Cost-Based Filtering for Shorter Path Constraints*

Meinolf Sellmann

Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853
sello@cs.cornell.edu

Abstract. Many real world problems, e.g. in personnel scheduling and transportation planning, can be modeled naturally as Constrained Shortest Path Problems (CSPPs), i.e., as Shortest Path Problems with additional constraints. A well studied problem in this class is the Resource Constrained Shortest Path Problem. Reduction techniques are vital ingredients of solvers for the CSPP, that is frequently NP-hard, depending on the nature of the additional constraints. Viewed as heuristics, until today these techniques have not been studied theoretically with respect to their efficiency, i.e., with respect to the relation of filtering power and running time. Using the concepts of Constraint Programming, we provide a theoretical study of cost-based filtering for shorter path constraints on acyclic, on undirected and on directed graphs that do not contain negative cycles.

Keywords: constrained shortest paths, problem reduction, optimization constraints, relaxed consistency

1 Introduction

Real world problems can frequently be modeled as Shortest Path Problems with additional constraints. The best known Constrained Shortest Path Problem (CSPP) is probably the *Resource Constrained Shortest Path Problem* [1, 3, 6, 14, 16] that consists in the combination of a Shortest Path Problem and capacity constraints on a set of resources. Even on directed acyclic graphs (DAGs), for non-negative objective functions and for only one resource that problem is known to be NP-hard [13].

Standard applications for the Resource Constrained Shortest Path Problem are route planning in traffic networks and quality of service routing [29, 21]. The Crew Scheduling Problem is another example of a real world problem where CSPPs are used in many successful approaches: In a column generation process, CSPPs have to be solved to generate columns, which correspond to individual lines of work in this context [7, 30].

Generally, CSPPs appear very often as subproblems in column generation approaches. Examples range from route guidance [15] and duty scheduling in public transit [4] up to the scheduling of switching engines [19]. In [17], a general framework for constraint programming based column generation was developed that formalizes the use of optimization constraints in this context.

* This work was supported by the Intelligent Information Systems Institute, Cornell University (AFOSR grant F49620-01-1-0076).

To solve Constrained Shortest Path Problems, state of the art solvers compute lower and upper bounds on the problem and then close the duality gap. The latter task is carried out by an enumeration procedure such as a tree search [3], dynamic programming [20] or a k-shortest path algorithm [14]. Particularly in a tree search, but also in the other approaches the tightening of (sub-)problems is vital for an effective gap closing procedure. And therefore, it is essential for the overall performance and the practical success of the entire approach.

The first tightening strategy that was proposed goes back to a work done by Aneja et al. [1] for problem reduction of the Resource Constrained Shortest Path Problem. The basic idea consists in identifying nodes and arcs that cannot be visited by any path that obeys the given resource restrictions. The same method can also be used to identify nodes and arcs that cannot be visited by any improving path, which gives a first cost-based filtering algorithm for the problem. Dumitrescu and Boland [6] proposed a repeated problem reduction procedure that has shown to be very successful for hard constrained problems. Beasley and Christofides [3] have shown how a tighter global, Lagrangian relaxation based bound can be used for the elimination of nodes and arcs.

Apparently, none of these heuristics has been classified with respect to its filtering abilities. Moreover, the reduction techniques used all focus on the removal of nodes and arcs, but those arcs and nodes that must be visited by all paths of a certain quality remain undetected. However, with respect to the additional constraints of the CSPP this information can be very valuable as it may prove useful for an additional simplification of the problem.

Constraint Programming theory provides means for the state of consistency that a domain filtering algorithm achieves. In [8], we extended the notion of generalized arc-consistency (GAC) to the concept of *relaxed consistency* for optimization constraints. It allows to measure and compare heuristic filtering algorithms not only with respect to their running time, but also to their filtering power that is determined by the quality of the relaxation used. With respect to shorter path constraints, we study the complexity of achieving GAC. Since the problem is NP-hard in the general case, we introduce shortest-path relaxations and develop and compare different filtering algorithms for different graph classes.

Particularly, in Section 2, we review the notion of relaxed consistency, and in Section 3, we define shorter path constraints formally. In Section 4, we investigate the problem of achieving GAC for a shorter path constraint on undirected graphs, where it is shown to be NP-hard. We introduce a shortest-path relaxation and formulate a linear time algorithm that achieves a state of relaxed consistency. Finally, in Section 5, we develop cost-based filtering algorithms for shorter path constraints on directed acyclic and general directed graphs with non-negative costs or graphs that at least do not contain negative weight cycles.

2 Definitions and General Observations

Within a tree search, during the course of optimization we compute a sequence of feasible solutions. We refer to the best known feasible solution as the *incumbent solution*.

Obviously, once we have found a solution of a certain quality, we are searching for improving solutions only. Thus, we impose a restriction on the objective. That restriction in combination with other side-constraints of the original problem forms an *optimization constraint* [7, 10, 11, 17, 22], which is the core concept that we will be using throughout this paper. It was developed by a community that has been working on the integration of constraint programming (CP) and operations research (OR) in recent years. Though never explicitly stated as constraints, in the OR world optimization constraints are frequently used for bound computations and variable fixing. From a CP perspective, they can be viewed as *global constraints* that link the objective with some other constraints of the problem:

Given $n \in \mathbb{N}$, let $X_1, \ldots, X_n$ denote variables with finite domains $D_1 := D(X_1)$, $\ldots, D_n := D(X_n)$. Further, given a constraint $\zeta : D_1 \times \cdots \times D_n \rightarrow \{0,1\}$, and an objective function $Z : D_1 \times \cdots \times D_n \rightarrow \mathbb{Q}$, let $x_i \in D_i \; \forall \; 1 \leq i \leq n$.

Definition 1. *Let $B \in \mathbb{Q}$ denote an upper bound on the objective Z to be minimized.*

A function $\vartheta_{\zeta,Z}[B] : D_1 \times \cdots \times D_n \rightarrow \{0,1\}$ with $\vartheta_{\zeta,Z}[B](x_1, \ldots, x_n) = 1$ iff $\zeta(x_1, \ldots, x_n) = 1$ and $Z(x_1, \ldots, x_n) < B$ is called minimization *or, more generally,* optimization constraint.

The purpose of optimization constraints is twofold: first, they can be used for pruning by computing a lower bound on the objective, which is the common idea in branch and bound algorithms. Second, they may also be used to remove those values from variable domains that cannot be part of any improving solution, which may be viewed as a generalization of the variable fixing technique (for problems containing binary variables only, variable fixing and domain filtering are of course the same).

2.1 On the Complexity of Cost-Based Domain Filtering Problems

In order to achieve generalized arc-consistency (GAC) [2, 18] of an optimization constraint, we have to find and remove all assignments that cannot be extended to an *improving* solution that is *feasible* with respect to ζ. That is, if ζ is the only constraint of a combinatorial optimization problem (we call that optimization problem and the optimization constraint *corresponding to* or *associated with* each other), a GAC algorithm allows us to compute improving solutions in a backtrack-free search. Consequently, if the original problem is NP-hard, so is the problem of achieving GAC for the corresponding optimization constraint. As an example, consider e.g. the Knapsack Problem [8].

If the optimization problem associated with an optimization constraint is polynomial, then the problem of achieving GAC may also be polynomial. For example, consider the AllDifferent constraint with costs. The corresponding optimization problem is the Weighted Bipartite Matching Problem (WBMP) for which there exists a polynomial time algorithm. Now, since the removal of an edge or two nodes (when the edge between the nodes is chosen to be part of the matching) does not change the structure of the problem (i.e., the subproblem is again a WBMP), achieving GAC for the AllDifferent with costs can obviously be done in polynomial time [24, 25].

The situation may change, however, if the problem structure is not preserved when a variable is forced to take a specific value. Consider a Shortest Path Problem in an

arbitrary network, where we use a binary variable for each edge (whereby a value 1 means that the edge is chosen to be on the path, and a value 0 represents that the edge is not on the path). The problem of finding a shortest path is of course solvable in polynomial time. However, if we are to compute the set of edges that must or cannot be part of any simple path that does not exceed a certain length, we are facing an NP-hard problem, which is easy to see by reduction to the Two Vertex Disjoint Paths Problem [9].

2.2 Degrees of Consistency

The discussion shows that we cannot always hope for an efficient cost-based domain filtering algorithm that achieves GAC. Therefore, we may consider to develop less effective but polynomial time bounded filtering algorithms that may only achieve a weaker degree of consistency.

Regarding cost-based filtering, an idea that has been developed in OR to perform variable fixing on linear integer problems is the *reduced cost filtering* method: when solving the continuous relaxation bound on a linear combinatorial optimization problem with the help of a general LP solver (such as the simplex algorithm or interior point methods), we get dual information and reduced cost data for free. That data can be used to compute a lower bound on the loss of performance that we have to accept when adding a new constraint of the form $X = x$ (usually this is done by performing one dual simplex re-optimization step). And of course, if the loss is too large, we can deduce that x must be removed from the domain of X. In [8], we strengthened and generalized the basic idea by coupling optimization constraints and relaxations:

Definition 2. *Given a minimization constraint* $\vartheta_{\zeta,Z}[B] : D_1 \times \cdots \times D_n \to \{0,1\}$, *let* $\Delta := D_1 \times \cdots \times D_n$. *Further, denote with* 2^Δ *the set of all subsets of* Δ, *and let* $L : 2^\Delta \to \mathbb{Q}$ *such that for all* $M_i \subseteq D_i,\ 1 \le i \le n$,

$$L(M_1 \times \cdots \times M_n) \le \min\{Z(x_1,\ldots,x_n) \mid \zeta(x_1,\ldots,x_n) = 1,\ x_i \in M_i,\ 1 \le i \le n\},$$

where $\min \emptyset = \infty$. *We call* L *a* relaxation of $\vartheta_{\zeta,Z}$ *and say that* $\vartheta_{\zeta,Z}[B]$ *is* relaxed L-consistent, *iff for any given* $1 \le i \le n$ *and* $x_i \in D_i$, $L(D_1 \times \cdots \times \{x_i\} \times \cdots \times D_n) < B$.

As one would expect, the definition states that relaxed L-consistency can the easier be achieved the weaker the relaxation L is. For $L \equiv -\infty$, there is no work to do to achieve relaxed L-consistency, whereas GAC is enforced when $L(M_1 \times \cdots \times M_n) = \min\{Z(x_1,\ldots,x_n) \mid \zeta(x_1,\ldots,x_n) = 1,\ x_i \in M_i,\ 1 \le i \le n\}$. That is, the choice of L determines the degree of domain filtering.

In practice, L is usually chosen as a fairly tight bound that can still be computed quickly. For example, linear programming relaxations can be used, as it was done in [8]. Generally, within a tree search there is a trade-off between the time spent per search node and the total number of search nodes. Thus, the favorable choice of the accuracy of the relaxation is always subject to the optimization problem at hand. We introduced the concept of relaxed consistency because it allows to compare domain filtering algorithms not only with respect to the running time but also with respect to the degree of consistency they achieve.

3 Shorter Path Constraints

Definition 3. *Denote with $G = (V, E, c)$ a weighted (directed or undirected) graph with $||c||_\infty \in O(poly(|E|, |V|))$[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 4. *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.

Obviously, the shorter path constraint is an optimization constraint. Now, to ease the notation, for the remainder of this section we assume that a shorter path constraint is associated with a set variable $Y \subseteq E$ that represents the set of edges e_i for which $X_i = 1$. The (current) domains of the variables X will be represented by two sets: the set of *possible members* $pos(Y)$, and the set of *required members* $req(Y)$ of Y. In the subtree of the search rooted at the current choice point, we require $req(Y) \subseteq Y \subseteq pos(Y)$. That is, $req(Y)$ represents the set of variables for which it has been set $X_i = 1$, and the set $E \setminus pos(Y)$ represents the set of variables for which it has been decided to set $X_i = 0$ already. Then, in the current choice point, we have to search for admissible paths P such that $req(Y) \subseteq P \subseteq pos(Y)$. Note that we use the set variable Y only to ease the presentation. It has no impact on the implementation that is assumed to use only the variables X. Especially, the didactic use of a set variable has no impact on the state of GAC that we try to achieve[2]. To achieve GAC of a shorter path constraint, we must ensure:

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

[2] Note that we could also model the shorter path constraint with a set variable instead of m binary variables. Then, GAC for the binary model corresponds to bound-consistency in the set model.

– For all $e \in pos(Y)$, there exists an admissible path P with $req(Y) \cup \{e\} \subseteq P \subseteq pos(Y)$, and
– for all $e \notin req(Y)$, there exists an admissible path with $req(Y) \subseteq P \subseteq pos(Y) \setminus \{e\}$.

That is, we have to find the set of all edges that must or cannot be part of all/any paths with length lower than B.

Obviously, whether there exists an admissible path at all can be decided by applying a shortest path algorithm. However, to decide whether there exists a simple path that visits a set of edges is already an NP-hard task which can be shown by a simple reduction to the Two Vertex Disjoint Path Problem. Consequently, the problem of achieving GAC for the general shorter path constraint is also NP-hard.

4 Shortest Path Problems on Undirected Graphs

First, we consider shorter path constraints on undirected graphs with non-negative edge weights. Obviously, on the existence of an admissible path can be decided by applying a shortest path algorithm. However, it is easy to see that to decide whether there exists a simple path that visits a set of edges is an NP-hard task. Therefore, in the following we develop a cost-based filtering algorithm that achieves relaxed consistency rather than generalized arc-consistency. In order to introduce the relaxation we want to use, we start with

Definition 5. *Denote with $G = (V, E, c)$ a weighted (directed or undirected) graph.*

– *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.*
– *With $P(i, j) \in \pi(i, j)$ we refer to a shortest path from i to j (with respect to c). Then, to ease the notation, we set $c(i, j) := cost(P(i, j))$.*
– *Given a shorter path constraint, a k-simple path P from v_1 to v_n is called a k-*admissible path *iff $cost(P) < B$.*

Note that, in a graph with non-negative edge weights, a shortest admissible path is also a shortest 2-admissible path. Now, instead of checking for admissible paths only, we consider the following shortest path relaxation (see Definition 2): Denote with $D(Y)$ the domain of Y represented as the pair of sets $(req(Y), pos(Y))$. We set $H := \{P \mid P \in \pi(v_1, v_n)$ with $P \subseteq pos(Y)\}$ and $F_f := \{P \mid P$ is a 2-simple path from v_1 to v_n with $f \in P\}$ for all $f \in E$. Then, we define

$$L_1(D(Y)) := \max\{\ \min\{cost(P) \mid P \in H\},\ \max_{f \in req(Y)}\{\min\{cost(P) \mid P \in F_f\}\}.$$

Lemma 1. *L_1 is a shortest path relaxation.*

Proof: According to Definition 2, we have to show that

$$L_1(D(Y)) \leq \min\{cost(P) \mid P \in \pi(v_1, v_n), req(Y) \subseteq P \subseteq pos(Y)\}.$$

Let $P \in \pi(v_1, v_n)$ denote a shortest path in G with $req(Y) \subseteq P \subseteq pos(Y)$. Obviously, it holds that $P \in H$ and $P \in F_f$ for all $f \in req(Y)$. And therefore, $L_1(D(Y)) \leq cost(P)$. □

The big advantage of the above relaxation is that it allows to be checked for consistency very easily, as we shall see below. Note, however, that L_1 does not require that the 2-admissible paths must visit all nodes in $req(Y)$ simultaneously. Of course, this weakens the relaxation. In practice, we can reduce the negative effects by improving the probability that a 2-admissible path visits the edges in $req(Y)$: we set $c_{ij} := 0$ for all $\{i, j\} \in req(Y)$ and subtract $cost(req(Y))$ from B.

According to the definition, a shorter path constraint is relaxed L_1-consistent, iff

1. for all $f \in pos(Y)$, there exists a 2-admissible path $P \in F_f$, and
2. for all $f \notin req(Y)$, there exists an admissible path $P \in H$ with $f \notin P$.

In the following two sections, we show how relaxed L_1-consistency can be achieved efficiently.

4.1 Removing Edges from the Possible Set

First, for all edges in E, we have to check whether there exists a 2-admissible path in G that visits an edge $\{i, j\} \in E$. We observe that the shortest 2-simple path from v_1 to v_n that visits $\{i, j\}$ is either $(P(v_1, i), P(j, v_n))$ with costs $c(v_1, i) + c_{ij} + c(j, v_n)$ or $(P(v_1, j), P(i, v_n))$ with costs $c(v_1, j) + c_{ij} + c(i, v_n)$. Therefore, to check whether an edge has to be removed from $pos(Y)$ with respect to the relaxation L_1 it is sufficient to know the shortest-path distances from the source and to the sink of all nodes. Both values can be computed for all nodes by only two shortest-path computations in G in time $O(m + n \log n)$ by using Dijkstra's algorithm in combination with Fibonacci heaps [12]. In a random access machine (RAM) model, shortest paths on undirected graphs can be computed in time $O(m+n)$ when using the algorithm of Thorup (see [28] and the recent extension of Pettie and Ramachandran in [23]). Thus, the set of edges that has to be removed from $pos(Y)$ to achieve relaxed L_1-consistency can be computed in time $O(m + n \log n)$, and in time $O(m + n)$ on a RAM.

4.2 Adding Edges to the Required Set

After having removed all edges from G that cannot be part of any 2-admissible path, the edges that must be visited by all such paths can be characterized by

Theorem 1. *Assume that all edges in G are part of at least one 2-admissible path. Then, an edge $\{r, s\} \in E$ must be visited by all admissible paths, iff $\{r, s\} \in P(v_1, v_n)$, and $\{r, s\}$ is a bridge in G*[3].

We can prove the above theorem with the help of the following two lemmas:

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

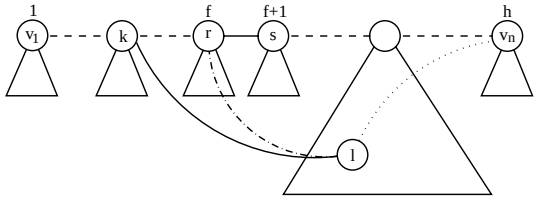

Fig. 1. The figure schematically shows an edge $\{k, l\} \in E$ that must exist according to Lemma 2. Solid lines mark edges in E, dashed lines parts of the shortest path between v_1 and v_n. The dotted line between l and v_n indicates that there exists a path between the two nodes that does not visit the edge $\{r, s\}$. The dashed lines between l and r indicate that the shortest path from l to v_n visits node r. The numbers on top of the nodes give their corresponding DFS numbers, and triangles mark DFS subtrees.

Lemma 2. *Assume that all edges in G are part of at least one 2-admissible path. Let $\{r, s\} \in E$ denote an edge that must be visited by all admissible paths and that can be removed from G without disconnecting v_1 and v_n. Then, there exists an edge $\{k, l\} \in E$ such that*

1. $\exists\ P \in \pi(v_1, v_n) :\ \{k, l\} \in P$ *and* $\{r, s\} \notin P$,
2. k *is a shortest-path predecessor of* r, *and*
3. $\{r, s\} \in P(l, v_n)$.

Proof: (See Fig. 1.) Assume we compute a shortest path $P = (i_1, \ldots, i_h) \in \pi(v_1, v_n)$. Then, $i_1 = v_1$, $i_h = v_n$ and $i_f = r$, $i_{f+1} = s$ for some $1 \leq f < h$. Next, we change the graph representation of G such that $\{i_g, i_{g+1}\}$ is the first outgoing edge of node i_g for all $1 \leq g < h$. For all nodes $j \in V$, denote with $d_j \in \{1, \ldots, n\}$ the ordering in which the nodes are first visited by a depth first search using the modified graph representation of G. Then, $d_{i_g} = g$ for all $1 \leq g \leq h$. Since the removal of $\{r, s\}$ does not disconnect v_1 and v_n, there exists a forward edge $\{k, l\} \in E$ with $d_k < f$ and $d_l > f + 1$. This implies the Statements 1 and 2.

It remains to show that $\{r, s\} \in P(l, v_n)$. By assumption, there exists a 2-admissible path R through the edge $\{k, l\}$. There are two possibilities: either R visits node k or node l first, which corresponds to:

a) $c(v_1, k) + c_{kl} + c(l, v_n) < B$, or
b) R visits l before k and $c(v_1, l) + c_{kl} + c(k, v_n) < B$.

In the first case, because $\{r, s\} \notin P(v_1, k)$ and $\{r, s\}$ must be visited by all admissible paths, it holds that $\{r, s\} \in P(l, v_n)$, and we are done.

So let us consider the second case. Let $Q \in \pi(v_1, l)$ denote a shortest path from v_1 to l with $\{r, s\} \notin Q$. Without loss of generality we may assume that k and l are chosen such that $\{k, l\} \in Q$. We observe that $\{r, s\} \in P(v_1, l)$, because otherwise this implies that $\{k, l\} \in Q = P(v_1, l)$. But then the 2-admissible path visits node k before node l. Now, because k is a shortest-path predecessor of r and $\{r, s\} \in P(v_1, l)$, it holds that $k \in P(v_1, l)$. And then,

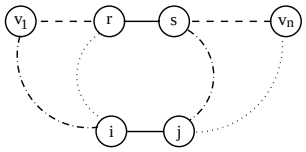

Fig. 2. The figure schematically shows an edge $\{i, j\} \in E$ that must exist according to Lemma 3. Solid lines mark edges in E, dashed lines mark parts of the shortest path between v_1 and v_n. Dashed lines indicate parts of the shortest path from v_1 to a node, dotted lines parts of the shortest path from a node to v_n. The proof of Theorem 1 shows that the path $(P(v_1, r), P(r, i), P(j, s), P(s, v_n))$ is two admissible and does not visit the edge $\{r, s\}$.

$$\begin{aligned} c(v_1, k) + c_{kl} + c(l, v_n) &\leq c(v_1, k) + c_{kl} + c(l, k) + c(k, v_n) \\ &= c(v_1, k) + c(k, l) + c_{kl} + c(k, v_n) \\ &= c(v_1, l) + c_{kl} + c(k, v_n) \\ &< B, \end{aligned}$$

which reduces this case to (a). □

Lemma 3. *Assume that all edges in G are part of at least one 2-admissible path. Let $\{r, s\} \in E$ denote an edge that must be visited by all admissible paths and that can be removed from G without disconnecting v_1 and v_n. Then, there exists an edge $\{i, j\} \in E$ such that $\{r, s\} \in P(i, v_n)$ and $\{r, s\} \notin P(j, v_n)$, and $\{r, s\} \notin P(v_1, i)$ and $\{r, s\} \in P(v_1, j)$.*

Proof: (see Fig. 2.) Denote with $\{k, l\} \in E$ an edge as in Lemma 2. Then, there exists a path $P \in \pi(l, v_n)$ with $\{r, s\} \notin P$ and $\{r, s\} \in P(l, v_n)$.

1. Due to $\{r, s\} \notin P(v_n, v_n)$, there exists an edge $\{i, j\} \in P$ such that $\{r, s\} \in P(i, v_n)$ and $\{r, s\} \notin P(j, v_n)$.
2. By assumption, there is a 2-admissible path that visits j. Since $\{r, s\} \notin P(j, v_n)$, it follows that $\{r, s\} \in P(v_1, j)$, because $\{r, s\}$ must be visited by all admissible paths. Finally, assume that $\{r, s\} \in P(v_1, i)$. Then, the shortest path visiting node i has costs

   $$c(v_1, r) + c_{rs} + c(s, i) + c(i, r) + c_{rs} + c(s, v_n).$$

   But the path from v_1 via r, i and s to v_n has costs

   $$c(v_1, r) + c(r, i) + c(i, s) + c(s, v_n),$$

   which is lower or equal to the cost of the shortest path visiting i. This implies that it is a shortest path visiting node i, too. But it does not visit some edges with zero costs. Particularly, it does not visit the edge $\{r, s\}$. Therefore, we may assume that $\{r, s\} \notin P(v_1, i)$.

□

Proof of Theorem 1:

$\Leftarrow$ Let $\{r, s\}$ be a bridge on the shortest path $P \in \pi(v_1, v_n)$. Then, the removal of $\{r, s\}$ disconnects the graph G. Since the node pairs (v_1, r) and (s, v_n) are still connected, the removal of $\{r, s\}$ also disconnects v_1 and v_n. Thus, for all $P \in \pi(v_1, v_n)$, it holds that $\{r, s\} \in P$. Therefore, also all admissible paths must visit $\{r, s\}$.

$\Rightarrow$ Obviously, if there exists any admissible path, then $P(v_1, v_n)$ is admissible, too. Thus, $\{r, s\} \in P(v_1, v_n)$. Now assume that the removal of $\{r, s\}$ does not disconnect v_1 and v_n. Then, according to Lemma 3, there exists an edge $\{i, j\} \in E$ such that $\{r, s\} \in P(i, v_n)$, $\{r, s\} \notin P(j, v_n)$, $\{r, s\} \notin P(v_1, i)$ and $\{r, s\} \in P(v_1, j)$. By assumption, there exists a 2-admissible path R visiting $\{i, j\}$. Without loss of generality we may assume that R visits node i before node j, because

$$\begin{aligned} c(v_1, j) + c_{ij} + c(i, v_n) &= c(v_1, r) + c_{rs} + c(s, j) + c_{ij} + c(i, r) + c_{rs} + c(s, v_n) \\ &\geq c(v_1, r) + c(r, i) + c_{ij} + c(j, s) + c(s, v_n) \\ &\geq c(v_1, i) + c_{ij} + c(j, v_n). \end{aligned}$$

But this implies that $\{r, s\} \notin R$, which is a contradiction to the assumption that every admissible path must visit $\{r, s\}$.

□

Using Theorem 1, after having removed all edges that cannot be part of any 2-admissible path, we can compute all edges that must be visited by all admissible paths in time $O(m + n)$: first, we compute a shortest path $P \in \pi(v_1, v_n)$ and mark all edges on this path. Then, we compute all bridges in G (which can easily be done in linear time, see [5]) and check which ones are visited by P. It follows:

Corollary 1. *On undirected graphs with non-negative edge weights, relaxed L_1-consistency of a shorter path constraint can be achieved in time $O(m + n \log n)$, and in time $O(m + n)$ on a RAM.*

5 Shortest Path Problems on Directed Graphs

On acyclic graphs, it is easy to see that arc-consistency can be achieved in linear time by computing shortest-path distances from the source and to the sink, and be determining bridges in the undirected version of the graph after the removal of arcs.

So let us consider general directed graphs with non-negative arc weights. In the end of this section, we will also give two theorems that we can prove for graphs that may contain negative arc weights but no negative cycles.

As for undirected graphs, achieving arc-consistency for shorter path constraints in general directed networks is NP-hard. Regarding the removal of arcs from the possible set, relaxed L_1-consistency on directed graphs with non-negative arc weights can be achieved in the same way as on undirected graphs. However, with respect to arcs that must be visited by all admissible paths, the situation is even more complicated. Recall the result from Section 4: After having removed the infeasible edges, in undirected graphs the edges that have to be required are exactly the ones on the shortest path that must be visited by *all* paths from v_1 to v_n.

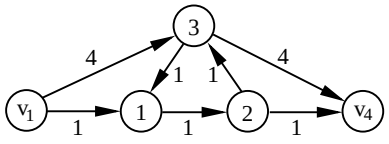

Fig. 3. A directed graph with non-negative arc weights. Assume we are given an upper bound $B = 8$. All arcs in the graph are part of an admissible path with costs lower than B. And every admissible path with costs lower than B must visit the arc $(1, 2)$. However, there exists a path $(v_1, 3, v_4)$ that does not visit this arc.

Unfortunately, this classification does not hold for directed graphs as can be seen in Figure 3. Thus, for all arcs $(i, j) \in P(v_1, v_n)$, we have to recompute the shortest-path value when removing (i, j) from E, which may require $n-1$ shortest-path computations in the worst case. It follows:

Theorem 2. *On directed graphs with non-negative arc weights, relaxed L_1-consistency can be achieved in time $O(n(m + n \log n))$.*

Since the computation time of the algorithm sketched in the above may not be efficient enough to be of profit when being applied in a tree search, in the following we consider another shortest path relaxation. Let $T \subseteq E$ denote a shortest-path tree in G rooted at v_1. Without loss of generality, we may assume that every node in G can be reached from 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. 4).

Obviously, $S_e^C \neq \emptyset$ iff $e \in T$. We set

$$\begin{aligned} J := \{P \mid\ & P \text{ is a 2-simple path from } v_1 \text{ to } v_n \text{ with} \\ & P \subseteq pos(Y) \text{ or, if } e \in P \setminus pos(Y), \text{ then there} \\ & \text{exists an arc } (i,j) \in P \setminus T \text{ such that} \\ & i \in S_e \text{ and } j \in S_e^C\}. \end{aligned}$$

And we define

$$\begin{aligned} L_2(D(Y)) := \max\{\ & \min\{cost(P) \mid P \in J\}, \\ & \max_{f \in req(Y)}\{\min\{cost(P) \mid P \in F_f\}\}. \end{aligned}$$

To understand the above shortest path relaxation better, we make the following observations:

- Obviously, because $H \subseteq J$, L_2 is dominated by L_1, i.e., $L_2 \leq L_1$. And therefore, L_2 is also a shortest path relaxation.
- The difference between relaxations L_1 and L_2 only consists in the set J that is used instead of H to determine the arcs that have to be required to achieve a state of relaxed consistency. In contrast to H, the set J also contains paths P that are not simple and that may visit arcs $e \notin pos(Y)$. However, if $e \in P \setminus pos(Y)$, then we enforce that P must also visit another arc $(i, j) \notin T$ that connects S_e with S_e^C. This implies $e \in T$, as otherwise $S_e^C = \emptyset$. Moreover, it holds that $cost(P) \geq \min\{c(v_1, i) + c_{ij} + c(j, v_n) \mid (i, j) \in (S_e \times S_e^C) \setminus T\}$.

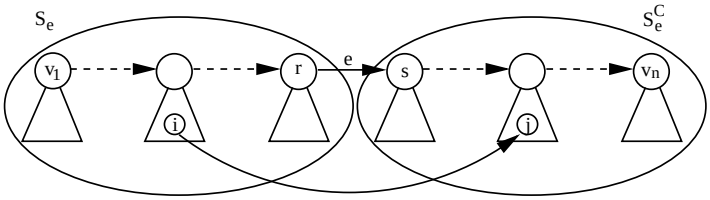

Fig. 4. The figure schematically shows a shortest-path tree T rooted at v_1. Solid lines denote arcs in G, dashed lines mark parts of the shortest path $P(v_1, v_n)$ from v_1 to v_n. The triangles symbolize shortest-path subtrees. For an edge $e = (r, s) \in P(v_1, v_n)$, the nodes in V are partitioned 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 $(i, j) \in (S_e \times S_e^C) \setminus T$.

- Like L_1, also L_2 does not force the 2-admissible paths to visit the nodes in $req(Y)$ simultaneously. Again we can improve the effectiveness of the filtering algorithm by setting $c_{ij} := 0$ for all $(i, j) \in req(Y)$ and by subtracting $cost(req(Y))$ from B.
- A shorter path constraint is relaxed L_2-consistent, iff
  1. for all $f \in pos(Y)$, there exists a 2-admissible path $P \in F_f$, and
  2. for all $f \notin req(Y)$, there exists a 2-admissible path $P \in J$ with $f \notin P$, or there exists an arc $e \in P \setminus T$ such that $e \in S_f \times S_f^C$.

We have seen that the relaxation L_2 is dominated by L_1. Nevertheless, we can show that cost-based filtering that achieves relaxed L_2-consistency is still at least as strong as ordinary reduced cost filtering:

Lemma 4. *If a shorter path constraint is relaxed L_2-consistent, reduced cost filtering is ineffective.*[4]

5.1 Relaxed L_2-Consistency

As relaxations L_1 and L_2 do not differ with respect to the definition of F_f, $f \in E$, to remove arcs from $pos(Y)$ we can simply follow the procedure sketched in Section 4.

Regarding the identification of arcs that have to be added to $req(Y)$ so as to achieve relaxed L_2-consistency, for all $e \in pos(Y) \setminus req(Y)$ we have to compute the cost of the shortest 2-simple path P from v_1 to v_n such that $e \notin P$ or such that there exists an edge $(i, j) \in P \setminus T$ with $(i, j) \in S_e \times S_e^C$, where T is a shortest-path tree in G rooted at v_1.

First, we compute the shortest paths from v_1 to v_n and v_n to v_1 in the reverse of G in time $O(m + n \log n)$. As a byproduct, we get $T \subseteq E$ and shortest-path distances $c(v_1, i)$, $c(i, v_n)$ for all $i \in V$. If $c(v_1, v_n) \geq B$, the current choice point is inconsistent, and we can backtrack. Otherwise, candidates to be added to $req(Y)$ are only the arcs $e \in P(v_1, v_n)$. Since $v_1 \in S_e$ and $v_n \in S_e^C$, the shortest 2-simple path P from v_1 to v_n with $e \notin P$ must contain an arc $(i, j) \in S_e \times S_e^C$. And since $T \cap S_e \times S_e^C = \{e\}$, we have

[4] The proof is omitted due to space restrictions. A full version of the paper can be found in [26].

that $(i,j) \notin T$. Therefore, it is sufficient to compute, for all $e \in P(v_1, v_n)$, the costs of the shortest 2-simple path P from v_1 to v_n that contains some $(i,j) \in (S_e \times S_e^C) \setminus T$.

Let $P(v_1, v_n) = (r_1, r_2, \ldots, r_h, r_{h+1})$, $h \in \mathbb{N}$, $r_1 = v_1$ and $r_{h+1} = v_n$, and denote with $(e_1, \ldots, e_h)$ the sequence of arcs that $P(v_1, v_n)$ visits, whereby $e_k = (r_k, r_{k+1})$ for all $1 \leq k \leq h$. Further, for all $1 \leq k \leq h$, denote with Q_k a shortest 2-simple path from v_1 to v_n with $(i,j) \in Q_k$ for some $(i,j) \in (S_{e_k} \times S_{e_k}^C) \setminus T$. Then,

$$cost(Q_k) = \min\{c(v_1, i) + c_{i,j} + c(j, v_n) \mid (i,j) \in (S_{e_k} \times S_{e_k}^C) \setminus T\}.$$

A brute force approach requires time $\Theta(nm)$ to determine these values. However, we can do better when we compute the values $cost(Q_k)$ for all $1 \leq k \leq h$ sequentially. Note that $S_{e_1} \subseteq \cdots \subseteq S_{e_h}$ and $S_{e_h}^C \subseteq \cdots \subseteq S_{e_1}^C$. We keep the nodes j in the current set $S_{e_k}^C$ in a min-heap, whereby the associated value of j in the heap is defined as

$$x_j := \min\{c(v_1, i) + c_{i,j} + c(j, v_n) \mid i \in S_{e_k} \text{ and } (i,j) \in E \setminus T\}.$$

Obviously, the smallest x_j in the heap determines $cost(Q_k)$. In the transition from one shortest-path arc e_k to the next e_{k+1}, the nodes $i \in S_{e_k} \setminus S_{e_{k+1}}$ have to be removed from the heap, and the values x_j must be updated. For each node $i \in S_{e_k} \setminus S_{e_k+1}$, we iterate over all outgoing arcs and perform a *decrease-key* on the adjacent nodes if necessary. Then, i is removed from the heap. Since every node in V leaves the heap at most once and never re-enters it, for all $1 \leq k \leq h$ this procedure requires at most m *decrease-key* operations and n *delete-min* operations. Therefore, when using a Fibonacci heap, the values $cost(Q_k)$ for all $1 \leq k \leq h$ can be determined in time $O(m + n \log n)$. Then, e_k is added to $req(Y)$ iff $cost(Q_k) \geq B$. It follows

Theorem 3. *On directed graphs with non-negative arc weights, relaxed L_2-consistency of a shorter path constraint can be achieved in time $O(m + n \log n)$.*

Finally, we would like to note that the results can be extended for directed graphs with no negative cycles (see [26] for proofs):

Theorem 4. *On directed graphs without negative cycles, relaxed L_1-consistency of a shorter path constraint can be achieved in time $O(n(m + n \log n))$.*

Theorem 5. *On directed graphs without negative cycles, relaxed L_2-consistency of a shorter path constraint can be achieved in time $O(nm)$. For $\Omega(\log n)$ calls to the filtering procedure with changing variable domains, relaxed L_2-consistency can be achieved in amortized time $O(m + n \log n)$.*

6 Conclusion

We summarize the results that we achieved (see Table 1): On arbitrary directed and on undirected graphs, achieving GAC is an NP-hard task. Therefore, we introduced the notion of relaxed consistency and developed two shortest path relaxations L_1 and L_2. Both relaxations are based on the class of 2-simple paths. We showed that L_1 dominates L_2, and cost-based filtering based on L_2 is superior to reduced cost filtering. On undirected graphs with non-negative edge weights, relaxed L_1-consistency (and therefore also relaxed L_2-consistency) can be achieved in time $O(m + n \log n)$ and in time

	Degree of Consistency			
Graph Type	GAC	L_1	L_2	RedCost
undirected, $c \geq 0$	NP-hard	$O(m + n \log n)$, $[RAM] O(m + n)$		
DAG	$O(m + n)$			
directed, $c \geq 0$	NP-hard	$O(n(m + n \log n))$	$O(m + n \log n)$	
directed, no negative cycles	NP-hard	$O(n(m + n \log n))$	O(nm) amort.$[\Omega(n)]$: $O(m + n \log n)$	

Table 1. The table gives an overview of the findings in this paper.

$O(m + n)$ on a RAM. On DAGs, generalized arc-consistency can be achieved in linear time. On general directed graphs with non-negative arc weights, relaxed L_1-consistency can be obtained in time $O(n(m + n \log n))$, and a state of relaxed L_2-consistency can be achieved in time $O(m + n \log n)$. Finally, in the presence of negative arc weights, we achieve relaxed L_1-consistency in time $O(n(m + n \log n))$, and L_2-consistency in time $O(nm)$ or $O(m + n \log n)$ for $\Omega(n)$ calls of the filtering algorithm with changing variable domains.

Note that these results are superior to the heuristics in [1], since we can also identify arcs that must be visited, which is a valuable information with respect to other constraints that may be present. With respect to the idea of an iterated reduction procedure as suggested in [6], we may assume that this is given by embedding the cost-based filtering algorithms in a CP solver. Regarding the tightening of lower bounds with respect to other linear constraints, e.g. as proposed in [3] for the Resource Constrained Shortest Path Problem, we refer the reader to the concept of CP-based Lagrangian relaxation presented in [27]. Finally, note that the algorithms we developed are all practicable and easy to implement (except of course the linear time shortest path algorithm on undirected graphs). Therefore, we expect this work to be relevant for many applications and practical approaches in the field of discrete optimization.

References

1. Y. Aneja, V. Aggarwal, K. Nair. Shortest chain subject to side conditions. *Networks*, 13:295-302, 1983.
2. K. R. Apt. The Rough Guide to Constraint Propagation. *Principles and Practice of Constraint Programming (CP)*, Springer LNCS 1713:1–23, 1999.
3. J. Beasley, N. Christofides. An Algorithm for the Resource Constrained Shortest Path Problem. *Networks*, 19:379-394, 1989.
4. R. Borndoerfer, A. Loebel. Scheduling duties by adaptive column generation. *Technical Report*, Konrad-Zuse-Zentrum fuer Informationstechink Berlin ZIB-01-02, 2001.
5. T.H. Cormen, C.E. Leiserson, R.L. Rivest. Introduction to Algorithms. *The MIT Press*, 1993.
6. 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.
7. 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.
8. T. Fahle, M. Sellmann. Cost-Based Filtering for the Constrained Knapsack Problem. *Annals of Operations Research*, 115:73–93, 2002.
9. S. Fortune, J. Hopcroft, J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

10. F. Focacci, A. Lodi, M. Milano. Cost-Based Domain Filtering. *Principles and Practice of Constraint Programming (CP)* Springer LNCS 1713:189–203, 1999.
11. F. Focacci, A. Lodi, M. Milano. Cutting Planes in Constraint Programming: An Hybrid Approach. *CP-AI-OR'00*, Paderborn Center for Parallel Computing, Technical Report tr-001-2000:45–51, 2000.
12. M. L. Fredmann, R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM* 34:596–615, 1987.
13. M. R. Garey, D. S. Johnson. Computers and Intractability, A Guide to the Theory of NP-Completeness. *Freeman*, San Francisco, 1979.
14. G. Handler, I. Zang. A Dual Algorithm for the Restricted Shortest Path Problem. *Networks*, 10:293-310, 1980.
15. O. Jahn, R. Moehring, A. Schulz. Optimal routing of traffic flows with length restrictions in networks with congestion. *Technical Report*, TU Berlin 658-1999, 1999.
16. H. Joksch. The Shortest Route Problem with Constraints. *Journal of Mathematical Analysis and Application*, 14:191-197, 1966.
17. U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. A Framework for Constraint programming based column generation. *Principles and Practice of Constraint Programming (CP)*, Springer LNCS 1713:261–274, 1999.
18. V. Kumar. Algorithms for Constraints Satisfaction problems: A Survey. *The AI Magazine, by the AAAI*, 13:32-44, 1992.
19. M. Luebbecke, U. Zimmermann. Computer aided scheduling of switching engines. *CASPT*, 2000.
20. K. Mehlhorn, M. Ziegelmann. Resource Constrained Shortest Paths. *Proc. 8th European Symposium on Algorithms (ESA)*, Springer LNCS 1879:326-337, 2000.
21. A. Orda. Routing with end to end QoS guarantees in broadband networks. *Conference on Computer Communications (Infocom)*, IEEE, 27-34, 1998.
22. G. Ottosson, E.S. Thorsteinsson. Linear Relaxation and Reduced-Cost Based Propagation of Continuous Variable Subscripts. *CP-AI-OR'00*, Paderborn Center for Parallel Computing, Technical Report tr-001-2000:129–138, 2000.
23. S. Pettie, V. Ramachandran. Computing undirected shortest paths using comparisons and additions. *ACM-SIAM Symposium on Discrete Algorithms*, January 2002.
24. J.C. Régin. Arc Consistency for Global Cardinality Constraints with Costs. *Principles and Practice of Constraint Programming (CP)*, Springer LNCS 1713:390–404, 1999.
25. M. Sellmann. An Arc-Consistency Algorithm for the Weighted All Different Constraint. *Principles and Practice of Constraint Programming (CP)*, Springer LNCS 2470:744–749, 2002.
26. M. Sellmann. Reduction Techniques in Constraint Programming and Combinatorial Optimization. *PhD Thesis*, University of Paderborn, Germany, http://www.upb.de/cs/sello/diss.ps, 2002.
27. 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.
28. M. Thorup. Undirected single source shortest paths in linear time. *Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE, 12–21, 1997.
29. G. Xue. Primal-dual algorithms for computing weight-constrained shortest paths and weight-constrained minimum spanning trees. *International Performance, Computing, and Communications Conference (IPCCC)*, IEEE, 271-277, 2000.
30. T. H. Yunes, A. V. Moura, C. C. Souza. A hybrid approach for solving large crew scheduling problems. *International Workshop on Practical Aspects of Declarative Languages (PADL)*, Springer LNCS 1753:293-307, 2000.