Algorithmic expedients for the S-labeling problem

Markus Sinnl[*1]

[1] *Department of Statistics and Operations Research, Faculty of Business, Economics and Statistics, University of Vienna, Vienna, Austria*

Abstract

Graph labeling problems have been widely studied in the last decades and have a vast area of application. In this work, we study the recently introduced S-labeling problem, in which the nodes get labeled using labels from 1 to $|V|$ and for each edge the contribution to the objective function, called S-labeling number of the graph, is the minimum label of its end-nodes. The goal is to find a labeling with minimum value. The problem is NP-hard for planar subcubic graphs, although for many other graph classes the complexity status is still unknown.

In this paper, we present different algorithmic approaches for tackling this problem: We develop an exact solution framework based on Mixed-Integer Programming (MIP) which is enhanced with valid inequalities, starting and primal heuristics and specialized branching rules. We show that our MIP formulation has no integrality gap for paths, cycles and perfect n-ary trees, and, to the best of our knowledge, we give the first polynomial-time algorithm for the problem on n-ary trees as well as a closed formula for the S-labeling number for such trees. Moreover, we also present a Lagrangian heuristic and a constraint programming approach. A computational study is carried out in order to (i) investigate if there may be other special graph classes, where our MIP formulation has no integrality gap, and (ii) assess the effectiveness of the proposed solution approaches for solving the problem on a dataset consisting of general graphs.

1. Introduction and motivation

Graph labeling problems have been widely studied in the last decades and they have a vast area of application, e.g., coding theory Bloom and Golomb (1977), computational biology Karp (1993), computer networks Jin and Yeh (2005), design of error-correcting codes Rodriguez-Tello et al. (2008) radio channel assignment Van den Heuvel et al. (1998) and more (we refer the reader to the surveys Díaz et al. (2002); Dehghan et al. (2013); Gallian (2009) for further details). In such problems, we are typically interested in assigning distinct positive integers (i.e., *labels*) to the nodes and/or edges of the graph subject to some constraints, such that a given objective function is optimized. In this work, we concentrate on the *S-labeling problem*, which is defined as follows.

Definition 1 (The S-labeling problem (S-LP))**.** *Let $G = (V, E)$ be a graph and $\phi : V \rightarrow \{1, \ldots, |V|\}$ be a labeling of the nodes. The S-labeling number $SL_\phi(G)$ with respect to the labeling $\phi(G)$ is defined as $\sum_{\{u,v\}\in E} \min\{\phi(u), \phi(v)\}$. The goal of the S-labeling problem (S-LP) is to find a labeling $\phi^*(G)$, such that $SL_{\phi^*}(G)$ has minimum value amongst all possible labelings for G.*

In the remainder of this paper, we simply write ϕ^* and SL_{ϕ^*}, when the graph is clear from the context. The S-LP was introduced in Vialette (2006) in the context of packing $(0, 1)$-matrices, where it was shown to be *NP-complete* for planar subcubic graphs. It is studied in detail in a series of papers Fertin and Vialette (2009); Fertin et al. (2015, 2017), which focused on deriving properties of optimal labelings. Based on these

[*] markus.sinnl@univie.ac.at

properties, exact and approximation algorithms for some graph classes are developed: Polynomial-time exact algorithms are given for *caterpillar graphs* (trees, where all nodes are at most distance one from the central path) and *split graphs* (graphs which can be partitioned into a clique and an independent set). Moreover, a greedy approximation algorithm, which gives a labeling ϕ with $SL_\phi < \frac{|E|(|V|+1)}{3}$ (resp., $\frac{|E|(|V|+1)}{4}$ if the graph is acyclic with the maximum degree of a node at least three) is presented. This algorithm is shown to be a $\frac{4|E|\Delta}{3|V|}$ approximation algorithm for general graphs, where Δ is the maximum degree of a node in the graph, and a 4/3 approximation algorithm for regular graphs; for some other special graphs, the approximation factor is further refined. Additionally, a fixed-point parameter tractable algorithm is also presented; this approach is based on partial enumeration, where the parameters is a positive integer k and the question is whether there is a labeling with $SL_\phi < |E| + k$. Finally, closed formulas for SL_{ϕ^*} for complete graphs, paths and cycles are given, but without proof. We note that one result of Fertin et al. (2015) (Lemma 3) claims that in any optimal labeling ϕ^*, the node i with label one is a node with maximum degree in the graph. We have a counter-example to this result, see Section 3.

Contribution and paper outline In this paper, we develop different modeling and algorithmic approaches to characterize and solve the S-LP. The paper is organized as follows. In Section 2, we describe an exact solution framework based on Mixed-Integer Programming (MIP), which includes starting and primal heuristics to construct high-quality feasible solutions during the solution process as well as a specialized branching-scheme. In Section 3 we investigate the Linear Programming (LP) relaxation of the MIPs presented in Section 2, and show that for paths, cycles and perfect n-ary trees the LP-relaxation exhibits no integrality gap. This is done by presenting a combinatorial procedure for solving the dual of the LP-relaxation and ad-hoc polynomial-time primal algorithms. To the best of our knowledge, we present the first polynomial-time algorithm for solving the S-LP on perfect n-ary trees. We also give a closed formula to compute SL_{ϕ^*} for perfect n-ary trees. In Section 4, we present further solution methods, namely a Lagrangian heuristic and a constraint programming approach. Section 5 contains a computational study. The purpose of this study is twofold: (i) to computationally investigate, if there are additional classes of graphs, where the LP-relaxation of our MIP formulation may have no integrality gap; and (ii) to assess the performance of the presented approaches for solving the S-LP on general graphs. Finally, Section 6 concludes the paper.

2. Mixed-Integer Programming approaches for the S-LP

In this section, we present MIP (exact) approaches for solving the S-LP. We first provide two MIP formulations for the problem and show that one of them is the projection of the other. After this, we describe branch-and-cut schemes based on these MIPs.

For the proposed formulations, let x_i^k, with $i, k \in \{1, \dots, |V|\}$, be a binary variable such that $x_i^k = 1$, if node i (or, more formally, the node with index i) gets labeled with the number k, and $x_i^k = 0$, otherwise. The following constraints (UNIQUE) and (ONELABEL) ensure that each label gets used exactly once, and that each node gets one label,

$$\sum_{i\in\{1,\dots,|V|\}} x_i^k = 1 \qquad \forall k \in \{1, \dots, |V|\} \tag{UNIQUE}$$

$$\sum_{k\in\{1,\dots,|V|\}} x_i^k = 1 \qquad \forall i \in \{1, \dots, |V|\}. \tag{ONELABEL}$$

To define the first formulation, for each edge $e \in E$ we introduce continuous variables $\theta_e \geq 0$, which are used to measure the contribution of edge e to the objective. Using these variables, our first formulation (F1) for the S-LP is given by

$$(F1) \quad SL_{\phi^*} = \min \quad \sum_{e \in E} \theta_e \tag{F1.1}$$

$$\text{(UNIQUE)}, \text{(ONELABEL)}$$

$$\theta_e \geq k - \sum_{l<k} (k-l)(x_i^l + x_{i'}^l) \qquad \forall k \in \{1, \dots, |V|\}|, \ \forall e = \{i, i'\} \in E \tag{F1.2}$$

$$x_i^k \in \{0,1\} \qquad \forall i, k \in \{1, \dots, |V|\}.$$

Constraints (F1.2) ensure that for each edge e the correct contribution is counted in the objective function: For an edge $e = \{i, i'\}$, let k^1 and k^2 (assume wlog $k^1 < k^2$) be the labels of the two end-nodes i and i' for a given solution encoded by $\bar{\mathbf{x}}$. For each $k \leq k^1$, the right-hand-side (rhs) of (F1.2) is k, for $k^1 < k < k^2$, the rhs is k^1, and for $k \geq k^2$, the rhs is $k^1 - k^2 < k^1$. Thus, θ_e takes the value k^1.

For the second model (F2), instead of using continuous variables $\theta_e \geq 0$ to measure the contribution of edge e, we introduce binary variables d_e^k, with $k \in \{1, \dots, |V| - 1\}$, such that $d_e^k = 1$ iff the contribution of edge e to the objective is k, $d_e^k = 0$ otherwise. With these variables, we can formulate the problem as follows,

$$(F2) \quad SL_{\phi^*} = \min \quad \sum_{e \in E} \sum_{k \in \{1, \dots, |V|-1\}} k d_e^k \tag{F2.1}$$

$$\text{(UNIQUE)}, \text{(ONELABEL)}$$

$$\sum_{k \in \{1, \dots, |V|\}} d_e^k = 1 \qquad \forall e \in E \tag{F2.2}$$

$$d_e^k \leq x_i^k + x_{i'}^k \qquad k \in \{1, \dots, |V|\}, \forall e = \{i, i'\} \in E \tag{F2.3}$$

$$x_i^k \in \{0,1\} \qquad i, k \in \{1, \dots, |V|\}$$

$$d_e^k \in \{0,1\} \qquad \forall e \in E, \ k \in \{1, \dots, |V|\}.$$

Constraints (F2.2) ensure that exactly one of the variables d_e^k is taken for each edge e, and constraints (F2.3) make sure that one of the two end nodes of edge e has label k, if the associated d_e^k-variable is chosen as one (and thus the contribution of the edge in the objective is k).

We observe that $(F1)$ has $O(|V|^2)$ variables, and $O(|V||E|)$ constraints, while $(F2)$ has $O(|V|^2 + |V||E|)$ variables, and $O(|V||E|)$ constraints. Thus $(F2)$ has considerably more variables, on the other hand, constraints (F2.3) are much sparser than their counterpart (F1.2). Note that the integrality of the variables d_e^k can be relaxed, as for fixed variables $x \in \{0,1\}^{|V| \times |V|}$ the remaining coefficient matrix in d_e^k becomes totally unimodular.

Finally, the following family of inequalities is valid for (F2);

Theorem 1. *Suppose the three nodes $i, i', i'' \in V$ form a triangle $e = \{i, i'\}, e' = \{i, i''\}, e'' = \{i', i''\}$ in G and let $V' \subset \{1, \dots, |V|\}$ be a subset of labels. Then*

$$\sum_{k \in V'} \left(d_e^k + d_{e'}^k + d_{e''}^k\right) \leq 1 + \sum_{k \in V'} \left(x_i^k + x_{i'}^k + x_{i''}^k\right) \tag{TRIANGLE}$$

is valid for (F2).

Proof. Clearly, the sum on the left-hand-side (lhs) can be at most three. If the sum is zero or one, validity is trivial and validity for two follows easily due to constraints (F2.3). Thus, suppose the sum on the lhs is three, this means all three edges of the triangle are in some of the label-levels given in V'. However, in this case, at least two of the variables on the right-hand-side must be one, as clearly not all three edges can be on the same labeling-level in any feasible solution. □

As there is an exponential number of (TRIANGLE) inequalities, we do not add them in the beginning, but separate them on-the-fly, i.e., we embed their separation in an branch-and-cut scheme; the separation is described in Section 2.3. The computational results in Section 5.2 reveal that these inequalities can be quite helpful in some instances.

2.1 Comparison between $(F1)$ and $(F2)$

In this section, we show that formulation $(F1)$ can be obtained from $(F2)$ by using Benders decomposition to project out the d_e^k-variables using Benders optimality cuts, i.e., consider a Benders master problem consisting of (UNIQUE), (ONELABEL) and variables θ_e that account for the contribution of edge e to the objective function, the obtained optimality cuts are exactly inequalities (F1.2).

In order to show this equivalence, we relax the integrality of the d_e^k-variables, and consider the dual of $(F2)$ for a fixed solution encoded by vector $\bar{x} \in \{0,1\}^{|V|\times|V|}$. In the following, for ease of readability, when we refer to dual of some formulation, we mean the dual of the corresponding LP-relaxation (with possible some subset of the variables fixed as mentioned). Observe that for fixed solution given by $\bar{x}$, the problem decomposes into one problem for each edge e. Let γ_e be the dual-variables associated with constraint (F2.2), and let δ_e^k be the dual variables associated with constraints (F2.3). The dual $(D_{(F2)(\bar{x},i,i')})$ of $(F2)$, for a given $\bar{x}$ and an edge $e = \{i, i'\}$, then reads as

$$(D_{(F2)(\bar{x},i,i')}) \quad \max \quad \gamma_e - \sum_{k\in\{1,\dots,|V|-1\}} (\bar{x}_i^k + \bar{x}_{i'}^k)\delta_e^k \qquad (D_{(F2)(\bar{x},i,i')}.1)$$

$$\gamma_e - \delta_e^k \leq k \qquad \forall k \in \{1,\dots,|V|\} \qquad (D_{(F2)(\bar{x},i,i')}.2)$$

$$\delta_e^k \geq 0 \qquad \forall k \in \{1,\dots,|V|\} \qquad (D_{(F2)(\bar{x},i,i')}.3)$$

Note that using standard duality rules, we would obtain $\delta_e^k \leq 0$ and positive coefficients associated with these variables in the objective and constraints. For ease of exposition, we write these dual variables as $\delta_e^k \geq 0$ by changing the objective function coefficients accordingly.

The optimal solution of $(D_{(F2)(\bar{x},i,i')})$ can be derived as follows. It is easy to see that in order to maximize the objective function $(D_{(F2)(\bar{x},i,i')}.1)$, γ_e needs to be increased. However, due to constraints $(D_{(F2)(\bar{x},i,i')}.2)$, if γ_e is increased to k, to ensure feasibility we need to compensate this, by setting $\delta_e^l = k - l$, whenever $l < k$. Due to constraints (UNIQUE) and (ONELABEL), the coefficients $(\bar{x}_i^k + \bar{x}_{i'}^k)$, of dual variable δ_e^k, take value of one only for two values of k, say l^1, l^2 with $l^1 < l^2$, and value of zero for the remaining values of k. Thus, as long as k is at most $k^1 = l^1 + 1$, setting $\delta_e^l = k - l$ does not influence the objective function, since the coefficients of the corresponding δ_e^l variables are zero, and the objective value obtained is k^1. For $k \in [k^1, \dots, l^2]$, the objective function also has the value k^1: If the value of γ_e is increased by one within this interval, then the value of $\delta_e^{k^1}$, whose objective function coefficient is minus one must also be increased by one. Finally, starting from $k^2 = l^2 + 1$, also $\delta_e^{k^2}$ would need to be increased by one to ensure feasibility, i.e., for each increase of γ_e by one, two variables with coefficient minus one in the objective must also be increased by one. Thus, the optimal solution value of the dual for the given $\bar{x}$ is k^1, and it is achieved by applying above procedure for any k with $k^1 \leq k < k^2$. The values of the dual variables in an optimal solution are $\gamma_e = k$ and $\delta_e^l = k - l$ for $l < k$ and zero otherwise. Thus, the associated Benders optimality cuts are inequalities (F1.2).

2.2 Starting and primal heuristic

In order to construct a feasible starting solution to initialize the branch-and-bound, we use the greedy algorithm proposed in Fertin et al. (2017), which works as follows. We pick any unlabeled node, say i, with maximum degree (ties broken randomly), and we label it with the smallest available label; then, we remove i from the graph and repeat the procedure until all nodes are labeled. After this, we try to improve the obtained solution with a local search-phase which is based on exchanging the label of pairs of nodes. The complete scheme is described in Algorithm 1. In the algorithm, $\phi_H^{-1}(k)$ denotes the node with label

k. In the local search-phase (lines 10-35), we exploit the fact that an exchange of labels between pairs of nodes i, i' does not require to recalculate the labeling number from scratch, as it is enough to recalculate the contribution of the edges $e \in E$ such that $e : \{i, \cdot\}$ or $e : \{i', \cdot\}$. We also do not try all label-pair-exchanges, but for a given label k limit the change to labels $k' \leq min(k, maxContribLabel)$, where $maxContribLabel$ is the maximum contribution to the objective of any edge, where one end-node is labeled with k.

```
   **input**  : instance $G = (V, E)$ of the $S$-LP
   **output**: feasible labeling $\phi_H$ with labeling number $z^H = SL_{\phi_H}$
 1 $G' \leftarrow G$
 2 $z^H \leftarrow 0$
   /* greedy algorithm phase                                    */
 3 for $1 \leq k \leq |V|$ do
 4     $i \leftarrow$ node with maximal degree in $G'$
 5     $\phi_H(i) \leftarrow k$
 6     $G' \leftarrow G' \setminus i$
 7     for $\{i, i'\} \in E$ with $i'$ unlabeled do
 8         $z^H \leftarrow z^H + k$
   /* local-search phase                                       */
 9 $improve \leftarrow true$
10 repeat
11     $improve \leftarrow false$
12     for $1 \leq k \leq |V|$ do
13         $i \leftarrow \phi_H^{-1}(k)$
14         $cost^i \leftarrow 0$
15         $maxContribLabel \leftarrow 0$
16         for $\{i, i''\} \in E$ do
17             $cost^i \leftarrow cost^i + \min(\phi_H(i), \phi_H(i''))$
18             $maxContribLabel \leftarrow \max(maxContribLabel, \min(\phi_H(i), \phi_H(i'')))$
19         for $1 \leq k' \leq \min(k, maxContribLabel)$ do
20             $i' \leftarrow \phi_H^{-1}(k')$
21             $cost^{i'} \leftarrow 0$
22             for $\{i', i''\} \in E$ do
23                 $cost^{i'} \leftarrow cost^{i'} + \min(\phi_H(i'), \phi_H(i''))$
24             $costExchange \leftarrow 0$
25             for $\{i, i''\} \in E$ do
26                 $costExchange \leftarrow costExchange + \min(k', \phi_H(i''))$
27             for $\{i', i''\} \in E$ do
28                 $costExchange \leftarrow costExchange + \min(k, \phi_H(i''))$
29             if $costExchange < cost^i + cost^{i'}$ then
30                 $z^H \leftarrow z^H - cost^i - cost^{i'} + costExchange$
31                 $\phi_H(i) \leftarrow k'$
32                 $\phi_H(i') \leftarrow k$
33                 $improve \leftarrow true$
34                 break
35 until $improve = false$
```

Algorithm 1: Starting heuristic

Based on this algorithm, we also designed a primal heuristic, guided by the LP-relaxation, which is embedded within the branch-and-bound search framework. Let $\tilde{x}$ be the value of the x-variables of the LP-relaxation at a branch-and-bound node. We solve the minimum assignment problem induced by constraints (UNIQUE) and (ONELABEL), setting the coefficient of variable x_i^k in the objective to $\tilde{x}_i^k$, the obtained solution gives a feasible labeling. Afterwards, we attempt to improve the obtained solution by applying the local search-part of the heuristic described in Algorithm 1.

2.3 Further enhancements

We now describe the additional ingredients that are part of the developed branch-and-cut algorithm, namely a branching strategy, an initialization procedure and separation procedures for violated inequalities.

Branching Constraints (ONELABEL) (and also (UNIQUE) and (F2.2)) are *generalized-upper-bound* (GUB) constraints (also known as *special-ordered-sets*). It is well-known that in presence of such constraints, branching on a single variable x_i^k may not be very efficient, as it will often lead to an "unbalanced" search tree: in one part of the search tree, one variable x_i^k is fixed to zero and in the other part, all other variables $x_i^{k'}, k' \neq k$ are fixed to zero. Instead, branching on a subset of the variables appearing in such a constraint could be more efficient, this strategy is often called GUB-branching (see, e.g., Conforti et al. (2014); Nemhauser and Wolsey (1988); Wolsey (1998) or also Cho and Linderoth (2015) for more details on implementing such a branching-scheme). In our case, we implemented GUB-branching based on constraints (ONELABEL), i.e., at every branch-and-bound-node, we select a node i to branch on, and then make two children-nodes where we enforce

$$\sum_{k \in K} x_i^k = 0 \quad \text{or} \quad \sum_{k \in \{1,\dots,|V|\} \setminus K} x_i^k = 0, \tag{1}$$

where $K \subset \{1, \dots, |V|\}$. This is a valid branching scheme, as in one child, node i must take a label in $\{1, \dots, |V|\} \setminus K$, and in the other it must take a label in K. Let $\tilde{x}$ be the value of the LP-relaxation at a branch-and-bound node and suppose we already selected a node i. The partition K is found by calculating $B = \lfloor \sum_{k \in \{1,\dots,|V|\}} k \tilde{x}_i^k \rfloor$ and putting every $k \leq B$ in K. To select node i to branch on, for each node i, with fractional variables $\tilde{x}_i^k$, we calculate the following score sc_i based on up-*pseudocosts* $\psi^+(i,k)$ and down-*pseudocosts* $\psi^-(i,k)$: $sc_i = \sum_{k \in \{1,\dots,|V|\}: \tilde{x}_i^k > 0} \psi^+(i,k)(1 - \tilde{x}_i^k) + \psi^-(i,k)\tilde{x}_i^k$, and select the node with the maximum score. If the maximum score is under 0.001, we then proceed with the default branching of the branch-and-cut solver (in preliminary tests, we also tried other scores, and this one was the most effective). Pseudocosts are a concept often used for branching decisions and are provided by CPLEX, which is the branch-and-cut solver we use (see, e.g., Achterberg et al. (2005); Conforti et al. (2014) for more on pseudocosts).

Initialization While both $(F1)$ and $(F2)$ have polynomially many constraints, for some instances, the size of the resulting models may still become prohibitive for efficient solving. Removing constraints initially, and only adding them on the fly when needed (i.e., using branch-and-cut) could be an useful option in such a case. However, in preliminary computations, for formulation $(F2)$, such a strategy did not turn out to be computationally successful. This may be explained in the structure of the constraints. Suppose that for some edge e, only a subset of the constraints (F2.3) is added. Then the d_e^k-variable with the smallest k, for which no constraint was added will be take value of one in the resulting relaxation. Thus, to ensure that the correct variable $d_e^{k'}$ is set to one, all constraints (F2.3) with $k \leq k'$ must be present. On the contrary, for formulation $(F1)$, a branch-and-cut approach was more useful. In this case, adding a single constraint (F1.2), for each edge e, is enough for having an effect on the objective value of the resulting relaxation. In our initialization approach for (F1), the set of initial constraints consists of all constraints (ONELABEL) and (UNIQUE), and a single-constraint (F1.2) for each e. The latter are determined by heuristically solving the dual problem of $(F2)$ with the algorithm described in the next section. For each edge e, we add the constraint (F1.2) for the largest k, for which the dual multiplier for $d_e^k \leq x_i^k + x_{i'}^k$ is non-zero. Separation of constraints (F1.2) is done by enumeration. During each separation round, we add one violated inequality (if there is any) for each e; if there is more than one violated, we add the one with the largest violation. By checking the potential violation of (F1.2) for each e increasingly by k, we do not need to re-calculate it from scratch for each k.

Separation of inequalities (TRIANGLE) We do not separate the inequalities for all subsets of labels $V' \subset \{1, \dots, |V|\}$, but simply check for all increasing subsets, starting with $\{1, 2\}$ until $\{1, 2, \dots |V| - 1\}$. The separation is then done by enumeration, i.e., we enumerate all *triangles* in the beginning and store them; afterwards, during a separation round at a given node of the branch-and-bound tree, we check the inequalities for each of the above mentioned subsets of V'. By performing this in an increasing order, we do not need to re-separate from scratch, but just need to add the contribution from the current label. For each triangle, we stop either when a violated inequality is found, or when the sum of fractional variables on the right-hand-side of the inequality has reached the value of three (since it cannot grow larger than that).

3. Analyzing the MIP-formulation for special classes of graphs

Clearly, any feasible solution to the dual of an LP-relaxation of a MIP gives a lower bound, which can be used in, e.g., a branch-and-bound algorithm to solve the MIP. Depending on the structure and size of the resulting dual, a high-quality dual solution can potentially be found using a combinatorial procedure instead of solving the corresponding LP model directly using, e.g., the simplex-algorithm. For example, for the facility location problem Erlenkotter (1978), the Steiner tree problem Wong (1984), or the network design problem Balakrishnan et al. (1989), there exist so-called *dual ascent* algorithms, which (heuristically) solve the dual by starting with a solution, where all variables are zero (such a solution is feasible for these problems), and then systematically increase the dual variables in order to get a good dual solution. In this section we present a combinatorial algorithm for solving $(D_{(F2)})$, which is inspired by these approaches. We show that for paths, cycles and perfect n-ary trees, the dual solution produced by our heuristic algorithm is in fact optimal.

3.1 A heuristic algorithm for solving the dual of formulation (F2)

In addition to the dual multipliers defined in the previous section, let α_k be the dual multipliers associated with (UNIQUE), and let β_i be the dual multipliers associated with (ONELABEL). The dual of (F2) denoted as $(D_{(F2)})$, is as follows (again changing the coefficients of δ^k_e to get $\delta^k_e \geq 0$);

$$(D_{(F2)}) \quad \max \sum_{k \in \{1,\dots,|V|\}} \alpha_k + \sum_{i \in V} \beta_i + \sum_{e \in E} \gamma_e \qquad (D_{(F2)}.1)$$

$$\alpha_k + \beta_i + \sum_{e:\{i,\cdot\} \in E} \delta^k_e \leq 0 \qquad k \in \{1, \dots, |V|\}, \forall i \in V \qquad (D_{(F2)}.2)$$

$$\gamma_e - \delta^k_e \leq k \qquad \forall e \in E, k \in \{1, \dots, |V|\} \qquad (D_{(F2)}.3)$$

$$\delta^k_e \geq 0 \qquad \forall e \in E, k \in \{1, \dots, |V|\} \qquad (D_{(F2)}.4)$$

Algorithm 2 outlines a (greedy) heuristic to solve $(D_{(F2)})$. We denote the degree of a node i with $\delta(i)$. While Algorithm 2 is a heuristic for general graphs, it gives the optimal dual solution for paths, cycles and perfect n-ary trees due to the particular structure of these graphs. At the end of the section, we discuss an extended version of the algorithm, which can produce better dual bounds for other graphs and also give a small example of this. Note that in both variants of our algorithm the dual variables are always kept integral, so in case the optimal dual solution has fractional values, it cannot be achieved with our algorithms.

We start with $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\delta})$ set to zero, and all γ_e set to one, which is a feasible solution, and we iteratively try to increase the values of some γ_e (which appear with coefficient one in the objective function $(D_{(F2)}.1)$) by one. Due to constraints $(D_{(F2)}.2)$ and $(D_{(F2)}.3)$, increasing some γ_e by one implies the increase of some δ^k_e and also the decrease of α_k or β_i to preserve feasibility of the solution. In our algorithm, we only decrease α_k and keep β_i at zero. As α_k appears with a coefficient of minus one in the objective function, our goal is to iteratively set the variables in such a way, that the net-change in the objective is positive at each step, and we stop, when the change is non-positive.

```
input  : instance $G = (V, E)$ of the $S$-LP
output: feasible solution $(\alpha, \beta, \gamma, \delta)$ of $(D_{(F2)})$ with value $z^D$
1  $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\delta}) \leftarrow 0, (\boldsymbol{\gamma}) \leftarrow 1, z^D \leftarrow |E|$
2  $\Delta = \max_{i \in V} \delta(i)$
3  for $\bar{k} \in \{1, \dots, |V|\}$ do
4      if $|E| - \bar{k} \cdot \Delta > 0$ then
5          $z^D \leftarrow z^D + |E| - \Delta$
6          for $k \leq \bar{k}$ do
7              $\alpha_k \leftarrow \alpha_k + \Delta$
8              for $e \in E$ do
9                  $\delta_e^k \leftarrow \delta_e^k + 1$
10         for $e \in E$ do
11             $\gamma_e \leftarrow \gamma_e + 1$
12     else
13         break
```

Algorithm 2: Heuristic solution procedure for the dual $(D_{(F2)})$

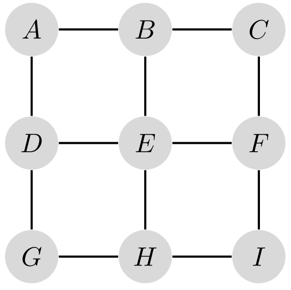
The key observations used in the design of the algorithm are the following: i) when we want to increase some γ_e from $\bar{k}$ to $\bar{k}+1$, all δ_e^k with $1 \leq k \leq \bar{k}+1$ need to be increased by one to keep feasibility, which in turn implies decreasing by one all α_k with $1 \leq k \leq \bar{k}+1$; ii) when α_k is set to some value $\bar{\alpha}_k$, for all adjacent edges e of any node i we can set $\bar{\alpha}_k$ variables δ_e^k to one and the solution remains feasible.

In our algorithm, we start with $\bar{k} = 1$ and a step consists of setting α_k, δ_e^k and γ_e for all $k \leq \bar{k}$ and some or all $e \in E$, then we proceed to the next step with $\bar{k} = \bar{k} + 1$. In the simplified version, we set α_k always to the maximum degree Δ of a node in the graph. Thus, we can always increase all γ_e at a step (as all δ_e^k can be increased), i.e., the positive change in the objective due to γ_e is $|E|$ at every step. Moreover, the negative change in the objective at every step due to setting the α_k is $\Delta \cdot \bar{k}$, as all α_k up to $\bar{k}$ need to be increased by Δ to keep the feasibility. Thus, the algorithm stops, when $\bar{k} \cdot \Delta \geq |E|$, and at each step before, the dual objective gets increased by $|E| - \bar{k} \cdot \Delta > 0$. Observe that the increase gets smaller at every step, as $\Delta \cdot \bar{k}$ grows with every step. The runtime of Algorithm 2 is $O(|V|^2|E|)$

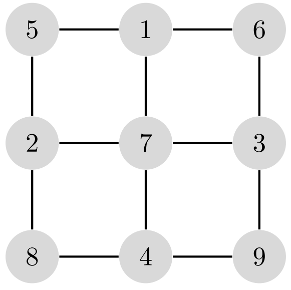
In the extended version of the algorithm, we do not just increase by Δ at a step, but try all the values up to Δ. If α_k is set to a value $\bar{\alpha}_k$ lower than Δ, for some $i \in V$, not all δ_e^k with $i \in e$ can be increased, which in turn means not all γ_e can be increased. Thus, a subset of e to increase needs to be chosen in these cases. We do this in a heuristic fashion: A list `active` of *active edges* is kept, and only such edges are considered for the remaining steps (i.e., increasing γ_e, δ_e^k and calculation of Δ which is re-calculated before every step considering only active edges). For each $\bar{\alpha}_k < \Delta$, we do the following: Make a temporary copy $\texttt{active}(\bar{\alpha}_k)$ of `active`. Let $\delta^a(i)$ be the degree of node i considering only edges in $\texttt{active}(\bar{\alpha}_k)$. We sort the nodes by $\delta^a(i)$, and then for each node i, we sort the adjacent active edges $e = \{i, i'\} \in \texttt{active}(\bar{\alpha}_k)$ by decreasing $\delta^a(i')$. We iterate through this list and set edges to be inactive in $\texttt{active}(\bar{\alpha}_k)$ until $\delta^a(i) \leq \bar{\alpha}_k$ and then proceed to the next node. After this procedure, increasing all γ_e in $\texttt{active}(\bar{\alpha}_k)$ (and also increasing resp., decreasing the associated δ_e^k, resp., α_k, $k \leq \bar{k}$) by one gives a feasible solution with a net-change $|\texttt{active}(\bar{\alpha}_k)| - \bar{k} \cdot \bar{\alpha}_k$ in the objective. At each step, we choose the value $\bar{\alpha}_k$ giving the largest positive net-change and set the dual variables and `active` accordingly (in case of ties, we take the smallest $\bar{\alpha}_k$ among the values giving the largest positive net-change). We stop, when there is no $\bar{\alpha}_k$ giving positive net-change at a step $\bar{k}$ (observe that as in the simplified variant, the net-change gets smaller in every step). The runtime of the extended version is $O(\Delta|V|^2|E|)$.

The following example illustrates both versions of the algorithm and also is a counter-example to Lemma 3 in Fertin et al. (2015), which claims that a node with maximum degree gets label one in any optimal labeling.

Example 1. *Consider the grid graph given in Figure 1a. The graph has $|V| = 9$, $|E| = 12$ and Δ is four and achieved by the node "E". An optimal labeling ϕ^* is given in Figure 1b with $SL_{\phi^*} = 30$. Note that only* *the nodes with label one, two, three and four contribute to the objective and node "E" does not have label one. Moreover, for this instance, when solving the LP-relaxation of (F2) we do not obtain SL_{ϕ^*} but* 29.6667*, i.e., (F2) has an integrality gap for this instance.*



(a) Instance



(b) Optimal labeling

Figure 1: Grid-graph instance and an optimal labeling ϕ^* for it, $SL_{\phi^*} = 3 \cdot 1 + 3 \cdot 2 + 3 \cdot 3 + 3 \cdot 4 = 30$.

The simplified variant of our algorithm proceeds as follows:

1. $\alpha_1 = 4$, $z^D = 20$ *(as* $|E| - \bar{k} \cdot \Delta = 12 - 1 \cdot 4 = 8$*)*
2. $\alpha_2 = 4, \alpha_1 = 8$, $z^D = 24$ *(as* $|E| - \bar{k} \cdot \Delta = 12 - 2 \cdot 4 = 4$*)*
3. *try* $\alpha_3 = 4, \alpha_2 = 8, \alpha_1 = 12$*, terminate as* $|E| - \bar{k} \cdot \Delta = 12 - 3 \cdot 4 = 0$

The extended variant of our algorithm proceeds as follows:

1. $\alpha_1 = 3$, $active_{BE} = false$, $z^D = 20$ *(as* $|active| - \bar{k} \cdot \delta^a = 11 - 1 \cdot 3 = 8$*)*
2. $\alpha_2 = 3, \alpha_1 = 6$, $active_{BE} = false$, $z^D = 25$ *(as* $|active| - \bar{k} \cdot \delta^a = 11 - 2 \cdot 3 = 5$*)*
3. $\alpha_3 = 3, \alpha_2 = 6, \alpha_1 = 9$, $active_{BE} = false$, $z^D = 27$ *(as* $|active| - \bar{k} \cdot \delta^a = 11 - 3 \cdot 3 = 2$*)*

Thus, the extended variant gives a better objective value of the dual solution.

3.2 Paths and Cycles

In Fertin and Vialette (2009); Fertin et al. (2015, 2017), the following closed formulas for SL_{ϕ^*} for paths P_n and cycles C_n with n nodes are given without proof: $SL_{\phi^*}(C_n) = SL_{\phi^*}(P_{n+1}) = \frac{n^2}{4} + \frac{n}{2}$ if n is even and $SL_{\phi^*}(C_n) = SL_{\phi^*}(P_{n+1}) = \frac{(n+1)^2}{4}$ if n is odd. We observe that paths are a special variant of caterpillar graphs, thus Fertin et al. (2015, 2017) give a polynomial-time algorithm for paths. In Algorithm 3, we give a linear-time algorithm for paths and in Algorithm 4 we do the same for cycles. We show that the objective value of the solution obtained by the respective algorithm (which is of course feasible for (F2) since it is a labeling) is the same value as the objective value of the solution for $D_{(F2)}$ produced by Algorithm 2. By strong duality this implies, that the solution is optimal and also means, that for paths and cycles, (F2) has no integrality gap.

```
input  : instance $G = (V, E)$ of the $S$-LP, with $G$ being a path
output: optimal labeling $\phi^*(G)$
1 start from an end of the path and label each even node with the smallest unused label
2 label the odd nodes arbitrarily
```

Algorithm 3: Optimal solution algorithm for the S-LP when the instance is a path

Theorem 2. *Suppose G is a path. Let z^P be the solution value obtained by Algorithm 3 and z^D be the solution value of $(D_{(F2)})$ for the dual solution obtained by Algorithm 2. We have $z^P = z^D$.*

Proof. We first calculate the value for z^P and then the value for z^D, and in both cases make a case distinction whether $|V|$ is even or odd.

- z^P
  - $|V|$ is odd
    By labeling every even node, we need $\frac{|V|-1}{2}$ labels to cover every edge, and each label covers two edges (the remaining labels do not contribute to the objective). Thus, $\phi = z^P$ for this labeling is
    $$2 \cdot \sum_{k=1}^{\frac{|V|-1}{2}} k = \frac{|V|-1}{2} \cdot \big(\frac{|V|-1}{2} + 1\big) = \frac{(|V|-1)^2}{4} + \frac{|V|-1}{2}$$
  - $|V|$ is even
    By labeling every even node, we need $\frac{|V|}{2}$ labels to cover every edge, and each label except the last covers two edges, and the last one covers one edge (the remaining labels do not contribute to the objective). Thus, $\phi = z^P$ for this labeling is
    $$\frac{|V|}{2} + 2 \cdot \sum_{k=1}^{\frac{|V|}{2}-1} k = \frac{|V|}{2} + \big(\frac{|V|}{2} - 1\big) \cdot \frac{|V|}{2} = \frac{|V|^2}{4}$$
- z^D
  In paths, we have $\Delta = 2$ and $|V| = |E|+1$. Thus, Algorithm 2 stops at step $\bar{k}$, when $(|V|-1)-2\cdot\bar{k} \leq 0$
  - $|V|$ is odd
    $(|V|-1) - 2 \cdot \bar{k} \leq 0$ when $\bar{k} = \frac{|V|-1}{2}$. As at each step k before termination, the net-change in z^D was $(|V|-1) - 2 \cdot k$ and the initial dual solution has value $|E|$, we obtain
    $$\begin{aligned} z^D &= \sum_{k=0}^{\frac{|V|-1}{2}-1} \Big((|V|-1) - 2 \cdot k\Big) \\ &= (|V|-1) \cdot \frac{|V|-1}{2} - 2 \cdot \sum_{k=0}^{\frac{|V|-1}{2}-1} k \\ &= \frac{(|V|-1)^2}{2} - \frac{|V|-1}{2} \cdot \big(\frac{|V|-1}{2} - 1\big) = \frac{(|V|-1)^2}{4} + \frac{|V|-1}{2}. \end{aligned}$$
  - $|V|$ is even
    $(|V|-1) - 2 \cdot \bar{k} \leq 0$ when $\bar{k} = \frac{|V|}{2}$. As at each step k before termination, the net-change in z^D was $(|V|-1) - 2 \cdot \bar{k}$ and the initial dual solution has value $|E|$, we obtain
    $$\begin{aligned} z^D &= \sum_{k=0}^{\frac{|V|}{2}-1} \Big((|V|-1) - 2 \cdot k\Big) \\ &= (|V|-1) \cdot \frac{|V|}{2} - 2 \cdot \sum_{k=0}^{\frac{|V|}{2}-1} k \\ &= \frac{|V|^2 - |V|}{2} - \frac{|V|}{2} \cdot \big(\frac{|V|}{2} - 1\big) = \frac{|V|^2}{4}. \end{aligned}$$

```
   input  : instance $G = (V, E)$ of the $S$-LP, with $G$ being a cycle
   output: optimal labeling $\phi^*(G)$
1  start with an arbitrary node and label each even node with the smallest unused label
2  label the odd nodes arbitrarily
```
Algorithm 4: Optimal solution algorithm for the S-LP when the instance is a cycle

□

Theorem 3. *Suppose G is a cycle. Let z^P be the solution value obtained by Algorithm 4 and z^D be the solution value of $(D_{(F2)})$ for the dual solution obtained by Algorithm 2. We have $z^P = z^D$.*

Proof. Similar to the proof for paths. □

3.3 Perfect n-ary trees

A perfect n-ary tree is a rooted tree, where all internal (i.e., non-leaf) nodes have k children and all leaf nodes are at the same depth d Black. Algorithm 5 gives a linear-time algorithm to solve the S-LP to optimality, and Theorem 4 shows that (F2) has no integrality gap for such trees and that $SL_{\phi^*} = \frac{(|V|-1)^2}{2\cdot(n+1)} + \frac{|V|-1}{2}$ in case d is odd, and $SL_{\phi^*} = \frac{(|V|-1-n)^2}{2\cdot(n+1)} + \frac{n\cdot(|V|-1-n)}{n+1} + \frac{|V|-1+n}{2}$ in case d is even.

```
   input  : instance $G = (V, E)$ of the $S$-LP, with $G$ being a perfect $n$-ary tree with depth $d$
   output: optimal solution $\phi^*(G)$ of the $S$-LP
1  if $d$ is odd then
2      take the nodes at even depths and label them with the smallest unused labels
3      arbitrarily label the remaining nodes
4  else
5      take the nodes at odd depths except the root node and label them with the smallest unused labels
6      label the root node with the smallest unused label
7      arbitrarily label the remaining nodes
```
Algorithm 5: Optimal solution algorithm for the S-LP when the instance is a perfect n-ary tree

Theorem 4. *Suppose G is a perfect n-ary tree with depth d. Let z^P be the solution value obtained by Algorithm 5 and z^D be the solution value of $(D_{(F2)})$ for the dual solution obtained by Algorithm 2. We have $z^P = z^D$.*

Proof. We first calculate the value for z^P and then the value for z^D, and in both cases make a case distinction whether d is even or odd.

- z^P
  - d is odd
    The nodes on the even depths cover every edge in the graph exactly once and each node covers $n+1$ edges (the edges going to its n children nodes, and the edge going to its parent node). Thus only these nodes contribute to the objective function and $\phi = z^P$ for this labeling is

$$(n+1)\cdot\sum_{k=1}^{\frac{|V|-1}{n+1}} k = \frac{(n+1)}{2}\frac{|V|-1}{n+1}\cdot\big(\frac{|V|-1}{n+1}+1\big) = \frac{(|V|-1)^2}{2\cdot(n+1)} + \frac{|V|-1}{2}$$

  - d is even
    The nodes on the odd depths cover every edge in the graph exactly once and each node except the root node covers $n+1$ edges (the edges going to its n children nodes, and the edge going to its parent node), while the root node covers n edges (the edges going to its n children nodes). Thus only these nodes contribute to the objective function and $\phi = z^P$ for this labeling is

$$(n+1) \cdot \sum_{k=1}^{\frac{|V|-1-n}{n+1}} k + n \cdot \big(\frac{|V|-1-n}{n+1} + 1\big) = \frac{n+1}{2} \cdot \frac{|V|-1-n}{n+1} \cdot \big(\frac{|V|-1-n}{n+1} + 1\big) + n \cdot \big(\frac{|V|-1-n}{n+1} + 1\big)$$
$$= \frac{(|V|-1-n)^2}{2 \cdot (n+1)} + \frac{|V|-1-n}{2} + \frac{n \cdot (|V|-1-n)}{n+1} + n = \frac{(|V|-1-n)^2}{2 \cdot (n+1)} + \frac{n \cdot (|V|-1-n)}{n+1} + \frac{|V|-1+n}{2}$$

- z^D
  We have $\Delta = n+1$ and $|V| = |E|+1$. Thus, Algorithm 2 stops at step $\bar{k}$, when $(|V|-1)-(n+1)\cdot\bar{k} \leq 0$
  - d is odd
    $(|V|-1)-(n+1)\cdot\bar{k} \leq 0$ when $\bar{k} = \frac{|V|-1}{n+1}$. As at each step k before termination, the net-change in z^D was $(|V|-1)-(n+1)\cdot k$ and the initial dual solution has value $|E|$, we obtain

$$\begin{aligned} z^D &= \sum_{k=0}^{\frac{|V|-1}{n+1}-1} \Big((|V|-1)-(n+1)\cdot k\Big) \\ &= (|V|-1) \cdot \frac{|V|-1}{n+1} - (n+1) \sum_{k=0}^{\frac{|V|-1}{n+1}-1} k \\ &= \frac{(|V|-1)^2}{n+1} - \frac{n+1}{2} \big(\frac{|V|-1}{n+1} - 1\big) \cdot \frac{|V|-1}{n+1} \\ &= \frac{(|V|-1)^2}{2\cdot(n+1)} + \frac{|V|-1}{2} \end{aligned}$$

  - d is even
    $(|V|-1)-(n+1)\cdot\bar{k} \leq 0$ when $\bar{k} = \frac{|V|-1-n}{n+1} + 1$. As at each step k before termination, the net-change in z^D was $(|V|-1)-(n+1)\cdot k$ and the initial dual solution has value $|E|$, we obtain

$$\begin{aligned} z^D &= \sum_{k=0}^{\frac{|V|-1-n}{n+1}} \Big((|V|-1)-(n+1)\cdot k\Big) \\ &= (|V|-n+n-1) \cdot \Big(\frac{|V|-1-n}{n+1} + 1\Big) - (n+1) \sum_{k=0}^{\frac{|V|-1-n}{n+1}} k \\ &= \frac{(|V|-1-n)^2}{n+1} + \frac{n\cdot(|V|-1-n)}{n+1} + |V| - 1 - \frac{n+1}{2} \cdot \Big(\frac{|V|-1-n}{n+1} + 1\Big) \cdot \frac{|V|-1-n}{n+1} \\ &= \frac{(|V|-1-n)^2}{2\cdot(n+1)} + \frac{n\cdot(|V|-1-n)}{n+1} + |V| - 1 - \frac{|V|-1-n}{2} \\ &= \frac{(|V|-1-n)^2}{2\cdot(n+1)} + \frac{n\cdot(|V|-1-n)}{n+1} + \frac{|V|-1+n}{2} \end{aligned}$$

□

4. Additional solution approaches

In this section, we describe a Lagrangian heuristic and a constraint programming formulation for the S-LP.

4.1 Lagrangian heuristic

Let

$$z^* = \min \left\{ \mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \le \mathbf{b},\ H\mathbf{x} \le \mathbf{h} \text{ and } \mathbf{x} \in \mathbb{Z}^n \right\}, \tag{COP}$$

be combinatorial optimization problem (COP), which can be formulated as MIP with a set of *easy* constraints $A\mathbf{x} \le \mathbf{b}$ and *complicating* constraints $H\mathbf{x} \le \mathbf{h}$. *Easy* and *complicating* constraints means, that the problem (COP) without constraints $H\mathbf{x} \le \mathbf{h}$ is easy to solve, e.g., using a combinatorial algorithm. Lagrangian relaxation (see, e.g., (Fisher, 1981; Wolsey, 1998)) is an attractive way to solve solve such problems.

Let $\boldsymbol{\lambda} \ge 0$ be a vector of *dual multipliers* for $H\mathbf{x} \le \mathbf{h}$. The Lagrangian relaxation of (COP) for a given $\boldsymbol{\lambda}$ is defined as

$$z_R(\boldsymbol{\lambda}) = \min \left\{ \left(\mathbf{c}^T + \boldsymbol{\lambda}^T H \right) \mathbf{x} - \boldsymbol{\lambda}^T \mathbf{h} \mid A\mathbf{x} \le \mathbf{b} \text{ and } \mathbf{x} \in \mathbb{Z}^n \right\}. \tag{LR}$$

The value $z_R(\boldsymbol{\lambda})$ gives a lower bound for the objective of (COP), i.e., $z^* \ge z_R(\boldsymbol{\lambda})$, and to find the best lower bound, a maximization problem in $\boldsymbol{\lambda}$, called the *Lagrangian dual problem*,

$$\max_{\boldsymbol{\lambda} \ge 0} z_R(\boldsymbol{\lambda}) \tag{LD}$$

is solved. We use a subgradient method to solve (LD). Within the subgradient method, the Lagrangian relaxation (LR) gets iteratively solved for different multipliers $\boldsymbol{\lambda}$ and the best value of $z_R(\boldsymbol{\lambda})$ is taken as lower bound z_{LB}. The value of $\boldsymbol{\lambda}^{t+1}$ at iteration $t+1$ of the subgradient method, is obtained from the current solution $\mathbf{x}^t$ with the help of a subgradient $\mathbf{g}^t$, which is calculated as $\mathbf{g}^t = \mathbf{h} - H\mathbf{x}^t$. We use a standard variant for updating the multipliers, which we describe in the following (see, e.g., (Conforti et al., 2014; Wolsey, 1998) for more details): The value of $\boldsymbol{\lambda}^{t+1}$ is calculated as $\boldsymbol{\lambda}^{t+1} = \max\{0, \boldsymbol{\lambda}^t - \mu^t \mathbf{g}^t\}$, with the step-size $\mu^t = \beta \frac{z^I - z_R(\boldsymbol{\lambda}^t)}{\|\mathbf{g}^t\|}$, where z^I is the value of the best feasible solution found so far, and β a given parameter in $(0, 2]$. We initialize β with two, and if there are $\tau = 7$ iterations without an improvement of $z_R(\boldsymbol{\lambda}^t)$, we set $\beta = \beta/2$. We have an iteration limit of 500 iterations as stopping criterion, moreover, we stop when either $z_I - z_{LB} < 1$, $\mu^t < 10^{-5}$ or $||g^t|| < 10^{-6}$.

To apply Lagrangian relaxation to the S-LP, we take formulation (F2), and relax constraints (F2.3), let $\delta_e^k \ge 0$ be the associated dual multipliers. We obtain the following relaxed problem $(LR_{(F2)})$.

$$\begin{aligned}
(LR_{(F2)}) \quad \min \quad & \sum_{e \in E} \sum_{1 \le k \le |V|-1} (k + \delta_e^k) d_e^k - \sum_{i \in V} \sum_{1 \le k \le |V|-1} \Big(\sum_{e \in E: i \in e} \delta_e^k \Big) x_i^k && ((LR_{(F2)}).1) \\
& \text{(UNIQUE)}, \qquad \text{(ONELABEL)} \\
& \sum_{1 \le k \le |V|} d_e^k = 1 \qquad \forall e \in E && ((LR_{(F2)}).1) \\
& x_i^k \in \{0,1\} \qquad 1 \le i, k \le V \\
& d_e^k \in \{0,1\} \qquad \forall e \in E, 1 \le k \le V \\
& \delta_e^k \ge 0 \qquad \forall e \in E, 1 \le k \le V
\end{aligned}$$

For a fixed set of multipliers $\bar{\delta}_e^k$, it is easy to see, that the problem decomposes into a maximum assignment problem in the x-variables, i.e.,

$$\begin{aligned}
\max \quad & \sum_{i \in V} \sum_{1 \le k \le |V|-1} \Big(\sum_{e \in E: i \in e} \bar{\delta}_e^k \Big) x_i^k \\
& \text{(UNIQUE)}, \text{(ONELABEL)} \\
& x_i^k \in \{0,1\} \qquad 1 \le i, k \le V
\end{aligned}$$

and for each edge $e \in E$ into a simple problem, where the index k with minimum objective coefficient needs to be selected, i.e.,

$$\begin{aligned} \min \quad & \sum_{1 \leq k \leq |V|-1} (k + \bar{\delta}_e^k) d_e^k \\ & \sum_{1 \leq k \leq |V|} d_e^k = 1 \\ & d_e^k \in \{0, 1\} \quad 1 \leq k \leq V \end{aligned}$$

Every solution of the assignment problem during the course of the subgradient algorithm gives a feasible labeling ϕ_H and we use the local-search phase of Algorithm 1 and try to improve the obtained ϕ_H. We initialize the multipliers δ_e^k with the values obtained by the extended version of the dual heuristic described in Algorithm 2. We also use inequalities (TRIANGLE), by adding and relaxing all of them for the increasing subsets of the labels up to $\{1, 2, \ldots, |V| - 1\}$. To generate an initial starting solution, we use Algorithm 1.

4.2 A Constraint Programming formulation

Let y_i be an integer variable denoting the label of node $i \in V$. Using the *min*-constraint and the *alldifferent*-constraint, S-LP can be formulated as constraint programming problem as follows

$$\min \sum_{e=\{i,i'\} \in E} \min\{y_i, y_{i'}\} \tag{C.1}$$

$$alldifferent(y) \tag{C.2}$$

$$y_i \in \{1, 2, \ldots, |V|\}, \quad \forall i \in V \tag{C.3}$$

5. Computational results

All approaches were implemented in C++, the branch-and-bound/cut frameworks were implemented using CPLEX 12.7, and the constraint programming solver was also implemented using the same CPLEX version. To solve the assignment problems arising as subproblems in the Lagrangian relaxation, and also in the primal heuristic of the branch-and-bound framework, we used the algorithm available at `http://dlib.net/`, which implements the Hungarian method Kuhn (1955). The runs were made an Intel Xeon E5 v4 CPU with 2.2 GHz and 3GB memory and using a single thread. The timelimit for a run was set to 600 seconds and all CPLEX-settings were left at default, except when solving (F1), details of the changed settings in this case are given in Section 5.2.

5.1 Computational tests for special graph classes

First, we are interested, if there may be additional graph classes, for which the LP-relaxation of our MIPs exhibits no integrality gap. Thus we generated the following sets of graphs using the graph generators of the `NetworkX`-package Hagberg et al. (2008).

- `grid`: We generated $|V| \times |V|$ grid graphs for $|V| \in \{3, 4, \ldots, 12\}$ using the `grid_graph`-function.
- `bipartite graphs`: We generated (n, m) bipartite graphs, with edge probability p, for $(n, m) \in \{(5, 5), (10, 10), (15, 15), (20, 20), (25, 25)\}$ and $p \in \{0.25, 0.5\}$ using the `random_graph`-function of the `bipartite`-module.
- `caterpillar`: A caterpillar is a tree, which reduces to a path, when all leaf nodes are deleted. In Fertin et al. (2015, 2017) a polynomial-time algorithm for the S-LP on caterpillar graphs is given. We

generated caterpillars with the expected number of nodes in the path in $\{10, 20, 30, 40, 50\}$ and the probability of adding edges to the underlying path in $\{0.25, 0.5\}$ using the `random_lobster`-function (and setting the probability of adding a second level edges to zero).

- `lobster`: A lobster graph is a tree, which reduces to a caterpillar, when all leaf nodes are deleted. We generated lobsters with the expected number of nodes in the path in $\{10, 20, 30, 40, 50\}$ and the probability of adding edges to the underlying path in $\{0.25, 0.5\}$ and also the same probability of adding a second level of edges using the `random_lobster`-function
- `tree`: We generated trees with $|V| = \{10, 15, 20, 25, 30, 35, 40, 50, 75, 100\}$ using the `random_tree`-function

Tables 1a to 1e show the value of the LP-relaxation (columns z_{LP}) of $(F2)$ (recall that both formulations have the same strength, as shown in Section 2.1) and the value of the optimal solution (columns z^*), obtained by solving the MIP, a bold entry in z_{LP} means there is no gap. Only for three of the tested graphs, there is a gap. Two of these three are grid graphs, and one is a lobster graph. Hence, for grid graphs and trees (as a lobster is a special case of a tree), $(F2)$ has an integrality gap. Interestingly, while for a lobster there is a gap, for the instances in `tree`, there is no gap. Thus, clearly caution is advised when drawing conclusions from these results regarding the integrality gap of $(F2)$. However, for caterpillars, these results plus the fact, that for these graphs, the problem is known to be polynomial-time solvable, this may suggest, that $(F2)$ gives no integrality gap. For bipartite graphs, it is also interesting to see that for the tested instances, there is no gap, as the complexity of the S-LP is still unknown for these graphs. Unfortunately, we were unable to prove results regarding the integrality gap for both graph types.

Table 1: Values of LP-relaxation and optimal solution for special graphs

name	z^*	z_{LP}
gridgraph3	30	29.67
gridgraph4	96	**96**
gridgraph5	242	241.67
gridgraph6	514	**514**
gridgraph7	972	**972**
gridgraph8	1692	**1692**
gridgraph9	2750	**2750**
gridgraph10	4254	**4254**
gridgraph11	6296	**6296**
gridgraph12	9016	**9016**

(a) Grid graphs

name	z^*	z_{LP}
bipartite5-5-0.25	6	**6**
bipartite5-5-0.5	19	**19**
bipartite10-10-0.25	85	**85**
bipartite10-10-0.5	211	**211**
bipartite15-15-0.25	254	**254**
bipartite15-15-0.5	707	**707**
bipartite20-20-0.25	880	**880**
bipartite20-20-0.5	1893	**1893**
bipartite25-25-0.25	1837	**1837**
bipartite25-25-0.5	3641	**3641**

(b) Bipartite graphs

name	z^*	z_{LP}
caterpillar10-0.25	41	**41**
caterpillar10-0.5	57	**57**
caterpillar20-0.25	434	**434**
caterpillar20-0.5	503	**503**
caterpillar30-0.25	306	**306**
caterpillar30-0.5	397	**397**
caterpillar40-0.25	508	**508**
caterpillar40-0.5	583	**583**
caterpillar50-0.25	682	**682**
caterpillar50-0.5	932	**932**

(c) Caterpillar graphs

name	z^*	z_{LP}
lobster10-0.25-0.25	41	**41**
lobster10-0.5-0.5	72	**72**
lobster20-0.25-0.25	492	**492**
lobster20-0.5-0.5	793	**793**
lobster30-0.25-0.25	361	**361**
lobster30-0.5-0.5	551	**551**
lobster40-0.25-0.25	533	**533**
lobster40-0.5-0.5	766	**766**
lobster50-0.25-0.25	737	**737**
lobster50-0.5-0.5	1267	1266.5

(d) Lobster graphs

name	z^*	z_{LP}
tree10	19	**19**
tree15	40	**40**
tree20	77	**77**
tree25	129	**129**
tree30	152	**152**
tree35	186	**186**
tree40	235	**235**
tree50	386	**386**
tree75	943	**943**
tree100	1692	**1692**

(e) Tree graphs

5.2 Computational tests on general graphs

In this section, we are interested in investigating the computational effectiveness of the proposed solution approaches for the S-LP when dealing with general graphs. The considered graphs are as follows:

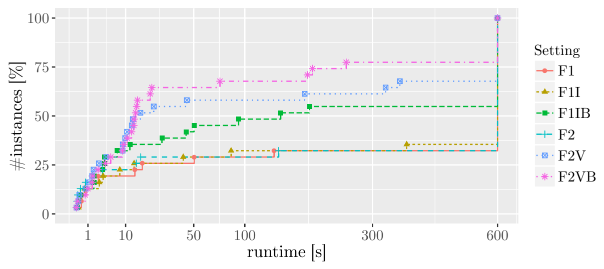
- HB: Graphs from the Harwell-Boeing Sparse Matrix Collection, which is a "is a set of standard test matrices arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines", see `https://math.nist.gov/MatrixMarket/collections/hb.html`. These graphs have for example be used when testing approaches for graph bandwidth problems, which are another type of labeling problem Duarte et al. (2011); Rodriguez-Tello et al. (2015). Details of the number of nodes and edges of the individual graphs are given in the results-tables in columns $|V|$ and $|E|$. For testing, we partitioned the set into small and large graphs, denoted by the suffixes `sm` and `lg` in the names of the set. Graphs with $|V| \leq 118$ are considered as small, and all others are considered as large. There are 28 graphs, and 12 are large.

- RND: Random graphs with $|V|$ nodes and $|E|$ edges, generated using the `gnm_random_graph`-function of the `NetworkX`-package. We generated five graphs each for the following $(|V|, |E|)$-pairs: $(|V|, |E|) \in \{(50, 100), (50, 150), (50, 200), (500, 1000), (500, 1500), (500, 2000)\}$, the graphs with 50 nodes are considered as small (`sm`) and the ones with 500 nodes as large (`lg`).

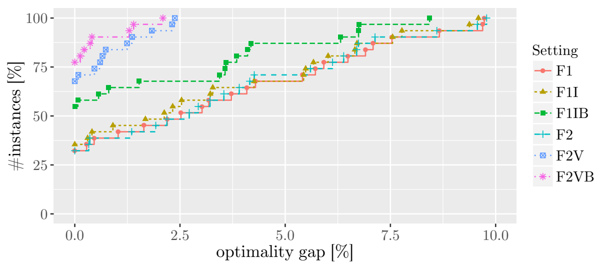
We first tested different configurations of our MIP approaches:

- `F1`: The branch-and-bound based on (F1) with the starting and primal heuristic described in Section 2.2; without the branching-scheme and initialization-scheme described in Section 2.3
- `F1I`: Setting `F1` and the initialization-scheme as described in Section 2.3
- `F1IB`: Setting `F1I` and the branching-scheme as described in Section 2.3
- `F2`: The branch-and-bound based on (F2) with the starting and primal heuristic described in Section 2.2; without the branching-scheme described in Section 2.3 and the valid inequalities (TRIANGLE)
- `F2V`: Setting `F2` and the valid inequalities (TRIANGLE)
- `F2VB`: Setting `F2V` and the branching-scheme as described in Section 2.3

Figures 2a and 2b show the runtime and optimality gap for these settings and the small instances (i.e., `HB-sm` and `RND-sm`). The optimality gap $g[\%]$ is calculated as $100 \cdot (z^D - z^*)/z^*$, where z^D the value of the dual bound and z^* is the value of the best solution found by the setting.



(a) Runtime



(b) Optimality gap

Figure 2: Runtime and optimality gap for the small instances and different settings

In the figures, we see that `F2VB` is the most effective setting, managing to solve about 75% of the instances within the given timelimit, and the maximum gap is under 2.5%. Settings `F2V` and `F1IB` also manage to solve over 50% of the instances. In terms of optimality gap, `F2VB` and `F2V` are quite similar, while `F1IB`has gaps up to 7.5%. The remaining settings are also quite similar, with about 37.5% of solved instances and gap of up to 10%. Thus, the valid inequalities (TRIANGLE) (used in `F2VB`and `F2V`) and the branching-scheme (used in `F2VB` and `F1IB`) seem to be quite helpful for solving the problem.

Tables 2 and 3 give detailed results for approaches `F1IB`, `F2VB`, `LAG` and `HEUR` and instance sets `HB-sm` and `RND-sm`, the performance of the extended version of Algorithm 2 is also reported denoted as `DUAL`. We note that the values z_D of `DUAL` and z^* of `HEUR` are always worse than the corresponding values obtained by `F1IB`, `F2VB`, `LAG` as the latter are initialized based on `DUAL` and `HEUR` (see Sections 2.2,2.3,4.1). For `F1IB` and `F2VB` we also report the LP-bound at the root node in column z_R. Bold entries in the columns mean that the method found the best values of z_D and z^* amongst all considered methods. We see that on the primal side (i.e., z^*), `F1IB`, `F2VB`, `LAG` all find the best solution value for all instances, while `CP` and the stand-alone heuristic `HEUR` only find the best solution value for some instances. Looking at the runtime, `LAG` takes at most five seconds, compared with the MIP-based methods `F1IB` and `F2VB`, which sometimes run until the timelimit, thus `LAG` can be a good option, if one wants to find a good solution value quickly. Looking at the dual side, we see that `F2VB` provides the best lower bound for all instances and the root-bound z_R is better than the root-bound of `F1IB` for nearly all instances, so the valid inequalities (TRIANGLE) seem quite helpful. `LAG`, which also uses (TRIANGLE) sometimes has better bounds than z_R of `F1IB`(the MIP-approaches also benefit from the general-purpose cuts of CPLEXs). The bounds provided by `DUAL` are about 10% worse than z_R of `F1IB` for most of the instances, however, for some they are also considerably smaller (e.g., instance `D-bcsstk01`), and for instance `rgg010` `DUAL` even manages to find the LP-bound. Regarding the runtime for the MIP-approaches, the size of the instance seems to be important, as for all except one of the largest instances (i.e., the ones with $|E| = 200$) of `RND-sm`, both `F1IB` and `F2VB` terminate due to the timelimit, and also for `HB-sm`, there is a connection between termination due to timelimit and number of edge of an instance.

Table 2: Results for instances class `HB-sm`

			`F1IB`					`F2VB`					`LAG`				`CP`		`HEUR`		`DUAL`	
name	$\|V\|$	$\|E\|$	z_D	z^*	$g[\%]$	$t[s]$	z_R	z_D	z^*	$g[\%]$	$t[s]$	z_R	z_D	z^*	$g[\%]$	$t[s]$	z^*	$t[s]$	z^*	$t[s]$	z_D	$t[s]$
A-pores-1	30	103	811.4	**818**	0.81	*TL*	744.00	**818**	**818**	0.00	14	807.77	756.27	**818**	7.55	0	**818**	*TL*	832	0	723	0
B-ibm32	32	90	**651**	**651**	0.00	2	626.50	**651**	**651**	0.00	2	651.00	613.16	**651**	5.81	0	**651**	*TL*	**651**	0	584	0
can-24	24	68	418.5	**425**	1.53	*TL*	374.50	**425**	**425**	0.00	12	417.48	390.88	**425**	8.03	0	**425**	*TL*	**425**	0	359	0
C-bcspwr01	39	46	**332**	**332**	0.00	0	332.00	**332**	**332**	0.00	0	332.00	328.06	**332**	1.19	0	**332**	*TL*	**332**	0	328	0
D-bcsstk01	48	176	2037.39	**2225**	8.43	*TL*	1986.34	**2220.05**	**2225**	0.22	*TL*	2210.31	2047.87	**2225**	7.96	1	2226	*TL*	2244	0	1936	0
E-bcspwr02	49	59	**471**	**471**	0.00	0	471.00	**471**	**471**	0.00	0	471.00	457.87	**471**	2.79	0	**471**	*TL*	**471**	0	452	0
F-curtis54	54	124	1341	**1342**	0.07	*TL*	1295.13	**1342**	**1342**	0.00	15	1341.00	1292.06	**1342**	3.72	1	1347	*TL*	1347	0	1233	0
G-will57	57	127	1361.25	**1369**	0.57	*TL*	1296.83	**1369**	**1369**	0.00	11	1369.00	1254.20	**1369**	8.39	1	1375	*TL*	1379	0	1211	0
H-impcol-b	59	281	3222.25	**3363**	4.19	*TL*	3162.00	**3349.33**	**3363**	0.41	*TL*	3346.62	3071.46	**3363**	8.67	4	3387	*TL*	3378	0	3001	0
I-ash85	85	219	4114.28	**4412**	6.75	*TL*	4071.16	**4412**	**4412**	0.00	185	4412.00	4093.26	**4412**	7.22	5	4600	*TL*	4444	0	3890	0
jgl009	9	32	**95**	**95**	0.00	0	84.75	**95**	**95**	0.00	1	92.17	85.86	**95**	9.62	0	**95**	52	**95**	0	83	0
jgl011	11	49	**175**	**175**	0.00	4	150.75	**175**	**175**	0.00	5	165.73	151.47	**175**	13.45	0	**175**	*TL*	**175**	0	147	0
J-nos4	100	247	5455	**5658**	3.59	*TL*	5416.00	**5539.5**	**5658**	2.09	*TL*	5523.50	5285.67	**5658**	6.58	3	5996	*TL*	5667	0	5218	0
K-dwt–234	117	162	**2169**	**2169**	0.00	3	2169.00	**2169**	**2169**	0.00	2	2169.00	2108.79	**2169**	2.78	2	2172	*TL*	**2169**	0	2105	0
L-bcspwr03	118	179	**3557**	**3557**	0.00	28	3546.25	**3557**	**3557**	0.00	13	3557.00	3440.42	**3557**	3.28	3	3695	*TL*	**3557**	0	3418	0
rgg010	10	45	**165**	**165**	0.00	93	135.00	**165**	**165**	0.00	193	145.29	139.21	**165**	15.63	0	**165**	*TL*	**165**	0	135	0

Table 3: Results for instances class `RND-sm`

			`F1IB`					`F2VB`					`LAG`				`CP`		`HEUR`		`DUAL`	
name	$\|V\|$	$\|E\|$	z_D	z^*	$g[\%]$	$t[s]$	z_R	z_D	z^*	$g[\%]$	$t[s]$	z_R	z_D	z^*	$g[\%]$	$t[s]$	z^*	$t[s]$	z^*	$t[s]$	z_D	$t[s]$
random50-100-1	50	100	**911**	**911**	0.00	1	911.00	**911**	**911**	0.00	1	911.00	871.84	**911**	4.30	0	**911**	*TL*	**911**	0	858	0
random50-100-2	50	100	**1053**	**1053**	0.00	4	1033.00	**1053**	**1053**	0.00	4	1053.00	971.08	**1053**	7.78	0	**1053**	*TL*	1062	0	956	0
random50-100-3	50	100	**994**	**994**	0.00	2	987.00	**994**	**994**	0.00	3	994.00	946.17	**994**	4.81	0	996	*TL*	999	0	923	0
random50-100-4	50	100	**1039**	**1039**	0.00	50	1000.00	**1039**	**1039**	0.00	22	1023.50	962.79	**1039**	7.33	0	**1039**	*TL*	1048	0	933	0
random50-100-5	50	100	**978**	**978**	0.00	7	960.00	**978**	**978**	0.00	14	972.50	909.90	**978**	6.96	0	985	*TL*	990	0	887	0
random50-150-1	50	150	**1599**	**1599**	0.00	44	1543.50	**1599**	**1599**	0.00	9	1599.00	1487.13	**1599**	7.00	1	1610	*TL*	1611	0	1447	0
random50-150-2	50	150	1674	**1736**	3.57	*TL*	1606.00	**1736**	**1736**	0.00	250	1711.00	1554.26	**1736**	10.47	1	1739	*TL*	1741	0	1520	0
random50-150-3	50	150	**1593**	**1593**	0.00	12	1553.70	**1593**	**1593**	0.00	9	1593.00	1484.96	**1593**	6.78	1	1598	*TL*	**1593**	0	1469	0
random50-150-4	50	150	**1612**	**1612**	0.00	145	1532.50	**1612**	**1612**	0.00	21	1608.22	1461.63	**1612**	9.33	1	1623	*TL*	1628	0	1421	0
random50-150-5	50	150	**1618**	**1618**	0.00	188	1543.00	**1618**	**1618**	0.00	13	1618.00	1510.53	**1618**	6.64	1	1621	*TL*	1621	0	1473	0
random50-200-1	50	200	2236.5	**2326**	3.85	*TL*	2181.53	**2323**	**2326**	0.13	*TL*	2308.88	2131.75	**2326**	8.35	1	2331	*TL*	2340	0	2059	0
random50-200-2	50	200	2314	**2470**	6.32	*TL*	2254.00	**2438.06**	**2470**	1.29	*TL*	2424.60	2233.40	**2470**	9.58	1	2484	*TL*	2478	0	2152	0
random50-200-3	50	200	2276.5	**2374**	4.11	*TL*	2209.50	**2365.31**	**2374**	0.37	*TL*	2344.50	2153.92	**2374**	9.27	1	2386	*TL*	**2374**	0	2089	0
random50-200-4	50	200	2213	**2373**	6.74	*TL*	2128.00	**2339.81**	**2373**	1.40	*TL*	2318.12	2078.76	**2373**	12.40	1	2381	*TL*	2381	0	2000	0
random50-200-5	50	200	2244	**2324**	3.44	*TL*	2155.50	**2324**	**2324**	0.00	74	2317.47	2136.98	**2324**	8.05	1	2332	*TL*	2364	0	2056	0

Finally, we turn our attention to the larger instances, i.e., `HB-lg` and `RND-lg`. For these instances, we tested `LAG`, `F1I` (due to the size of the instances, the LP-solving time becomes prohibitive for any setting based on (F2)), `HEUR` and also report on the performance of `DUAL`. For dealing with these larger instances with `F1I`, instead of the default CPLEX setting for the LP-algorithm and pricing-strategy, we explicitly set the primal simplex algorithm with reduced-cost pricing. This turned out to be beneficial in preliminary runs and is motivated by the fact, that there are many more columns than rows in the formulation and the formulation is very dense. Note that for these larger instances and the given timelimit `F1I` and `F1IB`are the same, as the root node of the branch-and-bound tree is not finished within the timelimit for any instance. Table 4 gives the results for `HB-lg` and Table 5 for `RND-lg`.

We see, that regarding primal solutions, `LAG` find the best solution value for all instances, while `HEUR` for none and `F1I` only for one, so the Lagrangian approach seems to be quite helpful for finding good primal solutions. However, compared to the small instances, `LAG` now becomes more time consuming and for instances of `HB-lg` some runs terminate due to the timelimit. Regarding dual bounds, for instance class `HB-lg` `F1I` finds the best bound for five instances, and `LAG` for seven, on the other hand, for `RND-lg` `F1I` finds the best bound for all instances, while `LAG` for none. The gaps provided by both `F1I` and `LAG` are comparable and between 3% and 18%, for `HB-lg`, `LAG` seems slightly better and for `RND-lg` `F1I`. The dual bounds provided by `DUAL` are not far from the bounds provided by `F1I` and `LAG`, while the runtime is much faster (i.e., at most one second). Thus, a branch-and-bound based on a dual heuristic like `DUAL` could be interesting to explore in further work.

Table 4: Results for instances class `HB-lg`

			`F1I`			`LAG`				`HEUR`		`DUAL`	
name	$\|V\|$	$\|E\|$	z_D	z^*	$g[\%]$	z_D	z^*	$g[\%]$	$t[s]$	z^*	$t[s]$	z_D	$t[s]$
M-bcsstk06.mtx	420	3720	309263.31	377439	18.06	**323231.00**	**376169**	14.07	*TL*	383166	0	306337	2
N-bcsstk07.mtx	420	3720	309263.31	377439	18.06	**323231.00**	**376169**	14.07	*TL*	383166	0	306337	2
O-impcol-d.mtx	425	1267	**97924.49**	103300	5.20	94650.20	**102501**	7.66	277	103312	0	93511	0
P-can–445.mtx	445	1682	170217.77	197273	13.71	**178140.00**	**196762**	9.46	*TL*	198158	0	169953	0
Q-494-bus.mtx	494	586	**43972.02**	**43999**	0.06	42620.30	**43999**	3.13	125	44418	0	42477	0
R-dwt–503.mtx	503	2762	258088.51	316834	18.54	**270426.00**	**316403**	14.53	*TL*	317347	0	250309	1
S-sherman4.mtx	546	1341	**162642.47**	169284	3.92	162372.00	**168914**	3.87	66	171364	0	162372	0
T-dwt–592.mtx	592	2256	295127	342013	13.71	**309821.00**	**341088**	9.17	*TL*	342583	0	295133	1
U-662-bus.mtx	662	906	**93056.25**	95509	2.57	91209.90	**95173**	4.16	347	96009	0	91005	0
V-nos6.mtx	675	1290	208656	211923	1.54	**208658.00**	**211908**	1.53	99	214026	0	208658	0
W-685-bus.mtx	685	1282	**150721.43**	162327	7.15	150436.00	**161821**	7.04	569	162327	0	148193	0
X-can–715.mtx	715	2975	409861.59	465576	11.97	**426068.00**	**464250**	8.22	*TL*	465576	0	409014	1

Table 5: Results for instances class `RND-lg`

			`F1I`			`LAG`				`HEUR`		`DUAL`	
name	$\|V\|$	$\|E\|$	z_D	z^*	$g[\%]$	z_D	z^*	$g[\%]$	$t[s]$	z^*	$t[s]$	z_D	$t[s]$
random500-1000-1	500	1000	**87139.6**	92236	5.53	84429.00	**91746**	7.98	47	92851	0	83903	0
random500-1000-2	500	1000	**87546.29**	91361	4.18	84553.00	**91124**	7.21	45	91607	0	84120	0
random500-1000-3	500	1000	**86763.58**	90970	4.62	83530.50	**90636**	7.84	161	91605	0	82662	0
random500-1000-4	500	1000	**87855.94**	91805	4.30	84837.00	**91470**	7.25	47	92578	0	84551	0
random500-1000-5	500	1000	**85586.29**	89547	4.42	82675.00	**89305**	7.42	159	90019	0	82139	0
random500-1500-1	500	1500	**140414.03**	151689	7.43	136347.00	**150407**	9.35	226	152341	0	134706	0
random500-1500-2	500	1500	**139672.08**	152517	8.42	136926.00	**151704**	9.74	237	152517	0	135266	0
random500-1500-3	500	1500	**139741.86**	149669	6.63	135483.00	**149240**	9.22	224	150462	0	133744	0
random500-1500-4	500	1500	**140033.31**	153100	8.53	136242.00	**151839**	10.27	235	153315	0	135021	0
random500-1500-5	500	1500	**141017.18**	154663	8.82	138273.00	**154249**	10.36	65	155512	0	137289	0
random500-2000-1	500	2000	**189879.38**	214580	11.51	189065.00	**213581**	11.48	320	214778	0	187004	1
random500-2000-2	500	2000	**193594.01**	221148	12.46	192783.00	**219807**	12.29	325	221148	0	190568	1
random500-2000-3	500	2000	**192637.36**	214499	10.19	188478.00	**212952**	11.49	332	214499	0	186467	1
random500-2000-4	500	2000	**196162.85**	217207	9.69	192015.00	**215966**	11.09	301	217594	0	189697	1
random500-2000-5	500	2000	**198630.17**	220397	9.88	194450.00	**219567**	11.44	322	220397	0	192649	1

6. Conclusions

In this work, we studied the recently introduced S-labeling problem, in which the nodes get labeled using labels from 1 to $|V|$ and for each edge the contribution to the objective function, called S-labeling number of the graph, is the minimum label of its end-nodes. The goal is to find a labeling ϕ^* with minimum value. We presented two Mixed-Integer Programming (MIP) formulations for the problem and developed branch-and-cut solution frameworks based on it. These frameworks were enhanced with with valid inequalities, starting and primal heuristics and specialized branching rules. Moreover, we showed that one of the MIP formulations is the projection of the other and, with the help of a (heuristic) algorithm to solve the dual in the spirit of dual-ascent algorithms, that our MIP formulations have no integrality gap for paths, cycles and perfect n-ary trees. We gave, to the best of our knowledge, the first polynomial-time algorithm for the problem on n-ary trees as well as a closed formula for the S-labeling number. Finally, we also presented a Lagrangian heuristic and a constraint programming approach.

We assessed the efficiency of our proposed solution methods in a computational study. The study reveals, that for caterpillar graphs and bipartite graphs our formulation may also have no integrality gap, unfortunately, we were not able to prove any results on this. Moreover, our MIP-approaches are quite effective for solving the S-LP to optimality on general graphs with up to around 100 nodes within the timelimit of 600 seconds, and the proposed enhancements, especially the valid inequalities, are helpful. For larger graphs with up to 1000 nodes, the Lagrangian heuristic produces solutions with an optimality gap of about 5-15% for most of the considered instances and the smaller-sized MIP formulation also provides good results (the size of the MIP formulations becomes burdensome for these larger graphs). There are various avenues for further work: i) further enhancing the presented MIP-approaches by e.g., additional valid inequalities or other techniques; ii) development of alternative (smaller-sized) MIP-approaches or (meta-)heuristic algorithms to deal with large-scale instances, a (combinatorial) branch-and-bound based using the dual heuristic or the Lagrangian relaxation for providing bounds could also be interesting in this regard; iii) further investigation, if there are additional graph classes, where the problem can be solved in polynomial-time, in particular, bipartite graphs could be an interesting class, as our presented MIP may have no integrality gap.

Acknowledgements

The research was supported by the Austrian Research Fund (FWF, Project P 26755-N19).

References

References

T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

A. Balakrishnan, T. L. Magnanti, and R. T. Wong. A dual-ascent procedure for large-scale uncapacitated network design. *Operations Research*, 37(5):716–740, 1989.

P. E. Black. Perfect k-ary tree. In *Dictionary of Algorithms and Data Structures*. URL `http://www.nist.gov/dads/HTML/perfectKaryTree.html`. [online], accessed 25.08.2018.

G. S. Bloom and S. W. Golomb. Applications of numbered undirected graphs. *Proceedings of the IEEE*, 65 (4):562–570, 1977.

N. Cho and J. Linderoth. Row-partition branching for set partitioning problems. In *Proceedings of the INFORMS Computing Society Meeting*, pages 119–133, 2015.

M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer programming*. Springer Berlin, 2014.

A. Dehghan, M.-R. Sadeghi, and A. Ahadi. Algorithmic complexity of proper labeling problems. *Theoretical Computer Science*, 495:25–36, 2013.

J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys (CSUR)*, 34 (3):313–356, 2002.

A. Duarte, R. Martí, M.G.C. Resende, and R.M.A. Silva. Grasp with path relinking heuristics for the antibandwidth problem. *Networks*, 58(3):171–189, 2011.

D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26(6): 992–1009, 1978.

G. Fertin and S. Vialette. On the S-labeling problem. *Electronic Notes in Discrete Mathematics*, 34:273–277, 2009.

G. Fertin, I. Rusu, and S. Vialette. Algorithmic aspects of the S-labeling problem. In *International Workshop on Combinatorial Algorithms*, pages 173–184. Springer, 2015.

G. Fertin, I. Rusu, and S. Vialette. The S-labeling problem: An algorithmic tour. *Discrete Applied Mathematics*, 2017.

M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

J. A. Gallian. A dynamic survey of graph labeling. *The Electronic Journal of Combinatorics*, 16(6):1–219, 2009.

A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

X. T. Jin and R. K. Yeh. Graph distance-dependent labeling related to code assignment in computer networks. *Naval Research Logistics (NRL)*, 52(2):159–164, 2005.

R. M. Karp. Mapping the genome: some combinatorial problems arising in molecular biology. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 278–285. ACM, 1993.

H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2 (1-2):83–97, 1955.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

E. Rodriguez-Tello, J.-K. Hao, and J. Torres-Jimenez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research*, 35(10):3331–3346, 2008.

E. Rodriguez-Tello, H. Romero-Monsivais, G. Ramirez-Torres, and F. Lardeux. Tabu search for the cyclic bandwidth problem. *Computers & Operations Research*, 57:17–32, 2015.

Jan Van den Heuvel, Robert A Leese, and Mark A Shepherd. Graph labeling and radio channel assignment. *Journal of Graph Theory*, 29(4):263–283, 1998.

S. Vialette. Packing of (0, 1)-matrices. *RAIRO-Theoretical Informatics and Applications*, 40(4):519–535, 2006.

L. Wolsey. *Integer Programming*. Wiley, 1998.

R. T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28(3):271–287, 1984.