Bounded Dijkstra (BD): Search Space Reduction for Expediting Shortest Path Subroutines

Amaury Van Bemten, Jochen W. Guck, Carmen Mas Machuca and Wolfgang Kellerer

Lehrstuhl für Kommunikationsnetze

Technical University of Munich

Email: {amaury.van-bemten, guck, cmas, wolfgang.kellerer}@tum.de

Abstract—The *shortest path* (SP) and *shortest paths tree* (SPT) problems arise both as direct applications and as subroutines of overlay algorithms solving more complex problems such as the *constrained shortest path* (CSP) or the *constrained minimum Steiner tree* (CMST) problems. Often, such algorithms do not use the result of an SP subroutine if its total cost is greater than a given bound. For example, for delay-constrained problems, paths resulting from a least-delay SP run and whose delay is greater than the delay constraint of the original problem are not used by the overlay algorithm to construct its solution. As a result of the existence of these bounds, and because the Dijkstra SP algorithm discovers paths in increasing order of cost, we can terminate the SP search earlier, i.e., once it is known that paths with a greater total cost will not be considered by the overlay algorithm. This early termination allows to reduce the runtime of the SP subroutine, thereby reducing the runtime of the overlay algorithm without impacting its final result. We refer to this adaptation of Dijkstra for centralized implementations as *bounded Dijkstra* (BD). On the example of CSP algorithms, we confirm the usefulness of BD by showing that it can reduce the runtime of some algorithms by 75% on average.

Index Terms—search space reduction, Dijkstra algorithm, shortest path routing, early termination, subroutine

I. Introduction

The *shortest path* (SP) and *shortest paths tree* (SPT) routing problems arise in a wide range of practical problems, both as direct applications [1] and as subroutine of other more complex problems such as the *(multi-)constrained shortest path* (CSP and MCSP) and *multi-constrained path* (MCP) routing problems [2], which are encountered, e.g., when provisioning quality of service (QoS) in software-defined networks [3]–[5]. When used as a subroutine, the result of one or several SP/SPT search(es) is used to determine a solution to the original problem. For example, DCUR [6], [7] combines the least-delay and least-cost SPTs from all nodes to a given destination in order to compute a *delay-constrained least-cost* (DCLC) path, i.e., a CSP, from a single source to a single destination. As a result, optimizing SP/SPT procedures is an important issue, as it allows to reduce the runtime of the wide range of routing algorithms using them, those of which can have considerable runtime footprint [2], [8].

For SP/SPT problems, the Dijkstra algorithm [9] is commonly acknowledged as the fastest optimal algorithm for centralized implementations [10]. When using an SP/SPT algorithm as a subroutine, its result is often not used if its total cost is greater than a given bound. For example, DCUR
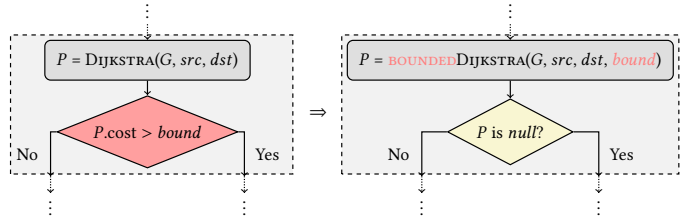

Fig. 1: When using a shortest path subroutine, algorithms often do not use the paths returned by Dijkstra if they are more costly than a given *bound*. The usage of bounded Dijkstra (BD) consists in incorporating this decision in the Dijkstra subroutine in order to avoid exploring these useless paths, thereby reducing the search space of Dijkstra. The input and output of the overlay algorithm are left unchanged, only its runtime is affected.

does not use the paths of its least-delay SPT search whose delay is greater than the delay constraint of the CSP problem. Indeed, these paths cannot lead to a feasible solution of the original problem. As a result, because Dijkstra discovers paths in increasing order of cost, the least-delay SPT search of DCUR can be stopped when paths with a delay higher than the delay constraint of the CSP problem are reached. This early termination of Dijkstra allows to reduce the runtime of the SP/SPT search, thereby reducing the runtime of the overlay algorithm (e.g., DCUR) without impacting its result. We refer to this simple adaptation of Dijkstra for centralized implementations as *bounded Dijkstra* (BD). BD can be used by any routing algorithm making use of one or several SP/SPT search(es) and able to provide a bound to these subroutines (Fig. 1).

First, in Sec. II, we present related work aiming at optimizing the runtime of SP/SPT searches. Second, after presenting the simple functioning and the benefits of BD in Sec. III, we show how BD can be used, i.e., how a bound can be provided to the SP/SPT subroutines, in the particular case of centralized CSP algorithms (Sec. IV). We show that BD can be used by a wide range of algorithms. Indeed, 20 out of 26 recently surveyed CSP algorithms [2] can make use of BD to improve their runtime. For each of them, we detail how bounds can be provided to the SP/SPT subroutines. Finally, in Sec. V, we evaluate the impact of BD on the performance of all these CSP algorithms. We observe that BD can reduce the runtime of some CSP algorithms by 75% on average. For favorable cases, BD reduces the runtime of several algorithms by 96% on average. We further confirm that using BD does not change the final solution found by the algorithms. These algorithms

hence have only benefits in using BD: reduced runtime at no cost. Due to the high number of algorithms, we only present the most interesting and insightful results and conclusions. The entire set of raw results and graphs is publicly available at *https://lora.lkn.ei.tum.de* [11].

II. RELATED WORK

The SP/SPT problem has been thoroughly investigated in the literature. In this section, we classify the attempts at making SP/SPT routines faster in six categories for which we list representative examples and with respect to which we highlight our contribution.

1) Heuristics: Some approaches improve the runtime of SP/SPT searches by accepting to find sub-optimal solutions [12]. In contrast, for positive metrics, BD guarantees to find the optimal result.

2) Hierarchical Routing: A way of reducing the complexity of SP/SPT routing is to apply hierarchical routing, thereby running SP/SPT algorithms on smaller graphs [13], [14]. As it does not modify the subject graph, BD can be used as part of any such hierarchical routing scheme.

3) Data Structure Optimizations: Several studies propose optimized data structures for the implementation of Dijkstra [15]–[17]. BD is independent of the data structure used and can hence be used with any of these data structures.

4) Improvements of Existing Algorithms: Several proposals introduce extensions to the well-known Dijkstra [18], [19] and Bellman-Ford [20], [21] algorithms. BD falls into this category but can be used in parallel with these improvements.

5) Bi-Directional Searches: Dijkstra explores the graph from the source towards the destination(s). For SP problems, bi-directional searches, starting from both the source and the destination simultaneously, have been proposed [22]–[24], potentially pruning parts of the individual searches when possible [22]. While bi-directional searches can only be used for SP problems, BD can be used for both SP and SPT problems.

6) Preprocessing: In the context of very large graphs such as road networks, a plethora of work [25] proposes to perform a preprocessing step computing intermediate information by, e.g., clustering nodes [26], defining overlay graphs [27], defining important transit nodes [28], or computing virtual links [29]. This preprocessed information is later used to solve routing requests faster. The precomputation step being costly, these approaches are only suitable for solving a batch of requests on the same static topology. Besides, these algorithms work well for large topologies but hardly outperform Dijkstra for general topologies [25]. On the other hand, BD can provide significant benefit on general and dynamic topologies. Some preprocessing algorithms involve SPT subroutines [28] and expedite these subroutines by pruning parts of the network because the corresponding information was already obtained from previous SPT runs. In contrast, BD does not exploit previous searches in order to improve runtime but rather information provided by an overlay algorithm. Further, these proposals simply prune parts of the network when a given condition is met, while BD completely terminates.

We note that bounded Dijkstra runs were already used in the literature [30]–[36], but only as part of the design of new specific algorithms. Our contribution consists in the formalization and generalization of such an approach for any generic algorithm using SP/SPT subroutines, and in the quantification of its benefits for these algorithms.

III. BOUNDED DIJKSTRA (BD)

In this section, we present the *bounded Dijkstra* (BD) algorithm. After presenting the context in which BD can be used (Sec. III-A), we describe the simple idea of the algorithm (Sec. III-B) and detail the impact it can have on SP (Sec. III-C) and SPT (Sec. III-D) searches. Then, we show that the same idea can also be applied to the *Bellman-Ford* (BF) [37], [38] and *Chong* [39] algorithms (Sec. III-E and III-F), respectively another SP/SPT algorithm and a kSP/kSPT algorithm.

A. Context: Centralized Bounded SP/SPT Subroutines

The *shortest path* (SP) and *shortest paths tree* (SPT) problems are core networking problems. Besides in their direct applications, these problems are often encountered as subproblems of other more complex centralized problem settings. For example, many *(multi-)constrained shortest path* algorithms (CSP and MCSP) use results of SP/SPT searches to determine a solution to their problem [2]. Similarly, *multi-constrained path* (MCP) algorithms such as H_MCP [40] or *constrained minimum Steiner tree* (CMST) algorithms such as BSMA [41] make use of an underlying SPT algorithm to construct a solution to their problem. When an SP/SPT algorithm is used as such a subroutine of a centralized algorithm, it often happens that paths with a total cost greater than a given bound are not used. For an SP search, this means that the result itself is not used. For an SPT search, this means that the paths to some destinations (too costly) are not considered, while others are. For example, for delay-constrained problems, paths resulting from a least-delay SP/SPT run which have a delay higher than the delay constraint are not considered. Similarly, for *delay-constrained least-cost* (DCLC), or CSP, problems, paths resulting from a least-cost SPT run which have a cost higher than the cost of the least-delay path will not be used, as these paths have a higher delay *and* cost than the least-delay path.

B. Idea: Early Termination for Search Space Reduction

As a result of the existence of these bounds, and because the Dijkstra algorithm [9] discovers paths in increasing order of cost, the SP/SPT searches can be stopped earlier, i.e., once it is known that paths with a greater total cost will not be considered by the overlay algorithm, thereby reducing the search space of Dijkstra and hence the runtime of the overlay algorithm. We refer to such a modified version of Dijkstra as *bounded Dijkstra* (BD). The pseudo-code of BD is shown in Fig. 2 and the required modification in the overlay algorithm is shown in Fig. 1. For positive metrics, BD does not influence

```
1: function BOUNDEDDIJKSTRA(G, src, dst, bound)
2:     Create empty priority queue Q
3:     for each node ∈ G do
4:         node.cost ← +∞
5:     src.cost ← 0
6:     Q.add(src)
7:     while not Q.empty do
8:         node ← Q.popLeastCostNode()
9:         if node is dst then return GETPREDECESSORS(dst)
10:         if node.visited then continue
11:         node.visited ← TRUE
12:         for each outgoing edge of node as edge do
13:             newCost ← node.cost + edge.cost
14:             if newCost < bound then
15:                 if newCost < edge.dst.cost then
16:                     edge.dst.cost ← newCost
17:                     edge.dst.predecessor ← node
18:                     Q.add(edge.dst)
19:     return NULL
```

Fig. 2: Pseudo-code of the Dijkstra algorithm and the BD adaptation (shown in light red). Note that, depending on the data structure in use (see Sec. II-3) for the priority queue (Q), the pseudo-code may vary slightly. We show here the most common pseudo-code using a heap, which we used for our implementation and which performs best among the available data structures [10].
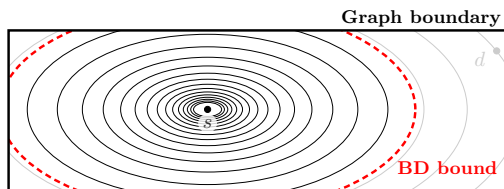

Fig. 3: Dijkstra discovers paths in increasing order of cost (as illustrated by the ellipses representing nodes which are equidistant from the source node) and stops once the destination (d) is reached. BD (dashed ellipse) terminates the search at a given cost from the source node, once it is known that longer paths will not be considered by the overlay algorithm.

the result of the overlay algorithm, as the latter considers having no path and having a too costly path identically. In order to use BD, an algorithm must of course be able to provide a bound value above which results are unnecessary. We detail in Sec. IV, on the example of CSP algorithms, how overlay algorithms can provide such bounds.

C. Impact on an SP Search

For an SP search, the Dijkstra algorithm [9] discovers paths in increasing order of cost from the source node and stops once the destination is reached. The pseudo-code is shown in Fig. 2. When a bound is provided to Dijkstra (or BD), two different cases can happen.

1) BD: The Destination is Further than the Bound: First, the provided bound can be lower than the cost of the SP to the destination. In this case, BD avoids exploring unnecessary parts of the network (Fig. 3) by preventing Dijkstra from considering paths with a cost greater than the provided bound and hence terminating before reaching the destination, i.e., before Dijkstra would have terminated. The path returned is then "NULL", which is considered by the overlay algorithm in the same way as a path which is too costly: it does not use it.

2) BD: The Destination is Closer than the Bound: Second, the provided bound can be greater than the cost of the SP to the destination. In this case, BD might appear useless, as it

will, like Dijkstra, terminate when reaching the destination. However, BD can also have a benefit in this case. Indeed, BD can avoid putting an element in the queue whose associated cost is greater than the provided bound (line 14 in Fig. 2). Because such elements can be reached before the destination, this allows BD to avoid unnecessary operations (lines 15-18) and to reduce the size of its priority queue, thereby increasing the speed of the upcoming popping operations (line 8). Let us consider an example where the cost of the SP to the destination is 16 and the BD bound is 18. BD pops an element with an associated cost value of 15 out of its queue (line 8) and expands it. This expansion leads to elements with associated costs 20, 22 and 24. While the traditional Dijkstra would execute lines 15-18, BD knows that these elements will never be used. Hence, BD can directly discard these elements (line 14), thereby preventing executing lines 15-18. As a result, even when the bound provided to BD is greater than the cost of the SP to the destination, BD can be beneficial. We will confirm this in our evaluations (Sec. V-B). However, the benefit is expected to decrease as the bound gets greater. Indeed, the number of elements that can be discarded will decrease. In the example above, a higher BD bound of 23 would for example allow to discard only one of the elements, rather than the three of them with a BD bound 18. Note that this phenomenon also happens when the destination is further than the provided bound (Sec. III-C1).

D. Impact on an SPT Search

For an SPT search, Dijkstra behaves as in the SP case (Fig. 2) but instead of stopping when reaching a given node (line 9 of Fig. 2), the algorithm stops when its priority queue is empty, i.e., when it has reached all the nodes.

When provided with a bound, BD can potentially stop the SPT expansion before exploring the whole graph. That is, BD can avoid waiting for reaching some nodes which are too far away. As for the single-destination case, this allows to reduce the runtime by preventing the exploration of unnecessary parts of the network (see Sec. III-C1) and by avoiding unnecessary operations and the addition of unnecessary elements to the priority queue (see Sec. III-C2).

E. BD Idea for Bellman-Ford (BF)

The *Bellman-Ford* (BF) algorithm [37], [38] is another algorithm for solving SP/SPT problems. Because of its structure, the algorithm is more often used in distributed implementations. However, it is also used as a subroutine of other complex algorithms where Dijkstra cannot be used (e.g., DEB, see Sec. IV-D). While the structure of the BF algorithm is very different from the structure of Dijkstra, it also discovers paths in increasing order of cost. Hence, the BD idea can also be applied to BF by simply discarding paths more costly than the given bound. As a result, algorithms making use of the BF algorithm as a subroutine can also apply the BD principle.

F. BD Idea for Chong's Algorithm

The problem of finding the *k shortest paths* (kSP) between two nodes (or the kSPT from one node to several destinations)

Algorithm	Delay				Cost	Comb.
	BF	SP	SPT	kSPT	SPT	SPT
Algorithms that Cannot Use BD (Sec. IV-A)						
CBF [42]						
A*Prune [43]						
kSPMC [2]						
E_MCOP [44]						
SMS-PBO [45]						
kLARAC [46]						
Algorithms that Can Use BD for SP Only (Sec. IV-B)						
LDP [2]		1				
FB [47]		(0, 1)				
LARAC [48]–[51]		1				
LARACGC [49]		1				
SCRC [52]		1				
DCCR [53]		1				
SSR+DCCR [53]		1				
Algorithms that Can Use BD for SPT (Sec. IV-C)						
DCUR [6], [7]			1			(0, 1)
SF-DCLC [54]			1			(0, 1)
SMS-CDP [45]			1			(0, 1)
SMS-RDM [45]			1			
IAK [55]			1			
DCR [56]		1				(0, 1)
H_MCOP [40]			1			
kH_MCOP [40]			1			
NR_DCLC [57]		(0, 1)				≥ 0
MH_MCOP [44]			1			≥ 0
DCBF [46]			1			
kDCBF [46]				1		
Algorithm that Can Use BD for BF (Sec. IV-D)						
DEB [58]	1					

TABLE I: Number of times some constrained shortest path (CSP) algorithms can make use of BD based on the metric (cost, delay or a combination) and algorithm on which BD can be applied (SP/SPT refers to Dijkstra, kSPT to Chong and BF to Bellman-Ford for SP). When the number of times BD can be used depends on the routing request, the set of possible values is given between parentheses and unbounded values are given using the $\geq$ symbol. Underlined algorithms are optimal.

also arises often as a subroutine of more complex algorithms (e.g., kDCBF, see Sec. IV-C). *Chong's algorithm* [39] solves this problem by assuming that the k value is known a priori. The algorithm is identical to the Dijkstra algorithm but keeps track, at each node, instead of one single path, of the current k best paths found. Hence, the BD idea can be applied to Chong's algorithm in the same way as it is applied to Dijkstra. As a result, algorithms making use of Chong's algorithm as a subroutine can also apply the BD principle. BF can also be adapted to a static kSP algorithm by also simply keeping track of the current k best paths found towards each node. This adaptation can also apply the BD principle.

IV. APPLICATION: BD FOR CSP ROUTING

In this section, we show that BD can be used by a wide range of algorithms by showing *(i)* how existing *constrained shortest path* (CSP) algorithms can replace their SP and SPT subroutines with BD, and *(ii)* how bounds can be provided to these BD runs.

The CSP problem consists in finding the shortest path (in terms of a first metric referred to as *cost*) such that a second metric (referred to as *delay*) is lower than a given bound. CSP algorithms use SP/SPT subroutines using either the cost metric, the delay metric, or a combination of both [2] as optimization metric. Tab. I shows, for each algorithm, how many times it can replace a SP/SPT run with a BD run. The cases for which BD can be used are separated based on the metric (cost, delay or a combination) and on the algorithm on which the BD principle is applied (SP/SPT refers to Dijkstra, kSPT to Chong and BF to Bellman-Ford for SP).

In a recent survey [2], Guck *et al.* presented 26 different CSP algorithms, out of which only 6 cannot make use of BD. In the following sections (and in Tab. I), algorithms are referred to using their acronyms as defined in [2].

A. Algorithms that Cannot Use BD

First, *CBF* [42], *A*Prune* [43] and *SMS-PBO* [45] have a specific structure making use of no underlying (k)SP/SPT algorithm and can hence not make use of BD. Second, *kSPMC* [2], *E_MCOP* [44] and *kLARAC* [46] exclusively make use of kSP and SP algorithms to which no bound can be provided.

B. Algorithms that Can Use BD for SP Only

The *LDP* [2], *FB* [47], *LARAC* [48]–[51], *LARACGC* [49], *SCRC* [52], *DCCR* [53], and *SSR+DCCR* [53] algorithms run a least-delay SP procedure (i.e., optimizing the delay metric) which can make use of BD by using the bound of the original problem. After this least-delay SP run, the *LARAC*, *LARACGC*, *SCRC*, *DCCR*, and *SSR+DCCR* algorithms run one or several least-cost SP runs (i.e., optimizing the cost metric). These runs could be provided with the cost of the least-delay path as bound. However, if provided with this bound, this BD run will always be in the case described in Sec. III-C1 where the provided bound is greater than the cost of the shortest path to the destination. As we will see in Sec. V-B, on average, the usage of BD in such a case increases the runtime of the SP run. As a result, we do not consider the least-cost run as a BD run.

C. Algorithms that Can Use BD for SPT

The *DCUR* [6], [7], *SF-DCLC* [54], *SMS-CDP* [45], *SMS-RDM* [45], *IAK* [55], *DCR* [56], *H_MCOP* [40], *kH_MCOP* [40], *DCBF* [46] and *kDCBF* [46] algorithms run a least-delay search to which the delay bound of the CSP problem can be provided as a bound. While *DCR* runs a least-delay SP search and *kDCBF* a least-delay kSPT (Chong) search, all the others run a least-delay SPT search. DCUR,

SF-DCLC, SMS-CDP and DCR then possibly execute a least-cost SPT run to which the cost of the least-delay path from the source to the destination can be provided as a bound. Indeed, any path with a cost higher than the least-delay path will never be used by the algorithms, as they would then rather choose to follow the least-delay path, which has both a lower cost and delay. The *IAK*, *H_MCOP*, *kH_MCOP*, *DCBF* and *kDCBF* algorithms further execute a least-cost (k)SP search. As for the algorithms in Sec. IV-B, a bound could be provided to this least-cost run but, for the same reason, we do not consider it.

The *NR_DCLC* algorithm [57] starts like FB and can hence make use of BD in the same way. Then, if the problem is feasible, it runs several times H_MCP [40], [44] (an MCP algorithm), a modified version of H_MCOP, to improve on the least-delay path result. H_MCP uses a metric combining the cost and delay metrics for its SPT search. Since bounds on both the delay (the bound of the CSP problem) and on the cost (the cost of the best path found so far) are known, the first step of H_MCP can also make use of BD. Hence, NR_DCLC can further make use of BD by using H_MCP with BD.

The *MH_MCOP* algorithm [44] is similar to NR_DCLC but, instead of using H_MCP to improve on the least-delay path result, H_MCP is used to improve on the path found by H_MCOP. Hence, MH_MCOP can make use of BD by using both H_MCOP and H_MCP with BD.

D. Algorithm that Can Use BD for BF

The *DEB* algorithm [58] runs a least-cost and a least-delay SP search using BF. As for *LARAC*, a bound can be provided to both the least-delay and least-cost searches but we only consider the least-delay search as a BD run.

V. Evaluation

The goal of our evaluation is twofold. First, in order to confirm our expectations of Sec. III-C and III-D, we quantify the impact of BD on an SP and an SPT run. To do so, we observe the behavior of the LDP (Sec. V-B) and IAK (Sec. V-C) algorithms, which are using BD respectively for a single SP and a single SPT run based on the delay metric. Second, in order to confirm the applicability of BD, in Sec. V-D, we observe its impact on the performance of the CSP algorithms presented in Sec. IV. Because of the big amount of resulting data, we only present here the most insightful and representative results. The complete data and set of graphs has been made publicly available at *https://lora.lkn.ei.tum.de* [11].

Among all the runs performed during the evaluation, the paths returned by the algorithms with and without BD were always identical, thereby confirming that BD does not impact the output of the algorithms. Hence, in the following, we only discuss the runtime of the algorithms.

The algorithms have been implemented using Java 8 and evaluated on an Ubuntu 16.04 PC equipped with an Intel Core i7-4790 CPU @ 3.60GHz.

A. Setup

In this section, we define the three dimensions (Sec. V-A1 to V-A3) along which we run our evaluation and describe how our plots (Sec. V-A4) and routing requests (Sec. V-A5) are generated.

1) First Dimension: Distance between Nodes: From Fig. 3, we can expect that, if the source and destination nodes are far apart from each other, the impact of BD will be lower. Indeed, in most directions, the graph boundary itself will be expected to stop the expansion of Dijkstra before BD does it. If the source and destination nodes are closer to each other compared to the graph size (alternatively, if the graph boundary rectangle in Fig. 3 gets bigger), we can expect that the BD bound will be reached more often before the boundary of the network and hence BD will provide more benefit. That is, the impact of BD potentially depends on the relative distance (in terms of cost) between the source and destination nodes compared to the size of the topology. To ease the definition of this dimension, we only consider a grid topology of size $N \times N$. For a given grid size, we define 10 different so-called *distance buckets*. These buckets correspond to source and destination nodes pairs whose least-hop distance in the grid is between 0 and 10%, 10% and 20%, ..., 90% and 100% of the longest path in the grid (i.e., of $2 \times (N - 1)$).

2) Second Dimension: Tightness of the Constraint: The delay constraint can range from a loose value for which the least-cost path is feasible to tight values for which the problem is infeasible. Within this range, we define 7 ranges of equal sizes. We refer to these ranges as *delay levels*.

3) Third Dimension: Grid Size: We further consider the grid size N as an evaluation dimension. We vary N from 6 to 20. We observed that the grid size does not influence the impact of BD. Hence, we here ignore this dimension (we always aggregate all the results for all the sizes) but the corresponding graphs are available online [11].

4) Plots Generation: For each algorithm, we generate plots showing the distribution[1] of the *runtime ratios* observed for the different values of a given dimension (i.e., node distance, delay level or grid size). For a given request, the runtime ratio is defined as the runtime of the algorithm without BD divided by the runtime of the algorithm with BD. The hidden dimensions are either aggregated (i.e., the runtime ratios for all their values are incorporated in the distributions) or only a specific value of these dimensions is incorporated in the distributions.

5) Requests Generation: For each algorithm and for each combination of distance bucket, delay level and grid size, we generate random cost and delay values between 1 and 2 for each link and we randomly generate 5000 requests (within the corresponding distance bucket and delay level). The 5000 requests are then solved by the considered algorithm and its corresponding version with BD. The first 500 runs are used

---

[1]After removing the values below the 1% percentile and above the 99% percentile, the distributions are shown as boxplots showing the 10%, 25%, 50%, 75% and 90% percentiles. A red square identifies the average. Versions of the plots also showing the outliers are available on the accompanying web interface [11].
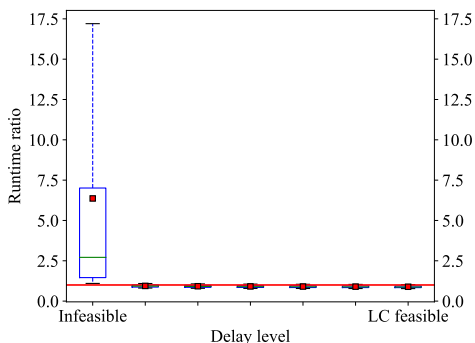
Fig. 4: Runtime ratios of LDP for different delay levels. On average, for single-destination SP searches, BD is only useful when the provided bound is lower than the cost of the shortest path to the destination.
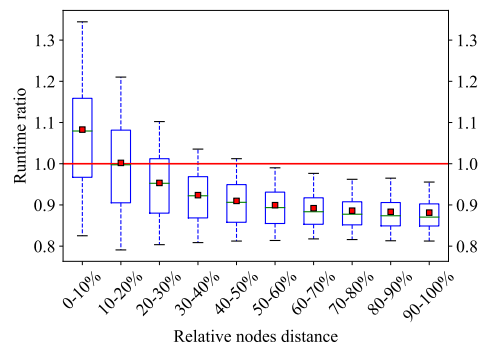

Fig. 5: Runtime ratios of LDP for the different distance buckets and for the first delay level that is feasible. For SP searches, in some favorable cases (short distances), BD can still be beneficial even if the provided bound is greater than the cost of the shortest path to the destination.

as warm-up for the Java HotSpot optimizer and their results are not considered. The order in which the algorithm and its BD version are run is alternating. This prevents the Java HotSpot optimizer from optimizing one of the run over the other. The distance bucket dimension cannot be aggregated simply by considering all the runtime ratios for all the different buckets. Indeed, considering random source-destination pairs, small distances are more probable than long distances. Hence, for plots aggregating the distance bucket dimension, for each algorithm and combination of delay level and grid size, we generate 50000 (out of which 5000 are used as warm-up) requests by randomly selecting a source and a destination node.

B. LDP: Influence of BD on an SP Search

We first observe the impact of BD on the runtime of LDP. This allows us to gain insight into the behavior of BD for SP runs (see Sec. III-C).

Fig. 4 shows the impact of BD on the runtime of LDP for the different delay levels, all the other dimensions being aggregated. As expected, we observe that BD allows to dramatically reduce (more than 6 times faster on average) the runtime of an SP search when the bound is lower than the cost of the shortest path to the destination (*infeasible* delay level, corresponding to the scenario described in Sec. III-C1). For all the other cases (the delay bound is greater than the cost of the shortest path to the destination – see Sec. III-C2), we however observe that, *on average*, the additional runtime induced by BD for checking if the bound is violated (line 14 in Fig. 2) is not compensated by its benefit. Indeed, we observe that the runtime ratios are, on average, slightly lower than 1.

However, interestingly, even if the provided bound is greater than the cost of the shortest path to the destination, there are cases for which BD reduces runtime (this is due to the fact that BD then avoids to place unnecessary elements in the priority queue – see. Sec. III-C2). Fig. 5 shows the impact of BD on the runtime of LDP for the different distance buckets, the grid size dimension being aggregated and for the first feasible delay level. The figure confirms that BD can also improve the runtime of an SP search when the provided bound is greater than the shortest path to the destination (see Sec. III-C2) but
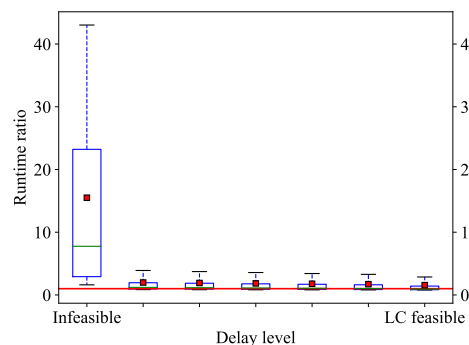

Fig. 6: Runtime ratios of IAK for different delay levels. For SPT searches, BD is beneficial in any case but better when the provided bound is lower.

that the benefit of BD only balances its additional overhead for low distances and tight bounds. Fig. 4 however shows that, *on average*, BD is only beneficial when the provided bound is lower than the cost of the shortest path to the destination.

As expected in Sec. V-A1, Fig. 5 also shows that, when the provided bound is greater than the shortest path to the destination, the impact of BD decreases as the distance between the nodes compared to the topology size increases. When the provided bound is lower than the shortest path to the destination, this effect is compensated by the fact that, when the distance is low, Dijkstra will anyway terminate before BD can stop it, thereby preventing BD from significantly reducing the search space. Hence, in this case, the impact of BD is relatively stable along the different distance buckets. This can be seen on the additional graphs available online [11].

C. IAK: Influence of BD on an SPT Search

We observe the impact of BD on the runtime of IAK. As IAK simply runs a least-cost SP search (without BD) and a least-delay SPT search with BD, this allows us to gain insight into the behavior of BD for SPT runs.

Fig. 6 shows the impact of BD on the runtime of IAK for the different delay levels, the other dimensions being aggregated. In comparison to Fig. 4, this shows that BD has a much higher impact for SPT runs (up to 15 times faster on average for infeasible cases, against 6 times faster for SP
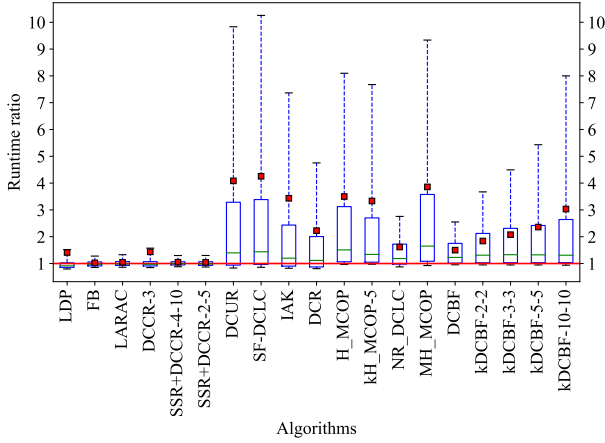
Fig. 7: Distributions of the runtime ratios of the different algorithms, all the dimensions being aggregated. We observe that BD can greatly reduce the runtime of some algorithms (up to 4 times faster, i.e., runtime reduced by up to 75% on average for some algorithms).
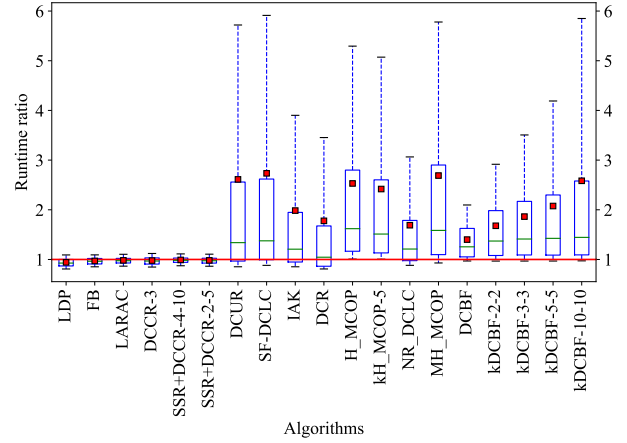

Fig. 8: Distributions of the runtime ratios of the different algorithms for the first feasible delay level, the other dimensions being aggregated. We observe that, even outside of the infeasible case, BD can provide significant benefit to some algorithms (around 3 times faster, i.e., runtime reduced by up to 66% on average for some algorithms).

searches). Further, we observe that, even when the problem is feasible, BD still provides benefit. This is because, even if the destination is closer than the provided bound, other nodes further away may be neglected by BD. Hence, this shows that, on average, BD is useful in any case for SPT searches. As expected, we observe that the impact of BD decreases as the delay bound gets looser.

In further evaluations available online [11], we have observed that, as for SP runs, the impact of BD on SPT searches decreases as the distance between the source and destination nodes of the original CSP problem increases.

D. BD Impact on All CSP Algorithms

In this section, we observe the impact of BD on all the CSP algorithms presented in Sec. IV. Algorithms requiring parameters have been configured as in [2]. In the plots, the parameters values are appended to the algorithm names.

Because too slow, SCRC, LARACGC, SMS-CDP, SMS-RDM, SMS-PBO and DEB were not able to run the evaluation in a reasonable amount of time. However, because of their similarity with LARAC, the impact of BD on LARACGC and SCRC is supposed to be similar to the impact on LARAC.

Fig. 7 shows the runtime ratios of all the algorithms, all the dimensions being aggregated. As can be seen, BD is, on average, beneficial for all the algorithms. However, we can see that algorithms which can only use BD for SP runs (LDP, FB, LARAC, DCCR and SSR+DCCR – Sec. IV-B) are only slightly improved. This was expected based on our observations of Sec. V-B. Indeed, the average impact of BD on SP runs is marginal. On the other hand, we can see that the runtime of algorithms which can use BD for SPT runs (DCUR, SF-DCLC, IAK, DCR, H_MCOP, kH_MCOP, NR_DCLC, MH_MCOP, DCBF and kDCBF – Sec. IV-C) can be dramatically improved by BD. For example, on average, DCUR, SF-DCLC and MH_MCOP are 4 times faster with BD, i.e., their runtime is reduced by 75% with BD. Also, most algorithms see their runtime improved by at least 20% in 50%

of the cases. DCUR, SF-DCLC, IAK and DCR present an interesting behavior. The two algorithms benefiting the most from BD are DCUR and SF-DCLC, because both their least-cost and least-delay SPT runs can use BD. Then, IAK benefits less because its least-cost SP run cannot benefit from BD. Finally, DCR benefits less than IAK even though its least-delay SP run and its least-cost SPT run can both benefit from BD. This shows that, while SPT runs benefit more from BD than SP runs, least-delay SPT runs benefit more from BD than least-cost SPT runs. MH_MCOP further shows that using BD for SPT runs based on a combination of the cost and delay metrics can provide as much benefit as for SPT runs based on the delay metric solely.

In order to highlight that BD can also provide significant benefit for feasible delay bounds, Fig. 8 shows the distributions of the runtime ratios of the different algorithms for the first feasible delay level, the other dimensions being aggregated. We can see that BD can still drastically reduce the runtime of some algorithms when the delay bound is feasible. For example, DCUR, SF-DCLC, H_MCOP, MH_MCOP and kDCBF-10-10 are, in this case, on average around 3 times faster with BD, i.e., their runtime is reduced by 66%. In further evaluations available online [11], by aggregating all dimensions except the distance between the source and destination nodes, we have observed that, when the problem is not infeasible, BD is beneficial to the algorithms as long as the relative distance stays below 40-50% of the topology size. When the nodes are further apart from each other, the graph boundary stops the expansion of Dijkstra before BD can have any significant impact.

We have seen that BD has potentially more impact when the delay constraint is tighter and the distance between the source and destination nodes is smaller. Fig. 9 shows the runtime ratios of all the algorithms for the infeasible delay level and for the 0-10% distance bucket, the grid size dimension being aggregated. We observe that, in this favorable case, BD
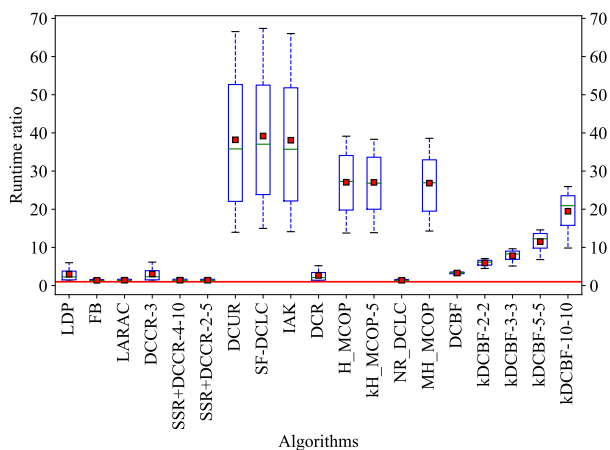
Fig. 9: Distributions of the runtime ratios of the different algorithms for favorable cases (infeasible delay constraint and the 0-10% distance bucket). We observe that, in these favorable cases, BD can drastically reduce the runtime of some algorithms (more than 25 times faster, i.e., runtime reduced by at least 96% on average for some algorithms).

allows to drastically reduce the runtime of all the algorithms, including those only using BD for SP runs. For example, SF-DCLC, IAK, DCUR, H_MCOP, kH_MCOP and MH_MCOP are more than 25 times faster with BD, i.e., they see their runtime reduced by more than 96% on average. Interestingly, DCR and NR_DCLC, which have a good average runtime improvement (see Fig. 7), do not benefit much in this favorable case. This is because, in the infeasible case, both algorithms only run SP searches, thereby having a benefit similar to the algorithms only using BD for SP searches (e.g., LDP and LARAC).

VI. Conclusions

Shortest path (SP) and *shortest paths tree* (SPT) algorithms are often used as subroutine of overlay algorithms solving more complex problems (e.g., the *(multi-)constrained shortest path* (CSP and MCSP), the *multi-constrained path* (MCP), and the *constrained minimum Steiner tree* (CMST) problems). In such a situation, it often happens that the result of an SP subroutine is not used if its total cost is greater than a given bound. Because Dijkstra discovers path in increasing order of cost, we can terminate the execution of Dijkstra as soon as it reaches paths which have a cost greater than the known bound. We refer to this adaptation of Dijkstra as *bounded Dijkstra* (BD). By terminating Dijkstra earlier, its search space is reduced, thereby reducing its runtime and hence the runtime of the overlay algorithm using it. BD can be used by any routing algorithm making use of an underlying SP/SPT algorithm and that can provide a bound to this algorithm. We evaluated the impact of BD on the specific example of CSP algorithms. We have shown that BD does not impact the output of the algorithms but can dramatically decrease their runtime. While BD can be beneficial for both SP and SPT searches, we showed that its benefit is greater for SPT runs. The runtime of some algorithms is reduced by 75% on average. We further showed that BD is more efficient for tight delay constraints and when

the source and destination nodes of the CSP problem are close to each other compared to the size of the topology. For these favorable cases, several algorithms see their runtime reduced by 96% on average (i.e., BD allows them to be more than 25 times faster).

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[2] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation," *IEEE Communications Surveys & Tutorials*, 2017.

[3] J. W. Guck, A. Van Bemten, and W. Kellerer, "DetServ: Network models for real-time QoS provisioning in SDN-based industrial environments," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1003–1017, 2017.

[4] A. Van Bemten, J. W. Guck, P. Vizarreta, C. M. Machuca, and W. Kellerer, "LARAC-SN and Mole in the Hole: Enabling routing through service function chains," in *4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 298–302.

[5] S. Zoppi, A. Van Bemten, H. M. Gürsu, M. Vilgelm, J. Guck, and W. Kellerer, "Achieving hybrid wired/wireless industrial networks with WDetServ: Reliability-based scheduling for delay guarantees," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2307–2319, 2018.

[6] H. F. Salama, D. S. Reeves, and Y. Viniotis, "A distributed algorithm for delay-constrained unicast routing," in *Proc. IEEE INFOCOM*, vol. 1, 1997, pp. 84–91.

[7] D. S. Reeves and H. F. Salama, "A distributed algorithm for delay-constrained unicast routing," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 239–250, Apr. 2000.

[8] A. Van Bemten, J. W. Guck, C. M. Machuca, and W. Kellerer, "Routing metrics depending on previous edges: The Mn taxonomy and its corresponding solutions," in *IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–7.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[10] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation," *Mathematical Programming*, vol. 73, no. 2, pp. 129–174, May 1996.

[11] LORA: The League of Routing Algorithms. [Online]. Available: https://lora.lkn.ei.tum.de

[12] L. Fu, D. Sun, and L. R. Rilett, "Heuristic shortest path algorithms for transportation applications: State of the art," *Computers & Operations Research*, vol. 33, no. 11, pp. 3324–3343, Nov. 2006.

[13] L. Kleinrock and F. Kamoun, "Hierarchical routing for large networks performance evaluation and optimization," *Computer Networks (1976)*, vol. 1, no. 3, pp. 155 158–174, 1977.

[14] G. R. Jagadeesh, T. Srikanthan, and K. Quek, "Heuristic techniques for accelerating hierarchical routing on road networks," *IEEE Transactions on intelligent transportation systems*, vol. 3, no. 4, pp. 301–309, 2002.

[15] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[16] B. V. Cherkassky, A. V. Goldberg, and C. Silverstein, "Buckets, heaps, lists, and monotone priority queues," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1326–1346, 1999.

[17] M. Thorup, "Integer priority queues with decrease key in constant time and the single source shortest paths problem," in *Proceedings of the 35th annual ACM Symposium on Theory of Computing*. ACM, 2003, pp. 149–158.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[19] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *Proceedings of the 16th annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2005, pp. 156–165.

[20] J. Y. Yen, "An algorithm for finding shortest routes from all source nodes to a given destination in general networks," *Quarterly of Applied Mathematics*, vol. 27, no. 4, pp. 526–530, Jan. 1970.

[21] M. J. Bannister and D. Eppstein, "Randomized speedup of the Bellman-Ford algorithm," in *Proc. of the SIAM Meeting on Analytic Algorithmics and Combinatorics*, 2012, pp. 41–47.

[22] J. B. Kwa, "BS*: An admissible bidirectional staged heuristic search algorithm," *Artificial Intelligence*, vol. 38, no. 1, pp. 95–109, 1989.

[23] T. Ikeda, M.-Y. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh, "A fast algorithm for finding better routes by ai search techniques," in *Vehicle Navigation and Information Systems Conference, 1994. Proceedings., 1994*. IEEE, 1994, pp. 291–296.

[24] M. N. Rice and V. J. Tsotras, "Bidirectional a* search with additive approximation bounds." in *SOCS*, 2012.

[25] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route planning in transportation networks," in *Algorithm Engineering*. Springer, 2016, pp. 19–80.

[26] M. Hilger and H. Schilling, "Fast point-to-point shortest path computations with arc-flags," *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, vol. 74, pp. 41–72.

[27] M. Holzer, F. Schulz, and D. Wagner, "Engineering multilevel overlay graphs for shortest-path queries," *Journal of Experimental Algorithmics (JEA)*, vol. 13, p. 5, 2009.

[28] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 349–360.

[29] I. Abraham, D. Delling, A. Goldberg, and R. Werneck, "Hierarchical hub labelings for shortest paths," *Algorithms–ESA 2012*, pp. 24–35, 2012.

[30] P. Bose, P. Carmi, M. Farshi, A. Maheshwari, and M. Smid, "Computing the greedy spanner in near-quadratic time," in *Scandinavian Workshop on Algorithm Theory*. Springer, 2008, pp. 390–401.

[31] A. Lingas and E.-M. Lundell, "Efficient approximation algorithms for shortest cycles in undirected graphs," *Information Processing Letters*, vol. 109, no. 10, pp. 493–498, 2009.

[32] M. Bourdellès and N. Menegale, "Routing optimization for network coding," in *2012 IFIP Wireless Days*. IEEE, 2012, pp. 1–6.

[33] M. N. Rice and V. J. Tsotras, "Parameterized algorithms for generalized traveling salesman problems in road networks," in *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2013, pp. 114–123.

[34] S. Varone, "On a many-to-one shortest paths for a taxi service," Haute école de gestion de Genève, Tech. Rep. HES-SO/HEG-GE/C14/1/1-CH, 2014.

[35] S. Varone and V. Janilionis, "Insertion heuristic for a dynamic dial-a-ride problem using geographical maps," in *10ème Conférence Francophone de Modélisation, Optimisation et Simulation (MOSIM)*, 2014.

[36] P. Simari, G. Picciau, and L. De Floriani, "Fast and scalable mesh superfacets," in *Computer Graphics Forum*, vol. 33, no. 7. Wiley Online Library, 2014, pp. 181–190.

[37] L. R. Ford Jr, "Network flow theory," DTIC Document, Tech. Rep., 1956.

[38] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, Apr. 1958.

[39] E. I. Chong, S. Maddila, and S. Morley, "On finding single-source single-destination k shortest paths," *J. Computing and Information, Special Issue ICCI*, vol. 95, pp. 40–47, 1995.

[40] T. Korkmaz and M. Krunz, "Multi-constrained optimal path selection," in *Proc. IEEE INFOCOM*, vol. 2, 2001, pp. 834–843.

[41] Q. Zhu, M. Parsa, and J. Garcia-Luna-Aceves, "A source-based algorithm for delay-constrained minimum-cost multicasting," in *INFOCOM. Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 1, 1995, pp. 377–385.

[42] R. Widyono, "The design and evaluation of routing algorithms for real-time channels," International Computer Science Institute Berkeley, Tech. Rep. TR-94-024, 1994.

[43] G. Liu and K. Ramakrishnan, "A*Prune: An algorithm for finding k shortest paths subject to multiple constraints," in *Proc. IEEE INFOCOM*, vol. 2, 2001, pp. 743–749.

[44] G. Feng, K. Makki, N. Pissinou, and C. Douligeris, "Heuristic and exact algorithms for QoS routing with multiple constraints," *IEICE Trans. on Commun.*, vol. 85, no. 12, pp. 2838–2850, 2002.

[45] R. Sriram, G. Manimaran, and C. S. R. Murthy, "Preferred link based delay-constrained least-cost routing in wide area networks," *Computer Communications*, vol. 21, no. 18, pp. 1655–1669, Dec. 1998.

[46] Z. Jia and P. Varaiya, "Heuristic methods for delay-constrained least-cost routing problem using k-shortest-path algorithms," in *Proc. IEEE INFOCOM*, 2001, pp. 1–9.

[47] W. C. Lee, M. G. Hluchyi, and P. A. Humblet, "Routing subject to quality of service constraints in integrated communication networks," *IEEE Network*, vol. 9, no. 4, pp. 46–55, Jul./Aug. 1995.

[48] Y. P. Aneja and K. Nair, "The constrained shortest path problem," *Naval Research Logistics Quarterly*, vol. 25, no. 3, pp. 549–555, Sep. 1978.

[49] G. Y. Handler and I. Zang, "A dual algorithm for the constrained shortest path problem," *Networks*, vol. 10, no. 4, pp. 293–309, Winter 1980.

[50] D. Blokh and G. Gutin, "An approximate algorithm for combinatorial optimization problems with two parameters," *Australasian Journal of Combinatorics*, vol. 14, pp. 157–164, 1996.

[51] A. Jüttner, B. Szviatovski, I. Mécs, and Z. Rajkó, "Lagrange relaxation based method for the QoS routing problem," in *Proc. IEEE INFOCOM*, vol. 2, 2001, pp. 859–868.

[52] L. Santos, J. Coutinho-Rodrigues, and J. R. Current, "An improved solution algorithm for the constrained shortest path problem," *Transportation Research Part B: Methodological*, vol. 41, no. 7, pp. 756–771, Aug. 2007.

[53] L. Guo and I. Matta, "Search space reduction in QoS routing," *Computer Networks*, vol. 41, no. 1, pp. 73–88, Jan. 2003.

[54] W. Liu, W. Lou, and Y. Fang, "An efficient quality of service routing algorithm for delay-sensitive applications," *Computer Networks*, vol. 47, no. 1, pp. 87–104, Jan. 2005.

[55] K. Ishida, K. Amano, and N. Kannari, "A delay-constrained least-cost path routing protocol and the synthesis method," in *Proc. IEEE Int. Conf. on Real-Time Computing Systems and Applications*, 1998, pp. 58–65.

[56] Q. Sun and H. Langendörfer, "A new distributed routing algorithm for supporting delay-sensitive applications," *Computer Communications*, vol. 21, no. 6, pp. 572–578, May 1998.

[57] G. Feng, C. Douligeris, K. Makki, and N. Pissinou, "Performance evaluation of delay-constrained least-cost QoS routing algorithms based on linear and nonlinear lagrange relaxation," in *Proc. IEEE Int. Conf. on Communications (ICC)*, vol. 4, 2002, pp. 2273–2278.

[58] G. Cheng and N. Ansari, "A new heuristics for finding the delay constrained least cost path," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM)*, vol. 7, 2003, pp. 3711–3715.