Shorter Path Constraints for the Resource Constrained Shortest Path Problem

Thorsten Gellermann
University of Paderborn
Computer Science
Fuerstenallee 11
33098 Paderborn

Meinolf Sellmann
Brown University
Computer Science
115 Waterman St
Providence, RI 02912

Robert Wright
Air Force Research Lab
Inform. Directorate
525 Brooks Road
Rome, NY 13441

Abstract. Recently, new cost-based filtering algorithms for shorter-path constraints have been developed. However, so far only the theoretical properties of shorter-path constraint filtering have been studied. We provide the first extensive experimental evaluation of the new algorithms in the context of the resource constrained shortest path problem. We show how reasoning about path-substructures in combination with CP-based Lagrangian relaxation can help to improve significantly over previously developed problem-tailored filtering algorithms and investigate the impact of required-edge detection, undirected versus directed filtering, and the choice of the algorithm optimizing the Lagrangian dual.

1 Introduction

Path constraints play a key role in many applications. Examples range from airline crew scheduling [8, 14] and vehicle routing [20] to the traveling salesman [2] and the resource constrained shortest path problem [1, 4, 6, 7, 12]. Of special interest in the context of combinatorial optimization are path constraints that incorporate the objective function. Shorter path constraints do exactly that by stating that a set of binary variables that are semantically linked to the edges of a graph must form a path from some designated source to a designated sink, whereby the length of this path must not exceed a given threshold value. Unfortunately, in [17] it was shown that achieving generalized arc-consistency for shorter path constraints is NP-hard. Consequently, filtering algorithms were developed that achieve weaker, so called *relaxed consistency* in the same time that it takes to solve the shortest path problem itself. This work was purely theoretical, though. Therefore, we consider it of interest to evaluate the performance of these filtering algorithms in practical experiments.

For this purpose, we focus on the resource constrained shortest path problem (RCSPP) that consists in finding a shortest route from some given source to a designated sink such that some given resources that are consumed while traversing the edges are not exhausted. While the RCSPP is of interest in itself, for example in the context of traffic guiding systems and route planners for cars and trucks, the problem also evolves as a natural subproblem in the context of even more complex problems like vehicle routing [20].

Based on the new filtering algorithms presented in [17], we provide an evaluation of relaxed consistency for shorter path constraints in the context of the resource constrained shortest problem. In the following section, we briefly review the filtering algorithms developed in [17]. In Section 3 we define the resource constrained shortest path problem formally and present a filtering approach that considerably outperforms previous filtering algorithms for this problem, as we will then see in Section 4.

2 Relaxed Consistency for Shorter Path Constraints

Before we review the filtering algorithms that we are going to use for the RCSPP later, let us start out by defining what shorter path constraints are. In words, they express our wish to search for paths in a (directed or undirected) graph such that the length is smaller than some given threshold value. Formally, we define:

Definition 1. *Denote with $G = (V, E, c)$ a weighted (directed or undirected) graph with $||c||_\infty \in O(\text{poly}(|E|, |V|))^1$, and let $h \in \mathbb{N}$.*

- *A sequence of nodes $P = (i_1, \ldots, i_h) \in V^h$ with $(i_f, i_{f+1}) \in E$ for all $1 \leq f < h$ is called a* path from i_1 to i_h in G.
- *A path P is called* simple *iff P visits every node at most once. For all $i, j \in V$, denote with $\pi(i, j)$ the set of all simple paths from i to j.*
- *For all paths P, nodes $i \in V$ and edges $(i, j) \in E$, we write $i \in P$ or $(i, j) \in P$ iff P visits node i or the edge (i, j), respectively. For a set of nodes or edges S, we write $S \subseteq P$, iff $s \in P$ for all $s \in S$. Correspondingly, we write $P \subseteq S$ iff $s \in S$ for all $s \in P$.*
- *The cost of a path $P = (i_1, \ldots, i_h)$ is defined as $cost(P) := \sum_{1 \leq j < h} c_{i_j i_{j+1}}$. Accordingly, for any set $S \subseteq E$ we define $cost(S) := \sum_{(i,j) \in S} c_{ij}$.*

Definition 2. *Let $G = (V, E, c)$ denote a (directed or undirected) graph with $n = |V|$ and $m = |E|$, a designated source $v_1 \in V$ and sink $v_n \in V$, and arc costs $c_{ij} \in \mathbb{Z}$. Further, assume we are given binary variables $X_1, \ldots, X_m$, and an objective bound $B \in \mathbb{Z}$.*

- *A constraint $SPC(X_1, \ldots, X_m, G, v_1, v_n, B)$ that is true, iff*
  1. *the set $\{e_i \mid X_i = 1\} \subseteq E$ determines a simple path in the graph G from the source v_1 to the sink v_n, and*
  2. *the cost of the path defined by the instantiation of X is lower than B*

  is called a shorter path constraint.
- *We call every simple path in G from source to sink with costs less than B* admissible.
- *A path P is called a* k-simple path in G *iff for all $j \in V$ the path P visits j at most k times. Note that a 1-simple path is a simple path in G.*
- *Given a shorter path constraint, a k-simple path P from v_1 to v_n is called a k-*admissible path *iff $cost(P) < B$.*

As mentioned before, it can be shown that achieving generalized arc consistency for shorter path constraints is an NP-hard task. Therefore, in [17] filtering algorithms for shorter path constraints on arbitrary undirected and directed graphs were developed that achieve *relaxed consistency*. With that term we denote filtering algorithms for minimization constraints [9] that only guarantee that those variable assignments are identified that would cause a bound rather than the optimal solution in the current subtree itself to exceed the current best known upper bound.

The formal relaxations considered in [17] are very technical and do not give a particularly useful insight into the task of filtering shorter path constraints. Therefore, we do not to repeat them here but just outline the filtering algorithms that we will use later.

^{1} This is the common *similarity assumption* that states that the largest cost is bounded by some polynomial in $|E|$ and $|V|$.

1. On both directed and undirected graphs, the filtering algorithm starts with two shortest path computations once from the source and the other starting at the sink node whereby, in the directed case, the computation is performed in the reverse graph.
2. As a result, we get the shortest path value from source to sink. If this value exceeds the objective bound B, the constraint fails.
3. Otherwise, as a byproduct of the shortest path computations we get the shortest path distances from the source and to the sink of every node for free. We use this information to identify those nodes and arcs of the graph for which the shortest 2-simple path that visits them is above the threshold B. For the nodes, we get this value by adding the shortest path distance from the source and that to the sink, for edges, we add the weight of the edge to that value.

2.1 Exploiting Bridges in Undirected Graphs

After having shrunk the graph in step 3, as a last step of our filtering algorithm we try to identify those edges that must be visited by all paths having a length below the given threshold. This step will be different for undirected and directed graphs. In the undirected case, there exists a simple exact classification of the edges that must be visited. In [17], it was shown that the edges to be required are exactly the bridges[2] in the reduced graph that fall onto the shortest path:

4a. We compute the set of bridges in the reduced graph. The bridges that are also on the shortest path from source to sink must be visited by all admissible paths, and we mark them as required.

On top of this last step of the filtering procedure that was proposed in [17], we add one more idea: We observe that bridges that are not on the shortest path cannot be visited by any simple path from source to sink. Therefore, we can remove those bridges and the entire part of the graph behind them as well:

5a. Remove all bridges from the graph that are not on the shortest path.

2.2 Required Arcs in Directed Graphs

Unfortunately, we do not know a similar classification of required arcs as we have it for undirected graphs where the edges to be required are exactly the bridges in the reduced graph that we get after step 3. The algorithm in [17] tries to bound the shortest path distance when having to detour around an arc on the shortest path. When implementing this algorithm, we realized that actually we do not need to compute this bound after having reduced the graph in step 3 of the algorithm. While preserving the same filtering effectiveness of the original algorithm, we can save the overhead of using a heap data structure, because it is completely sufficient to know whether such a detour still exists; since the arc that we use in our detour has not been deleted, we know already that the value of the detour will not exceed the given path-length threshold.

In order to state the last step of our filtering algorithm for directed graphs, we briefly repeat some of the terminology introduced in [17]. Let $T \subseteq E$ denote a shortest-path

[2] A bridge is an edge whose removal disconnects the graph.

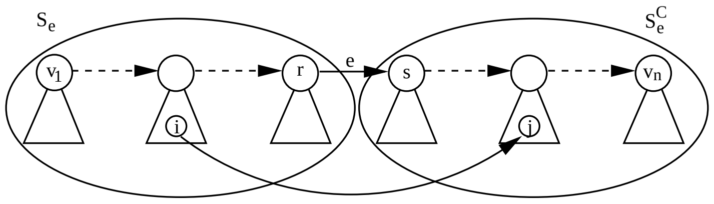

Fig. 1. The figure schematically shows a shortest-path tree T rooted at v_1. Solid lines denote arcs in G, dashed lines mark parts of the shortest path $P(v_1, v_n)$ from v_1 to v_n. The triangles symbolize shortest-path subtrees. For an edge $e = (r, s) \in P(v_1, v_n)$, the nodes in V are partitioned into two non-empty sets S_e and S_e^C. If e is removed from the graph, the shortest path from v_1 to v_n must visit an edge $(i, j) \in (S_e \times S_e^C) \setminus T$.

tree in G rooted at v_1. Without loss of generality, we may assume that every node in the graph $G = (V, E)$ can be reached from the source node v_1. Obviously, when $e \in E$ is removed from T, the nodes in V are partitioned into two sets: the set $v_1 \in S_e \subset V$ of nodes that are still connected with v_1 in $T \setminus \{e\}$, and the complement of S_e in V, S_e^C (see Fig. 1). Using these naming conventions, the last step of our filtering algorithm for directed graphs reads:

4b. Denote with E^R the reduced arc set after step 3. For all arcs $e = (r, s)$ on the shortest path from v_1 to v_n, check whether there exists any other arcs $e \neq f = (i, j) \in (S_e \times S_e^C) \cap E^R$. If not, then e must be visited by all paths from source to sink and it is therefore required.

Note that this last step can be implemented with the help of a simple set data structure for an asymptotic cost of $O(m + n)$.

3 A Filtering Approach for Resource Constrained Shortest Paths

In order to evaluate the filtering algorithms for directed and undirected graphs as described in the previous section, we apply them in the context of resource constrained shortest path:

Definition 3. *Given a (directed or undirected) graph $G = (V, E)$, $n = |V|$, $m = |E|$, with $R+1$ edge-weight functions $l^k : E \to \mathbb{N}$, $0 \leq k \leq R$, R resource limits $L^1, ..L^R$, and two designated source- and sink-nodes $v_1, v_n \in V$, the* resource constrained shortest path problem (RCSPP) *consists in the computation of a path $P \subseteq E$ such that $\sum_{e \in P} l_e^k \leq L^k$ for all $1 \leq k \leq R$ and $\sum_{e \in P} l_e^0$ is minimal.*

When we denote the best known solution value found with B and set $L^0 := B$, any RCSPP-instance can be modeled as a conjunction of $R+1$ shorter path constraints $SPC(X_1, \ldots, X_m, (V, E, l^k), v_1, v_n, L^k)$ for $0 \leq k \leq R$.

Of course, we could use these constraints to perform an ordinary tree search. However, for the RCSPP it was found that tree search approaches perform rather poorly.

Instead, to solve the RCSPP, state-of-the-art solvers compute lower and upper bounds on the problem first and then close the duality gap. The latter task is carried out by an enumeration procedure such as dynamic programming [15] or labeling approaches [6]. The tightening of the initial problem is vital for an effective gap closing procedure and is therefore essential for the overall performance and the practical success of the entire approach.

Following this framework, we assume that an initial upper bound B has been computed before the filtering phase that will, as a byproduct, also provide a lower bound on the problem. Instead of having the shorter path constraints communicate via variable domains only, we use the CP-based Lagrangian relaxation framework as published in [16, 18, 19]. Precisely, we relax all linear constraints $\sum_{1\leq i\leq m} l_i^k X_i \leq L^k$, $1 \leq k \leq R$, and penalize their violation in the objective function. Given any vector of Lagrangian multipliers $0 \leq \lambda \in \mathbb{Q}^R$, we consider the constraint

$$SPC(X_1, \dots, X_m, (V, E, l^0 + \sum_{1\leq k\leq R} \lambda_k l^k), v_1, v_n, L^0 + \sum_{1\leq k\leq R} \lambda_k L^k).$$

As usual, the question arises how to compute good Lagrangian multipliers that will yield a good lower bound on the problem and allow us to filter effectively. In general, we can use any subgradient, bundle, or volume algorithm for this purpose [3, 5, 10, 11]. Since most benchmark sets for the RCSPP contain only one resource (i.e., $R = 1$), we use a specialized algorithm for the optimization of the Lagrangian dual with only one multiplier.

3.1 Maximizing One-Parameter Piecewise Linear Concave Functions

A schematic view on the Lagrangian dual for RCSPP-instances with $R = 1$ is given in Figure 2. Assume that we know an interval $[A, B]$ in which the function (let us denote it with f) must take its maximum.

Interval Partitioning One way to find the function's maximum in the given interval is to trisect the interval by introducing two interior points $A < X < Y < B$. When we evaluate the function at X and Y and find that $f(X) > f(Y)$ ($f(X) < f(Y)$), due to its concaveness we can deduce that f must take its maximum in the interval $[A, Y]$ ($[X, B]$). Thus, we have found a smaller interval in which the function must take its maximum. We can repeat this process until the width of the interval has become small enough. Of course, in every iteration we could choose new interior points in the current interval. However, in order to save some evaluations of the function f we should try to reuse one of the former inner points. If we partition the current interval according to the golden section, we know that the interval length will decrease geometrically and that one inner point can always be reused, which means that we need to perform only one evaluation of f in each iteration. The procedure is sketched graphically in Figure 2.

For the optimization of the Lagrangian dual this means that we can ϵ-approximate the best Lagrangian multiplier λ in $O(\log \frac{L}{\epsilon})$ iterations, whereby L denotes the width of the initial interval. Each iteration involves the solution of only one Lagrangian subproblem. In the context of the RCSPP, the subproblem is a shortest-path problem. Moreover, assuming that there exists a path that obeys the resource restriction (i.e., when a primal

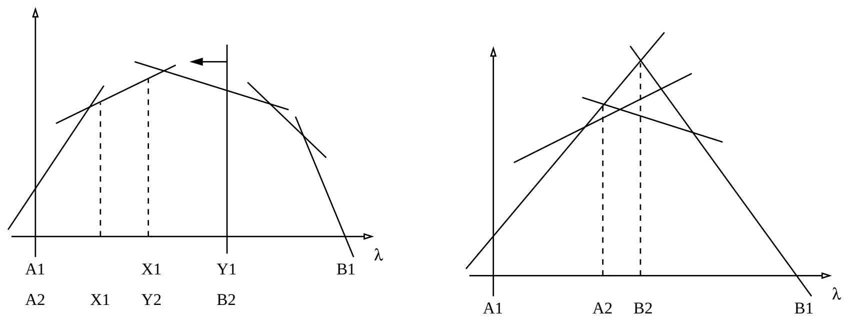

Fig. 2. Maximizing a piece-wise linear concave function by interval partitioning (left) or by cutting planes (right).

solution exists at all and consequently the dual is not unbounded), it is easy to show that the optimal Lagrangian multiplier cannot be greater than $n||l^0||_\infty$. Consequently, we can solve the Lagrangian dual in time $O(\log \frac{n||l^0||_\infty}{\epsilon}(m + n \log n))$.

With this method, the proposed filtering algorithm for the RCSPP with one resource works as follows: We choose an initial interval $[0, n||l^0||_\infty]$. Denote with L the length of the current interval with left end-point l. Then, we solve the Lagrangian shortest-path subproblem for interior points $X = l + \frac{\sqrt{5}-1}{2}L$ and $Y = l + \frac{\sqrt{5}+1}{2}L$. While solving the shortest-path problem, we also apply our filtering algorithm as described in the previous section. Depending on the point that achieves a larger shortest-path value, we cut off either the right or the left part of the interval and proceed by solving one more Lagrangian dual and filtering in each successive iteration of the algorithm.

Cutting Planes Another way of computing the function's maximum in the given interval is to use a cutting plane algorithm [13]. Clearly, if the function has a negative slope on the left end point of the interval, then this is the point where the function takes its maximum in the given interval. Analogous reasoning holds for the right end point of the interval if the slope there is positive. Now, if the left end point has a positive slope and the right end point a negative one, then the two lines intersect for some point in the middle of the interval. We evaluate the function at that point and check whether the slope is positive or negative (if it is horizontal then we are obviously done). If it is positive (negative), then this point becomes the new left (right) end point of our search interval, and we continue until the computed inner point does not change anymore.

With this method, the proposed filtering algorithm for the RCSPP with one resource works as follows: We choose an initial interval $[0, n||l^0||_\infty]$. We solve the Lagrangian subproblem at the two end points and perform cost-based filtering. Then we repeatedly intersect the slopes at the end points of our interval to determine inner points for which we process the Lagrangian subproblem again. Depending on whether the solution to the subproblem exceeds the resource limit or not, the corresponding Lagrange multiplier becomes the new left or right end point of our search interval (see Figure 2).

Note that our filtering routine is actually changing the problem while we are solving the Lagrangian dual. In general, this is problematic since really both algorithms designed for maximizing concave functions over convex polytopes are not designed to cope with changes in the problem during the optimization. Instead, one could just mark the changes to be made. However, we found that both our algorithms for maximizing the Lagrangian dual were very robust and yielded good results even when incorporating the changes “on the fly”. As a matter of fact, this was very beneficial since the successive calls to the shortest path algorithm become cheaper and cheaper, since the graph size reduces considerably during filtering, as we will see in the following section.

4 Numerical Results

We have outlined the shorter-path filtering algorithms and described how CP-based Lagrangian relaxation can be applied for two-shorter-path-constraint problems. The latter correspond to resource constrained shortest path problems (RCSPPs). We have chosen to base our experimentation on the RCSPP for various reasons. Of course, when evaluating the practical efficiency of shorter-path constraint filtering, we would like to eliminate all possible side effects caused by other constraints of the problem under consideration. Therefore, the purest evaluation would be to consider the shortest path problem. However, since all filtering algorithms for shorter-path constraints are actually based on efficient shortest-path algorithms, this is not a feasible choice.

Note also that the application of filtering algorithms usually only makes sense for NP-hard problems. So the natural idea is to consider a problem that consists in the conjunction of two shorter-path constraints, which corresponds to the search of improving solutions for the RCSPP with one resource. This problem is NP-hard and filtering methods for it have been studied a long time before the idea of constraint programming was developed [1, 4]. Therefore, it is of particular interest to investigate how CP performs in comparison with those problem-tailored filtering algorithms.

This being said, it is important to note here that we do not aim at providing a complete state-of-the-art algorithm for the RCSPP itself. Our goal is instead to evaluate the practical performance of shorter-path constraint filtering, and the RCSPP appears as a very reasonable benchmark for such an evaluation. There exist very efficient algorithms for the optimization of the RCSPP [1, 4, 6, 7, 12]. Most of them incorporate a filtering component, but it could be interweaved with the specific algorithm. Note also that an upper bound is required to perform filtering. Now, in order to avoid that we are actually measuring the performance of an upper bounding procedure and not the quality of shorter-path filtering, we do not provide a primal heuristic for the RCSPP. Instead, we base our experimentation on upper bounds of predefined and controlled accuracy, so that we are able to evaluate the performance of the existing and the new filtering algorithms when the quality of the primal heuristic varies.

Thus, when interpreting the following experimental results, keep in mind that shorter-path filtering is just one component in an RCSPP solver, and that we do not provide a complete solver for this problem here. Especially, we do not provide algorithms for the computation of good upper bounds.

	Austria S	Austria B	Scotland S	Scotland B	Road S	Road B
Undirected	46160	165584	65024	252432	50826	171536
LSA	7.589	252.590	7.784	-	0.651	3.472
MZ	0.492	2.186	0.891	12.378	0.791	3.273
Directed	46160	165584	65024	252432	50826	171536
LSA	6.489	257.361	7.581	-	0.651	3.461
MZ	0.474	2.339	2.914	11.053	0.792	3.283
	Curve 1	Curve 2	Curve 3	Curve 4	Curve 5	Curve 6
Undirected	19890	39580	99890	199580	199890	399580
LSA	5.607	18.603	155.749	-	-	-
MZ	0.055	1.436	2.438	0.596	0.797	-
Directed	9945	19790	49945	99790	99945	199790
LSA	3.537	12.286	29.939	-	-	-
MZ	0.039	1.128	2.078	0.394	0.594	183.034

Table 1. The table shows the initial number of edges in the undirected and directed versions of the test files and the time needed by LSA and MZ to solve them. A '-' indicates a solver was unable to compute a solution due to exhaustive memory consumption.

4.1 Overview of Experiments and Benchmark Sets Used

In our experiments, we run tests to determine under which parameters our algorithms, that combine shorter-path filtering with CP-based Lagrangian relaxation (SPFCP), perform best. The performance of the algorithms is measured by the number of edges filtered and the CPU time taken. We seek to answer the following questions. Is there any advantage towards using the undirected version (marked by SPFCP-U) over the directed version (SPFCP-D) of our filtering method? Does using required-arc (-RE) and bridge detection (-BD) as part of the filtering have any benefit? And, which method for optimizing the Lagrangian dual is better, interval partitioning (-IP) or cutting plane (-CP)? Finally, we add a comparison of the SPFCP algorithm with two existing filtering algorithms when used for the RCSPP.

For the optimization of the RCSPP after the initial filtering phase, we use our own implementation of a standard RCSPP label setting algorithm (LSA) or our implementation of the RCSPP algorithm by Mehlhorn and Ziegelmann (MZ) [15]. The experiments measure CPU time in seconds and were performed on an Intel Pentium 4 2.5GHz, 1Gb RAM machine running Red Hat Linux 9. The filtering programs, LSA, and MZ were compiled using gcc version 3.2.2 with the optimizing flag.

We use the RCSPP benchmark files provided by Mehlhorn and Ziegelmann [15] [3]. All input graphs specify a designated source and a sink, edge cost and resource, and a resource limit. We use two variants of the benchmark files: the original directed files and converted undirected versions. The latter were generated by viewing the arcs as undirected and flipping a coin in case of multi-edges. Files that were generated in that way are marked with an extra '*'. Note that an undirected graph can be viewed as a bi-directed graph where resource and cost coefficient for all edges are the same in

[3] Data files are available at http://www.mpi-sb.mpg.de/˜mark/cnop/.

both directions. This interpretation allows us to use the directed version of our filtering algorithm on this benchmark set as well, so that we can compare the undirected and the directed filtering variants on this benchmark set. Table 1 shows information on the size of the graphs as well as the time needed to solve them using MZ and LSA. The following is a description of the types of RCSPP problems the input graphs represent.

Digital elevation models (DEM): These graphs are grid graphs representative of elevation data over areas of Austria and Scotland. The problem is to find the path with the minimum total height difference while satisfying a constraint on distance.

Road graphs: This benchmark set contains US road graphs. Edges in these graphs are weighted by distance and congestion. The problem is to find the route that takes minimal time while satisfying constraints on fuel consumption.

Curve approximation: In some applications, such as computer graphics programs, it is necessary to represent infinitely detailed curves with less complex functions. In this benchmark set, curves are estimated by many straight lines/edges joined at breakpoints/nodes which lay on the original curve. It is desirable to reduce the number of breakpoints used to estimate the curve while satisfying a constraint on the amount of error introduced. Modeled as an RCSPP, solutions to these instances minimize the number of sampling points when approximating a curve by a piecewise linear function.

4.2 Undirected and Directed SPFCP

In Section 2, we proposed two implementations of the SPFCP algorithm, one that filters on directed graphs and one that filters on undirected graphs. We explained how the undirected version has the advantage of being able to reason via the detection of bridges. We now want to compare the two variants by using the bi-directed benchmark set. We varied the upper bound on the objective from optimal to +5% optimal to examine how the performance of the SPFCP algorithms degrade. Table 2 shows the results of the comparison using both the directed and undirected versions of SPFCP with required-arc and bridge detection used.

When comparing the raw numbers, the directed version is capable of filtering more edges than the undirected version on the same graphs. However, on most of the tests where the algorithms were given an optimal bound on the objective the filtered graphs from the directed algorithm has exactly half as many edges as the graphs from the undirected algorithm. This is because the undirected algorithm must meet the constraint of leaving a bi-directed graph after filtering whereas the directed version does not. So, the filtered graphs from both algorithms when given an optimal bound on the objective are relatively the same, the directed version just additionally filters out return edges on the shortest path. The undirected version runs faster though, by 53% on average when given an optimal upper bound. When the value of the upper bound is 5% above the optimal value, the directed version filters on average 20% more edges than the undirected. However, the undirected version is still 45% faster on average.

In general, the time taken by the SPFCP algorithms to perform the filtering increases as the quality of the upper bound decreases. This phenomenon can easily be explained in that successive iterations of the filtering algorithm during the optimization of the Lagrangian dual require more time when previous iterations were not as effective at removing edges. While the undirected version works twice as fast as the directed variant

	Optimal				+5%			
Graph	SPFCP-D-RE		SPFCP-U-BD		SPFCP-D-RE		SPFCP-U-BD	
	# Edges	Time	# Edges	Time	# Edges	Time	# Edges	Time
Austria Small*	213	0.367	426	0.153	3667	0.443	6882	0.260
Austria Big*	436	1.650	872	1.017	8903	1.860	14742	1.243
Scotland Small*	652	0.547	1304	0.287	6879	0.613	11258	0.347
Scotland Big*	494	2.793	988	1.927	24155	3.170	30584	2.377
Road Small*	899	0.610	1596	0.320	1559	0.627	2180	0.340
Road Big*	1278	1.807	2476	0.997	2755	1.870	3904	1.000
Curve 1*	301	0.107	602	0.040	13555	0.127	15896	0.053
Curve 2*	300	0.193	600	0.063	15824	0.247	20138	0.110
Curve 3*	811	0.660	1622	0.287	99865	0.837	99886	0.430
Curve 4*	810	1.150	1620	0.423	188321	1.293	191684	0.577
Curve 5*	2018	1.380	4036	0.647	199890	1.820	199890	0.990
Curve 6*	2091	2.387	4182	0.953	392448	3.243	393974	1.697

Table 2. The table shows the number of remaining edges and the CPU-time in seconds taken to filter the bi-directed graphs using both the directed and undirected versions of the SPFCP algorithm with bridge and required-arc detection. We vary the quality of the upper bounds between optimal and 5%. The Lagrangian dual is optimized using the cutting plane algorithm.

of the SPFCP, the directed version is more effective and more general since it can filter both directed and undirected graphs.

4.3 Required-Arc and Bridge Detection

Next, we would like to investigate what the benefit of identifying edges that must be visited by any improving path is. Note that this aspect was one of the main contributions in [17]. We found that, in the DEM and curve approximation graphs the required-arc and bridge detection algorithms were ineffective. This is caused by the structure of these graphs that have many alternate optimal routes. However, on the road graph test files required-arc and bridge detection turned out to be quite effective and also caused the filtering of more edges than just using the SPFCP algorithm without the detection of required arcs. Table 3 shows the results for running both the undirected and directed versions of the SPFCP algorithm on the bi-directed road graphs with and without required-arc and bridge detection.

The test results show how required-arc and bridge detection improve the SPFCP algorithm's ability to filter edges on all of the road graph test files. They also show that, as the value of the initial upper bound on the objective deviates from optimality required-arc and bridge detection becomes more valuable. In the case of using the undirected SPFCP on the Road Small* test file, bridge detection filters 7% more edges with an optimal upper bound and 13% more with an upper bound of +5% from optimal. Generally, SPFCP-U-BD takes less time to complete than SPFCP-D-RE and SPFCP-U. This can be attributed to the bridge detection being most effective in the early iterations of the filtering algorithm and is illustrated in Figure 3.

Graph	Algo	Optimal		+1%		+3%		+5%	
		# Edges	Time	# Edges	Time	# Edges	Time	# Edges	Time
Road Small SPFCP-	-D	1181	1.457	1314	1.460	1597	1.487	2294	1.657
	-D-RE	899	1.570	965	1.560	1171	1.563	1813	1.770
	-U	1718	1.357	1812	1.360	2046	1.383	2640	1.583
	-U-BD	1596	1.137	1658	1.143	1852	1.153	2332	1.323
Road Big SPFCP-	-D	1414	2.097	1493	2.093	1717	2.127	2100	2.127
	-D-RE	1278	2.217	1307	2.217	1385	2.243	1672	2.223
	-U	2528	1.417	2584	1.423	2724	1.427	3178	1.460
	-U-BD	2476	1.360	2506	1.360	2574	1.373	2980	1.407

Table 3. The table shows the number of remaining edges and the CPU-time, in seconds, taken to filter the bi-directed road graphs using both the directed and undirected versions of the SPFCP algorithm with and without bridge and required-arc detection. The Lagrangian dual is optimized using the interval partitioning algorithm.

4.4 Interval Partitioning vs Cutting Plane

In the following experiments, we compare the performance of SPFCP-D while using the two algorithms for closing the duality gap, cutting plane and interval partitioning. Table 4 summarizes our results. Using the cutting plane algorithm improves the speed of the SPFCP filtering algorithm dramatically over using interval partitioning: SPFCP-D-CP is 63% faster on average when given an optimal upper bound and 65% faster on average when given an upper bound of +5% from optimal. In the optimality proof, both methods filtered roughly the same amount of edges, while interval partitioning is slightly more effective than the cutting plane algorithm.

The faster computation times and the slightly diminished effectivity of the cutting plane algorithm are explained by the fact that the method is able to close the duality gap in far fewer iterations, which can be seen by comparing Figures 4 and 5. We can see clearly how the cutting plane algorithm considers more meaningful Lagrangian multipliers much earlier in the search, which results in a much quicker computation of the lower bound as well as more filtering at earlier stages of the optimization. Also, it needs less iterations close to the optimal multipliers where the interval partitioning algorithm considers quite a few very near optimal multipliers before the desired approximation quality is achieved. The cutting plane needs just one iteration once that it is close enough to the optimum. We believe it is for that reason that the algorithm is slightly less effective in its filtering abilities. Still, we prefer the cutting plane algorithm over interval partitioning since it is able to filter almost as many edges in a fraction of the time.

4.5 Filtering for the RCSPP

In this section we compare the performance of the SPFCP algorithm against previously developed filtering algorithms for the RCSPP. Particularly, we compare against the algorithm from Aneja et al. (AN) and the algorithm from Beasley and Christofides (BC) [1,

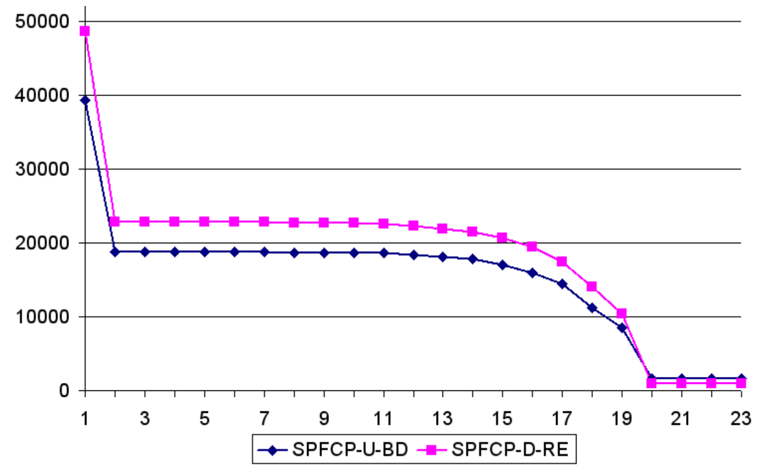

Fig. 3. This figure shows the remaining size of the Road Small* instance after each iteration of the interval partitioning algorithm.

4]. Both AN and BC only remove edges from the graph without detecting those edges that must be visited by all improving paths. AN considers the pure shorter-path constraints on the objective and the resource only, without integrating them in a Lagrangian fashion. BC performs filtering for the optimal Lagrangian multiplier. It is important to note here that suboptimal Lagrangian multipliers can have stronger filtering abilities than the optimal ones [16]. Therefore, the idea of CP-based Lagrangian relaxation makes sense, i.e. it is a reasonable approach to filter even during the optimization of the Lagrangian dual and not just for optimal multipliers only.

Table 5 shows the results for experiments using the directed input graphs and SPFCP-D-BD versus AN and BC. Comparing BC and AN first, we find that BC filters much better, but also takes significantly more time to do so. This is not surprising, since BC needs to solve the Lagrangian dual whereas AN works by just four shortest-path computations. We observe that SPFCP can increase the filtering effectiveness further (by 40% on average) while using less computation time than BC but still about twice as much as AN. The fact that SPFCP runs faster than BC has to be attributed to the algorithm's ability to filter out most of the edges in the first few iterations of solving the Lagrangian dual, thereby reducing the graph size and making successive iterations quicker. While SPFCP filtering works slower than AN, from the solution times of the RCSPP algorithms LSA and MZ on the filtered graphs we see that the additional effort is very worthwhile in the context of the RCSPP. Since this problem, though NP-hard, is still relatively easy to solve, we conjecture that the improved filtering power of shorter-path filtering in combination with CP-based Lagrangian relaxation will probably pay off even more in the context of more complex problems that incorporate shorter-path constraints.

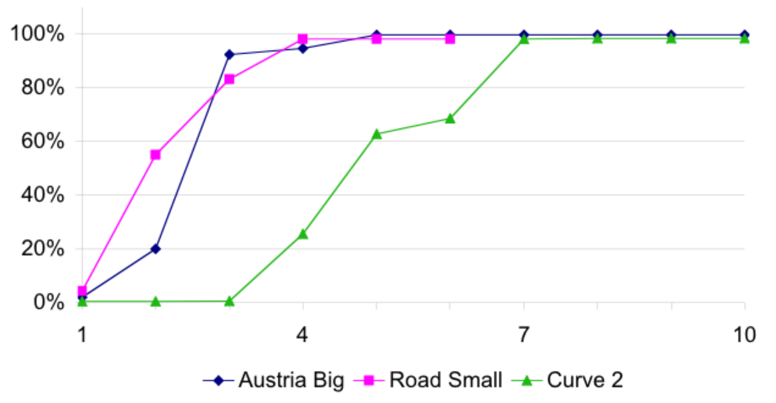

Fig. 4. This figure shows the percentage of edges filtered by SPFCP-D at each iteration of the cutting plane algorithm used to solve the Lagrangian relaxation.

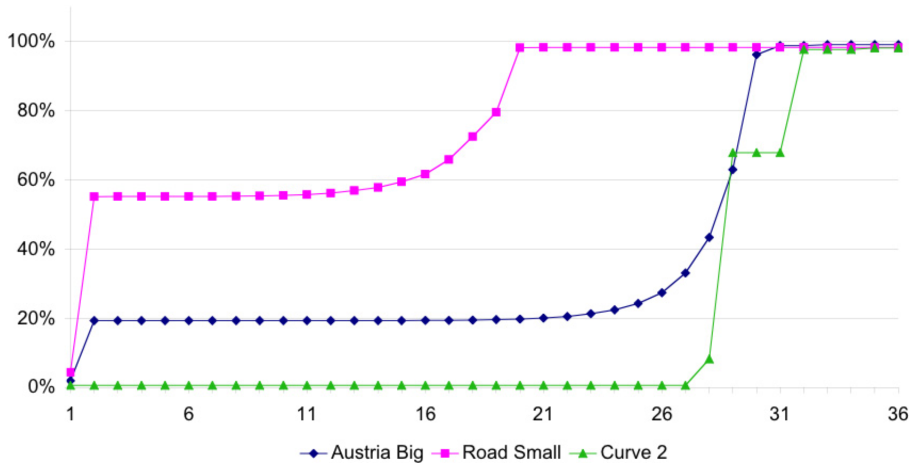

Fig. 5. This figure shows the percentage of edges filtered by SPFCP-D at each iteration of the interval partitioning algorithm used to solve the Lagrangian relaxation.

	interval partitioning				cutting plane			
Graph	Optimal		+5%		Optimal		+5%	
	# Edges	Time	# Edges	Time	# Edges	Time	# Edges	Time
Austria S	229	1.293	3150	1.750	231	0.367	3341	0.430
Austria B	410	7.623	7097	8.160	410	1.607	7323	1.767
Scotland S	304	2.040	3864	2.317	263	0.537	4737	0.607
Scotland B	494	14.113	17220	14.970	494	2.770	21578	3.113
Road S	899	1.563	1813	1.767	899	0.607	1559	0.620
Road B	1278	2.217	1672	2.223	1278	1.793	2755	1.813
Curve 1	306	0.217	7948	0.300	301	0.067	7948	0.090
Curve 2	300	0.270	10067	0.403	300	0.107	10069	0.170
Curve 3	836	1.090	49943	1.797	811	0.447	49943	0.600
Curve 4	803	1.643	95842	2.673	810	0.710	95842	0.847
Curve 5	2018	2.587	99945	4.067	2018	0.953	99945	1.310
Curve 6	2034	3.953	196987	5.947	2091	1.553	196987	2.263

Table 4. The table shows the number of remaining edges and the CPU-time, in seconds, taken by SPFCP-D-RE using the cutting plane and interval partitioning algorithms for solving the Lagrangian dual. The quality of the upper bound was varied from optimal to +5% from optimal.

5 Conclusions

We provided an experimental evaluation of shorter-path filtering by applying it to the resource constrained shortest path problem. We have compared the undirected and the directed versions of shorter-path filtering and found that the undirected version, where applicable, works about twice as fast while the directed version is more effective and enjoys wider applicability. Regarding the identification of edges that must be visited by all improving routes, we found that this ability is of use only in rare special cases where no alternative improving paths exist.

Further, we have seen that, in the context of CP-based Lagrangian relaxation, the choice of the algorithm solving the Lagrangian dual can have a significant impact on the overall performance of the filtering algorithm. For one-parameter relaxations, we have found that a method based on cutting planes can be much more efficient than an interval partitioning algorithm.

Finally, our experiments showed that, even for this comparably simple problem, an increase in filtering power can yield to significant performance improvements and that shorter-path constraint filtering outperforms previously developed filtering algorithms for the RCSPP.

Graph	AN				BC				SPFCP D-IP				SPFCP D-CP			
	Filter		Solver		Filter		Solver		Filter		Solver		Filter		Solver	
	Edges	Time	LSA	MZ	Edges	Time	LSA	MZ	Edges	Time	LSA	MZ	Edges	Time	LSA	MZ
Austria S	28480	.287	5.113	.338	245	.620	.002	.039	229	1.293	.002	.040	231	.367	.002	.040
Austria B	131647	1.190	258.596	1.786	410	3.143	.007	.049	410	7.623	.007	.052	410	1.607	.007	.049
Scotland S	43667	.423	7.221	2.683	906	.883	.020	.064	304	2.040	.005	.044	263	.537	.003	.040
Scotland B	209546	1.933	-	10.34	1287	4.823	.033	.084	494	14.113	.008	.052	494	2.770	.009	.050
Road S	21446	.430	.288	.318	1309	1.010	.011	.054	899	1.563	.009	.053	899	.607	.009	.053
Road B	109711	1.770	1.222	1.818	1641	4.247	.019	.072	1278	2.217	.017	.065	1278	1.793	.018	.063
Curve 1	9945	.040	3.693	.038	9945	.067	3.581	.038	306	.217	.004	.001	301	.067	.004	.001
Curve 2	19679	.073	12.225	1.143	305	.133	.004	.040	300	.270	.004	.040	300	.107	.004	.040
Curve 3	49945	.223	30.175	2.078	826	.450	.009	.053	836	1.09	.009	.004	811	.447	.009	.004
Curve 4	99790	.387	-	.394	99790	.653	-	.393	803	1.643	.009	.004	810	.710	.010	.002
Curve 5	99945	.470	-	.592	2018	.940	.049	.018	2018	2.587	.037	.014	2018	.953	.036	.018
Curve 6	199790	.810	-	190.62	199790	1.513	-	190.105	2034	3.953	.036	.006	2091	1.553	.038	.031

Table 5. The table shows the number of remaining edges after filtering wrt to an optimal lower bound plus one (so that the solvers still need to compute the optimum after filtering), the CPU-time in seconds taken for the filtering on directed graphs, and the CPU-time taken by the RCSSP solvers LSA and MZ to find an optimal solution for the filtered graphs. A '-' indicates that a solver was unable to find a solution due to exhaustive memory consumption.

References

1. Y. Aneja, V. Aggarwal, K. Nair. Shortest chain subject to side conditions. *Networks*, 13:295-302, 1983.
2. D. Applegate, R. Bixby, V. Chvátal, and W. Cook. On the solution of Traveling Salesman Problems. *Doc. Math. J. DMV*, Extra Volume ICM III, pp.645–656, 1998.
3. F. Barahona and R. Anbil. The Volume Algorithm: producing primal solutions with a subgradient algorithm. *Mathematical Programming*, 87:385–399, 2000.
4. J. Beasley, N. Christofides. An Algorithm for the Resource Constrained Shortest Path Problem. *Networks*, 19:379-394, 1989.
5. H. Crowder. Computational improvements for subgradient optimization. *Symposia Mathematica*, XIX:357–372, 1976.
6. J. Desrosiers, Y. Dumas, M. Solomon, F. Soumis. Time Constrained Routing and Scheduling. *Handbook in Operations Research and Management Science 8: Network Routing*, North-Holland, 8:35–139, 1995.
7. I. Dumitrescu, N. Boland. The weight-constrained shortest path problem: preprocessing, scaling and dynamic programming algorithms with numerical comparisons. *International Symposium on Mathematical Programming (ISMP)*, 2000.
8. T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, B. Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59-81, 2002.
9. F. Focacci, A. Lodi, M. Milano. Cost-Based Domain Filtering. *Principles and Practice of Constraint Programming (CP)* Springer LNCS 1713:189–203, 1999.
10. A. Frangioni. A Bundle type Dual-ascent Approach to Linear Multi-Commodity Min Cost Flow Problems. *Technical Report*, Dipartimento di Informatica, Universita di Pisa, TR-96-01, 1996.
11. A. Frangioni. Dual Ascent Methods and Multicommodity Flow Problems. *Doctoral Thesis*, Dipartimento di Informatica, Universita di Pisa, TD-97-05, 1997.
12. G. Handler, I. Zang. A Dual Algorithm for the Restricted Shortest Path Problem. *Networks*, 10:293-310, 1980.
13. J.E. Kelley. The Cutting Plane Method for Solving Convex Programs. *Journal of the SIAM*, 8:703–712, 1960.
14. A. Makri and D. Klabjan. A New Pricing Scheme for Airline Crew Scheduling. *INFORMS Journal on Computing*, 16:56–67, 2004.
15. K. Mehlhorn, M. Ziegelmann. Resource Constrained Shortest Paths. *Proc. 8th European Symposium on Algorithms (ESA)*, Springer LNCS 1879:326-337, 2000.
16. M. Sellmann. Theoretical Foundations of CP-Based Lagrangian Relaxation. *Principles and Practice of Constraint Programming (CP)*, Spirnger LNCS 3258:634–647, 2004.
17. M. Sellmann. Cost-Based Filtering for Shorter Path Constraints. *Principles and Practice of Constraint Programming (CP)*, Springer LNCS 2833:679–693, 2003.
18. M. Sellmann and T.Fahle. Coupling Variable Fixing Algorithms for the Automatic Recording Problem. *Annual European Symposium on Algorithms (ESA)*, Springer LNCS 2161: 134–145, 2001.
19. M. Sellmann and T. Fahle. Constraint Programming Based Lagrangian Relaxation for the Automatic Recording Problem. *Annals of Operations Research*, 118:17–33, 2003.
20. P. Toth and D. Vigo. The Vehicle Routing Problem. *Monographs on Discrete Mathematics and Applications*, SIAM, Philadelphia, 2001.