular if the hull of the Pareto Front is not convex. Some methods try to mitigate this effect by using a k-shortest path approach to possibly find more feasible paths [39, 40], but such an extension may result in a large increase of execution time and may not provide more guarantees. Other heuristics rely on non-linear metric aggregation. While it seems to prevent loss of relevant information, at first glance, such algorithms expose themselves to maintain all non-dominated paths (towards all nodes) as the isotonicity does not hold anymore (while it holds with linear metrics). Since the Pareto Front may be exponential with respect to the size of the graph, those algorithms either simply impose a hard limit on the number of paths that can be maintained (*e.g.,* TAMCRA [15] and LPH [71]), or specifically chose the ones to maintain through previously acquired knowledge (HMCOP [41]). Finally, other works like [17, 32] rely on heuristics designed to solve a variant of DCLC, the MCP problem (Multi-Constrained Paths, the underlying NP-Complete decision version of DCLC – with no optimization objective). It mainly consists of sequential MCP runs using a conservative cost constraint iteratively refined.

Relying on heuristics is tempting, but their lack of guarantees can prevent to enforce strict SLAs even when a suitable path actually exists. One can argue it is particularly unfortunate, as DCLC is only *weakly* NP-hard: it can be solved exactly in pseudo-polynomial time, *i.e.,* polynomial in the numerical value of the input [23]. Said otherwise, DCLC is polynomial in the smallest largest weight of the two metrics once translated to integers. Consequently, it is possible to design FPTAS[3] solving DCLC while offering strong guarantees [58].

Approximations. Numerous FPTAS have been proposed to solve DCLC and related constrained shortest path problems. All the following algorithms fall into this category. The common principle behind these schemes is to reduce the precision (and/or magnitude) of the considered metrics. This can be performed either directly, by *scaling and rounding* the weights of each link, or indirectly, by dividing the solution space into intervals and only maintaining paths belonging to different intervals (*Interval partitioning*) [64]. Scaling methods usually consider either a high-level dynamic programming scheme or a low-level practical Dijkstra/Bellman-Ford core with pseudo-polynomial complexity, and round the link costs to turn their algorithms into an FPTAS (see for example Hassin [36], Ergun et al. [16] or Lorenz and Raz [47] methods). Goel et al. [29], in particular, chose to round the delay instead of the cost and can consider multiple destinations (as our own algorithm).

Most interval partitioning solutions explore the graph through a Bellman-Ford approach. The costs of the paths are mapped to intervals, and only the path with the lowest delay within a given interval is kept. The size of the intervals thus introduces a bounded error factor [36, 67]. In particular, HIPH [66] offers a dynamic approach between an approximation and exact scheme. It proposes to maintain up to x non-dominated paths for each node and stores eventual additional paths using an interval partitioning strategy. This allows the algorithm to be exact on simple instances (resulting in a limited Pareto front, *i.e.,* polynomial in the number of nodes, in particular when it is bounded by x) and offer strong guarantees on more complex ones. This versatility is an interesting feature, as most real-life cases are expected to be simple instances with a bounded Pareto front size, in particular because one of the metrics may be coarse by nature. For these reasons, not only approximation schemes can offer practical solutions (with a bounded margin error) but also exact algorithms (with controlled performance), as they may be viable in terms of computing time for simple real-life IP networking instances.

Exact methods. Numerous exact methods have indeed also been studied extensively to solve DCLC. Some methods simply use a k-shortest path approach to list all paths within the Pareto front [55, 57]. On the other hand, Constrained Bellman-Ford [70] (ironically, also called Constrained Dijkstra as it uses a priority queue – denoted PQ in the following) explores paths by increasing delays and lists all non-dominated paths towards each node. Several algorithms use the same principle but order the paths differently within the queue, relying either on a lexicographical ordering, ordered aggregated sums, or a simple FIFO/LIFO ordering [51, 52, 8]. Most notably, A* Prune [45] is a multi-metric adaptation[4] of A* relying on a PQ where paths known to be unfeasible are pruned. Two-phase methods [62] first find paths lying on the convex hull of the Pareto front through multiple Dijkstra runs, before finding the remaining non-dominated path through implicit enumerations.

Finally, `SAMCRA` [68] is a popular and well-known multi-constrained path algorithm. Similarly to other Dijkstra-based algorithms, `SAMCRA` relies on a PQ to explore the graph but instead of the traditional lexicographical ordering, it relies on non-linear cost aggregation. Among feasible paths (others are natively ignored) it first considers the one that minimizes its maximum distance to the multiple constraints. Such a path ranking to deal with the PQ is supposed to increase its performance with respect to other PQ organizations.

As we have seen so far, while many solutions exist, most possess certain drawbacks or lack certain features to reconcile both the practice and the theory. Heuristics do not always allow to retrieve the existing paths enforcing strict SLAs, while exact solutions are not able to guarantee a reasonable maximum running time when difficult in-

[3]Fully Polynomial Time Approximation Scheme.

[4]This adaptation is exact, *i.e.,* not a heuristic, as the estimated cost underestimates the actual distance towards the destination.

Algorithms	Practical Features			Exactitude vs Performance					
	Multi-Dest Single Run	SR Ready	Multi-thread Ready	Bounded Pareto Front		Coarse Metric		All Cases	
LARAC [40]	×	×	×	×	✓	×	✓	×	✓
LPH [71]	✓	×	×	✓	✓	~	✓	×	✓
HMCOP [41]	×	×	×	×	✓	×	✓	×	✓
HIPH [66]	✓	×	×	✓	✓	✓	✓	~	✓
Hassin [36]	×	×	×	~	✓	✓	✓	~	✓
Tsaggouris et al. [67]	✓	×	×	~	✓	✓	✓	~	✓
Raith et al. [62]	×	×	×	✓	✓	✓	✓	✓	×
A* Prune. [45]	×	×	×	✓	✓	✓	✓	✓	×
SAMCRA [68]	✓	×	×	✓	✓	✓	✓	✓	×
BEST2COP	✓	✓	✓	✓	✓	✓	✓	~	✓

Table 1: Qualitative summary of a representative subset of DCLC-compatible algorithms showcasing their practicality, exactitude, and performance. In the *Practical Features* column, the green check-mark indicates whether the algorithm supports the corresponding feature (while the red cross denotes the opposite). In the *Exactitude vs Performance* column, the two subcolumns associated which each three scenarios show how the latter impact (*i*) the exactitude (exact, strong guarantees, no guarantees) and (*ii*) the performance of the algorithm (polynomial time or not). While the orange tilde denotes strong guarantees in terms of exactitude, green check-marks (and red crosses respectively) either indicate exact results (no guarantees resp.) or polynomial-time execution (exponential at worst resp.) for performance. For both subcolumns *Bounded Pareto Front* and *Coarse Metric*, we consider the case where their spreading is polynomial with respect to the number of vertices in the input graph (and as such predictable in the design/calibration of the algorithm).

stances arise, although both features are essential for real-life deployment. On the other hand, FPTAS can provide both strong guarantees and a polynomial execution time. However, they are often found in the field of operational research where, at best, possible networking applications and assumptions are discussed, but are not investigated. Because of this, the deployment of the computed paths, with SR and its MSD in particular, is not taken into consideration. It is worth to note that the number of segments is not a standard metric as it is not simply a weight assigned to each edge in the original graph (that is, without a specific construct, it requires to be computed on the fly for each visited path). Considering the latter can have a drastic impact on the performance of the algorithms not designed with this additional metric in mind. In addition, not all the algorithms presented here and in Table 1 are single-source multiple-destinations. Finally, none of these algorithms evoke the possibility to leverage multi-threaded architectures, an increasingly important feature as such computations now tend to be performed by dedicated Path Computation Elements or even in the cloud.

Our contribution, `BEST2COP`, aims to close this gap by mixing the best existing features (such as providing both a limited execution time and strong guarantees in terms of precision in any cases) and adapt them for a practical modern usage in IP networks deploying SR. Table 1 summarizes some key features of a representative subset of the related work. Similarly to FPTAS, `BEST2COP` rounds one of the metrics of the graph. However, conversely to most algorithms, `BEST2COP` does not sacrifice accuracy of the cost metric, but of the measured delay. Because of the native inaccuracy of delay measurements (and the arbitrary nature of its constraint), this does not prevent `BEST2COP` from being technically exact in most practical cases. In addition (and akin to [66]), `BEST2COP` can easily be tuned to remain exact on all simple instances with a bounded Pareto front regardless of the accuracy of the metrics. Thus, `BEST2COP` can claim to return exact solutions in most scenarios and, at worst, ensure strict guarantees in others (for theoretical exponential instances). In all cases, `BEST2COP` possesses a pseudo-polynomial worst-case time complexity. `BEST2COP` was designed while keeping the path deployment aspect of the problem in mind. A single run allows to find all DCLC paths (and many variants as we will see later) to all destinations. The MSD constraint related to SR is taken into account natively. As a result, paths requiring more than MSD segments are excluded from the exploration space. The outer loop of `BEST2COP` can be easily parallelized, leading to a non-negligible reduction in the execution time. In Sec. 4, relying on a performance comparison between `BEST2COP` and `SAMCRA`, we will show that `BEST2COP` outperforms `SAMCRA` when running `SAMCRA` as published in [68]. While `SAMCRA` reaches similar performance when benefitting from SR-aware methods from our own design (explained thoroughly in the remainder of this paper), `BEST2COP` outperforms its competitor when relying on multi-threading.

Last but not least, `BEST2COP` has been adapted for multi-area networks and leverages the structures of the latter, allowing it to solve DCLC on very large ($\approx 105\,000$ nodes) domains in one second. To the best of our knowledge, such large-scale experiments and results have neither been

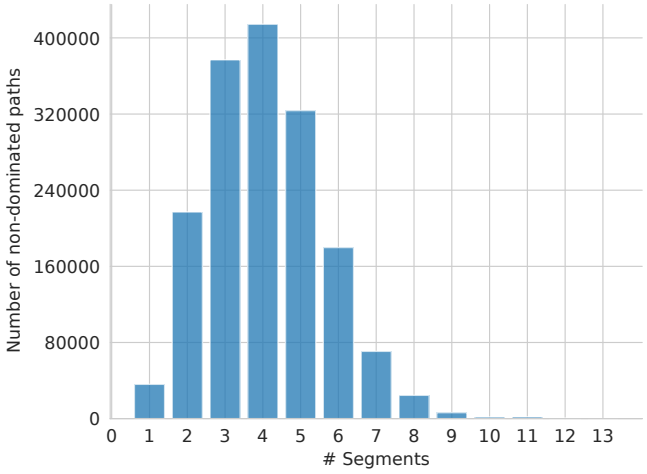


Figure 2: Required number of segments for all DCLC solutions, in a network of 45 000 nodes generated by YARGG, with delay constraints of up to 100ms.

conducted nor achieved within SR domains [5].

2.3. SR is Relevant for DCLC: MSD is not a Limit

Given the MSD constraint, one may question the choice of SR for deploying DCLC paths in practice. Indeed, in some cases, in particular if the metrics are not aligned[6], constrained paths may required more than MSD detours to satisfy a stringent latency constraint.

While it has been shown that few segments are required for most current SR usages (e.g. for TI-LFA [7] or when considering only one metric) [19, 5], to the best of our knowledge, there is no similar study for our specific use-case, i.e. massive scale networks with two valuation functions (delay and IGP cost). This is probably one of the most exciting challenge for SR as DCLC is a complex application. However, since such massive-scale computer network topologies are not available publicly, we rely on our own topology generator whose detailed description is available in section 4.2. These topologies follow a standard OSPF-like area division. and both metrics (delay and IGP cost) follow a realistic pattern. For this analysis, we opt for a worst-case graph having $\approx 45\,000$ nodes and $\approx 92\,000$ edges scattered in 140 areas.

For this analysis, we keep track, for each destination, of all the solutions solving DCLC for all delay constraints up to 100ms, and extract the necessary number of segments. In other words, we show the number of segments required to encode all non-dominated (and thus practically relevant for some given constraint) paths, considering all delay constraints up to 100ms. The results are shown in Fig. 2.

One can see that most paths require less than 10 segments, meaning that performant hardware should be able to deploy most DCLC paths. However, some corner-cases requiring more than 10 segments do exist, probably arising from stringent delay constraints. In addition, less performant hardware (*e.g.,* with MSD ≈ 5), while able to deploy the majority of DCLC paths, can not deploy *any* DCLC path. Note that several mechanisms exist to bypass this limit. Flexible Algorithms [60] allows to compute shortest paths and create segments with other metrics (*e.g.,* the delay). Binding segments [21] allows to "compress" a segment list in a single segment, which is uncompressed when popped from the list. However, both techniques increases the message exchange, number of states to maintain, and overall complexity. Their usage should thus be limited to a few corner cases.

Consequently, our analysis exhibits two main points. First, SR is appropriate to deploy TE paths. Indeed, the majority of DCLC paths should be deployable within the MSD constraint, if not all when using performant hardware. Second, since there may however exist DCLC paths requiring more than MSD segments, this limit must be considered to compute feasible paths correctly. Otherwise, a single non-feasible path dominating feasible ones is enough to lead to an incorrect algorithm. The underlying path computation algorithm must then efficiently consider delay, IGP cost and the number of segments to ensure its correctness.

3. BEST2COP(E): Efficient Data Structures and Algorithms for Massive Scale Networks

This section presents our contributions. We introduce and define preliminary notations and concepts used to design `BEST2COP`, before describing the data structures used by our algorithm. In section 3.3, we describe our algorithm, `BEST2COP`, and show how we extend it for massive-scale networks divided in several areas in section 3.4.

We have shown that SR seems indeed appropriate (as desired) for fine-grained delay-based TE. We thus aim to solve DCLC in the context of an ISP deploying SR, leading to the DCLC-SR problem that considers the IGP cost, the propagation delay, and the number of segments.

For readability purposes, we denote:

- M_0 the metric referring to the number of segments, with the constraint $c_0 = \text{MSD}$ applied to it;
- M_1 the delay metric, with a constraint c_1;
- M_2 the IGP metric being optimized.

Given a source s, **DCLC-SR** consists in finding, for *all* destinations, a *segment list* verifying two constraints, c_0

[5] Some elementary algorithms (such as Multi-constrained Dijkstra) and more intricate solutions exhibit impressive computing times on even more massive road networks [33]. However, such networks are less dense than IP ones (and with metrics that are also even more correlated).

[6] The delay and the IGP costs in particular. Since node segments represent best IGP paths, the IGP cost and the number of segments will most likely be aligned by design

[7] Topology Independent Loop Free Alternate

and c_1, respectively on the number of segments (M_0) and the delay (M_1), while optimizing the IGP distance (M_2). We denote this problem DCLC-SR(s, c_0, c_1). On Fig. 1, we would have DCLC-SR(*Frankfurt*, 3, 8) $\supset$ *Frankfurt – Budapest – Vienna*. This DCLC path (shown in yellow in Fig 1), is indeed the best option to reach Vienna when considering an arbitrary delay constraint of 8ms. Since the best IGP path from Frankfurt to Vienna (the green one) does not go through Budapest, encoding this DCLC path requires at least one detour, i.e. one segment (here, a node segment instructing the packet to go through Budapest first).

To solve such a challenging problem, efficient data structures are required. In the following, we first introduce the constructs we leverage and how we benefit from the inaccuracy of real delay measurements in particular.

3.1. DCLC and True Measured Delays

3.1.1. Leveraging measurement inaccuracy

As mentioned, DCLC is weakly NP-Hard, and can be solved exactly in pseudo-polynomial time. In other words, as long as either the cost of the delay possesses only a limited number of distinct values (i.e., paths can only take a limited number of distinct distances), the Pareto front of the paths' distances is naturally bounded in size as well, making DCLC tractable and efficiently solvable[8]. Such a metric thus has to be bounded and possess a coarse accuracy (i.e., be discrete). Although this has little impact when solving DCLC in a theoretical context, it can be strongly leveraged to solve DCLC efficiently thanks to the characteristics of real ISP networks.

We argue that the metrics of real ISP networks do indeed possess a limited number of distinct values. Although `BEST2COP` can be adapted to fit any metric, we argue that M_1, the propagation delay, is the most appropriate one. Indeed, IGP costs depend on each operators' configurations. For example, while some may rely on few spaced weights, others may possess intricate weight systems where small differences in weights may have an impact. Thus, bounding the size of the Pareto front based on the IGP costs is not only operator-dependant, but might still result in a very large front.

On the other hand, the delay (i) is likely strongly bounded, and (ii) can be handled as if having a coarse accuracy in practice. For TE paths, the delay constraint is likely to be very strict (10ms or less). Second, while the delay of a path is generally represented by a precise number in memory, the actual accuracy, *i.e.,* the trueness t of the measured delay is much coarser due to technical challenges [3, 2]. In addition, delay constraints are usually formulated at the millisecond granularity with a tolerance margin, meaning that some loss of information is acceptable.

Thus, floating numbers representing the delays can be truncated to integers, e.g., taking 0.1ms as unit. This allows to easily bound the number of possible non-dominated distances to $c_1 \times \gamma$, with γ being the desired level of accuracy of M_1 (the inverse of the unit of the delay grain, here 0.1ms). For example, with c_1 = 100ms and a delay grain of 0.1ms ($\gamma = \frac{1}{0.1} = 10$), we have only 1000 distinct (truncated) non-dominated pairs of distances to track at worst. This leads to a predictable and bounded Pareto front. One can then store non-dominated distances within a static array, indexed on the M_1-distance (as there can only be one non-dominated couple of distances (M_1, M_2) for a given M_1-distance).

In the remaining of the paper, Γ denotes the size allocated in memory for this Pareto front array (i.e., $\Gamma = c_1 \times \gamma$). When t, i.e., the real level of accuracy, is lower (or equal) than γ, the stored delay can be considered to be exact. More precisely, it is discretized but with no loss of relevant information. When t is too high, one can choose γ such that $\gamma < t$, to keep Γ at a manageable value. In this case, some relevant information can be lost, as the discretization is too coarse. While this sacrifices the exactitude of the solution (to the advantage of computation time), our algorithm is still able to provide predictable guarantees in such cases (*i.e.,* a bounded error margin on the delay constraint). This is further discussed in Section 3.5.2.

3.1.2. Fine, but which delay?

Referring to a path's delay may be ambiguous. Indeed, this characteristic is not monolithic. The total delay is mainly composed of the propagation delay and the queuing delay. Both delays may play an important part in the overall latency, though none can be stated to be the main factor [65]. Although the propagation delay is stable, the queuing delay may vary depending on the traffic load. However, in order to compute TE paths, the delay metric must be advertised (usually within the IGP itself). For this reason, it is strongly recommended to use a stable estimate of the delay, as varying delay estimations may lead to frequent re-computations, control-plane message exchanges, and fluctuating traffic distribution [26, 27].

For this reason, we use the propagation delay, as recommended in [26, 27]. The latter is usually measured through the use of a priority queue, ignoring so the queuing delay. Its value is deduced as a minimum from a sampling window, increasing so its stability [13]. Using this delay not only makes our solution practical (as we rely on existing measurements and respect protocol-related constraints), but is actually pertinent in our case. In practice, flows benefitting from DCLC paths benefit from a queue with high priority and experience negligible queueing delays. In addition, the amount of traffic generated by such premium interactive flows can be controlled to remain small

[8] Metric M_0 is omitted for now as this trivial distance is only required for SR and discussed in details later. While dealing with a three-dimensional Pareto front seems more complex at first glance, we will show that SR eventually reduces the exploration space because its operational constraint is very tight in practice and easy to handle efficiently.

enough if it is not limited by design. Consequently, not only is there no competition between premium flows and best-effort traffic, but these flows do not generate enough traffic to lead to significant competition between themselves. Thus, the experienced delay is actually agnostic of the traffic load for our use-case, making the propagation delay a relevant estimate. Consequently, we use the discretized propagation delay, enabling both practical deployment and the limitation of the number of non-dominated distances, within our structure used to encompass Segment Routing natively, the SR graph.

3.2. The SR Graph and 2COP

To solve DCLC-SR efficiently, as well as its comprehensive generalization, 2COP, we rely on a specific construct used to encompass SR, the delay, and the IGP cost: the *multi-metric SR graph.*

3.2.1. Turning the Physical Graph into a Native SR Representation

This construct represents the segments as edges to natively deal with the M_0 metric and its constraint, c_0 = MSD. The valuation of each edge depends on the distance of the path encoded by each segment. While the weights of an adjacency segment are the weights of its associated local link, the weights of a node segment are the distances of the ECMP paths it encodes: the (equal) IGP cost (*i.e.,* , M2-distance), and the lowest guaranteed delay (*i.e.,* , the M1-distance), *i.e.,* the worst delay among all ECMP paths. Hence, computing paths on the SR graph is equivalent to combining stacks of segments (and the physical paths they encode), as stacks requiring x segments are represented as paths of x edges in the SR graph (agnostically to its actual length in the raw graph). The SR graph can be built for all sources and destinations thanks to an All Pair Shortest Path (APSP) algorithm. Note that this transformation is inherent to SR and leads to a complexity of $O(n(n\log(n)+m))$, for a raw graph having n nodes and m edges, with the best-known algorithms and data structures. This transformation (or rather, the underlying APSP computation) being required for any network deploying TE with SR (the complexity added by our multi-metric extension being negligible), we do not consider it as part of our algorithm presented later.

This transformation is shown in Fig. 3, which shows the SR counterpart of the raw graph provided in Fig. 1. To describe this transformation more formally, let us denote $G=(V,E)$ the original graph, where V and E respectively refer to the set of vertices and edges. As G can have multiple parallel links between a pair of nodes (u,v), we use $E(u,v)$ to denote all the direct links between nodes u and v. Each link (u,v) possesses two weights, its delay $w_1^G((u,v))$ and its IGP cost $w_2^G((u,v))$. The delay and the IGP cost being additive metrics, the M_1 and M_2 distances of a path p (denoted $d_1^G(p)$ and $d_2^G(p)$ respectively) are the sums of the weights of its edges.

From G, we create a transformed multigraph, the SR graph denoted $G'=(V,E')$. While the set of nodes in G' is the same as in G, the set of edges differs because E' encodes segments as edges representing either adjacency or node segments encoding respectively local physical link or sets of best IGP paths (with ECMP). The M_i-weight of an edge in G' is denoted $w_i^{G'}((u,v))$. However, to alleviate further notations, we denote simply $d_i(p)$ the M_i distance of a path in G' instead of $d_i^{G'}(p)$. Note that if G is connected, then G' is a complete graph thanks to node segments.

SR graph: Node segment encoding. A node segment, encoding the whole set $P_G(u,v)$ of ECMP best paths between two nodes u and v, is represented by exactly one edge in $E'(u,v)$. The M_2-weight $w_2^{G'}((u,v))$ of a node segment is the (equal) M_2-distance of $P_G(u,v)$. Since, when using a node segment, packets may follow any of the ECMP paths, we can only guarantee that the delay of the path will not exceed the maximal delay out of all ECMP paths. Consequently, its M_1-weight $w_1^{G'}((u,v))$ is defined as the maximum M_1-distance among all the paths in $P_G(u,v)$. Links representing node segments in G' thus verify the following:

$$w_1^{G'}((u,v)) = \max_{p\in P_G(u,v)} d_1^G(p)$$
$$w_2^{G'}((u,v)) = d_2^G(p) \quad \text{for any } p\in P_G(u,v)$$

SR graph: Adjacency segment encoding. An adjacency segment corresponds to a link in the graph G and is represented by an edge (u_x,v) in $E'(u,v)$, whose weights are the ones of its corresponding link in G, only if it is not dominated by the node segment $(u,v)_{G'}$ for the same pair of nodes, *i.e.,* if $w_1^{G'}((u,v)) > w_1^G((u_x,v))$, or by any other non-dominated adjacency segments (u_y,v), *i.e.,* if $w_1^G((u_y,v)) > w_1^G((u_x,v))$ or $w_2^G((u_y,v)) > w_2^G((u_x,v))$, where (u_x,v) and (u_y,v) are two different outgoing links of u in $E(u,v)$[9].

Fig. 3 illustrates the result of such a transformation: one can easily identify the three non-dominated paths between Frankfurt and Vienna, bearing the same colors as in Fig. 1. The green path (*i.e.,* the best M_2 path) is encoded by a single node segment. The pink, direct path (*i.e.,* the best M_1 path) is encoded by an adjacency segment (the double line in Fig. 3). The yellow paths (the solution of DCLC-SR(Frankfurt, 3, 8) and an interesting tradeoff between M_1 and M_2) requires an additional segment, in order to be routed through Budapest. Note that in practice, the last segment is unnecessary if it is a node segment, as the packet will be routed towards its final IP destination through the best M_2 paths natively.

Our multi-metric SR graph (or equivalent constructs gathering the multi-metric all-pair shortest path data) is mandatory to easily consider the number of segments

[9] If two links have exactly the same weights, we only add one adjacency segment in G'

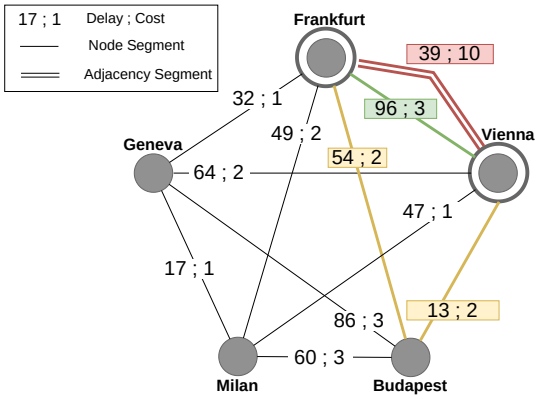


Figure 3: This Figure shows the network from Fig. 1 translated into an SR graph. The SR graph encodes segments as edges. Plain edges represent node segments, *i.e.,* sets of ECMP paths. Double-lines are adjacency segments, here only (*Frankfurt*, *Vienna*), and are visible only if they are not dominated by other segments. Colored edges refer to the paths highlighted in Fig. 1.

necessary to encode the paths being explored. However, its usage can differ in practice. We envision two modes which allow to consider this additional "off the graph" metric, using our SR Graph.

Using the SR Graph to perform path conversions. One of the two options is to run the path computation algorithm on the original topology, and convert the paths being explored to segment lists. Performing this conversion is however not trivial. One must return the minimal encoding of the given path (with respect to the number of segments) while correctly managing the (forced) path diversity brought by ECMP, which may exhibit heterogeneous delays. However, one can efficiently perform such conversion when relying on our SR graph. By summarizing the relevant information (*i.e.,* the worst-case delay within ECMP paths), the SR Graph allows to easily consider the ECMP nature of SR within a multi-metric context. However, the segment metric M_0 is peculiar. Extending a path does not always imply an increase in the number of necessary segments. Furthermore, the number of segments required to encode two distinct paths may evolve differently, even when the latter are extended from the same node with the same edge. Because of these properties, the way to check paths for dominancy must be revised. This extended dominance check may lead to an increased number of paths to extend, and thus to a higher worst-case complexity. Further details regarding the conversion algorithm and the extended dominancy check can be found in Appendix B.

Using the SR Graph natively. Another method is to run the path computation algorithm directly on the SR graph we described. Note that this forces the algorithm to run on a complete graph, which may significantly increase the overall complexity. However, the segment metric M_0, originally an "off the graph" metric with singular properties, becomes a standard graph metric, as it is now expressed by the number of edges that compose the paths (a path encoded by x segments has x edges within the SR Graph). This method also allows using standard, known algorithms as-is to solve the DCLC-SR problem.

When designing our algorithm, `BEST2COP`, we use the second approach. Indeed, by using a bellman-ford-like exploration of the SR Graph, one can not only easily prune paths requiring more than *MSD* segments, but also benefit from efficient Pareto front management and multi-threading. These various features allow `BEST2COP` to efficiently solve not only DCLC-SR, but also 2COP, a more general and practically relevant problem regarding the computation of constrained paths within an SR domain. Note, however, that we will provide our competitor with both approaches to make the evaluation as fair as possible.

3.2.2. The 2COP Problem(s)

Solving DCLC-SR exactly requires, by definition, to visit the entirety of the Pareto front for all destinations. However, although only some of these paths are DCLC-SR solutions for a given delay constraint, all paths visited during this exploration may be of some practical interest. In particular, some of them solve problems similar to DCLC but with different optimization strategies and constraints. By simply memorizing the explored paths (*i.e.,* storing the whole Pareto front within an efficient structure), one can solve a collection of practically relevant problems. For instance, one may want to obtain a segment path that minimizes the delay, another the IGP-cost, or the number of segments. Solving 2COP consists in finding, for all destinations, paths optimizing all three metrics independently, and respecting the given constraints. We formalize this collection of problems as 2COP. Solving 2COP enables more versatility in terms of optimization strategies and handles heterogeneous constraints for different destinations. Simply put, while DCLC-SR is a one-to-many DCLC variant taking MSD into account, 2COP is more general as it includes all optimization variants.

With initial constraints c_0, c_1, c_2, `BEST2COP` solves 2COP, *i.e.,* returns in a single run paths that satisfy smaller constraints c'_0, c'_1, c'_2 for any $c'_i < c_i$, $i = 0, 1, 2$, offering more flexibility than simply returning the DCLC-SR solution.

Definition. ***2-Constrained Optimal Paths (2COP)***
Let $f(M_j, c_0, c_1, c_2, s, d)$ be a function that returns a feasible segment path from s to d (if it exists), verifying all constraints $c_i, 0 \leq i \leq 2$ and optimizing $M_j, j \in 0, 1, 2$. For a given source s and given upper constraints c_0, c_1, c_2, we have

$$2COP(s, c_0, c_1, c_2) = \bigcup_{\substack{\forall d \in V, \\ \forall j \in \{0,1,2\}, \\ \forall c'_j \leq c_j}} f(M_j, c'_0, c'_1, c'_2, s, d)$$

Observe that, for any $s \in V$, DCLC-SR(s, c_0, c_1) consists of the paths in $2COP(s, c_0, c_1, \infty)$ minimizing M_2. Looking at Fig. 3, we have two interesting examples (we rely on the first capital letter of the cities):

$$f(M_2, 3, 70, \infty, F, V) = (F, B)|(B, V)\ (67, 4)$$
$$f(M_1, 3, \Gamma, \infty, G, B) = (G, M)|(M, B)\ (77, 4)$$

In the second example, recall that the M1-distances are truncated to obtain integer values and Γ is the maximum c_1 constraint we consider (multiplied by γ).

When the delay accuracy allows to reduce the problem's complexity sufficiently, `BEST2COP` can solve exactly any of the variants within 2COP and return any desired output of the image of f.

In Sec. 3.5.2, we detail how we can handle each 2COP variant with guarantees when the delay accuracy is too high to provide exact solutions while remaining efficient. Solving 2COP can be implemented as efficiently as solving only DCLC-SR.

3.3. Our Core Algorithm for Flat Networks

In this section, we describe `BEST2COP`, our algorithm efficiently solving 2COP (and so DCLC-SR). Its implementation is available online[10]. Although `BEST2COP` was already described in our previous contribution, a complete and far more detailed algorithmic description can be found in Appendix A for the interested reader.

Akin to the SR graph computation, `BEST2COP` can be run on a centralized controller but also by each router. Its design is centered around two properties. First, the graph exploration is performed so that paths requiring i node segments are found at the $i^{th} + 1$ iteration[11], to natively tackle the MSD constraint. Second, `BEST2COP`'s structure is easily parallelizable, allowing to benefit from multi-core architectures with low overhead.

Simply put, at each iteration, `BEST2COP` starts by extending the known paths by one segment (one edge in the SR graph) in a Bellman-Ford fashion (a not-in-place version to be accurate). Paths found during a given iteration are only checked loosely (and efficiently) for dominancy at first. This extension is performed in a parallel-friendly fashion that prevents data-races, allowing to easily parallelize our algorithm. Only once at the end of an iteration are the newly found paths filtered and thoroughly checked for dominancy, to reflect the new Pareto front. The remaining non-dominated paths are in turn extended at the next iteration. These steps only need to be performed $MSD \approx 10$ times, ignoring so all paths that are not deployable through SR. When our algorithm terminates, the results structure contains, for each segment number, all the distances of non dominated paths from the source towards all destinations. The interested reader may find further details regarding how 2COP solutions are extracted from the results structure in Appendix 3.

The good performance of `BEST2COP` comes from several aspects. First, the fact that paths requiring more than MSD segments are natively excluded from the exploration space. Second, well-chosen data structures benefiting from the limited accuracy of the delay measurements to limit the number of paths to extend. This allows to manipulate arrays of fixed size, because the Pareto front of distances towards each node is limited to Γ at each step (enabling very efficient read/write operations). Third, using a Bellman-Ford approach allows not only to easily parallelize our algorithm but also to perform lazy efficient update of the Pareto front. Indeed, a newly found path may only be extended at the *next* iteration. Thus, we can efficiently extract the non-dominated paths from all paths discovered during the current iteration in a single pass, once at the end of the iteration. Conversely, other algorithms tend to either check for dominancy whenever a path is discovered (as the later may be re-extended immediately), or not bother to check for dominancy at all, *e.g.,* by relying solely on interval partitioning to limit the number of paths to extend.

3.4. For Massive Scale, Multi-Area Networks

As shown in [50], this algorithms exhibits great performance on large-scale networks of up to 1000 nodes ($\approx$ 15ms). However, since the design of `BEST2COP` implies a dominant factor of $|V|^2$ in term of time complexity[12] (the SR graph being complete), recent SR deployments with more than 10 000 nodes would not scale well enough. The sheer scale of such networks, coupled with the inherent complexity of TE-related problems, makes 2COP very challenging if not impossible to practically compute at first glance. In fact, even `BEST2COP` originally exceeds 20s when dealing with $\approx$15 000 nodes. As we will see in the evaluations, this is much worse with concurrent options.

In this section, we extend `BEST2COP` in order to deal efficiently with massive scale networks. By leveraging the physical and logical partitioning usually performed in such networks, we manage to solve 2COP in $\approx$ 1s even in networks of 100 000 nodes.

3.4.1. Scalabity in Massive Network & Area decomposition

The scalability issues in large-scale networks do not arise solely when dealing with TE-related problems. Standard intra-domain routing protocols encounter issues past several thousands of nodes. Naive network design creates a large, unique failure domain resulting in numerous computations and message exchanges, as well as tedious management. Consequently, networks are usually divided, both logically and physically, in *areas*. This notion exists in

[10] `https://github.com/talfroy/BEST2COP`

[11] Note that each adjacency segment translates to at least one necessary segment, two if they are not globally advertised and not subsequent.

[12] The detailed complexity is given in section 3.5.1

both major intra-domain routing protocols (OSPF and IS-IS). In the following, we consider the standard OSPF architecture and terminology but our solution can be adapted to fit any one of them

Areas can be seen as small, independent sub-networks (usually of around 100 - 1000 nodes at most). Within OSPF, routers within an area maintain a comprehensive topological database of their own area only. Stub-areas are centered around the *backbone*, or area 0. *Area Border Routers*, or ABRs, possess an interface in both the backbone area and a stub area. Being at the intersection of two areas, they are in charge of sending a summary of the topological database (the best distance to each node) of one area to the other. There are usually at least two ABRs between two areas. We here (and in the evaluation) consider two ABRs, but the computations performed can be easily extended to manage more ABRs. Summaries of a non-backbone area are sent through the backbone. Upon reception, ABRs inject the summary within their own area. In the end, all routers possess a detailed topological database of their own area and the best distances towards destinations outside of their own area.

3.4.2. Leveraging Area Decomposition

This partitioning creates obvious separators within the graph, the ABRs. Thanks to the latter, we can leverage this native partition in a similar divide-and-conquer approach, adapted to the computation of 2COP paths, by running `BEST2COP` at the scale of the areas before exchanging and combining the results. We do not only aim to reduce computation time, but also to keep the number and size of the exchanged messages manageable.

We now explain how we perform this computation in detail. For readability purposes, we rely on the following notations: $\mathcal{A}_x$ denotes area x. A_x denotes the ABR between the backbone and $\mathcal{A}_x$. When necessary, we may distinguish the two ABR $A1_x$ and $A2_x$. Finally, $\text{b2cop}(\mathcal{A}_x, s, d)$ denotes the results (the non-dominated paths) from s to d within $\mathcal{A}_x$. When d is omitted, we consider all routers within $\mathcal{A}_x$ as destination. Figure 4 illustrates a network with three areas, x, y and 0, the backbone area.
We here chose to detail a simple distributed and router-centric variant of our solution. However, our solution may well be deployed in other ways, e.g. relying on controllers, or even a single one. In such cases, the computation could be parallelized per area if needed. Such discussion is left for future work.

Working at area scale. Due to the area decomposition, routers do not possess the topological information to compute a full, complete SR graph of the whole network. Thus, we make routers only compute the SR graph of their own area(s). Because exchanging the SR graphs themselves implies a large volume of information to share, we instead make the ABRs exchange their 2COP paths (i.e., the non dominated paths to all destinations of their areas) since we limit their numbers to Γ at worst. This exchange still provides enough information for all routers to compute all 2COP paths for every destination.

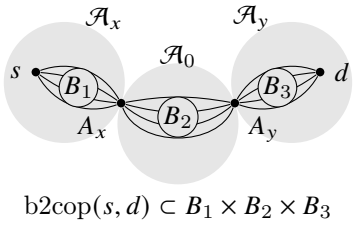


Figure 4: The set of solutions across areas is obtained from the cartesian product of the solutions in each area.

More formally, each ABR A_x computes $\text{b2cop}(\mathcal{A}_x, A_x)$ and exchange the results with $A_y, \forall y \neq x$. Areas being limited to a few hundreds routers on average, this computation is very efficient. Note that ABRs also compute $\text{b2cop}(\mathcal{A}_0, A_x)$, but need not exchange it, as all ABRs perform this computation. Exchanging the computed 3D Pareto front has a message complexity of $|V| \times c_0 \times \Gamma$ at worst in theory. In practice, we expect both the size of Pareto fronts and the number of relevant destinations to consider to be fairly low ($<< \Gamma$ and $<< |V|$ resp.). In the case of non-scalable Pareto fronts, one can opt for sending only part of them but at the cost of relaxing the guarantees brought by `BEST2COP`.

After exchanging messages, any ABR A_x should know the non-dominated paths from itself to $A_y, \forall y \neq x$, and the non-dominated paths from A_y to all nodes within $\mathcal{A}_y$. By combining this information, we can compute the non-dominated paths from A_x to all nodes within $\mathcal{A}_y$, as we will now detail.

Cartesian product. Since ABRs act as separators within the graph, to reach a node within a given area $\mathcal{A}_y$, it is necessary to go through one of the corresponding ABRs A_y. It thus implies that non-dominated paths to nodes within $\mathcal{A}_y$ from A_x can be found by combining $\text{bcop}(\mathcal{A}_0, A_x, A_y)$ with $\text{bcop}(\mathcal{A}_y, A_y)$. In other words, by combining, with a simple cartesian product, the local non-dominated paths towards the ABRs of a given zone with the non-dominated paths from said ABRs to nodes within the corresponding distant areas, one obtains a superset of the non-dominated paths towards the destinations of the distant area. In practice, since several ABR can co-exist, it is necessary to handle the respective non-dominated paths ($\text{bcop}(\mathcal{A}_y, A1_y)$ and $\text{bcop}(\mathcal{A}_y, A2_y)$) with careful comparisons to avoid incorrect combinations.

Post-processing and merging. To ensure that the results obtained through the cartesian product aforementioned are correct, some post-processing is required. When combining segment lists, the latter are simply concatenated. More precisely, the resulting segment list necessarily possesses the following structure:

$(u_0, u_1)|\dots|(u_i, A)|(A, v_0)|\dots|(v_{j-1}, v_j)$, with A denoting an ABR. However, A being a separator, it is likely that the best IGP path from u_i to v_0 natively goes through A without the need of an intermediary segment. Thus, segments of the form $(u_i, A)|(A, v_0)$ can often be replaced by a single segment (u_i, v_0). Such anomalies should be corrected, as an additional useless segment may render the path falsely unfeasible, even though it actually fits the MSD constraint. This correction can be performed easily. Let $A1$ be the separator, if $(u_i, A1)$ and $(A1, v_0)$ are node segments, and all best IGP paths from u_i to v_0 go through $A1$ (or possess the same cost and delay as the best IGP ones going through $A2$), the two node segments can be replaced by a single one.

This correction is performed quickly and relies solely on information available to the router (the local SR graph and the received distances summary). Finally, after having performed and corrected the cartesian products for all the ABRs of the area, the latter are merged in a single Pareto front.

Once performed for all areas, an ABR A_x now possesses all 2COP paths to all considered destinations within the network. These can then be sent to routers within $\mathcal{A}_x$, who will need to perform similar computations to compute non-dominated paths to all routers within a different area. Note that the 2COP paths for each destination can be sent as things progress, so that routers can process such paths progressively (and in parallel) if needed.

Summary. By running `BEST2COP` within each area, before exchanging and combining the results, one can find all non-dominated paths to each destination within a network of 100 000 nodes in less than 900ms. The induced message complexity is manageable in practice and can be further tuned if required. Our method can be adapted for controller-oriented deployments.

3.5. A Limited Complexity with Strong Guarantees

3.5.1. An Efficient Polynomial-Time Algorithm

The flat `BEST2COP`. In the worst-case, for a given node v, there are up to $degree(v) \times \Gamma$ paths that can be extended towards it. Observe that $degree(v)$ is at least $|V|$ (because G' is complete) and depends on how many parallel links v has with its neighbors. With L being the average number of links between two nodes in G', on average we thus have $degree(v) = |V| \times L \times \Gamma$ paths to extend to a given node, at worst. These extensions are performed for each node v and up to MSD times, leading to a complexity of

$$O(c_0 \cdot \Gamma \cdot |V|^2 \cdot L)$$

Using up to $|V|$ threads, one can greatly decrease the associated computation time. Note that, while the algorithm is easily parallelizable with regular loads between cores, the exhibited speedup ultimately depends on the underlying hardware characteristics and the difficulty of the problem instance. As such, while we will see that the speedup observed is significant, it is unlikely that the theoretical (quasi-linear) speedup can be reached in practice.

The Cartesian Product. Its complexity is simply the size of the 2COP solution space squared, for each destination, thus at worst $O((c_0 \cdot \Gamma)^2 \cdot |V|)$. Note that we can reach a complexity of $O(c_0^2 \cdot \Gamma^2)$, again with the use of $|V|$ threads since each product is independent. This worst case is not expected in practice as metrics are usually mostly aligned to result in Pareto fronts whose maximal size is much smaller than $c_0 \cdot \Gamma$.

Overall, `BEST2COP-E` (multi-area) exhibits a complexity of

$$O\left(c_0 \cdot \Gamma \cdot \left(c_0 \cdot \Gamma + L \cdot max_{\forall i \in [1..m]}(|V_i|)\right)\right)$$

with V_i denoting the set of nodes in each area i (m being their number) and the use of $|V|$ threads and sufficient CPU resources (this bound is achievable ideally because the load is perfectly balanced and bottlenecks negligible). Note that the cartesian product dominates this worst-case analysis as long as the product $V_i \cdot L$ remains small enough. However, with realistic weighted networks, we argue that the contribution of the Cartesian product is negligible in practice, so `BEST2COP-E` is very scalable for real networking cases.

3.5.2. What are the Guarantees one can expect when the Trueness exceeds the Accuracy, i.e., *if $t > \gamma$?*

If propagation delays are measured with a really high trueness (*e.g.,* with a delay grain of 1 μs or less), `BEST2COP` (and so, `BEST2COP-E`) can either remain exact but slower, or, on the contrary, rapidly produce approximated results. In practice, if one prefers to favor performance by choosing a fixed discretization of the propagation delay (to keep the computing time reasonable rather than returning truly exact solutions), this may result in an array not accurate enough to store all non dominated delay values, *i.e.,* two solutions might end up in the same cell of such an array even though they are truly distinguishable. Nevertheless, we can still bound the margin errors, relatively or in absolute, regarding constraints or the optimization objective of the 2COP variant one aims to solve.

In theory, note that while no exact solutions remain tractable if the trueness of measured delays is arbitrarily high (for worst-case DCLC instances), it is possible to set these error margins to extremely small values with enough CPU power. If $t < \gamma$, each iteration of our algorithm introduces an absolute error of at most $\frac{1}{\gamma}$ for the M_1 metric, *i.e.,* the size of one cell in our array (recall that $\gamma = \frac{\Gamma}{c_1}$ is the accuracy level and is the inverse of the delay grain of the static array used by `BEST2COP`). So our algorithm may miss an optimal constrained solution p_d^* (for a destination d) only if there exists another solution p_d such that

$d_1(p_d) \geq d_1(p_d^*)$ but the M_1 distance of both solutions associated to the same integer (that is stored in the same cell of the `dist` array) *i.e.,* only if $d_1(p_d) \leq d_1(p_d^*) + \frac{c_0}{\gamma}$. In this case, we have $d_2(p_d) \leq d_2(p_d^*)$ because otherwise, p_d^* would have been stored instead of p_d. From this observation, depending on the minimized metric, `BEST2COP` ensures the following guarantees.

If one aims to minimize M_0 or M_2 (*e.g.,* when solving DCLC), then `BEST2COP` guarantees a solution p_d that optimizes the given metric, but this solution might not satisfy the given delay constraint $c \leq c_1$. As an example, for DCLC-SR (optimizing M_2), we have

$$d_0(p_d) \leq c_0$$
$$d_1(p_d) < c + \frac{c_0}{\gamma}$$
$$d_2(p_d) \leq d_2(p_d^*)$$

With p_d^* denoting the optimal constrained solution. When minimizing M_1, the solution returned by `BEST2COP` for a given destination d, p_d, will indeed verify the constraints on M_0 and M_2, and we have $d_1(p_d) < d_1(p_d^*) + \frac{c_0}{\gamma}$. The induced absolute error of c_0/γ regarding the delay of paths becomes negligible as the delay constraint increases. If $c \approx c_1$, the latter translates to a small relative error of c_0/Γ. Conversely, it becomes significant if $c << c_1$. When minimizing M_0 or M_2, it is thus recommended to set c_1 as low as possible regarding the relevant sub-constraint(s) $c \leq c_1$ if necessary. Similarly, to guarantee a limited relative error when minimizing M_1, it is worth running our algorithm with a small c_1 as we can have $d_1(p_d^*) << c_1$. However, note that this later and specific objective (in practice less interesting than DCLC in particular) requires some a priori knowledge, either considering the best delay path without any c_2 and c_0 constraints, or running twice `BEST2COP` to get $d_1(p_d)$ as a first approximation to avoid set up c_1 blindly initially (here c_1 is not a real constraint, only c_2 and c_0 apply as bounds of the problem, c_1 just represents the absolute size of our array and, as such, the accuracy one can achieve).

Even though `BEST2COP` exhibits strong and tunable guarantees, it may not return exact solutions once two paths end up in the same delay cell, which may happen even with simple instances exhibiting a limited Pareto front. Fortunately, a slight tweak in the implementation is sufficient to ensure exact solutions for such instances. Keeping the original accuracy of M_1 distances, one can rely on truncated delays only to find the cell of each distance. Then, one possible option consists of storing up to k distinct distances in each cell[13]. Thus, some cells would form a miniature, undiscretized Pareto front of size k when required. This trivial modification allows the complexity to remain bounded and predictable: as long as there exists less than k distances within a cell, the returned solution is exact. Otherwise, the algorithm still enforces the aforementioned guarantees. While this modification increases the number of paths we have to extend to $k \cdot \Gamma$ at worst, such cases are very unlikely to occur in average. Notably, our experiments show that 3D Pareto fronts for each destination contain usually less than ≈ 10 elements at most on realistic topologies, meaning that a small k would be sufficient in practice. In summary, `BEST2COP` is efficient and exact to deal with simple instances and/or when $t \geq \gamma$, while it provides approximated but bounded solutions for difficult instances if $t < \gamma$ to remain efficient and so scalable even with massive scale IP networks.

[13] In practice, note that several implementation variants are possible whose one consists of using the array only when the stored Pareto front exceeds a certain threshold. Moreover, k can be set up at a global scale shared for all cells or even all destinations, instead of a static value per cell, to support heterogeneous cases more dynamically. These approaches were also evoked in [66]

4. Performance Evaluation

In this section, we evaluate the computation time of our solution. We start by evaluating `BEST2COP` on various flat network instances, ranging from worst-case scenario to real topologies, and compare it to another existing approach based on the *Dijkstra* algorithm, `SAMCRA` [68]. Then, after having introduced our multi-area topology generator, we evaluate the extended variant of our solution, `BEST2COP-E`, on massive scale networks. In the following, we consider our discretization to be exact (*i.e.,* Γ is high enough to prevent loss of relevant information).

To conduct our evaluations, we consider that:

- $c_0 = MSD = 10$, as it is close to the best hardware limit;
- $L = 2$: while some pairs of nodes may have more than two parallel links connecting them in G, we argue that, on average in G', one can expect that the total number of links in E' is lower than $2|V|^2$.
- $\Gamma = 1000$, although this value is tunable to reflect the expected product trueness-constraint on M_1, we consider here a fixed delay grain of 0.1ms (so an accuracy level of $\gamma = 10$) regarding a maximal constraint $c_1 =$ 100ms. This Γ limitation is realistic in practice and guarantees the efficiency of `BEST2COP` even for large complex networks as it becomes negligible considering large $|V|$.

Note that the delays fed to `SAMCRA` are *also* discretized in the same fashion as for `BEST2COP`, allowing the number of non-dominated paths that `SAMCRA` has to consider to be bounded and reduced. In addition, as `SAMCRA` is not designed with the SR Graph in mind, it is difficult to know which of the two methods mentioned in Section 3.2.1 is the most suited to consider the segment metric. Thus, we compare ourselves to both variants. First, we run `SAMCRA` on the fully-meshed SR Graph, which allows to

use the SAMCRA algorithm nearly as-is. We call this variant SAMCRA+SRG. Second, we implement our conversion algorithm, which allows to efficiently convert multi-metric paths to segment lists. This method requires however further modification of the SAMCRA algorithm, not only by adding the conversion algorithm but also by extending its dominancy checks (details can be found in Appendix Appendix B). We refer to this variant as SAMCRA+LCA.

Finally, note that our implementation of SAMCRA (both SAMCRA+SRG and SAMCRA+LCA) is purely sequential. While it may be possible to parallelize some inner loops (or the outer one by adapting methods used to parallelize the Dijkstra algorithm [14]), doing so raises several challenges to verify the correctness and actual efficiency of the resulting algorithm in our context. As these issues were, to the best of our knowledge, not discussed for the SAMCRA algorithm, we prefer to remain close to its original design and leave such challenges for future works.

All our experiments are performed on an Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz × 8.

4.1. Computing Time & Comparisons for Flat Networks

This section illustrates the performance of our algorithm BEST2COP using three flat network scenarios. In particular, we do not take advantage of any area decomposition to mitigate the computing time.

As shown in [50] by forcing BEST2COP to explore its full iteration space, our algorithm *cannot* exceed 80s at worst on topologies of 1000 nodes. This upper bound can however be drastically reduced through the use of multi-threading, reaching a worst-case of $\approx 10s$ when relying on 8 threads, highlighting the parallel nature of our algorithm [14]. Additional information can be found in Appendix Appendix C for the interested reader.

We will see that in practice, BEST2COP is far from reaching these upper bounds, even on random networks. In the following, we will evaluate BEST2COP and compare it to SAMCRA in two main scenarios: a real network with real link valuations, and random networks of up to 10 000 nodes.

Real network. We start by considering a real IP network topology. We use our largest available ISP topology, consisting of more than 1100 nodes and 4000 edges. This topology describes the network of a Tier-1 operator and is not available to the public [15]. While the IGP costs of each link were available, we do not have their respective delays. We thus infer delays thanks to the available geographical locations we do possess: we set the propagation delays as the orthodromic distances between the connected nodes divided by the speed of light, and run both algorithms on the obtained topology. The execution times are then shown in Fig. 5. BEST2COP (1, 2, 8 threads) and SAMCRA (with LCA and SRG) are run for every node as source, resulting in the distributions showcased.

One can see that SAMCRA+SRG (*i.e.,* SAMCRA run directly on the SR Graph) exhibits the worst execution times out of all the algorithms and variants presented, averaging at 100ms, and reaching 250ms at worst. Interestingly, this shows that exploring the SR Graph itself may be detrimental to some algorithms (in particular priority-queue-based ones) due to its high density. Hence, algorithms not designed to take advantage of its features may fare better by exploring the original, sparser topology, and using the information within the SR Graph to compute the number of necessary segments to encode the paths being explored. This is visible on SAMCRA+LCA computation times. Our construct, coupled with our conversion algorithm, allowed SAMCRA+LCA to reach computation times very similar to the mono-threaded variant of BEST2COP, with an average execution time of ≈ 60ms. Note that BEST2COP, which runs on the SR Graph itself, shows equivalent execution time when relying on a single thread. However, when relying on multiple threads, BEST2COP outperforms its competitor in all runs, reaching a computation time of ≈ 25ms at worst when using 8 threads, *i.e.,* three times faster than its competitor.

These low execution times are not only due to the efficiency of the algorithms presented, but also to the realistic link valuations, which tend to be correlated in practice. In realistic cases, BEST2COP can thus work with $\Gamma > 1000$ and so with a supported accuracy $t >> 0.1$ms (to deal with a micro-second grain) for small enough delay constraint (*i.e.,* , << 100ms), while keeping the execution time in the hundreds of milliseconds. One may notice that (almost) perfectly aligned metrics reduce the usefulness of any DCLC-like algorithm, but such metrics are not always aligned for all couples in practice (even with realistic cases, we observe that the average size of the 3D Pareto front is strictly greater than 1, typically ≈ 4). Our algorithm deals efficiently with easy cases and remains exact[16] and efficient for more complex cases, *e.g.,* with random graphs.

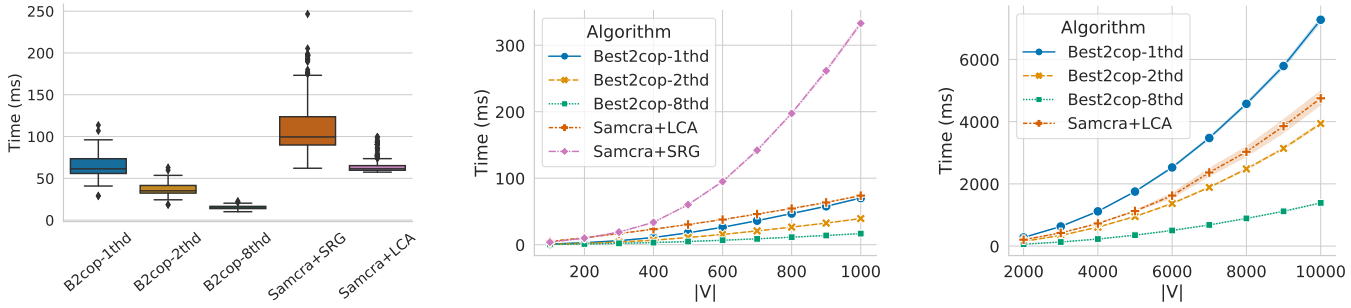
Random networks. The number of publicly available large topologies being limited, we continue our evaluation with random scenarios to assess the computation time of the aforementioned algorithms on a larger number of scenarios.

We generate raw connected graphs of $|V|$ nodes by using the Erdos-Rényi model. The generated topologies have a degree of $log(|V|)$. Both the delays and the IGP weights are picked uniformly at random. IGP weights are chosen within the interval $[1, 2^{32}/|V|/10]$, to ensure that no paths possesses a cost higher than 2^{32}. Delays are chosen within the interval $[0, 0.01 \times c_1]$, with $c_1 = 100ms$, to ensure that a high number of feasible paths exist.

[14] Additional experiments on a high-performance grid showed that BEST2COP may reach a speed-up of 23 when running on 30 cores.

[15] While public topology datasets exist, these topologies are often too small for our use-case and/or do not possess any link valuation.

[16] Or at least near exact for difficult instances having both high trueness and exponential increasing Pareto fronts.



(a) Computation time of BEST2COP (1, 2, 8 threads) & SAMCRA on a real network with real link valuation.

(b) Computation time of BEST2COP & SAMCRA on random networks with random valuation for $|V| = 100$ to 1000

(c) Computation time of BEST2COP & SAMCRA on random networks with random valuation for $|V| = 1000$ to 10 000

Figure 5: Computation time of BEST2COP and SAMCRA on various experiments. Although the results can be close when considering mono-threaded BEST2COP and SAMCRA, our algorithm always outperforms its competitor when using multi-threading. In some cases, multi-threading is not even necessary

We start by running BEST2COP and SAMCRA for $|V|$ ranging from 100 to 1000 (with steps of 100). To account for the randomness of both valuation functions, we generate 30 differently weighted distinct topologies for each value of $|V|$. We run BEST2COP and SAMCRA for 30 nodes selected as representative sources (randomly picked uniformly). Computing times are shown in Fig 5.

While the computation times are slightly higher (due to the random valuations which lead to a higher number of non-dominated paths), the results are similar to the previous experiment. These results display more clearly that SAMCRA does not benefit from exploring the SR Graph. Indeed, on random networks, SAMCRA+SRG is about 7 times slower than the other algorithms displayed. However, as on real networks, SAMCRA+LCA shows results close (if not equal) to BEST2COP execution time. Nevertheless, even on random networks, BEST2COP remains three times faster than its competitor when relying on 8 threads.

Interestingly, BEST2COP mono-threaded and SAMCRA+LCA computation times get closer as $|V|$ increases. Thus, we continue our comparison on networks of 2000 to 10 000 nodes. Given the long computation times of SAMCRA+SRG, we here only consider SAMCRA+LCA. The results are shown in Fig. 5. On such networks, BEST2COP (mono-threaded) exhibits an execution time of $7s$, while SAMCRA+LCA remains under $5s$. The quadratic complexity of BEST2COP (whose main factor is $|V|^2$) is here clearly visible. SAMCRA+LCA exhibits a less steep growth. However, when relying on multiple-thread, BEST2COP remains far more efficient. While two threads already allow to reach an execution time slightly lower than SAMCRA (4s), 8 threads allow BEST2COP to remain $\approx$ 3.3 times faster than its competitor.

Summary. The way to use the SR Graph has a high impact on the underlying algorithm. As the SR Graph is not at the core of SAMCRA's design, exploring the latter lead to high execution time due to its density. However, adding our conversion algorithm (which relies on the SR Graph data) within SAMCRA allowed the latter to reach competitive execution times while solving 2COP. BEST2COP, which explores the SR Graph directly, exhibits an execution similar to SAMCRA+LCA when relying on a single thread. When using multi-threading, BEST2COP clearly outperforms its competitor in all scenarios.

In any case, it is interesting to note that even BEST2COP takes more than one second on networks of 10 000 nodes. To showcase the performance of our contribution on massive-scale networks, we now evaluate the execution time of its extension, BEST2COP-E, which supports and leverages OSPF-like area division. This version is adapted to tackle TE problems in massive-scale hierarchical networks. In the following, we only consider our approach. Indeed, the latter showcased better performance that SAMCRA even without relying on multiple threads when considering a topology size $|V| < 1000$, which encompasses the size of standard OSPF areas.

However, before continuing analyzing the computing time results, we first introduce our generator for massive-scale, multi-areas, realistic network having two valuations (IGP cost & delays).

4.2. Massive Scale Topology Generation

To the best of our knowledge, although such networks exist in the wild, there are no massive scale topologies made *publicly* available which exhibit IGP costs, delays, and area subdivision. For example, the graphs available in the topology zoo (or sndlib) datasets do not exceed 700 nodes in general. Moreover, the ones for which the two metrics can be extracted, or at least inferred, are limited to less than 100 nodes. Thus, at first glance, performing a practical massive-scale performance evaluation of BEST2COP-E is highly challenging if not impossible. There exist a few topology generators [61, 54] able to generate networks of arbitrary size with realistic

networking patterns, but specific requirements must be met to generate topologies onto which `BEST2COP-E` can be evaluated, in particular the need for two metrics and the area decomposition.

Topology generation requirements. First, the experimental topologies must be large, typically between 10 000 and 100 000 nodes. Second, they must possess two valuation functions as realistic as possible, one for the IGP cost and the other modeling the delay. Third, since the specific patterns exhibited by real networks impact the complexity of TE-related problems, the generated topologies must possess realistic structures (*e.g.,* with respect to redundancy in the face of failures in particular). Finally, for our purposes, the topology must be composed of different areas centered around a core backbone, typically with two ABRs between each to avoid single points of failure. Since we do not know any generator addressing such requirements, we developed YARGG (Yet Another Realistic Graph Generator), a python tool(Code available online [17]) which allows one to evaluate its algorithm on massive-scale realistic IP networks. In the following, we describe the generation methods used to enforce the required characteristics.

High-level structure. One of the popular ISP structure is the three-layers architecture [11], illustrated in Fig. 7. The *access layer* provides end-users access to the communication service. Traffic is then aggregated in the *aggregation layer*. Aggregation routers are connected to the *core routers* forming the last layer. The aggregation and access layers form an area, and usually cover a specific geographical location. The core routers, the ABRs connecting the backbone other areas, and their links, form the backbone area that interconnect the stub areas, *i.e.,* the aggregation and access layers of the different geographical locations. Core routers are ABRs and belong both to a sub-area, per couple of 2 for redundancy. Thus, while the access and aggregation layers usually follow standard structures and weight systems recommended by different network vendors, the backbone can vastly differ among different operators, depending on geographical constraints, population distribution, and pre-existing infrastructure. Taking these factors into account, YARGG generates large networks by following this 3-layer model, given a specific geographical location (*e.g.,* a given country).

Generating the core network and the areas. YARGG is a heuristic that generates the core network by taking the aforementioned considerations into account: existing infrastructures, population, and geographical constraints. An example of a core network as generated by YARGG may be seen in Fig. 6. Given a geographical location (e.g., a country or a continent), YARGG builds the structure of the core network by

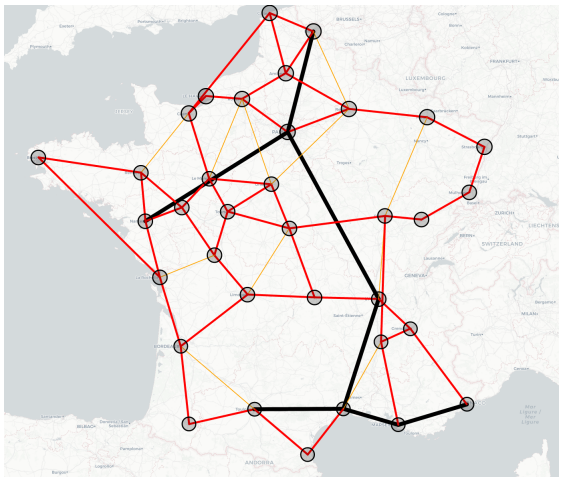


Figure 6: Core network (before step 5) generated by YARGG in France. While we consider the road distances, we represent the links in an abstract fashion for readability purposes. The color and width of the links represent their bandwidth (and thus their IGP costs).

1. *Extracting the x most populated cities in the area.* Close cities are merged in a single entity. The merge trigger value may change (the exact values used here can be found in [49]).
2. *Constructing a minimum Spanning-Tree covering all cities of the area*, using road distances as the metric. Links between cities totaling more than 30% of the total population are normalized in order to be highly prioritized.[18]
3. *Removing articulation-points.* YARGG picks one biconnected component, and adds the smallest link (in terms of road distances) that bridges this component with another. This process is repeated until no articulation point remains (*i.e.,* the topology is biconnected).
4. *Adding links increasing the connectivity and resilience for a limited cost.* YARGG considers all links meeting certain criterias. The two cities/nodes must be closer than 20% of the largest road distance. Their current degree must be lower than 4. The link, if added, should reduce the distance (and so, the delay) between the nodes by at least 25%. Among these links, YARGG adds the one with the highest attractiveness, expressed as the sum of the distance reduction and the population of the cities (normalized).
5. *Doubling the obtained topology.* The topology is doubled. There are now two nodes (/routers) per city. Links are added between the two routers of the same city, making the topology tri-connected.

[17] `https://github.com/JroLuttringer/YARGG`

[18] The parameters tuned may be easily modifier. For now, the latter are purely empirical.

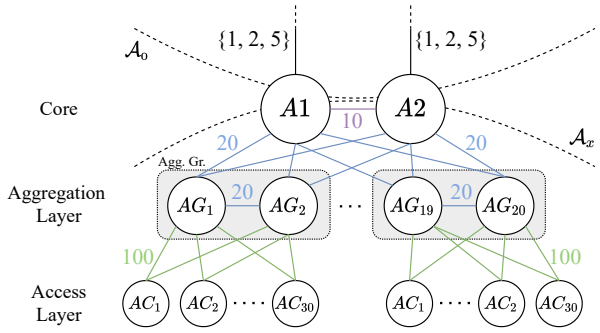


Figure 7: Weights and structures of an area generated by YARGG.

The couple of routers located at each city within this generated backbone area become the ABRs between the backbone and their area, which is generated next.

Access & aggregation layers. These last two layers make up a non-backbone area and span a reduced geographical area. Thus, one access and one aggregation layer are located in each city considered by YARGG. Several network equipment vendors recommend a hierarchical topology, such as the three-layer hierarchical model [12]. An illustration can be seen in Fig. 7. Simply put, there should be two core routers (the ABRs) at the given location (a city in YARGG's case). Each core router is connected to all aggregation routers. For better resiliency, the aggregation layer is divided into aggregation groups, composed of two connected routers. Finally, routers within an aggregation group are connected to access-layer routers. To achieve areas of ≈ 300 nodes, we consider 30 access routers per aggregation group. This results in a large, dense, and realistic graph.

Weights. In the backbone, the weights generated by YARGG are straightforward. The delays are extracted from the road distances between the cities, divided by 60% the speed of light (close to the best performing fiber optic). The IGP cost is 1 for links between large cities since these links usually have a high bandwidth (in black in Fig. 6), 2 for standard links, necessary to construct a tri-connected graph (added at step 3, in red in Fig. 6), and 5 for links that are not mandatory, but that increase the overall connectivity (added at step 4, in orange in Fig. 6).

Within an area, the IGP costs follow a set of realistic constraints, according to two main principles: (i) access routers should not be used to route traffic (except for the networks they serve), (ii) links between routers of the same hierarchical level (*e.g.,* between the two core routers or the two aggregation routers of a given aggregation group) should not be used, unless necessary (*e.g.,* multiple links or node failures). These simple principles lead to the IGP costs exhibited in Fig. 7. The delays are then chosen uniformly at random. Since access routers and aggregation routers are close geographically, the delay of their links is chosen between 0.1 and 0.3ms. The delay between aggregation routers and core routers is chosen between 0.3ms and the lowest backbone link delay. Thus, links within an area necessarily possess a lesser delay than core links.

Summary. YARGG computes a large, realistic, and multi-area topology. The backbone spans a given geographical location and possesses simple IGP weights and realistic delays. Other areas follow a standard three-layer hierarchical model. Weights within a stub area are chosen according to a realistic set of usual ISP constraints. Delays, while chosen at random within such areas, remain consistent with what should be observed in practice.

4.3. Computing Time for Massive Scale Multi-Areas Networks

Using YARGG, we generate five massive scale, continent-wide topologies, and run `BEST2COP` on each one of them. The topologies ranges from 10 000 to 100 000 nodes. Each non-backbone area possess around 320 nodes. The topologies, their geographical representations and some of the associated network characteristics can be found online [49].

We run `BEST2COP` on each ABR as a source (around $|V|/320 \times 2$ sources). The time corresponding to the message exchange of the computed Pareto front (step 2 of `BEST2COP-E`) is not taken into consideration. Thus, the computation time showcased is the sum of the average time taken by ABRs to perform the preliminary intra-area `BEST2COP` (and the distances to segment lists conversions) plus the time taken to perform the $|V|/320 \times 2 - 2$ Cartesian products (for all other ABRs of all other areas).

Note that we consider an ABR as a source and not an intra-area destination. In practice, the ABR would send the computed distances to the intra-area nodes, who in turn would have to perform a Cartesian product of these distances with its own distances to said ABR. However, both the ABR and the intra-area node have to consider the same number of destinations ($|V|$), and the results computed by the ABR can be sent as they are generated (destination per destination), allowing both the ABR and the intra-area nodes to perform their Cartesian product at the same time. In addition, intra-area nodes may benefit from several optimizations regarding their Cartesian product, if the constraints of the desired paths are known (these optimizations will not be used nor detailed in this paper). For these reasons, we argue that the time measured here, using an ABR as a source, is representative of the total actual time required, *i.e.,* the overall worst time for the last treated destination at each source.

The results of this experiment are shown in the violin plot of Fig. 8. By leveraging the network structure, `BEST2COP-E` exhibits very good performance despite the scale of the graph. For 10 000 nodes, `BEST2COP-E` exhibits a time similar to the one taken by its flat variant for $|V| = 2000$. Furthermore, `BEST2COP-E` seems to scale

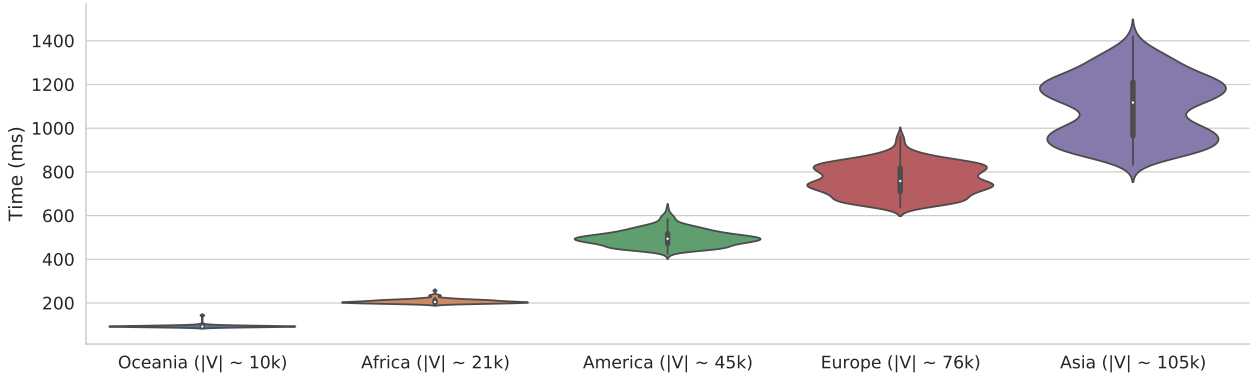


Figure 8: BEST2COP-E computation time on 5 continent-wide topologies generated by YARGG.

linearly with the number of nodes, remaining always under one second for $|V| \approx 75\,000$. Even once the network reaches a size of $\approx 100\,000$, BEST2COP-E is able to solve 2COP in less than one second for a non-negligible fraction of the sources, and never exceeds 1.5s.

Note that the times showcased here rely on a single thread. While BEST2COP-E's Cartesian product can be parallelized locally (both at the area and the destination scale), this parallelization hardly has any effect. This is explained by the fact that these individual computations are in fact fairly efficient, hence the overhead induced by the creation and management of threads is heavier than their workload. In addition, since BEST2COP deals with very large topologies, some complex memory-related effects might be at play. Indeed, we notice these results to surprisingly vary depending on the underlying system, operating system, and architecture due to the differences in terms of memory management.

Thus, while massive scale deployments seem to a priori prevent the usage of fine-grained TE, their structures can be leveraged, making complex TE problems solvable in less than one second even for networks reaching 100 000 nodes. The computations performed for each area can also be distributed among different containers within the cloud, if handled by a controller.

5. Conclusion

While the overhead of MPLS-based solutions lead to a TE winter in the past decade, Segment Routing marked its rebirth. In particular, SR enables the deployment of a practical solution to the well-known DCLC problem. Our algorithm, BEST2COP [50] (Best Exact Segment Track for 2-Constrained Optimal Paths), iterates on the SR Graph to natively solve DCLC in SR domains with strong guarantees, through simple and efficient data structures and concepts.

In this paper, we went several steps further with the following achievements:

- experimentally demonstrating that SR is a relevant technology to deploy DCLC paths;
- for massive scale ISPs relying on area-subdivision, we extend BEST2COP to BEST2COP-E, partitioning 2COP into smaller sub-problems, to further reduce its overall complexity (time, memory and churn);
- through extensive evaluations, relying on multi-threading and our own multi-metric/multi-areas network generator, we have shown that BEST2COP-E is very efficient in practice. This was confirmed through a comparison with a relevant state-of-the-art algorithm, which benefited from a novel path to segment multi-metric conversion algorithm that we designed.

To the best of our knowledge, BEST2COP is the first practically exact and efficient solution for 2COP within SR domains, making it the most practical candidate to be deployed for such a TE flavor in today ISPs. It is able to solve 2COP on massive scale realistic networks having 100 000 nodes in less than a second. For large areas having thousands of routing devices, we have shown that BEST2COP(-E) can easily deal with random topologies while its competitors do not scale.

More advanced and flexible structures can be envisioned to deal with high trueness requirements, while deploying novel flex-algo strategies can help to mitigate the rare SR limit drawbacks. Finally, exploring the efficiency of parallel Dijkstra-based solutions and associated trade-offs in a multi-metric context is also an interesting perspective.

Acknowledgment

This work was partially supported by the French National Research Agency (ANR) project Nano-Net under contract ANR-18-CE25-0003.

References

[1] Adams, H., . Segment routing in the 5g era. https://www.juniper.net/assets/de/de/local/pdf/articles/3200083-en.pdf. (Accessed on 06/21/2021).

[2] Almes, G., Kalidindi, S., Zekauskas, M., 1999. A Round-trip Delay Metric for IPPM. RFC 2681. RFC Editor.

[3] Almes, G., Kalidindi, S., Zekauskas, M., Morton, A., 2016. A One-Way Delay Metric for IP Performance Metrics (IPPM). STD 81. RFC Editor.

[4] Aneja, Y.P., Nair, K.P.K., 1978. The constrained shortest path problem. Naval Research Logistics Quarterly 25, 549–555. doi:10.1002/nav.3800250314.

[5] Aubry, F., . Models and algorithms for network optimization with segment routing , 175.

[6] Aubry, F., Lebrun, D., Vissicchio, S., Khong, M.T., Deville, Y., Bonaventure, O., 2016. Scmon: Leveraging segment routing to improve network monitoring, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9. doi:10.1109/INFOCOM.2016.7524410.

[7] Bhatia, R., Hao, F., Kodialam, M., Lakshman, T.V., 2015. Optimized network traffic engineering using segment routing, in: 2015 IEEE Conference on Computer Communications (INFOCOM), pp. 657–665.

[8] Brumbaugh-Smith, J., Shier, D., 1989. An empirical investigation of some bicriterion shortest path algorithms. European Journal of Operational Research 43, 216–224. doi:10.1016/0377-2217(89)90215-4.

[9] Brundiers, A., Schüller, T., Aschenbruck, N., 2021. On the benefits of loops for segment routing traffic engineering, in: 2021 IEEE 46th Conference on Local Computer Networks (LCN), pp. 32–40. doi:10.1109/LCN52139.2021.9524958.

[10] Carpa, R., Glück, O., Lefevre, L., 2014. Segment routing based traffic engineering for energy efficient backbone networks, in: 2014 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS), pp. 1–6. doi:10.1109/ANTS.2014.7057272.

[11] Cisco Networks, 2008. Campus network for high availability design guide. https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Campus/HA_campus_DG/hacampusdg.html. (Accessed on 07/12/2021).

[12] Cisco Networks, 2014. Hierarchical network design overview (1.1) > cisco networking academy connecting networks companion guide: Hierarchical network design | cisco press. https://www.ciscopress.com/articles/article.asp?p=2202410&seqNum=4. (Accessed on 09/08/2021).

[13] Cisco Networks, 2020. Cisco content hub - performance measurement for traffic engineering. https://content.cisco.com/chapter.sjs?uri=/searchable/chapter/content/en/us/td/docs/ios-xml/ios/seg_routing/configuration/xe-17/segrt-xe-17-book/performance_measure_TE.html.xml. (Accessed on 06/21/2021).

[14] Crauser, A., Mehlhorn, K., Meyer, U., Sanders, P., 1998. A parallelization of dijkstra's shortest path algorithm, in: MFCS.

[15] De Neve, H., Van Mieghem, P., 2000. Tamcra: A tunable accuracy multiple constraints routing algorithm. Comput. Commun. 23, 667–679. URL: https://doi.org/10.1016/S0140-3664(99)00225-X, doi:10.1016/S0140-3664(99)00225-X.

[16] Ergun, F., Sinha, R., Zhang, L., 2002. An improved fptas for restricted shortest path. Information Processing Letters 83, 287–291. doi:10.1016/S0020-0190(02)00205-3.

[17] Feng, G., Makki, K., Pissinou, N., Douligeris, C., 2002. Heuristic and exact algorithms for qos routing with multiple constraints. IEICE Transactions on Communications 85, 2838–2850.

[18] Filsfils, C., 2019. Segment routing - cisco live barcelona 2019. https://www.segment-routing.net/conferences/2019-01-30-CLEUR-3122/. (Accessed on 08/30/2021).

[19] Filsfils, C., 2020. Network programming with srv6. https://orbi.uliege.be/bitstream/2268/245292/4/network-programming-phd-final.pdf. (Accessed on 06/28/2021).

[20] Filsfils, C., Michielsen, K., Talaulikar, K., 2017. Segment Routing Part I. Number ptie. 1 in Segment Routing, CreateSpace Independent Publishing Platform. URL: https://books.google.fr/books?id=1zMyMQAACAAJ.

[21] Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., Shakir, R., 2018. Segment Routing Architecture. RFC 8402. RFC Editor.

[22] Foerster, K., Parham, M., Chiesa, M., Schmid, S., 2018. Ti-mfa: Keep calm and reroute segments fast, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 415–420.

[23] Garey, M.R., Johnson, D.S., 1990. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA. doi:10.5555/574848.

[24] Garroppo, R.G., Giordano, S., Tavanti, L., 2010. A survey on multi-constrained optimal path computation: Exact and approximate algorithms. Computer Networks 54, 3081–3107. doi:10.1016/j.comnet.2010.05.017.

[25] Gay, S., Hartert, R., Vissicchio, S., 2017. Expect the unexpected: Sub-second optimization for segment routing. IEEE INFOCOM 2017 - IEEE Conference on Computer Communications , 1–9.

[26] Giacalone, S., Ward, D., Drake, J., Atlas, A., Previdi, S., 2015. OSPF Traffic Engineering (TE) Metric Extensions. RFC 7471. RFC Editor.

[27] Ginsberg, L., Previdi, S., Giacalone, S., Ward, D., Drake, J., Wu, Q., 2019. IS-IS Traffic Engineering (TE) Metric Extensions. RFC 8570. RFC Editor.

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

[29] Goel, A., Ramakrishnan, K., Kataria, D., Logothetis, D., 2001. Efficient computation of delay-sensitive routes from one source to all destinations, in: Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213), pp. 854–858 vol.2. doi:10.1109/INFCOM.2001.916276.

[30] Guck, J.W., Van Bemten, A., Reisslein, M., Kellerer, W., 2018. Unicast qos routing algorithms for sdn: A comprehensive survey and performance evaluation. IEEE Communications Surveys & Tutorials 20, 388–415. doi:10.1109/COMST.2017.2749760.

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

[32] Guo, L., Matta, I., 1999. Search space reduction in qos routing, in: Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No.99CB37003), p. 142–149. doi:10.1109/ICDCS.1999.776515.

[33] Hanusse, N., Ilcinkas, D., Lentz, A., 2020. Framing algorithms for approximate multicriteria shortest paths, in: 20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020), pp. 11:1–11:19. URL: https://hal.archives-ouvertes.fr/hal-03034585, doi:10.4230/OASIcs.ATMOS.2020.11.

[34] Hao, F., Kodialam, M., Lakshman, T.V., 2016. Optimizing restoration with segment routing, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9.

[35] Hartert, R., Schaus, P., Vissicchio, S., Bonaventure, O., 2015. Solving segment routing problems with hybrid constraint programming techniques, in: Pesant, G. (Ed.), Principles and Practice of Constraint Programming, Springer International Publishing, Cham. pp. 592–608.

[36] Hassin, R., 1992. Approximation schemes for the restricted shortest path problem. Mathematics of Operations Research 17, 36–42.

[37] Hou, X., Wu, M., Zhao, M., 2018. An optimization routing algorithm based on segment routing in software-defined networks. Sensors 19, 49. doi:10.3390/s19010049.

[38] Jaffe, J.M., 1984. Algorithms for finding paths with multiple constraints. Networks 14, 95–116. doi:10.1002/net.3230140109.

[39] Jia, Z., Varaiya, P., 2006. Heuristic methods for delay constrained least cost routing using k-shortest-paths. IEEE Transactions on Automatic Control 51, 707–712. doi:10.1109/TAC.2006.872827.

[40] Juttner, A., Szviatovski, B., Mecs, I., Rajko, Z., 2001. Lagrange relaxation based method for the qos routing problem, in: Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213), pp. 859–868 vol.2.

[41] Korkmaz, T., Krunz, M., 2001. Multi-constrained optimal path selection, in: Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213), p. 834–843 vol.2. doi:10.1109/INFCOM.2001.916274.

[42] Kuipers, F., Van Mieghem, P., Korkmaz, T., Krunz, M., 2002. An overview of constraint-based path selection algorithms for qos routing. IEEE Communications Magazine 40, 50–55. doi:10.1109/MCOM.2002.1106159.

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

[44] Lee, W.C., Hluchyi, M.G., Humblet, P.A., 1995. Routing subject to quality of service constraints in integrated communication networks. IEEE Network 9, 46–55.

[45] Liu, G., Ramakrishnan, K., 2001. A*prune: an algorithm for finding k shortest paths subject to multiple constraints, in: Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213), pp. 743–749 vol.2. doi:10.1109/INFCOM.2001.916263.

[46] Liu, W., Lou, W., Fang, Y., 2005. An efficient quality of service routing algorithm for delay-sensitive applications. Computer Networks 47, 87–104. doi:10.1016/S1389-1286(04)00213-0.

[47] Lorenz, D.H., Raz, D., 2001. A simple efficient approximation scheme for the restricted shortest path problem. Operations Research Letters 28, 213–219. doi:10.1016/S0167-6377(01)00069-4.

[48] Loreti, P., Mayer, A., Lungaroni, P., Lombardo, F., Scarpitta, C., Sidoretti, G., Bracciale, L., Ferrari, M., Salsano, S., Abdelsalam, A., et al., 2021. Srv6-pm: A cloud-native architecture for performance monitoring of srv6 networks. IEEE Transactions on Network and Service Management 18, 611–626. doi:10.1109/TNSM.2021.3052603.

[49] Luttringer, J.R., Alfroy, T., Bramas, Q., Clad, F., Mérindol, P., Pelsser, C., 2021. Deploying Near Optimal Delay Constrained Paths with Segment-Routing in Massive Scale Networks. URL: https://doi.org/10.5281/zenodo.5535430, doi:10.5281/zenodo.5535430.

[50] Luttringer, J.R., Alfroy, T., Mérindol, P., Bramas, Q., Clad, F., Pelsser, C., 2020. Computing delay-constrained least-cost paths for segment routing is easier than you think, in: 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA), pp. 1–8. doi:10.1109/NCA51143.2020.9306706.

[51] Martins, E.Q.V., 1984. On a multicriteria shortest path problem. European Journal of Operational Research 16, 236–245. doi:10.1016/0377-2217(84)90077-8.

[52] Martins, E.Q.V., Santos, J.L.E.D., 1999. The labelling algorithm for the multiobjective shortest path problem .

[53] Matsushima, S., Filsfils, C., Ali, Z., Li, Z., Rajaraman, K., 2021. SRv6 Implementation and Deployment Status. Internet-Draft draft-matsushima-spring-srv6-deployment-status-11. IETF Secretariat. URL: https://www.ietf.org/archive/id/draft-matsushima-spring-srv6-deployment-status-11.txt. https://www.ietf.org/archive/id/draft-matsushima-spring-srv6-deployment-status-11.txt.

[54] Medina, A., Lakhina, A., Matta, I., Byers, J., 2001. Brite: an approach to universal topology generation, pp. 346–353. doi:10.1109/MASCOT.2001.948886.

[55] Namorado Climaco, J.C., Queirós Vieira Martins, E., 1982. A bicriterion shortest path algorithm. European Journal of Operational Research 11, 399–404. doi:10.1016/0377-2217(82)90205-3.

[56] Nordic Gateway for Research and Education, 2014. GÉant topology map. https://www.nordu.net/content/géant. (Accessed on 06/21/2021).

[57] Paixão, J., Santos, J.L., 2008. A new ranking path algorithm for the multi-objective shortest path problem. undefined URL: https://www.semanticscholar.org/paper/A-new-ranking-path-algorithm-for-the-shortest-path-Paix%C3%A3o-Santos/d113dc9676607aa8018e972220f6cbb70838146d.

[58] Papadimitriou, C., Yannakakis, M., 2000. On the approximability of trade-offs and optimal access of web sources, in: Proceedings 41st Annual Symposium on Foundations of Computer Science, p. 86–92. doi:10.1109/SFCS.2000.892068.

[59] Programme, G.F.N., . Network slicing : Use case requirements. https://www.gsma.com/futurenetworks/wp-content/uploads/2020/01/2.0_Network-Slicing-Use-Case-Requirements-1.pdf. (Accessed on 05/26/2021).

[60] Psenak, P., Hegde, S., Filsfils, C., Talaulikar, K., Gulko, A., 2020. IGP Flexible Algorithm. Internet-Draft draft-ietf-lsr-flex-algo-07. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-ietf-lsr-flex-algo-07.txt.

[61] Quoitin, B., Van den Schrieck, V., Francois, P., Bonaventure, O., 2009. Igen: Generation of router-level internet topologies through network design heuristics, in: 2009 21st International Teletraffic Congress, pp. 1–8.

[62] Raith, A., Ehrgott, M., 2009. A comparison of solution strategies for biobjective shortest path problems. Computers & Operations Research 36, 1299–1331. doi:10.1016/j.cor.2008.02.002.

[63] Reeves, D., Salama, H., 2000. A distributed algorithm for delay-constrained unicast routing. IEEE/ACM Transactions on Networking 8, 239–250. doi:10.1109/90.842145.

[64] Sahni, S., 1977. General techniques for combinatorial approximation. Operations Research 25, 920–936.

[65] Savage, S., Collins, A., Hoffman, E., Snell, J., Anderson, T., 1999. The end-to-end effects of internet path selection. SIGCOMM Comput. Commun. Rev. 29, 289–299. URL: https://doi.org/10.1145/316194.316233, doi:10.1145/316194.316233.

[66] Song, M., Sahni, S., 2006. Approximation algorithms for multiconstrained quality-of-service routing. IEEE Transactions on Computers 55, 603–617. doi:10.1109/TC.2006.67.

[67] Tsaggouris, G., Zaroliagis, C., 2009. Multiobjective optimization: Improved fptas for shortest paths and non-linear objectives with applications. Theory of Computing Systems 45, 162–186. doi:10.1007/s00224-007-9096-4.

[68] Van Mieghem, P., Kuipers, F.A., 2004. Concepts of exact qos routing algorithms. IEEE/ACM Transactions on Networking 12, 851–864.

[69] Ventre, P.L., Salsano, S., Polverini, M., Cianfrani, A., Abdelsalam, A., Filsfils, C., Camarillo, P., Clad, F., 2020. Segment Routing: a Comprehensive Survey of Research Activities, Standardization Efforts and Implementation Results. arXiv:1904.03471 [cs] ArXiv: 1904.03471.

[70] Widyono, R., Group, T., 1994. The design and evaluation of routing algorithms for real-time channels.

[71] Xin Yuan, Xingming Liu, 2001. Heuristic algorithms for multiconstrained quality of service routing, in: Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213), pp. 844–853 vol.2.

[72] Zheng Wang, Crowcroft, J., 1996. Quality-of-service routing for supporting multimedia applications. IEEE Journal on Selected Areas in Communications 14, 1228–1234.

[73] Zhou, J., 1998. A new distributed routing algorithm for supporting delay-sensitive applications, in: ICCT'98. 1998 Interna-

tional Conference on Communication Technology. Proceedings (IEEE Cat. No.98EX243), pp. 7 pp. vol.2–. doi:10.1109/ICCT.1998.741046.

Appendix A. The BEST2COP Algorithm: A detailed description

This Appendix aims at describing BEST2COP in a complete and detailed fashion. It also showcases the pseudo-code of our algorithm.

BEST2COP's main procedure is shown in Alg. 1. The variable `pfront`, the end result returned by our algorithm, contains, for each iteration, the Pareto front of the distances towards each node `n`. In other words, `pfront[i]` contains, at the end of the i^{th} iteration, all non-dominated (M_1, M_2) distances of feasible paths towards each node `n`.

The variable `dist` is used to store, for each vertex, the best M_2-distance found for each M_1-distance to each node. Since the M_1-distance of any feasible path in G' is bounded by Γ, we can store these distances in a static array `dist[v]`. Note that during iteration i, `dist` will contain the Pareto front of the current iteration (non-dominated distances of i segments) in addition to distances that may be dominated. Keeping such paths in `dist` allows us to pre-filter paths before ultimately extracting the Pareto front of the current iteration from `dist` later on. This variable is used in conjunction with `pf_cand`, a boolean array to remember which distances within `dist` were found at the current iteration.

The variable `extendable` is a simple list that contains, at iteration i, all non-dominated distances discovered at iteration $i-1$. More precisely, `extendable` is a list of tuples (u, d_list), where d_list is the list of the best-known paths towards u. The variable `nextextendable` is a temporary variable allowing to construct `extendable`.

After the initialization of the required data structures, the main loop starts. This loop is performed MSD times, or until no feasible paths are left to extend. For each node v, we extend the non-dominated distances found during the previous iteration towards v (originally (0,0) towards `src`). Extending paths in this fashion allows to easily parallelize the main `for` loop (*e.g.,* through a single *pragma* Line 14). Indeed, each thread can manage a different node v towards which to extend the non-dominated paths contained within `extendable`. As threads will discover distances towards different nodes v (written in turn in structures indexed on v), this prevents data-races. Note that in raw graphs, this method may lead to uneven workloads, as not all paths may be extendable towards any node v. However, since an SR graph is (at least) complete, any path may be extended towards any node v, leading to similar workloads among threads.

The routine `ExtendPaths`, detailed in Alg. 2, takes the list of extendable paths, *i.e.,* non-dominated paths discovered at the previous iteration, and a node v. It then extends the extendable paths to u further towards v. The goal is to update `dist[v]` with new distances that may belong to the Pareto front. Before being added to `dist[v]`, extended distances go through a pre-filtering. Indeed, the newly found distance to v may be dominated or may be part of the Pareto front. While this check is performed

Algorithm 1: BEST2COP(G',src)

```
1  pfront := Array of size MSD
2  forall i ∈ [0..MSD] do
3      pfront[i] := Array of size |V| of Empty Lists
4  add (pfront[0][src], (0,0))
5  dist := Array of size |V|
6  forall n ∈ V do
7      dist[n] := Array of size Γ
8  dist[src][0] = (0,0)
9  optimal constrained extendable:= Empty List (of Empty
     Lists)
10 add (extendable, (src, [ (0,0) ]))
11 nextextendable:= Array of size |V| of Empty Lists
12 i:= 1, max_d1:= 0
13 while extendable ≠ [ ] and i ≤ MSD do
14     #pragma omp parallel for
15     forall v ∈ V do
16         pf_cand := Array of size Γ
17         nb, imax := ExtendPaths (v, extendable, pf_cand,
             dist[v])
18         max_d1 = max (imax, max_d1)
           // How to iterate on dist to get new PF
19         if nb log nb + nb + |pfront[i-1][v]| < max_d1 then
20             d1_it := mergesortd1 (pfront[i-1][v],
                 pf_cand)
21         else
22             d1_it := [0...max_d1]
23         nextextendable[v] = [ ]
           // Extract new PF from dist
24         CptExtendablePaths (nextextendable[v],
             pfront[i][v], pf_cand, d1_it, dist[v])
       // Once each thread done, gather ext.  paths
25     extendable = []
26     forall v ∈ V | nextextendable[v] ≠ [ ] do
27         add (extendable, (v,nextextendable[v]))
28     i = i + 1
29 return pfront
```

thoroughly later, we can already easily prune some paths: if the new paths to v violate either constraint, there is no point in considering it. Furthermore, recall that `dist` stores, for all Γ M_1-distances towards a node, the best respective M_2-distance currently known. Thus, if the new M_2-distance is worst than the one previously stored in `dist` at the same M_1 index, this path is necessarily dominated and can be ignored. Otherwise, we add the distances to `dist` and update `pf_cand` to remember that a new distance which may be non-dominated was added during the current iteration. Note that `ExtendPaths` returns the number of paths updated within `dist`, as well as the highest M_2-distance found. This operation is performed for efficiency reasons detailed here.

Once returned, `dist` contains distances either dominated or not. We thus need to extract the Pareto front of the current iteration. This operation is performed in a lazy fashion once for all new distances (and not for each edge extension). Since this Pareto front lies within `dist`, one can simply walk through `dist` by order of increasing M_1 distance from 0 to the highest M_1 distance found yet and filter all stored distances to get the Pareto front of the current iteration. This may not be effective as most of the entries of `dist` may be empty.

However, the precise indexes of all active distances that need to be examined (to skip empty entries) can be constructed by merging and filtering the union of the current Pareto front and the new distances (`pf_cand`). Thus, if the sorting and merging of the corresponding distance indexes is less costly than walking through `dist`, the former method is performed in order to skip empty or useless entries. Otherwise, a simple walk-through is preferred. The merging of the M_1 distances of the Pareto front and new M_1 distances is here showcased at high-level (Line 20, Alg. 1). The usage of more subtle data structures in practice allows to perform this operation at the cost of a simple mergesort.

After the list of distance indexes to check and filter is computed, the actual Pareto front is extracted during the `CptExtendablePaths` procedure, as shown in Alg. 3. This routine checks whether paths of increasing M_1 distance do possess a better M_2 distance than the one before them. If so, the path is non-dominated and is added to the Pareto front, as well as to the paths that are to be extended at the next iteration. Finally, once each thread is terminated, `nextextendable` contains $|V|$ lists of non-dominated distances towards each node. These lists are merged within `extendable`, to be extended at the next iteration.

Note that most approximations algorithms relying on interval partitioning or rounding do not bother with dominance check. In other words, the structure they maintain is similar to our `dist`: the best M_2 distance for each M_1 distance to a given node. The latter may thus contain dominated paths which are considered and extended in future iterations. In contrast, by maintaining the Pareto front efficiently, we ensure to consider the minimum set of paths required to remain exact, and thus

profit highly from small Pareto front.

Algorithm 2: ExtendPaths(v, `extendable`, `pf_cand`, `dist_v`)

```
1  imax = 0, nb = 0
2  forall (u, d_list) ∈ extendable do
3      forall l ∈ E'(u,v) do
4          forall (d1u, d2u) ∈ d_list do
5              d1v = d1u + w1(l)
6              d2v = d2u + w2(l)
               // Filters:  constraints and dist
7              if d1v ≤ c1 and d2v ≤ c2
8              and d2v < dist_v[d1v] then
9                  dist_v[d1v] = d2v
10                 if not pf_cand[d1v] then
11                     nb ++
12                     pf_cand[d1v] = True
13                 if d1v > imax then
14                     imax = d1v
15 return nb, imax
```

Algorithm 3: CptExtendablePaths (`nextextendable_v`, `pfront_iv`, `pf_cand`, `d2_it`, `dist_v`)

```
1  last_d2 = ∞
2  forall d1 ∈ d1_it do
3      if dist_v[d1] < last_d2 then
4          add (pfront_iv, (d1,d2))
5          last_d2 = dist_v[d1]
6          if pf_cand[d1] then
7              add (nextextendable_v, (d1,d2))
```

The output of BEST2COP. When our algorithm terminates, the `pfront` array contains, for each segment number, all the distances of non dominated paths from the source s towards each destination d. To answer the 2COP problem, for each d and for all (stricter sub-)constraints $c'_0 \leq c_0$, $c'_1 \leq c_1$ and $c'_2 \leq c_2$, we can proceed as follows in practice:

- for $f(M_1, c'_0, \Gamma, c'_2, s, d)$, *i.e.,* to retrieve the distance from s to d that verifies constraints c'_0 and c'_2 minimizing M_1, we look for the *first* element in $\texttt{pfront}[c'_0 - 1][d]$ verifying constraint c'_2 (the first feasible distance is also the one minimizing M_1 because they are indexed on the later metric).
- for $f(M_2, c'_0, c'_1, \infty, s, d)$, we look for the *last* element in $\texttt{pfront}[c'_0 - 1][d]$ verifying constraint c'_1. The path minimizing M_2 being, by design, the last element.
- to compute $f(M_0, \infty, c'_1, c'_2, s, d)$, let us first denote k the smallest integer such that $\texttt{pfront}[k][d]$ contains an element verifying constraints c'_1 and c'_2. The resulting image is then any of such elements in $\texttt{pfront}[k][d]$.

As one might notice, computing $f(M_j, c'_0, c'_1, c'_2, s, d)$, $j = 0, 1, 2$ cannot always be achieved in constant time (for $j = 0$ and sub-constraints in particular). Indeed, we favor a simple data structure. A search in an ordered list of size Γ is needed for stricter constraints (and may be performed $\log(MSD)$ times when optimizing M_0). To improve the time efficiency of our solution, each $\texttt{pfront}[i][d]$ may be defined as or converted into a static array in the implementation.

Finally, for simplicity, we did not show in our pseudo-code the structure and operations that store and extend the lists of segments. In practice, we store one representative of the best predecessors and a posteriori retrieve the lists using a backward induction for each destination.

Appendix B. SR Graph & Live Conversion Algorithm

The multi-metric SR Graph may be used in different ways. The path computation algorithm may be run directly on it. While this induces the cost of exploring a fully-meshed graph, it allows to treat the segment metric as a standard graph metric (in the form of the number of edges). Another way to use the SR Graph is to use the information it contains to perform conversions. The algorithm thus explores the original, sparser topology and converts the explored paths into segment lists on the fly to consider their number of segments. Note that the information within the SR Graph may be stored in different fashions (*e.g.,* by keeping the results of the APSP computation separated by source rather than merging it in a single SR Graph). However, the multi-metric APSP computation itself is mandatory.

The second method requires to design an efficient conversion algorithm, able to transform a path into a minimal segment list which respects the cost *and* delay of the original path. This conversion is not trivial, as one must take into account the (forced) path diversity brought by ECMP. Indeed, using a node segment implies that the packet may follow any of the ECMP paths, which may possess heterogeneous delays. On the other hand, relying too much on adjacency segments will not lead to the minimal segment encoding of the considered paths. In addition, the number of segments, being an "off the graph" metric, exhibits peculiar behavior and requires to slightly extend the way path are checked for dominancy.

Appendix B.1. Conversion Algorithm

We start by introducing the following notations. We consider a network $G = (V, E)$. Let $p = (x_0, \ldots, x_l)$ be a path within G, with $x_i \in V$. Let $p_{[x_i, x_j]} = (x_i, \ldots, x_j)$. We note $d(p)$ the couple of distances $(d_1(p), d_2(p))$ of the path p.

We denote *Dag* a shortest path tree (regarding the M_2 metric, *i.e.,* the IGP cost). $Dag_{[u,v]}$ denotes the paths and distances from u to v, within the shortest path tree

Dag rooted at u. To encode a given path, the *Dag* rooted at each node $u \in V$ must be known. The latter must account for all ECMP path. Alternatively, one may compute the SR graph of the graph G, which contains the minimal amount of information required to perform any path encoding.

We denote SR_G the SR graph of G, computed as explained in the main body of the paper (Section 3.2), but with the adjacency segments removed (the latter are not necessary to perform path encoding). We denote segments $S_i = (t_i, s_i, s_{i+1})$ with $s_i \in V$ and $t_i \in \{AdjSeg, NodeSeg\}$. The latter may either encode the best ECMP paths from s_i to s_{i+1} (in which case, $t_i = NodeSeg$) or a specific link between the two nodes (in which case, $t_i = AdjSeg$). We denote $SR(u, v)$ the best guaranteed distances when using a node segment from u to reach v (*i.e.,* the maximum delay among all path within $Dag_{[u,v]}$). In other words, it is the weight of the edge (u, v) within SR_G. We define a list of segments as $S = (S_i)_{0 \leq i \leq l-1}$.

We now define the *encoding* of a path p.

Definition 1 (Encoding). *Let $p = (x_0, \ldots, x_l)$ be a path. Let $S = (S_i)_{0 \leq i \leq l-1}$ be a segment list. S encodes the path p if both conditions are verified:*

- *$\forall i$, $\exists j$ such that $s_i = x_j$*
- *$\forall i$, either*
  - *$p_{[s_i,s_{i+1}]} \in Dag_{[s_i,s_{i+1}]} \wedge SR(s_i, s_{i+1}) \leq d(p_{[s_i,s_{i+1}]})$*
  - *$t_i = AdjSeg \wedge (s_i, s_{i+1}) \in p$.*

We then define the *delay-cost path encoding problem.*

Definition 2 (Delay-cost path encoding problem). *Given a path p, the delay-cost path encoding problem consists in finding a segment list L that encodes p with the minimal number of segments.*

To solve the delay-cost path encoding problem, we start by presenting an algorithm which finds, given a path p, the longest prefix of p encodable by a single segment. The pseudo-code of the algorithm is shown in Alg. 4.

Algorithm 4: 1SegLongestPrefix(p))

```
1 if p_[x0,x1] ∉ Dag_[x0,x1] then
2     return (AdjSeg, x0, x1)
3 for i = 2 ... l do
4     if p_[x0,xi] ∉ Dag_[x0,xi] or SR(x0, xi) > d(p_[x0,xi]) then
5         return NodeSeg, x0, x_{i-1})
6 return NodeSeg, x0, xl)
```

The test performed at Line 1 checks whether the first edge of p is within the shortest path tree of rooted at x_0. If so, the edge is not part of the ECMP path from x_0 to x_1. Thus, an adjacency segment is required. Otherwise, the algorithms checks, for each $i = 2 \ldots l$ whether the path $p_{[x_0,x_i]}$ is encodable in a single segment. This is equivalent to checking whether the path is within the shortest path tree rooted at x_0, and if the latter has, among all the ECMP paths $Dag_{[x_0,x_i]}$, the highest delay. Once this checks does not hold anymore, the algorithm returns.

Note that because the SR graph SR_G encodes the minimal amount of information for such conversion, both conditions (Line 1 and 4) can be checked easily and efficiently by relying on the SR graph SR_G. Indeed, the condition at Line 4 can be expressed equivalently as $SR(x_0, x_{i-1}) + d(x_{i-1}, x_i) \neq SR(x_0, x_i)$. Similarly, Line 1 can be expressed as $SR(x_0, x_1) \neq d(x_0, x_1)$.

We then generalize this algorithm to encode any given path p with a minimal number of segments, as shown in Alg. 5.

Algorithm 5: encode(p)

```
1 s := x0
2 S := ∅
3 do
4     S_i = (t, s, v) = 1SegLongestPrefix(p_[s,xl])
5     L.push(S_i)
6     s := v
7 while s ≠ xl;
8 return L
```

Alg. 5 starts by finding the segment S_i encoding the longest prefix in p encodable in a single segment. Suppose that S_i encodes the subpath $p_{[s,v]}$. The algorithm adds the segment to the segment list S and repeats the procedure, considering the reminder of the path $p_{[v,x_l]}$ and v as the new source. This is repeated until v is equal to the last node in p, *i.e.,* x_l.

Lemma 1. *Let S be a segment list that encodes p. Then, $|S| \geq |encode(p)|$.*

Proof: Let us consider two segment lists $S = ((S_i)_{0 \leq i \leq l-1})$ and $S' = ((S'_i)_{0 \leq i \leq l'-1})$, both encoding p. Let $S' = encode(p)$ (*i.e.,* S' is the result of Alg. 5).

For the sake of contradiction, let us assume that $l < l'$. Then, there must exist k, $0 \leq k \leq l-1$, such that

1. s_k appears before, or at the same place as s'_k in path p
2. s_{k+1} appears strictly after s'_{k+1} in path p.

Let us start by considering that $s_k = s'_k$. Then, 2. contradicts with the fact that S'_k is a path encoding the longest prefix of p (recall that S'_k is the output of procedure 1SegLongestPrefix on s'_k).

Otherwise, s_k appears strictly before s'_k in path p. Following the definition of encoding, $p_{[s_k,s_{k+1}]} \subset Dag_{[s_k,s_{k+1}]}$. In other words, $p_{[s_k,s_{k+1}]}$ is a shortest path (regarding M_2). Since $s'_k \in p_{[s_k,s_{k+1}]}$, then, $p_{[s'_k,s_{k+1}]}$ is a shortest path as well, in other words

$$p_{[s'_k,s_{k+1}]} \subset Dag_{[s'_k,s_{k+1}]}$$

and by definition of SR,

$$d(p_{[s'_k,s_{k+1}]}) = d(s'_k, s_{k+1}) \leq SR(s'_k, s_{k+1})$$

meaning that $(NodeSeg, s'_k, s_{k+1})$ encodes $p_{[s'_k,s_{k+1}]}$ which contradicts the fact that S'_k is the longest encoding.

□

Note that within our code, the conversion algorithm is implemented in a slightly different fashion for performance sake. During the exploration of the graph, path are extended edges by edges. To avoid recomputing the encoding of a path from scratch after each extension, we save for each path states allowing to efficiently update, for a given path p, the number of segments required if the latter is extended by an edge e.

Appendix B.2. Dominancy checks

Although the algorithms presented in the previous section allow to find the number of segments to encode a path, it is not sufficient to correctly take into consideration the number of segments when computing paths. Indeed, conventional DCLC algorithm tend to only extend non-dominated paths. However, some precautions must be taken when comparing the number of segments of two paths. Indeed, because of the peculiar nature of the metric, the latter may evolve differently. As a consequence, paths that seem dominated must sometimes be extended nevertheless, as they may end up requiring less segments than their dominating competitors.

To illustrate this issue, let us consider two paths p and p'. Let $S = (S_i)_{0\leq i\leq l-1} = \text{encode}(p)$ and $S' = (S'_i)_{0\leq i\leq l'-1} = \text{encode}(p')$. Let $S'_{l-1} = (AdjSeg, s'_{l-1}, s'_l)$ and $S_{l-1} = (NodeSeg, s_{l-1}, s_l)$. Then, intuitively, if p' is to be extended by an edge $e = (u, v)$, a new segment is mandatory, as S'_{l-1} may not encode more than a single link. However, a new segment is not necessarily mandatory to extend p, as S_{l-1} may very well be *extended* to include e, if $p_{[s_{l-1},v]} \in Dag_{[s_{l-1},v]}$ and if $SR(s_{l-1}, v) < d(p_{[s_l,v]})$.

Note that this effect may also occur (although less obviously) even when both p and p' end with a node segment. Indeed, it is possible that $p_{[s_{l-1},v]} \in Dag_{[s_{l-1},v]}$ while $p_{[s'_{l-1},v]} \notin Dag_{[s'_{l-1},v]}$

Consequently, even though a path may seem dominated, this may not be definitive. Thus, some paths that *seem* dominated must be nevertheless be extended. More precisely, a path p *should not* be considered dominated by a path p', regardless of their cost and delay, if the three following conditions are met:

1. $d_0(p) = d_0(p')$
2. $s_{l-1} \neq s'_{l-1}$
3. $t_{l-1} = NodeSeg$

By modifying the dominancy condition as such, and relying on the conversion algorithms presented, a standard multi-metric path computation algorithm is able to correctly take into account the number of segments as another path metric. Although the overhead induced by our conversion algorithm should be limited, note that this dominancy condition implies that some paths that are ultimately dominated may have to be extended nonetheless, increasing the overall worst-case complexity of the algorithm. The optimal usage of the SR Graph thus depends on the overhead of the conversion and extended dominancy check, compared to the cost of exploring a fully meshed graph.

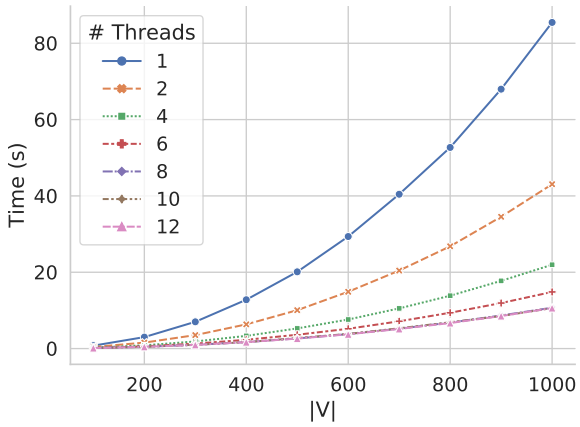


Figure C.9: Upper bound of BEST2COP execution time regarding the number of nodes and threads.

Appendix C. BEST2COP upper bound with multi-threading

To test BEST2COP upper bound, we force it to explore its full iteration space. In other words, the algorithm behaves as if $|V|^2 \times L \times \Gamma$ distances have to be extended at each iteration. The results are shown in Fig. C.9 for an increasing number of nodes and threads. BEST2COP does not exceed 84s when using a single thread, considering $|V| = 1000$ and $L = 2$, the average number of parallel links in the transformed graph. This time, reasonable given the unrealistic nature of the experience, is significantly reduced when relying on multi-threading. Using 8 threads, BEST2COP execution time decreases to around 10s, highlighting both the parallelizable nature of our algorithm and its inherently good performance (past this number of threads, there is no more speedup as we exceed the number of physical cores of our machine). Additional experiments conducted on a high-performance grid show that BEST2COP reaches a speedup factor of 23 when run on 30 cores. Ultimately, the seepd-up exhibits is also highly dependent on the underlying hardware and the difficulty of the problem instance.