Improved CP-Based Lagrangian Relaxation Approach with an Application to the TSP

Raphaël Boudreault and Claude-Guy Quimper
Université Laval, Québec, Canada
raphael.boudreault.1@ulaval.ca, claude-guy.quimper@ift.ulaval.ca

Abstract

CP-based Lagrangian relaxation (CP-LR) is an efficient optimization technique that combines cost-based filtering with Lagrangian relaxation in a constraint programming context. The state-of-the-art filtering algorithms for the WEIGHTEDCIRCUIT constraint that encodes the traveling salesman problem (TSP) are based on this approach. In this paper, we propose an improved CP-LR approach that locally modifies the Lagrangian multipliers in order to increase the number of filtered values. We also introduce two new algorithms based on the latter to filter WEIGHTEDCIRCUIT. The experimental results on TSP instances show that our algorithms allow significant gains on the resolution time and the size of the search space when compared to the state-of-the-art implementation.

1 Introduction

In constraint programming (CP), an efficient way to perform domain filtering for an optimization problem is cost-based filtering [Focacci et al., 1999]. Given a minimization problem and an upper bound on the objective value, the cost of a relaxed subproblem is used as lower bound. If assigning a variable to a value increases this lower bound beyond the upper bound, this assignment is inconsistent and the value is filtered out from the variable domain. In linear programming, Lagrangian relaxation is a common technique to obtain lower bounds. Difficult constraints are moved into the objective function while adding weights, called Lagrangian multipliers, that penalize the objective when these constraints are violated. Maximizing the objective function over the multipliers provides better lower bounds. During this optimization process, one could apply cost-based filtering to each of the resulting subproblems. From this idea, CP-based Lagrangian relaxation (CP-LR) was introduced [Sellmann and Fahle, 2001] and used to solve many problems [Fahle and Sellmann, 2002; Sellmann and Fahle, 2003; Menana and Demassey, 2009; Bergman et al., 2015; Cambazard and Fages, 2015]. In particular, the state-of-the-art filtering algorithms for the WEIGHTEDCIRCUIT constraint that encodes the traveling salesman problem (TSP) are based on a CP-LR approach [Benchimol et al., 2012].

We propose an improved CP-LR approach that adds a step before applying the cost-based filtering. This step globally increases the number of filtered values by locally modifying the Lagrangian multipliers. It results in a stronger filtering and a greater pruning of the search tree.

Section 2 presents the theory of cost-based filtering, Lagrangian relaxation, CP-LR, the TSP, and the WEIGHTEDCIRCUIT constraint. Section 3 describes our improved CP-LR approach. Section 4 presents two new algorithms based on this approach to filter the WEIGHTEDCIRCUIT constraint. Experiments on the TSP (Section 5) show a significant gain on both the resolution time and the size of the search space when compared to the state-of-the-art implementation of WEIGHTEDCIRCUIT [Jussien et al., 2008; Fages, 2014].

2 Background

2.1 Cost-Based Filtering

Consider the generic optimization problem \( \min f(x) \), where \( x = (x_1, \ldots, x_n) \) subject to unspecified constraints. CP generally uses a branch-and-bound approach to explore, within a search tree, the possible assignments for \( x \). Say, at some point of the search, the best solution found has an objective value \( U \) and that we can compute a lower bound \( L \) by considering a relaxed version of the original problem. Cost-based filtering [Focacci et al., 1999] consists in using the information of the current relaxed subproblem to filter values from the variable domains. If \( L > U \), infeasibility is raised. Let \( L[x_i = \mu] \) be the optimal objective value of the relaxed subproblem with the extra constraint \( x_i = \mu \). If \( L[x_i = \mu] > U \), then the value \( \mu \) is filtered out from \( \text{dom}(x_i) \). This method requires an efficient algorithm to compute \( L[x_i = \mu] \). In the context of integer linear programming, reduced costs can be used to filter specific values with a similar reasoning (reduced cost fixing, see e.g. [Wolsey, 2020]).

2.2 Lagrangian Relaxation

Consider the following linear program formed of two constraint families, \( A : Ax \leq b \) and \( B : Bx \leq d \), where \( X \subseteq \mathbb{R}^n \) is an arbitrary set:

\[
Z = \min \{ c^T x : Ax \leq b, Bx \leq d, x \in X \} \quad (P)
\]

Suppose \( A \) consists of difficult constraints. The Lagrangian relaxation technique lets these constraints go in the objective
function while keeping a global view on the original problem. Introducing *Lagrangian multipliers* $\lambda_i \geq 0$, the objective function is penalized when the constraints of $A$ are violated, resulting in the following relaxed problem:

$$Z_{LR}(\lambda) = \min \ c^T x + \lambda^T (Ax - b) \quad \text{s.t.} \quad Bx \leq d, \ x \in X$$

Any $\lambda \geq 0$ makes $Z_{LR}(\lambda)$ a valid lower bound of $Z$. In order to get the tightest bound, the Lagrangian multiplier problem is to find $\lambda$ that maximizes $Z_{LR}(\lambda)$ subject to $\lambda \geq 0$. Various methods exist in the literature to solve this problem including subgradient descent algorithms [Beasley, 1993; Sellmann, 2004], where the choice of multipliers is guided by the solution of $(LR(\lambda))$ until convergence.

### 2.3 CP-Based Lagrangian Relaxation

Assume we are given the linear program (P) and that an efficient filtering algorithm $\text{PROP}(B)$ is known for $B$. *CP-based Lagrangian relaxation* (CP-LR) [Sellmann and Fähle, 2001; Sellmann, 2004] consists of optimizing the Lagrangian multipliers for $A$ while using $\text{PROP}(B)$ for each subproblem $(LR(\lambda(A)))$ encountered during the gradient descent. Hence, while maximizing the lower bound $Z_{LR}(\lambda)$ over $\lambda$, CP-based filtering is applied on the corresponding substructure $B$.

As shown by Sellmann [2004], suboptimal multipliers can be more efficient for filtering than the multipliers that optimize the bound. This justifies why the filtering should be performed during the multipliers optimization process rather than once at the end. While optimizing the bound is the main objective, one could also want to maximize the quantity of filtered values each time $\text{PROP}(B)$ is called. Thus, it is a hint that multipliers should play a greater role in the filtering step.

### 2.4 TSP and WeightedCircuit

Let $G = (V, E)$ be an undirected graph with nodes set $V := \{1, \ldots, n\}$, edges set $E \subseteq \{(x, y) : x, y \in V, x \neq y\}$ and weight function $w : E \to \mathbb{Z}$. For an edge $\{i, j\} \in E$, we write $w(i, j)$ for its weight. The (symmetric) traveling salesman problem (TSP) consists in finding a Hamiltonian cycle in $G$ of minimum weight, i.e. a minimal path visiting all nodes and returning to its starting point.

Let $\delta(i) := \{e \in E : i \in e\}$ be the set of edges adjacent to node $i$. Introducing binary variables $x_e$ for $e \in E$, the TSP can be modeled as an integer linear program [Applegate et al., 2006]:

$$Z = \min \sum_{e \in E} w(e) x_e \quad \text{s.t.} \quad \begin{align*} &\sum_{e \in \delta(i)} x_e = 2 \quad \forall i \in V \\
&\sum_{i \in N} x_{(i,j)} \leq |N| - 1 \quad \forall N \subsetneq V, |N| \geq 3 \\
&x_e \in \{0, 1\} \quad \forall e \in E \end{align*}$$

Equations (1) are known as the *degree constraints* and require that each node have exactly two adjacent edges. Inequalities (2) are known as the *subtour elimination constraints* and ensure the connectivity of the tour.

The *1-tree relaxation* of Held and Karp [1970; 1971] results from relaxing the degree constraints (1). Let $G' = (V', E')$ be the graph $G$ from which an arbitrary node labeled 1 is removed, i.e. with $V' := V \setminus \{1\}$ and $E' := E \setminus \delta(1)$. A *1-tree* of $G$ is a spanning tree of $G'$ to which we add two distinct edges adjacent to node 1. The 1-tree relaxation consists in finding a 1-tree of $G$ of minimal weight. The sum of the edges’ weights in the 1-tree is a lower bound of the TSP. Introducing Lagrangian multipliers $\lambda = (\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^n$ with $\lambda_1 = 0$, this lower bound can be improved with a Lagrangian relaxation of the degree constraints (1). Let $\deg_A(i) := \{|e \in A : i \in e\}$ be the degree of node $i \in V$ in a set of edges $A$. We obtain the following relaxed problem:

$$Z_{LR}(\lambda) = \min \sum_{e \in T} w(e) + \sum_{i \in V} \lambda_i (\deg_{T(i)} - 2) \quad \text{s.t.} \quad T \text{ is a 1-tree, where } \forall e \in E \quad x_e \in \{0, 1\}$$

Observe that the objective function can be rewritten as

$$\sum_{i \in T} (w(i, j) + \lambda_i + \lambda_j - 2 \sum_{i \in V} \lambda_i \quad \text{s.t.} \quad \forall e \in E \quad x_e \in \{0, 1\}$$

Thus, $Z_{LR}(\lambda)$ can be found by computing a minimum spanning tree of $G'$, denoted $S$, and adding the two minimal edges adjacent to node 1, denoted $m_1$ and $m_2$, using the weight function $\tilde{w}(i, j) := w(i, j) + \lambda_i + \lambda_j$, $\forall \{i, j\} \in E$.

In CP, given the binary variables $x = (x_{e_1}, \ldots, x_{e_{|E|}})$, the weight function $w$, and an integer variable $z$, the *WeightedCircuit*(x, w, z) constraint [Benchimol et al., 2012] is satisfied if the edges $e \in E$ with $x_e = 1$ form a Hamiltonian cycle on the nodes $V$ with total weight at most $z$. The TSP is thus formulated as a minimization problem on $z$ subject to this constraint. Its filtering algorithms rely on the identification of *mandatory* edges that must be part of any solution and *forbidden* edges that cannot be part of any solution, through the costs stemming from the 1-tree structure.

Let $M$ be the set of mandatory edges and suppose that forbidden edges were removed from $E$. Let $\lambda$ be the Lagrangian multipliers, $T := S \cup \{m_1, m_2\}$ the minimum 1-tree and $U$ a known upper bound on the value of $Z$.

For an edge $e \in E \setminus T$, the *support edge* $s \in T$ is the edge that needs to be removed from $T$ if $e$ is forced into the minimum 1-tree. If $e$ is not adjacent to node 1, let $C_e \subseteq S$ be the edges lying on the unique cycle in $S \cup \{e\}$. The non-mandatory edge $e \in C_e \setminus M$ with maximum weight is the support edge of $e$. Else, $e$ is adjacent to node 1 and the support edge is the one in $\{m_1, m_2\} \setminus M$ with the greatest weight. The *reduced cost of* $e$, denoted $\bar{c}(e)$, is the increase of the objective value when forcing $e$ in $T$. Thus, we have $\bar{c}(e) = \bar{w}(e) - \bar{w}(s)$ or $\infty$ if $s$ does not exist. Cost-based filtering implies that $e \in E \setminus T$ is forbidden if $Z_{LR}(\lambda)[x_e = 1] = Z_{LR}(\lambda) + \bar{c}(e) > U$.

For an edge $e \in T$, the *replacement edge* $r$ is the edge that replaces $e$ in the minimum 1-tree if $e$ is removed. If $e$ is not adjacent to node 1, removing $e$ from $S$ divides the tree into two components. We define $R_e \subseteq E' \setminus S$ to be the cut-set of $G'$ induced by the removal of edge $e$ from $S$, i.e. the edges in $E'$ adjacent to both components in $S \setminus \{e\}$. The replacement edge $r$ is the one with minimum weight in $R_e$. Otherwise, $e$ is adjacent to node 1 and the replacement edge is the one in
Suppose we know an efficient cost-based filtering algorithm \( x \). We define a cost-based algorithm for finding the minimum 1-tree and the support/replacement edges of every edge in the graph. A brute-force algorithm directly derives from the definitions computes the reduced/replacement costs as well as the sets \( C_e \) or \( R_e \) for each edge \( e \in E \) in overall time \( O(|V| |E|) \). We propose to apply the improved framework of Section 3 suppressing this pre-processing step was done.

Let the relaxed solution \( x^* \) correspond to the minimum 1-tree \( T \) which gives the lower bound \( Z_{LR}(\lambda) \). By definition of the 1-tree relaxation, this solution is optimal if, and only if, \( T \) is a minimum 1-tree of \( G \). Thus, for each edge \( e \in E \setminus T \), the search for Lagrangian multipliers \( \lambda \) is subject to the condition that \( T \) remains a minimum 1-tree of \( G \).

Given an edge \( (i,j) \in E \setminus M \), Lemma 1 specifies the conditions on how to find \( \lambda \) that increases the value of \( Z_{LR}(\lambda)[x_{(i,j)}] = 0 \) or \( Z_{LR}(\lambda)[x_{(i,j)}] = 1 \).

**Lemma 1.** Let \( (i,j) \in E \setminus M \) be an edge of \( G \). Suppose \( \{k,l\} \) is the support (resp. replacement) edge of \( (i,j) \) and \( \lambda' := (\lambda_1, \ldots, \lambda_n + v) \) where \( v \in \mathbb{R} \) and \( s \in V \setminus \{i,j\} \). If \( v \) is chosen such as under this modification \( T \) remains a minimum 1-tree of \( G \) and \( \{k,l\} \) the support (resp. replacement) edge of \( (i,j) \), then

\[
Z_{LR}(\lambda')[x_{(i,j)}] = 1 = Z_{LR}(\lambda)[x_{(i,j)}] = 1
\]

\[
+ v \cdot \deg_T(s) - 2 + v \cdot (1_{(i,j)}(s) - 1_{(k,l)}(s))
\]

where \( 1_A(x) = 1 \) iff \( x \in A \) is the indicator function.

**Proof.** Suppose that \( (i,j) \in E \setminus T \) and that \( \{k,l\} \) is its support edge (the proof is similar for \( \{i,j\} \in T \)). Considering \( \lambda' \), let \( \tilde{w}(e) \) be the new weight of edge \( e \). Since \( T \) is still a minimum 1-tree of \( G \) and \( \{k,l\} \) the support edge of \( (i,j) \), \( Z_{LR}(\lambda')[x_{(i,j)}] = 1 = Z_{LR}(\lambda') + \tilde{w}(i,j) - \tilde{w}(k,l) \) by definition of the reduced cost of \( (i,j) \). We have

\[
Z_{LR}(\lambda') = \sum_{e \in T} w(e) + \sum_{i \in V} \lambda_i(\deg_T(i) - 2)
\]

\[
+ \sum_{i \in V \setminus \{s\}} \lambda_i(\deg_T(i) - 2) + (\lambda_s + v)(\deg_T(s) - 2)
\]

and

\[
\tilde{w}(i,j) - \tilde{w}(k,l) = w(i,j) + \lambda_i + \lambda_j - w(k,l) - \lambda_k - \lambda_l + v \cdot c
\]

where \( c = 1_{(i,j)}(s) - 1_{(k,l)}(s) \in \{-1, 0, 1\} \). Recalling that

\[
Z_{LR}(\lambda)[x_{(i,j)}] = 1 = Z_{LR}(\lambda) + \tilde{w}(i,j) - \tilde{w}(k,l)
\]

and putting it all together, the result follows.

Even though Lemma 1 imposes the condition that the support/replacement edge of \( (i,j) \) remains unchanged, which is more than what is required by the step 1, it leads in the following to a SIMPLE algorithm (Section 4.1) and an \( \alpha \)-SETS algorithm (Section 4.2).
4.1 The SIMPLE Algorithm

Given the edge \( \{i, j\} \in E \setminus M \), the SIMPLE algorithm (Algorithm 1) checks whether \( \lambda_i \) and \( \lambda_j \) can be both modified so that \( \text{dom}(x_{i,j}) \) is filtered. If \( \{i, j\} \in T \), line 1 computes the value \( \Delta \) corresponding to how much the bound and the replacement cost of \( \{i, j\} \) needs to be increased in order to declare the edge mandatory. For \( \lambda_i \), line 2 calls MAXDECREASE to compute a value \( v \geq 0 \) such that \( \lambda_i + v \) is a modification allowed by the hypotheses of Lemma 1 and that can only increase \( Z_{LR}(\lambda) |_{x_{i,j}} = 0 \). Line 3 performs the same process for \( \lambda_j \). If SIMPLE is applied for \( \{i, j\} \in E \setminus T \), we aim at increasing the values of the multipliers \( \lambda_i \) and \( \lambda_j \) so that \( \{i, j\} \) is identified as forbidden. Line 4 computes the value \( \Delta \) of how much the bound and the reduced cost of \( \{i, j\} \) need to be increased. For \( \lambda_i \), line 5 calls MAXINCREASE to compute a value \( v \geq 0 \) such that \( \lambda_i + v \) is a modification allowed by the hypotheses of Lemma 1 and that increases \( Z_{LR}(\lambda) |_{x_{i,j}} = 1 \). Line 6 repeats the process for \( \lambda_j \).

For \( \{i, j\} \in T \) and the multiplier \( \lambda_i \), line D1 of function MAXDECREASE computes a value \( \alpha \geq 0 \) ensuring \( T \) remains a minimum 1-tree under the modification \( \lambda_i - \alpha \). Line D2 restricts this value not to exceed \( \beta \) to ensure that the replacement edge of \( \{i, j\} \) remains unchanged. For \( \{i, j\} \in E \setminus T \), the value \( \alpha \) computed on line D1 of function MAXINCREASE ensures that \( T \) remains a minimum 1-tree while the value \( \beta \) computed on line D2 restricts the support edge of \( \{i, j\} \) to remain unchanged. Both functions run in \( O(|V|) \).

In the following, we show that for an edge \( \{i, j\} \in T \), the SIMPLE algorithm correctly computes new values for \( \lambda_i \) and \( \lambda_j \) that can only increase \( Z_{LR}(\lambda) |_{x_{i,j}} = 0 \). The proof is similar in the case of an edge \( \{i, j\} \in E \setminus T \). For the modification of \( \lambda_i \), we first need the following lemma.

Lemma 2. Considering \( \lambda' = (\lambda_1, \ldots, \lambda_i - \alpha, \ldots, \lambda_n) \) and the corresponding new reduced costs \( \bar{c}(e) \) for \( e \in E \setminus T \), we have \( \bar{c}(e) \geq 0 \) for all \( \alpha \geq 0 \).

Proof. Consider \( e \in E \setminus T \) and let \( s, s' \in T \) be respectively the support edge of \( e \) before and after considering the multipliers \( \lambda' \). We have

\[
\bar{c}(e) = \bar{w}'(e) - \bar{w}'(s') \geq \bar{w}'(e) - \bar{w}(s') \geq \bar{w}'(e) - \bar{w}(s)
\]

because \( \bar{w}'(x) \leq \bar{w}(x) \) for all \( x \in E \) and by definition of a support edge, \( \bar{w}(s') \leq \bar{w}(s) \). Now, if \( e \in \delta(i) \), we have

\[
\bar{w}'(e) - \bar{w}(s) = (\bar{w}(e) - \bar{w}(s)) = \bar{c}(e) \geq 0
\]

by definition of \( \alpha \) in line D1. Else, \( e \notin \delta(i) \) and

\[
\bar{w}'(e) - \bar{w}(s) = \bar{w}(e) - \bar{w}(s) = \bar{c}(e) \geq 0.
\]

In every case, \( \bar{c}(e) \geq 0 \).

By Lemma 2, the reduced costs remain positive, thus \( T \) remains a minimum 1-tree under the modification of \( \lambda_i \). For every edge \( a \) adjacent to \( i \) in \( R_{\{i,j\}} \), \( a \) is a candidate to be a replacement edge of \( \{i, j\} \). If \( r \) is the current replacement edge of \( \{i, j\} \), the value computed by \( \beta \) guarantees that \( \bar{w}'(a) \geq \bar{w}'(r) \) with the multipliers \( \lambda' \), i.e., that \( r \) remains the replacement edge. Since \( v \leq 0 \) and the conditions \( i \notin \{1, k, l\} \land \text{deg}_T(i) \leq 2 \) hold, the conclusion follows from Lemma 1. The same process is performed with \( \lambda_j \) without recomputing the reduced costs. Indeed, the modification of \( \lambda_j \) cannot decrease the reduced cost of edges adjacent to node \( j \).

SIMPLE only considers specific cases of Lemma 1 and some opportunities of increasing \( Z_{LR}(\lambda) |_{x_{i,j}} = \mu \) are ignored. This allows us to assure the previous properties are true and keep the algorithm simple and efficient.

The values computed on lines D2 and D2 require the sets \( C_e \) and \( R_e \). As an alternative that only uses the reduced costs computed by a faster pre-processing [Benchimol et al., 2012], we propose to replace the sets on lines D2 and D2 by

\[
\\{\bar{w}(i, k) - \bar{w}(r) : \{i, k\} \in E \setminus T, \bar{w}(i, k) \geq \bar{w}(r) \} \cup \{\infty\},
\\{\bar{w}(s) - \bar{w}(i, k) : \{i, k\} \in T \setminus M, \bar{w}(s) \geq \bar{w}(i, k) \} \cup \{\infty\}.
\]

In the following, we refer to the choice of the original sets as...
the \textit{complete} policy and of these latter as the \textit{relaxed} policy.
In both cases, the overall time complexity is in $O(|V|)$.  

4.2 The $\alpha$-SETS Algorithm

Following step 1, we derive the conditions on the multipliers
in order for the 1-tree $T = S \cup \{m_1, m_2\}$ to remain minimal.

\begin{equation}
\lambda_a' + \lambda_b' - \lambda_d' \leq w(a,b) - w(c,d)
\forall(a,b) \in S \setminus M, \forall(c,d) \in R_{a,b}
\tag{A}
\end{equation}

\begin{equation}
\lambda_b' - \lambda_d' \leq w(1,d) - w(1,b)
\forall(1,b) \in m_1, m_2 \setminus M, \forall(1,d) \in \delta(1) \setminus \{m_1, m_2\}
\tag{B}
\end{equation}

The constraints (A) come from the \textit{cut property} of minimum spanning trees stating that the cost of an edge in the tree \{a, b\} should not be greater than the cost of any edge \{c, d\}
in its cut-set $R_{a,b}$. The constraints (B) ensure that $m_1$ and $m_2$ are the two minimal edges adjacent to node 1. For an edge \{i,j\} $\in E \setminus M$, Lemma 1 also requires that its support/replacement edge \{k,l\} $\in E \setminus M$ remains unchanged.

By definition, this can be formulated as

\begin{equation}
\lambda_k' - \lambda_a' \leq w(a,b) - w(k,l)
\forall(a,b) \in R_{i,j}, \text{ if } \{i,j\} \in S;
\tag{C}
\end{equation}

\begin{equation}
\lambda_l' - \lambda_b' \leq w(1,b) - w(1,l)
\forall(1,b) \in \delta(1) \setminus \{m_1, m_2\}, \text{ if } \{i,j\} \in m_1, m_2;
\tag{D}
\end{equation}

\begin{equation}
\lambda_a' - \lambda_k' \leq w(k,l) - w(a,b)
\forall(a,b) \in C_{i,j}, \text{ if } \{i,j\} \in E' \setminus S;
\tag{E}
\end{equation}

\begin{equation}
\lambda_b' - \lambda_l' \leq w(1,l) - w(1,b)
\forall(1,b) \in m_1, m_2, \text{ if } \{i,j\} \in \delta(1) \setminus \{m_1, m_2\}.
\tag{F}
\end{equation}

All of these constraints are linear and could be used to derive a linear program that maximizes $Z_{LR}(X')|x_{(i,j)} = \mu|$. However, given there are $O(|V|^4)$ constraints, even the best linear solvers take too much time for the filtering to pay off. Therefore, we solve an easier problem where constraints are gradually added and multipliers are locally modified.

Given an edge \{i,j\} $\in E \setminus M$ and Lagrangian multipliers $\lambda$, the $\alpha$-SETS algorithm tries to find an $\alpha$-set $A \subseteq V$ in the graph: a set of nodes that will have their corresponding multiplier simultaneously modified and that will lead to an increased $Z_{LR}(X')|x_{(i,j)} = \mu|$ value. Each node $u \in V$ is labeled with a value $\sigma_u \in \{-1,0,1\}$ initially set to 0, corresponding to whether the multiplier $\lambda_u$ will decrease (-1), increase (+1), or remain unchanged (0). For a variable $a \geq 0$ and each node $u$, we look for multipliers $\lambda_u = \lambda_u + \sigma_u \cdot \alpha$. Starting from an initial node, the algorithm computes incrementally a set $A$ of nodes and a set $\Omega$ of constraints taken from (A) to (F). During any step in the process, the substitution of $\lambda_u$ by $\lambda_u + \sigma_u \cdot \alpha$ $\forall u \in V$ in each constraint $\omega \in \Omega$ leads to a system of linear inequalities that can be written in the form $c_\omega \cdot \alpha \leq m_\omega$, where the coefficient $c_\omega \in \{-2,-1,0,1,2\}$ and $m_\omega \geq 0$ are two known constants. Maximizing $\alpha$ under these constraints, a value $\alpha^* \geq 0$ is found. If $\alpha^* = 0$, the algorithm looks for the restraining constraint and appends one of its related nodes to $A$ in order to loosen the inequality. Doing so, new constraints must be taken into account and added to $\Omega$. This process is repeated until $\alpha^* > 0$, leading to a valid $\alpha$-set $A$ and new multipliers $\lambda'$. The increased value $Z_{LR}(X')|x_{(i,j)} = \mu|$ directly follows from the sum of all the changes according to Lemma 1, without having to re-compute a minimum 1-tree. If it is still insufficient to filter $\text{dom}(x_{(i,j)})$, the procedure can be re-executed with the multipliers $\lambda := \lambda'$ previously found and looks for another $\alpha$-set.

This procedure searches for an $\alpha$-set $A$ where $\{k,l\} \in E \setminus M$ is the support/replacement edge of $\{i,j\}$.

1. Initialize a set of constraints $\Omega$ with the constraints from (C) to (F), depending on the nature of $\{i,j\}$.

2. Choose an initial node $u \in \{i,j,k,l\} \setminus \{1\}$ and $\sigma_u \in \{-1, +1\}$. Following Lemma 1, this choice must satisfy

\begin{itemize}
\item[(a)] $\{i,j\} \in E \setminus T \Rightarrow \sigma_u \cdot (\deg_T(u) - 2 + 1_{(i,j)}(u) - 1_{(k,l)}(u)) > 0$;
\item[(b)] $\{i,j\} \in T \Rightarrow \sigma_u \cdot (\deg_T(u) - 2 + 1_{(k,l)}(u) - 1_{(i,j)}(u)) > 0$.
\end{itemize}

If none of these 8 combinations works, no set $A$ exists and the algorithm halts without filtering $\text{dom}(x_{(i,j)})$.

3. Append node $u$ to the set $A$.

4. Add to $\Omega$ all the constraints not already considered of type (A)-(B) where $\lambda' \in \Omega$ has coefficient $\sigma_u$.

5. For each constraint $\omega \in \Omega$, compute the values $c_\omega$ and $m_\omega$. Since $\omega$ is of the form $\lambda_a' + \lambda_b' - \lambda_k' \leq w(1,a,b) - w(c,d)$, we have $c_\omega = \sigma_a + \sigma_b - \sigma_c - \sigma_d$ and $m_\omega = w(a,b) - w(c,d)$. The maximal value $\alpha \geq 0$ can take, subject to the constraints in $\Omega$, is given by

\begin{equation}
\alpha^* = \min \left\{ \frac{m_\omega}{c_\omega} : \omega \in \Omega \land c_\omega > 0 \right\}.
\end{equation}

6. If $\alpha^* > 0$, return the value $\alpha^*$ and the set $A$ is found.

7. If $\alpha^* = 0$, find the constraint $\omega \in \Omega$ that prevents obtaining a value $\alpha > 0$. This constraint is of the form $c_\omega \cdot \alpha \leq w(a,b) - w(c,d)$, where $a,b,c,d \in V$.

8. Choose a node $u' \in \{a,b,c,d\} \setminus \{1\}$ and $\sigma_{u'} \in \{-1, +1\}$ such that

\begin{itemize}
\item[(a)] $u' \notin P \cup \{(a,b) \cap (c,d)\}$;
\item[(b)] $(u' \in \{a,b\} \land \deg_T(u') \geq 2) \Rightarrow \sigma_{u'} = +1$;
\item[(c)] $(u' \in \{c,d\} \land \deg_T(u') \leq 2) \Rightarrow \sigma_{u'} = -1$;
\item[(d)] $u' \in \{i,j,k,l\} \Rightarrow \text{conditions of step 2}$.
\end{itemize}

If none of these 8 combinations works, backtrack and reconsider the last choice. If no possibility remains, no set $A$ exists and the algorithm halts without filtering $\text{dom}(x_{(i,j)})$. Else, set $u := u'$ and go to step 3.

Since the procedure can be repeated as long as an $\alpha$-set $A$ is found and $\text{dom}(x_{(i,j)})$ is not filtered, a maximum number of iterations can be added. Also, to avoid considering too many constraints at the same time, a maximum cardinality $C_m$ on $A$ can be imposed. Our implementation follows an iterative-deepening search that gradually increases $C_m$. For an $\alpha$-set of $C$ nodes, step 5 deals with $C \cdot O(|V| |E|)$ constraints. As 4 nodes are possible for each choice, it leads to a worst-case time complexity in $O(C_m |V| |E|^3)$.

This algorithm can be combined with the SIMPLE algorithm and the \textit{complete} policy: we call the latter first to obtain new multipliers $\lambda'$, and if it was not enough to filter the
considered edge, we use them as the initial multipliers for \( \alpha \)-SETS. We refer to this process as the HYBRID algorithm.

4.3 Example

On the graph of Figure 1a with \( Z_{LR}(\lambda) = 19 \) and \( U = 23 \), \( \{a, b\} \) is the support edge of \( \{b, c\} \) and \( Z_{LR}(\lambda)|_{x_{\{b,c\}} = 1} = Z_{LR}(\lambda) + \tilde{w}(b, c) - \tilde{w}(a, b) = 20 \).

We apply HYBRID on edge \( \{b, c\} \). First, SIMPLE finds that \( \lambda_c \) can be increased by 1, obtained from the minimum between \( \alpha = 1 \) (line 11) and \( \beta = 2 \) (line 12). This leads to the multipliers \( \lambda' \) on Figure 1b, but \( Z_{LR}(\lambda')|_{x_{\{b,c\}} = 1} = 22 \leq U \).

Choosing again to increase node \( c \) by \( \alpha \geq 0 \), \( \alpha \)-SETS finds the constraint \( \alpha \leq \tilde{w}(d, e) - \tilde{w}(c, d) = 0 \). A valid choice is to simultaneously increase node \( c \) by \( \alpha \) leading to the new constraint \( \alpha \leq \tilde{w}(d, e) - \tilde{w}(c, e) = 1 \). Since \( \alpha^* > 0 \), the multipliers \( \lambda' \) are updated (Figure 1c) and \( Z_{LR}(\lambda)|_{x_{\{b,c\}} = 1} = 24 > U \). The edge \( \{b, c\} \) is thus forbidden.

5 Experiments

The algorithms were implemented\(^1\) in Java 14 using the solver Choco 4.0.6 [Jussien et al., 2008] and its extension Choco Graph 4.2.3 [Fages, 2014]. The experiments were performed on a CentOS Linux 7 machine using an Intel Xeon Silver 4110 CPU at 2.10 GHz and 32 GB of RAM. The TSP was modeled using the WEIGHTEDCIRCUIT constraint already implemented in Choco Graph using the state-of-the-art algorithms [Benchimol et al., 2012]. It uses a subgradient descent algorithm to optimize the Lagrangian multipliers. We kept the default parameters, but we did not restart the algorithm when the lower bound of the 1-tree was increased. As an initial upper bound on the objective variable, we gave the bound provided by the LHK-2.0.9 heuristic [Helsgaun, 2000].

The search strategy was fixed to maxCost with the LCFirst policy [Fages et al., 2016]. We chose the symmetric TSP instances from the TSPLIB library [Reielt, 1991] between 96 and 500 nodes that could be solved by Choco under 8 hours and with at least 100 search nodes.

We compare the SIMPLE and HYBRID algorithms against the one from Choco. Our algorithms were only called at the very last iteration of the subgradient process. Furthermore, for HYBRID, since \( \alpha \)-SETS is slowed down by the number of constraints, it was only called when \( |E| \leq 2|V| \) with a limit of 2 on the cardinality of the \( \alpha \)-sets and a maximum of 10 iterations when trying to reach the fixed point.

---

\(^1\)The code is available at http://www2.ift.ulaval.ca/~quimper/publications.php.
References


