

Unified Label-Setting Algorithm for variants of the Shortest Path Problem with Resource Constraints

Bjørn Petersen
bjorn@diku.dk

*DIKU Department of Computer Science, University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark*

November 20, 2006

Abstract

The Shortest Path Problem with Resource Constraints is a common subproblem in many Branch-Cut-and-Price algorithms (BCP), e.g., routing and scheduling problems. This paper presents a general label-setting algorithm for solving various shortest path problems. An integration of the ‘Shortest Path Problem with Resource Constraints and k -cycle elimination’ with the ‘Partial Elementarity’ is introduced, and so is a bi-directional label-setting algorithm for solving general Shortest Path Problem with Resource Constraints. Moreover it is shown how to handle the so-called *weighted subset penalties*; which say that given a subset of nodes T each weighted with $w_t \in \mathbb{Q} : \forall t \in T$ a penalty σ has to be paid for every whole visit to T , i.e., penalty for path P is $\lfloor \sum_{i \in P \cap T} w_i \rfloor \sigma$. These additional constraints makes it possible to handle Chvatal Rank 1 cuts applied to a Set-Packing like master problem of a BCP algorithm.

Keywords: Label Setting, Shortest Path Problem with Resource Constraints and k -cycle Elimination, Decremental State Space Relaxation, Vehicle Routing Problem, weighted subset row inequalities

1 Introduction

The Shortest Path Problem with Resource Constraints (SPPRC) can be stated as: Given a weighted directed graph $G(V, E)$ with nodes V and edges E , and a set of resources R . For each edge $e \in E$ and resource $r \in R$ three parameters are given: A lower limit $a_r(e)$ on the accumulation of resource r when traversing edge $e \in E$; an upper limit $b_r(e)$ on the accumulation of resource r when traversing edge $e \in E$; and an amount $c_r(e)$ of resource r consumed by traversing edge $e \in E$. In general $c_r(e)$ can be a function and can also be dependent on other resources, e.g., $c_r(e, r_1, r_2) : r_1, r_2 \in R$, but will for ease of notation be denoted $c_r(e)$ throughout this paper. The objective is to find a minimum cost path P , i.e., minimize the cost resource \bar{c} , from a source node $o \in V$ to a target node $o' \in V$, where the accumulated resources of P satisfy the limits for all resources $r \in R$. Without loss of generality we assume that the limits must be satisfied at the start of each edge e , i.e., before $c_r(e)$ has been consumed.

It is noted that equivalent upper and lower limits and consumptions on the nodes can be “pushed” onto the edges, e.g., the ingoing edges of the node.

An additional Weighted Subset Penalty criteria can be added to the SPPRC, where a penalty σ_T has to be paid for every whole visit to a set $T \in \hat{T}$ of nodes in the graph, where each node $t \in T$ is

weighted with w_t – this have to be done for all $T \in \hat{T}$. That is, the cost $\hat{c}(P)$ of path P , given the nodes in P as $V(P)$ and the edges in P as $E(P)$, is

$$\hat{c}(P) = \sum_{e \in E(P)} c_{\bar{e}}(e) + \sum_{T \in \hat{T}} \sigma_T \left[\sum_{i \in V(P) \cap T} w_i \right].$$

The Shortest Path Problem with Resource Constraints and k -cycle Elimination (k -cyc-SPPRC) can be stated as the SPPRC but with an additional constraint that the path is k -cycle free. In k -cycle free paths, cycles of size k or smaller are not allowed, i.e., paths containing $(\dots, v_0, v_1, \dots, v_{k-1}, v_0, \dots)$ are forbidden.

The Elementary Shortest Path Problem with Resource Constraints (ESPPRC) can also be stated as the SPPRC but with an additional constraint that path P is cycle free, i.e., no node $v \in V$ is in P more than once.

Relaxing the ESPPRC so that all nodes does not have to be elementary gives rise to the Partial Elementary Shortest Path Problem with Resource Constraints (PESPPRC), where only a subset $S \subseteq V$ of the nodes are not allowed to be in the path more than once.

Finally the integration of k -cyc-SPPRC and PESPPRC demands that the nodes in $S \subseteq V$ are not allowed to be in the path more than once at the same time as none of the other nodes $\bar{S} = V \setminus S$ among them self form a cycle of size k or smaller. That is, paths containing $(\dots, v_0, e_0, v_1, e_1, \dots, v_{k-1}, e_{k-1}, v_0, \dots)$ where $v_i \in \bar{S} \wedge e_i \subseteq S \cup \emptyset : 0 \leq i \leq k-1$ are forbidden.

Dror (1994) showed that the ESPPRC is strongly $\mathcal{N}P$ -hard, hence a relaxation of the ESPPRC was used as a pricing problem in earlier BCP. Christofides et al. (1981) denoted the SPPRC solutions as q -routes. Later the relaxed pricing problem where non-elementary paths are allowed was denoted the Shortest Path Problem with Resource Constraints (SPPRC) by Desrochers (1986), this problem can be solved in pseudo-polynomial time, e.g., by use of label-setting algorithms. To improve lower bounds of the master problem Desrochers et al. (1992) used 2-cycle elimination which was later extended by Irnich and Villeneuve (2006) to k -cycle elimination (k -cyc-SPPRC), still with pseudo-polynomial running time. This has shown to significantly improve lower bounds of the master problem compared to the SPPRC.

Beasley and Christofides (1989) proposed to solve the ESPPRC using Lagrangian relaxation. However, recently label-setting algorithms have become the most popular approach to solve the ESPPRC, see e.g., Dumitrescu (2002) and Feillet et al. (2004). When solving the ESPPRC with a label-setting algorithm a binary resource for each node is added which increases the complexity of the algorithm compared to solving the SPPRC or the k -cyc-SPPRC. Righini and Salani (2004) developed a label-setting algorithm using the idea of Dijkstra's bi-directional shortest path algorithm that expands both forward from the source node o and backward from the target node o' and connects paths in the middle, thereby potentially reducing the running time of the algorithm. Furthermore Righini and Salani (2005) and Boland et al. (2006) proposed to solve ESPPRC by use of a decremental state space algorithm that iteratively solves a SPPRC by applying resources forcing nodes to be visited at most once. Recently Chabrier (2005), Danna and Pape (2005), and Salani (2005) successfully solved several previously unsolved instances of the VRPTW from the benchmarks of Solomon (1987) using a label-setting algorithm for the ESPPRC.

The paper is outlined as follows: In Section 2 is given a quick introduction to the concept of label setting and a thorough description of how it is applied to general shortest paths. In Section 3 the Weighted Subset Penalties are introduced which can be modeled using additional resources but can also be handled much more efficiently if their special structure is exploited. Section 4 describes

how to make the search for shortest paths bidirectional and a proof of correctness is given. In Section 5 extensive computational results on the Gehring and Homberger benchmarks and the Solomon benchmarks are shown. Finally, in Section 6, concluding remarks.

2 Label-Setting Algorithm

There already exist several articles covering the basics of solving shortest path problems by use of label-setting algorithms, so it is beyond the scope of this article to go into these details. However, a short introduction to settle the notation will be given. For a detailed description see e.g. Irnich and Desaulniers (2004).

The central part of the algorithms is the use of labels which represent partial paths rooted at node o . Each label has associated a set of attributes:

- A node to which it belongs $\bar{v} \in V$
- A pointer to the label of the parent node p
- The accumulated consumption of each resource $r \in R$ (including the cost resource \bar{c})
- Ordered set of last $k - 1$ visited nodes $\pi \subseteq \bar{S}$

Thus, a label L with $\bar{v}(L) = v$ represents a partial path from node o to node v and all the accumulated resources along the path. We will use $f(L)$ to refer to attribute f of a label. E.g. $r(L)$ refer to the accumulated consumption of resource r in label L . The parent $p(L)$ of label L is the label L_p that was extended to create L . L_p is recursively used to find the path $P(L)$ that label L represents. $V(P(L))$ (or shorthand $V(L)$) is the multiset of the predecessors and $E(P(L))$ (or shorthand $E(L)$) is the edges on P . The attributes r and π are not strictly necessary and are only present for notational and computational reasons, they can always be computed by following the chain of parent labels L_p, L_{p-1}, \dots, L_o back to the starting node o .

In the following it is assumed that all resources are bounded strongly from above, and weakly from below, i.e., if the current resource accumulation is below the lower limit on a given edge e , it is allowed to fill up the resource to the lower limit, e.g. waiting for a time window to open. This means that two consecutive labels L_u and L_v related by an edge $e = (u, v)$, i.e., L_u is extended and creates L_v , where $\bar{v}(L_u) = u$ and $\bar{v}(L_v) = v$, must satisfy

$$r(L_v) \leq b_r(e) \quad \forall r \in R \quad (1)$$

$$r(L_v) = \max\{r(L_u) + c_r(e), a_r(e)\} \quad \forall r \in R \quad (2)$$

$$v \neq w \quad \forall w \in \pi \quad (3)$$

Here (1) demands that L_v satisfies the upper limit of resource r corresponding to edge $e = (u, v)$, while (2) states that resource r at label L_v corresponds to the resource consumption at label L_u plus the amount consumed by traversing edge e , respecting the lower limit on edge e and (3) secures no cycles of size smaller than k .

The concept of a label-setting algorithms is to iteratively extend labels in the following way until there are no more labels left. When a label has been extended it is considered treated:

```

LABEL-SETTING( $G, o, o'$ )
1   $L_{init} \leftarrow \text{FIRST-LABEL}(o)$ 
2   $PQ.\text{ENQUEUE}(L_{init})$ 
3  while  $PQ \neq \emptyset$ 
4      do REMOVE-DOMINATED( $PQ$ )
5       $L \leftarrow PQ.\text{DEQUEUE}()$ 
6      for each node  $v \in \text{EXTENDABLES}(L)$   $\triangleright$  Nodes to which  $L$  can be extended
7          do  $L_v \leftarrow \text{EXTEND-LABEL}(L, v)$ 
8              if  $\bar{v}(L_v) = o'$ 
9                  then STORE-SOLUTION( $L_v, sol$ )
10                 else  $PQ.\text{ENQUEUE}(L_v)$ 
11 return  $sol$ 

```

From the pseudocode it is clear that without REMOVE-DOMINATED in line 4 this results in a complete enumeration of all feasible paths.

2.1 Dominance

The goal of dominance is to reduce the number of labels that are created during the execution of the label-setting algorithm, since it is not desirable to extend labels that are not part of an optimal solution. Unfortunately it is not known in advance which labels span optimal solutions, but it might be possible to decide for some labels that they are not part of any optimal solution. If just any optimal solution is sought, dominance is to reduce the number of labels extended and still be able to find an optimal solution. A label is thus said to be dominated if its removal during the run of the algorithm does not remove all optimal solution.

In the following it is assumed that all the extension functions $c_r(e)$ are non-decreasing. A non-decreasing function $c_r(e)$ has the property:

Definition 1. A function f is non-decreasing iff:

$$x \leq y \Rightarrow f(x) \leq f(y) \quad \forall x, y$$

In relation to dominance it is necessary to consider extensions of labels. For this reason three definitions are presented (slightly modified from Irnich and Villeneuve (2006)):

Definition 2. The set of all feasible paths from label L to node u considering the resource consumption of label L is defined as $\mathcal{F}(L, u)$.

Definition 3. The set of all feasible paths from label L to node u considering the k -cycle elimination and the partial elementarity is defined to be $\mathcal{S}(L, u)$.

Definition 4. All feasible extensions of label L is defined as:

$$\mathcal{E}(L) = \mathcal{F}(L, t) \cap \mathcal{S}(L, t)$$

With Definition 4 as a building block the following definition of domination is now given:

Definition 5. A set of labels \mathcal{L}_i dominates label L_j if:

$$\bar{v}(L_i) = \bar{v}(L_j) \quad \forall L_i \in \mathcal{L}_i \quad (4)$$

$$\bar{c}(L_i) \leq \bar{c}(L_j) \quad \forall L_i \in \mathcal{L}_i \quad (5)$$

$$\mathcal{E}(L_j) \subseteq \bigcup_{L_i \in \mathcal{L}_i} \mathcal{E}(L_i) \quad (6)$$

In other words, the paths corresponding to labels in \mathcal{L}_i and the path L_j should end at the same node $\bar{v}(L_i) = \bar{v}(L_j) \in V : \forall L_i \in \mathcal{L}_i$, each path corresponding to some label $L_i \in \mathcal{L}_i$ should cost no more than the path corresponding to label L_j , and finally any feasible extension of L_j is also a feasible extension of some $L_i \in \mathcal{L}_i$.

Definition 5 implies that if L_j is dominated then any path $P(L_j, \epsilon)$ consisting of L_j concatenated with a feasible extension $\epsilon \in \mathcal{E}(L_j)$ is not a unique optimal solution, since at least one other label $L_i \in \mathcal{L}_i$ can also be concatenated with ϵ and make a path $P(L_i, \epsilon)$ that is as least as cheap, because $\bar{c}(L_i) + c_{\bar{c}}(\epsilon, r(L_i)) \leq \bar{c}(L_j) + c_{\bar{c}}(\epsilon, r(L_j))$ due to Definition 1, that is:

$$\begin{aligned} & \forall \epsilon \in \mathcal{E}(L_j) \quad \exists L_i \in \mathcal{L}_i : \epsilon \in \mathcal{E}(L_i) \wedge \bar{c}(L_i) \leq \bar{c}(L_j) \\ \Rightarrow & \forall \epsilon \in \mathcal{E}(L_j) \quad \exists L_i \in \mathcal{L}_i : \epsilon \in \mathcal{E}(L_i) \wedge \bar{c}(P(L_i, \epsilon)) \leq \bar{c}(P(L_j, \epsilon)) \end{aligned}$$

Each node in the set of elementary nodes S can be modeled using a binary resource. Feillet et al. (2004) suggested to consider the set of nodes in S that cannot be reached from a label L_i and compare the set with the unreachable nodes of a label L_j in order to determine if some extensions are impossible. Or in other words: update the node resources in an eager fashion instead of a lazy. The following definition is a generalization of (Feillet et al. 2004, Definition 3).

Definition 6. Given a start node $o \in V$ and a label L with $\bar{v}(L) = u$, a node $v \in V$ is considered unreachable if v has already been visited on the path from o to u , i.e., $v \in V(L)$ or if a resource window is violated, e.g.:

$$\exists r \in R \quad r(L) + \ell_r(u, v) > b_r(v)$$

where $\ell_r(u, v)$ is a lower bound on the consumption of resource r on all feasible paths from u to v . The node resources are then given as: $v(L) = 1$ indicates that node $v \in V$ is unreachable from node $\bar{v}(L) \in V$, and $v(L) = 0$ otherwise.

In the following $\overline{\mathcal{E}}(\pi(L))$ (or shorthand $\overline{\mathcal{E}}(L)$) will be the set of all feasible extension for label L only considering the k -cycle elimination constraint, and is equivalent to the concept of *Hole-Sets* as defined by Irnich and Villeneuve (2006). The \subseteq -operator and the \bigcup -operator are also as defined by Irnich and Villeneuve (2006).

To determine if (6) holds can be quite cumbersome, as the straightforward definition demands that we calculate all extensions of the involved labels. Therefore a sufficient criteria for (6) is sought which can be computed faster. If label L_i has consumed less resources than label L_j then no resources are limiting the possibilities of extending L_i compared to L_j , hence the following proposition can be used as a relaxed version of the dominance criteria in Definition 5.

Proposition 7 (Sufficient condition). *A set of labels \mathcal{L}_i dominates label L_j if:*

$$\bar{v}(L_i) = \bar{v}(L_j) \quad \forall L_i \in \mathcal{L}_i \quad (7)$$

$$r(L_i) \leq r(L_j) \quad \forall r \in R, \forall L_i \in \mathcal{L}_i \quad (8)$$

$$\bar{\mathcal{E}}(L_j) \subseteq \bigcup_{L_i \in \mathcal{L}_i} \bar{\mathcal{E}}(L_i) \quad (9)$$

and node resources are set according to Definition 6.

Proof. We check Definition 5. Equation (4) follows directly from (7), and (5) follows from (8) with $r = \bar{c}$, i.e., the cost resource. The remaining concern is if (6) holds for \mathcal{L}_i and L_j . The proof is by contradiction. Assume that (7), (8) and (9) are satisfied but that (6) is not satisfied. Then there must exist an extension $\varepsilon \in \mathcal{E}(L_j) \setminus \bigcup_{L_i \in \mathcal{L}_i} \mathcal{E}(L_i)$, i.e., ε is feasible for L_j but not for any $L_i \in \mathcal{L}_i$. Let L^u denote the label that is obtained with $\bar{v}(L^u) = v_u$ after L has recursively been extended through ε , let L^u be equivalently defined, let $v_1, \dots, v_{h-1}, v_h, \dots$ be the nodes on ε and let v_h be the first node on ε preventing the extension of all $L_i^{h-1} \in \mathcal{L}_i^{h-1}$. There are only three conditions where this can happen for each $L_i^{h-1} \in \mathcal{L}_i^{h-1}$:

- 1) $v_h(L_i^{h-1}) = 1$
- 2) $\exists r \in R, \quad r(L_i^{h-1}) + l_r(v_{h-1}, v_h) > b_r(h)$
- 3) $\varepsilon \notin \bar{\mathcal{E}}(L_i)$

Since L_j can be extended with ε the, equivalent conditions for L_j^{h-1} are:

- 1) $v_h(L_j^{h-1}) = 0$
- 2) $r(L_j^{h-1}) + c_r(v_{h-1}, v_h) \leq b_r(h), \quad \forall r \in R$
- 3) $\varepsilon \in \bar{\mathcal{E}}(L_j)$

Since all resources are consumed according to Definition 1 on ε until v_{h-1} for all $L_i \in \mathcal{L}_i$ and L_j , the above conditions contradicts that (7), (8) and (9) are satisfied. Hence, $\mathcal{E}(L_j) \setminus \bigcup_{L_i \in \mathcal{L}_i} \mathcal{E}(L_i) = \emptyset$ which implies $\mathcal{E}(L_j) \subseteq \bigcup_{L_i \in \mathcal{L}_i} \mathcal{E}(L_i)$ and (6) holds. That is, Definition 5 holds and \mathcal{L}_i dominates L_j . \square

Using Proposition 7 as a dominance criteria is a relaxation of the dominance criteria of Definition 5 since only a subset of labels satisfying (4), (5) and (6) satisfies inequalities (7), (8) and (9).

It is noted that Condition (8) can be tightened by being lazy with $r(L_i)$ and eager with $r(L_j)$. Furthermore, if $k \leq 1$ Condition (9) is automatically satisfied so $|\mathcal{L}_i| = 1$.

Decreasing extension-functions can always be handled by use of equality on the affected resources, but can be tightened if a lower and an upper bound is known, see Pisinger and Reinhardt (2006) for further details.

Being more aggressive in REMOVE-DOMINATED, i.e., removing non-dominated labels, yields a heuristic solution but with likely improved running time.

3 Weighted Subset Penalties

The Subset Penalties originates from SR–inequalities in the master problem in a Branch-Cut-and-Price (BCP) algorithm that needs to be handled in the pricing problem, i.e., in an SPPRC, see Jepsen et al. (2006). The Weighted Subset Penalties are a generalization of these and can handle Chvatal Rank 1 cuts:

$$\sum_{p \in P} \left[\sum_{i \in T} w_i \alpha_{ip} \right] \lambda_p \leq \left[\sum_{i \in T} w_i \right] \quad (10)$$

Given a set of subsets of nodes $\hat{T} : T \in \hat{T} \wedge T \subseteq V$, for each $T \in \hat{T}$ a penalty σ_T has to be paid for every whole visit to T , where each node $t \in T$ is weighted with w_t .

Definition 8. A *Weighted Subset Penalty* is an extra cost σ_T that have to be paid for path P for every whole visit to a set $T \in V$ of weighted nodes resulting in an extra cost of:

$$\bar{c}_T(P) = \sigma_T \left[\sum_{i \in V(P) \cap T} w_i \right] \quad (11)$$

P can contain cycles so $V(P)$ can be a multiset (several occurrences of the same element). The \cap -operator on multisets here results, for n_i occurrences of element $e \in I$ and n_j occurrences of element $e \in J$, in $n_i \times n_j$ occurrences of element $e \in I \cap J$.

We analyze how these additional costs \bar{c}_T for $\forall T \in \hat{T}$ can be handled in the label-setting algorithm. The cost of a label L can be expressed as:

$$\hat{c}(L) = \bar{c}(L) + \sum_{T \in \hat{T}} \sigma_T \left[\sum_{i \in V(L) \cap T} w_i \right] \quad (12)$$

A new resource m_T can be used to compute the coefficient of penalty σ_T for label L , i.e., $m_T(L) = \sum_{i \in V(L) \cap T} w_i$, the weighted amount of visit to T along partial path L . Note that the consumption of resource m_T is w_i for each e.g. outgoing edge of the involved nodes $i \in T$. Therefore the usual dominance criteria of Proposition 7 can be used. Note that in case L_i dominates L_j then for all $L_i \in \mathcal{L}_i$ holds: $\bar{c}(L_i) \leq \bar{c}(L_j)$ and $m_T(L_i) \leq m_T(L_j) \forall T \in \hat{T}$ so $\hat{c}(L_i) \leq \hat{c}(L_j)$ since $\sigma_T > 0$ for all $T \in \hat{T}$. Hence the penalty term must only be considered on the last edge to the target node to compute the reduced cost $\hat{c}(P)$ of path P . This will not ruin the non-decreasing structure of the cost functions, however, further labels can be eliminated by exploiting the structure of (12).

For a label L let

$$\tau_T(L) = \left(\sum_{i \in V(L) \cap T} w_i \right) \bmod 1$$

be the number of weighted visits made to T since the last penalty was paid for visiting T a whole number of times. Recall $\mathcal{E}(L)$ as the set of feasible extensions from the label L to the target node o' and note that when the set of labels \mathcal{L}_i dominates label L_j , their common extensions are $\mathcal{E}(L_j)$ due to (6). The following cost dominance criteria is obtained for a comparison of two labels and a single Weighted Subset Penalty:

Proposition 9. If $\tau(L_i) \leq \tau(L_j)$ and $\hat{c}(L_i) \leq \hat{c}(L_j)$, then label L_i dominates label L_j on cost.

Proof. Consider any common extension $\varepsilon \in \mathcal{E}(L_j)$. The comparison of the number of future penalties for the two labels is:

$$\left\lfloor \sum_{i \in \varepsilon \cap T} w_i + \mathcal{T}(L_i) \right\rfloor \leq \left\lfloor \sum_{i \in \varepsilon \cap T} w_i + \mathcal{T}(L_j) \right\rfloor$$

since $\mathcal{T}(L_i) \leq \mathcal{T}(L_j)$. Hence L_j will always pay at least the same penalties as L_i using their common extensions and the current cost $\hat{c}(L_i)$ of L_i can be used during dominance. \square

Proposition 10. *If $\mathcal{T}(L_i) > \mathcal{T}(L_j)$ and $\hat{c}(L_i) + \sigma_T \leq \hat{c}(L_j)$, then label L_i dominates label L_j on cost.*

Proof. Consider any common extension $\varepsilon \in \mathcal{E}(L_j)$. The comparison of the number of future penalties for the two labels are:

$$\left\lfloor \sum_{i \in \varepsilon \cap T} w_i + \mathcal{T}(L_i) \right\rfloor \geq \left\lfloor \sum_{i \in \varepsilon \cap T} w_i + \mathcal{T}(L_j) \right\rfloor$$

since $0 \leq \mathcal{T}(L_j) < \mathcal{T}(L_i) < 1$. Applying an additional penalty to L_i gives:

$$1 + \left\lfloor \sum_{i \in \varepsilon \cap T} w_i + \mathcal{T}(L_i) \right\rfloor \leq \left\lfloor \sum_{i \in \varepsilon \cap T} w_i + \mathcal{T}(L_j) \right\rfloor$$

Adding one additional penalty σ_T to the cost of L_i during dominance ensures that L_j always pays at least the same penalties as L_i using their common extensions. \square

Observe that, if $\mathcal{T}(L_i) + \sum_{i \in \varepsilon \cap T} w_i < 1$ for all $\varepsilon \in \mathcal{E}(L_j)$, it is not possible to visit T enough to trigger a penalty, i.e., the temporary penalty to the cost of L_i can be disregarded.

The new dominance criteria is as follows:

Proposition 11. *Let $Q = \{q : \mathcal{T}_q(L_i) > \mathcal{T}_q(L_j)\}$. Then labels \mathcal{L}_i dominates label \mathcal{L}_j if:*

$$\bar{v}(L_i) = \bar{v}(L_j) \quad \forall L_i \in \mathcal{L}_i \quad (13)$$

$$\bar{c}(L_i) - \sum_{q \in Q} \sigma_q \leq \bar{c}(L_j) \quad \forall L_i \in \mathcal{L}_i \quad (14)$$

$$r(L_i) \leq r(L_j) \quad \forall r \in R \setminus \{\bar{c}\}, \forall L_i \in \mathcal{L}_i \quad (15)$$

$$\bar{\mathcal{E}}(L_j) \subseteq \bigcup_{L_i \in \mathcal{L}_i} \bar{\mathcal{E}}(L_i) \quad (16)$$

Proof. The validity of (14) follows directly from Propositions 9 and 10. The validity of (13), (15) and (16) is equivalent to the proof of Proposition 7. \square

4 Bidirectional Search

The concept of bidirectionality is to look for the shortest path from node o to node o' by finding paths from o to ‘the middle’ and ‘reverse paths’ from o' to ‘the middle’. The paths meeting in ‘the middle’ is then spliced together, and thereby a shortest path is obtained. ‘The middle’ is defined by the consumption of a monotone resource r_{mono} , i.e. $c_{r_{mono}}$ is either non-negative or non-positive with only non-zero cycles.

The reason for doing this for ESPPRC is to halve the exponential factor in the worst case number of labels, e.g. $O(V!2^V)$ can be reduced to $O(\frac{V}{2}!2^{V/2})$ by selecting r_{mono} as the number of visited nodes. For k -cyc-SPPRC and pure SPPRC the theoretical worst case number of labels is not affected but a better practical running time is hoped for. For PESPPRC the running time is dependent on the number of nodes in S and will be somewhere in between the running time for SPPRC and ESPPRC.

The bidirectional algorithm consist of the following three parts:

- Finding the shortest *forward* path from s to t the same time as finding the shortest *backward* ‘reverse path’ from t to s .
- Combine a forward label L_f and backward label L_b with $\bar{v}(L_f) = \bar{v}(L_b)$ to obtain a path $P(L_f, L_b)$.
- Stop at the ‘middle’, e.g. stop when the consumption of resource r_{mono} in a label reaches $x_{stop} = \lceil \max_{v \in V} (b_{r_{mono}}(v)) / 2 \rceil$.

Forward and Backward Paths

The algorithm from Section 2 and Section 3 can be reversed by starting with a label L_b in node t with the consumption of each resource set to the upper bound $r(L) = b_r(t)$ for all $r \in R$. Then go towards node s and treat extensions and dominance equivalently – this of cause is only possible if there exist an inverse of the extension function. The algorithm from Section 2 and Section 3 will be referred to as the *forward algorithm* and the reversed counterpart will be referred to as the *backward algorithm*. Using equivalent argumentation as for the forward algorithm it is clear that the backward algorithm also yields optimal solutions to the problems.

The bidirectional algorithm works by running a forward algorithm together with a backward algorithm keeping two sets of labels: The forward labels \mathcal{L}^f and the backward labels \mathcal{L}^b . The following pseudocode shows how the bidirectional algorithm works.

```

BIDIRECTIONAL-LABEL-SETTING( $G, s, t$ )
1  $L_s \leftarrow \text{FIRST-LABEL-FORWARD}(s)$ 
2  $L_t \leftarrow \text{FIRST-LABEL-BACKWARD}(t)$ 
3  $PQ_f.\text{ENQUEUE}(L_s)$ 
4  $PQ_b.\text{ENQUEUE}(L_t)$ 
5 while  $PQ_f \neq \emptyset$  or  $PQ_b \neq \emptyset$ 
6   do if  $PQ_f.\text{MIN}().\text{VALUE}() < x_{stop} - PQ_b.\text{MAX}().\text{VALUE}()$ 
7     then  $\text{REMOVE-DOMINATED}(PQ_f)$ 
8        $L \leftarrow PQ_f.\text{DEQUEUE}()$ 
9     else  $\text{REMOVE-DOMINATED}(PQ_b)$ 
10       $L \leftarrow PQ_b.\text{DEQUEUE}()$ 
11   for each node  $v \in \text{EXTENDABLES}(L)$ 
12     do  $L_v \leftarrow \text{EXTEND-LABEL}(L, v)$ 
13     for each label  $L \in \text{SPLICEABLE}(L_v)$ 
14       do  $path \leftarrow \text{SPLICE}(L_v, L)$ 
15          $\text{STORE-SOLUTION}(path, sol)$ 
16 return  $sol$ 

```

The functions in the pseudocode have knowledge about the direction (forward or backward) and behaves accordingly. Line 6 lets the two directions grow in parallel.

It is clear that the algorithm will find at least two optimal paths. One is found going forward and one is found going backwards. These two paths may be identical.

Splicing the Paths

At any time during the execution of the algorithm above there are two sets of labels $\mathcal{L}_v^f : L \in \mathcal{L}^f \wedge \bar{v}(L) = v$ and $\mathcal{L}_v^b : L \in \mathcal{L}^b \wedge \bar{v}(L) = v$ belonging to each node $v \in V$. Consider a label $L_f \in \mathcal{L}_v^f$ and a label $L_b \in \mathcal{L}_v^b$. If the sub-path L_b is in the extension $L_b \in \mathcal{E}(L_f)$ of L_f , the two labels can be combined to form a feasible solution P , this is denoted *splicing*. Since a path may use several nodes, a given path P may be the product of several different splicings, e.g. one for each of the $|P|$ nodes in P .

For obvious reasons it is desirable only to get unique paths, so when searching for a path P two labels $L_f \in \mathcal{L}_v^f$ and $L_b \in \mathcal{L}_v^b$ in P is only spliced when at an unique node $v \in V(P)$. One way to find this unique node v to splice at was proposed by Righini and Salani (2004) and is defined as the node $v \in V(P)$ where L_f and L_b are as close as possible to having the same consumption of r_{mono} , a tie is broken arbitrarily, e.g. L_f takes priority.

Following we propose another way to find the unique node v . If more than half the upper limit

$$x_{stop} = \left\lceil \frac{\max_{v \in V}(b_{r_{mono}}(v))}{2} \right\rceil$$

of resource r_{mono} is consumed on path P , one edge $(i, j) \in E(P)$ either crosses x_{stop} or ends at x_{stop} , choosing node j as the splicing point for P will be unique. If the consumption of $r_{mono}(P) \leq x_{stop}$ choosing the first (or the last) node of P as splicing point will be unique.

Stop at the Middle

It is clear that at least one sub-path from the forward algorithm or one sub-path from the backward algorithm has to contain the edge $(i, j) \in E(P)$ that crosses the middle if it is crossed, and at least one of them has to contain the first (or last) edge.

From the description of splicing nodes above, it is clear that there is no reason, for the algorithm without the responsibility, to extend a label if a consumption of more than x_{stop} will be obtained. For the algorithm with the responsibility, there is no reason to extend a label further when a consumption of x_{stop} is obtained. Therefore both algorithms can be stopped early, and the optimal path P is found.

Proposition 12. *The bidirectional algorithm returns an optimal solution for any value of x_{stop} .*

Proof. Without loss of generality assume that the forward algorithm crosses x_{stop} if $r_{mono}(P) > x_{stop}$, the last node is chosen for splicing if $r_{mono}(P) \leq x_{stop}$, and the optimal path P is unique. Let $P = v_1 \rightarrow \dots \rightarrow v_n$, let $L_f^i : \bar{v}(L_f^i) = v_i, \forall v_i \in V(P)$ be the labels representing P for the forward algorithm, and let $L_b^i : \bar{v}(L_b^i) = v_i, \forall v_i \in V(P)$ be the labels representing P for the backward algorithm.

The proof is by contradiction. Assume that the optimal path P is not found. This can only happen in three cases:

- 1) For some node $v \in V(P)$ neither L_f^v nor L_b^v is created.
- 2) For some node $v \in V(P)$ neither L_f^v nor L_b^v exist after domination.
- 3) There is no node $v \in V(P)$ where both L_f^v and L_b^v exist after domination.

It will now be show that none of the three cases can happen.

Since both the forward and the backward algorithm find P , for ‘Case 1’ to happen, the stopping criteria must have stopped both of them before node v was reached. This means that $r_{mono}(L_f^v) > x_{stop} > r_{mono}(L_b^v) \Rightarrow L_f^v \notin E(L_b^v)$ which contradict that P is feasible.

For ‘Case 2’ to happen at least one of L_f^v and L_b^v must have been deleted during domination, which is in contradiction with Definition 5 or that P is unique and optimal.

‘Case 3’ can be divided into two case: one where $r_{mono}(P) \leq x_{stop}$ and another where $r_{mono}(P) > x_{stop}$.

If $r_{mono}(P) \leq x_{stop}$, then the splicing must be done at v_n . L_b^n clearly exist, so L_f^n must be absent. This can only happen when $r_{mono}(L_f^{n-1}) > x_{stop}$ which contradict that $r_{mono}(P) \leq x_{stop}$.

If $r_{mono}(P) > x_{stop}$, then there must be a node $v_i \in V(P)$ where L_b^i cannot be extended more due to $r_{mono}(L_b^i) - r_{mono}(e(v_{i-1}, v_i)) = r_{mono}(L_b^{i-1}) < x_{stop}$. Since L_f^i does not exists, $r_{mono}(L_f^{i-1}) > x_{stop}$. This means that $r_{mono}(L_f^{i-1}) > x_{stop} > r_{mono}(L_b^{i-1}) \Rightarrow L_f^{i-1} \notin E(L_b^{i-1})$ which contradicts that P is feasible. \square

Since any value of x_{stop} yields an optimal solution, x_{stop} can be adjusted to balance the amount of labels created by the forward and the backward algorithms respectfully. As long as $x_{stop} \leq \min(PQ_b)$ the value of x_{stop} can be raised.

5 Computational Results

The results in Table 1 were obtained by running the bi-directional algorithm with $k = 3$ in conjunction with CVPR in a branch-and-cut-and-price framework.

Instance	Labels		Time (s)		Cliques
	Mono	Bi	Mono	Bi	

A-n44-k6	2677	2370	0.32	0.23	0
A-n44-k6	8558	5868	3.01	1.53	14
A-n45-k7	4540	3928	0.70	0.50	0
A-n45-k7	12290	8441	4.61	2.48	16
A-n53-k7	5576	5330	1.18	0.80	0
A-n53-k7	17020	12897	12.69	5.61	17
B-n45-k6	14344	12853	5.57	3.26	0
B-n45-k6	54738	33469	126.98	32.70	32
P-n55-k15	6653	6875	1.12	0.94	0
P-n55-k15	9839	9576	3.31	2.50	13
P-n60-k10	4149	3367	0.58	0.37	0
P-n60-k10	18303	8985	5.43	1.81	34

Table 1: k -cyc-SPPRC with $k=3$

In Table 2 is shown the results obtained by running the bi-directional algorithm with elementarity in conjunction with VPRTW in a branch-and-cut-and-price framework.

Instance	Labels		Time (s)		Cliques
	Mono	Bi	Mono	Bi	
r206-i1	10196	7642	0.37	0.76	0
r206-i2	9005	7165	0.30	0.65	0
r206-i3	8809	6904	0.31	0.58	0
r206-i4	20811	12854	1.34	1.87	14
r206-i5	27491	13833	2.28	2.63	14
r206-i6	77546	41435	17.39	12.68	42
r206-i7	243204	96942	304.67	65.87	64
r206-i8	159785	69437	109.27	38.31	66
r206-i9	149572	66405	92.91	36.65	60
r206-i10	152166	66873	100.10	37.57	63
r206-i11	150864	66904	96.63	37.56	63
r206-i12	169239	71288	130.43	45.17	74
c203-i1	159655	57619	107.22	10.27	0
c203-i2	220308	63737	222.76	12.50	0
c203-i3	119710	40551	31.73	6.82	0
c203-i4	81265	29904	16.23	3.37	0
c203-i5	150753	45410	77.20	6.40	0
c203-i6	69351	27717	6.71	2.51	0
c203-i7	71317	27863	7.51	2.38	0

Table 2: ESPPRC

6 Concluding Remarks

Handling Subset Penalty Constraints does not come for free, but in hard problems they are worth the effort.

Bi-directional algorithm improves the running time of the pricing problem in most instances especially the difficult.

References

- Beasley, J.E., N. Christofides. 1989. An algorithm for the resource constrained shortest path problem. *Networks* **19** 379–394.
- Boland, N., J. Dethridge, I. Dumitrescu. 2006. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operation Research Letters* (1) 58–68.
- Chabrier, A. 2005. Vehicle routing problem with elementary shortest path based column generation. In press: *Computers and Operations Research*.
- Christofides, N., A. Mingozzi, P. Toth. 1981. State space relaxation for the computation of bounds to routing problems. *Networks* **11** 145–164.
- Danna, E., C. Le Pape. 2005. *Accelerating branch-and-price with local search: A case study on the vehicle routing problem with time windows*, chap. 3 in *Column Generation*, G. Desaulniers, J. Desrosiers, and M. M. Solomon (editors). Kluwer Academic Publishers, 90–130.
- Desrochers, M. 1986. La fabrication d’horaires de travail pour les conducteurs d’autobus par une méthode de génération de colonnes. Ph.D. thesis, Centre de recherche ser les Transport, Université de Montréal, Montréal, Canada. Publication #470, in French.
- Desrochers, M., J. Desrosiers, M. Solomon. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* **40** 342–354.
- Dror, M. 1994. Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research* **42** 977–979.
- Dumitrescu, I. 2002. Constrained path and cycle problems. Ph.D. thesis, Department of Mathematics and Statistics, University of Melbourne, Australia.
- Feillet, D., P. Dejax, M. Gendreau, C. Gueguen. 2004. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks* **44**(3) 216–229.
- Irnich, S., G. Desaulniers. 2004. Shortest path problems with resource constraints. Tech. rep., HEC. Les Cahiers du GERAD.
- Irnich, S., D. Villeneuve. 2006. The shortest path problem with resource constraints and k -cycle elimination for $k \geq 3$. Forthcoming in: *INFORMS Journal of Computing*.
- Jepsen, Mads, Bjørn Petersen, Simon Spoorendonk, David Pisinger. 2006. A non-robust branch-and-cut-and-price algorithm for the vehicle routing problem with time windows. Tech. Rep. 06-03, Department of Computer Science (DIKU), University of Copenhagen, Denmark, Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark.
- Pisinger, D., L. B. Reinhardt. 2006. Multi-objective non-additive shortest path. Submitted.
- Righini, G., M. Salani. 2004. Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. Tech. Rep. 66, Note del Polo - Ricerca, Dipartimento di Tecnologie dell’Informazione, Università degli studi di Milano. Submitted to *Discrete Applied Mathematics*.
- Righini, G., M. Salani. 2005. New dynamic programming algorithms for the resource constrained shortest path problem. Tech. Rep. 69. Submitted to *Networks*.

- Salani, M. 2005. Branch-and-price algorithms for vehicle routing problems. Ph.D. thesis, Università Degli Studi Di Milano, Facoltà di Scienza Matematiche, Fisiche e Naturali Dipartimento di Tecnologie dell'Informazione, Milano, Italy.
- Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research* **35** 234–265.