Solving the Constrained Single-Row Facility Layout Problem with Decision Diagrams

Vianney Coppé
UCLouvain, Louvain-la-Neuve, Belgium

Xavier Gillard
UCLouvain, Louvain-la-Neuve, Belgium

Pierre Schaus
UCLouvain, Louvain-la-Neuve, Belgium

Abstract

The Single-Row Facility Layout Problem is an NP-hard problem dealing with the ordering of departments with given lengths and pairwise traffic intensities in a facility. In this context, one seeks to minimize the sum of the distances between department pairs, weighted by the corresponding traffic intensities. Practical applications of this problem include the arrangement of rooms on a corridor in hospitals or offices, airplanes and gates in an airport or machines in a manufacture. This paper presents two novel exact models for the Constrained Single-Row Facility Layout Problem, a recent variant of the problem including positioning, ordering and adjacency constraints. On the one hand, the state-of-the-art mixed-integer programming model for the unconstrained problem is extended to incorporate the constraints. On the other hand, a decision diagram-based approach is described, based on an existing dynamic programming model for the unconstrained problem. Computational experiments show that both models outperform the only mixed-integer programming model in the literature, to the best of our knowledge. While the two models have execution times of the same order of magnitude, the decision diagram-based approach handles positioning constraints much better but the mixed-integer programming model has the advantage for ordering constraints.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases Discrete Optimization, Mixed-Integer Programming, Decision Diagrams, Constrained Single-Row Facility Layout Problem

Digital Object Identifier 10.4230/LIPIcs.CP.2022.14

Supplementary Material *Software (Source Code)*: `https://github.com/vcoppe/csrflp-dd`
Software (Source Code): `https://github.com/vcoppe/csrflp-mip`

1 Introduction

The *Single-Row Facility Layout Problem* (SRFLP) is an ordering problem considering a set of departments in a facility, with given lengths and pairwise traffic intensities. Its goal is to find a linear ordering of the departments minimizing the weighted sum of the distances between department pairs. The SRFLP is applied in different fields to arrange items such as rooms on a corridor in hospitals or offices [52], airplanes and gates in an airport [53], machines in a manufacture [30], books on a shelf and files in disk cylinders [50]. When all facilities have equal lengths and the traffic intensities are binary, the problem is known as the *Minimum Linear Arrangement Problem* (MinLA). It is a well-known graph layout problem which has been proved to be NP-hard [20] and consequently, so is the SRFLP.

Due to its difficulty in being solved by exact methods, many heuristic techniques have been designed to find good quality solutions to the SRFLP problem [19, 28, 29, 41] and more recently [18, 23, 40, 47, 51]. The first attempt to solve the SRFLP optimally was a branch-and-bound algorithm with interesting lower bounds [52]. Later, the DP approach presented in [39] was applied to the SRFLP in [50]. More recent techniques include non-linear programming [31], linear *mixed-integer programming* (MIP) [2, 3, 46], branch-and-cut [4, 5] and semidefinite programming [7, 8, 9, 35, 36].

In [37], *positioning*, *ordering* and *relation* constraints were suggested for the SRFLP to model real-life situations. The resulting problem is called the *Constrained Single-Row Facility Layout Problem* (cSRFLP). They also proposed a permutation-based genetic algorithm to solve this new problem and reported very good results, with objective values deviating by only a few percents from the best known solutions to the unconstrained problem for instances with up to 100 departments. In [44], the first MIP model solving the cSRFLP is introduced and a constrained improved fireworks algorithm is described. The latter is shown to find solutions of better quality than the genetic algorithm of [37].

This paper begins with a formal definition of the SRFLP in Section 2 and of the constraints that constitute the cSRFLP. We then present two novel exact models to solve the problem. In Section 3, we model the constraints of the cSRFLP on top of the state-of-the-art MIP model for the SRFLP [4]. Likewise, Section 4 recalls the *dynamic programming* (DP) model for the SRFLP from [50] and shows how the new constraints can be integrated. This DP model will be used as the basis of a decision diagram-based approach described in detail in Sections 5 and 6.

A *decision diagram* (DD) is a data structure used to encode sets in a compressed form through a graphical representation. They first appeared as binary decision diagrams for the representation of Boolean functions and were successfully used for circuit design and formal verification [1, 15, 34, 43]. Among the wide variety of domains in which the DDs were applied through the years [45, 56], the compactness which they provide was exploited in constraint programming [32, 48, 55] and optimization [10, 25, 26, 27, 42]. Recently, a complete framework for discrete optimization with decision diagrams was introduced in [13]. It relies on a DP model of the problem, which can represent the solution space in a compact form. In spite of their compactness, DDs encoding hard optimization problems may not fit in memory. The exact optimization method is therefore built upon *relaxed* and *restricted* DDs. These approximate DDs were introduced in [6, 11, 14] for their ability to provide tight lower and upper bounds [16, 17, 33, 54]. An adapted branch-and-bound algorithm based exclusively on DDs was presented in [13].

In Section 7, the results of our computational experiments are presented. They show that our two new models outperform the MIP model from [44] in terms of solving time. Other than that, there is no clear winner between the DD approach and the new MIP model. The former seems to handle positioning constraints better while the latter is particularly efficient for relation constraints. However, the ability to parallelize the DD approach is unmatched by the MIP solver. The paper concludes with a summary of our contributions and directions for future work.

2 Problem Definition

This section is organized as follows, a formal definition of the SRFLP is given in Section 2.1 which is then completed in Section 2.2 with the constraints that constitute the cSRFLP.

2.1 SRFLP

The SRFLP is a linear ordering problem considering a set $N = \{1, 2, \ldots, n\}$ of departments in a facility. Each department i has a given length l_i and is connected to all other departments by a traffic intensity c_{ij}. Both the lengths and the traffic intensities are positive integers. It is imposed that $c_{ij} = c_{ji}$ but it is not a modeling restriction since a trip in any direction covers the same distance, the traffic intensities can thus concentrate both directions [52].

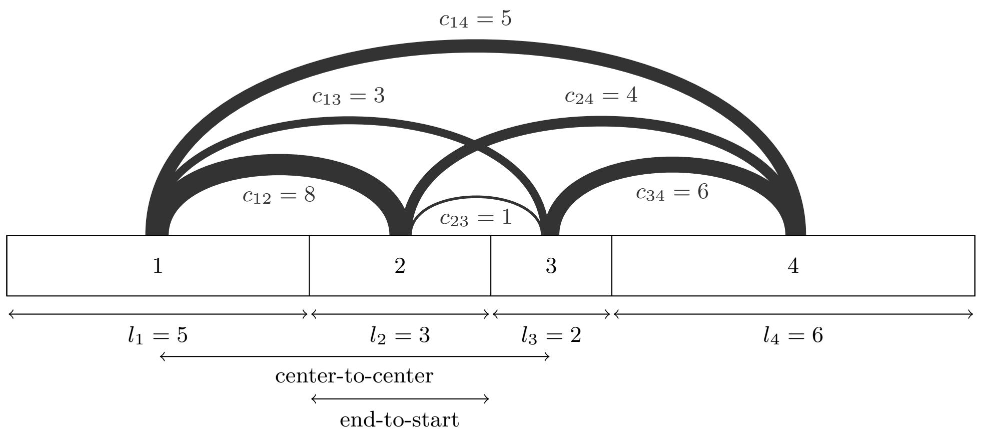

Figure 1 An instance of the SRFLP with 4 departments ordered optimally. The lengths of the departments are noted below them and the pairwise traffic intensities are given on the edges connecting pairs of departments. Center-to-center and end-to-start distances between departments one and three are shown.

A solution to the SRFLP is an ordering of the departments on a line, defined by the bijection $\pi : N \to \{1, 2, \ldots, n\}$. If d^{π}_{ij} is the center-to-center distance between departments i and j for ordering π, the cost function to minimize is formulated as follows:

$$SRFLP\,(\pi) = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i<j}}^{n} c_{ij} d^{\pi}_{ij}. \tag{1}$$

It is a measure of the total distance traveled by components or products within the facility. Using center-to-center distances implies that we must deal with half department lengths. However, one can notice that wherever a department i is placed with respect to a department j, the center-to-center distance between i and j will be at least $\frac{l_i+l_j}{2}$. This leads to a reformulation that simplifies the coming formulas:

$$SRFLP\,(\pi) = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i<j}}^{n} c_{ij} \tilde{d}^{\pi}_{ij} + K \qquad \text{with} \qquad K = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i<j}}^{n} c_{ij} \frac{l_i + l_j}{2} \tag{2}$$

where $\tilde{d}^{\pi}_{ij}$ is the end-to-start distance (see Figure 1) separating departments i and j and K is a constant accounting for all contributions of half department lengths [52].

▶ **Example 1.** Let us illustrate the computation of the objective function on the facility given in Figure 1. We first compute the value of the constant K:

$$\begin{aligned} K &= c_{12}\frac{l_1+l_2}{2} + c_{13}\frac{l_1+l_3}{2} + c_{14}\frac{l_1+l_4}{2} + c_{23}\frac{l_2+l_3}{2} + c_{24}\frac{l_2+l_4}{2} + c_{34}\frac{l_3+l_4}{2} \\ &= 8\frac{5+3}{2} + 3\frac{5+2}{2} + 5\frac{5+6}{2} + 1\frac{3+2}{2} + 4\frac{3+6}{2} + 6\frac{2+6}{2} \\ &= 8 \cdot 4 + 3 \cdot 3.5 + 5 \cdot 5.5 + 1 \cdot 2.5 + 4 \cdot 4.5 + 6 \cdot 4 = 114.5 \end{aligned}$$

and then the cost of the ordering $\pi(i) = i, \forall i \in N$ as shown in Figure 1:

$$\begin{aligned} SRFLP(\pi) &= c_{12}\tilde{d}^{\pi}_{12} + c_{13}\tilde{d}^{\pi}_{13} + c_{14}\tilde{d}^{\pi}_{14} + c_{23}\tilde{d}^{\pi}_{23} + c_{24}\tilde{d}^{\pi}_{24} + c_{34}\tilde{d}^{\pi}_{34} + K \\ &= c_{12} \cdot 0 + c_{13} l_2 + c_{14}(l_2 + l_3) + c_{23} \cdot 0 + c_{24} l_3 + c_{34} \cdot 0 + K \\ &= 8 \cdot 0 + 3 \cdot 3 + 5 \cdot (3+2) + 1 \cdot 0 + 4 \cdot 2 + 6 \cdot 0 + 114.5 = 156.5. \end{aligned}$$

2.2 cSRFLP

The cSRFLP is obtained by adding three types of constraints to the SRFLP:

- *Positioning constraints:* A department is forced to be located at a specific position within the ordering. These constraints are described by a function $position : N \to N \cup \{0\}$ which maps positions to their corresponding department or to 0 if there is no constraint on the position. To simplify the coming equations, we also define the function $department : N \to N \cup \{0\}$ which is the inverse mapping, between departments and positions.
- *Ordering constraints:* These constraints impose that some department must come before another one in the ordering. Formally, the function $predecessors : N \to 2^N$ gives the set of *predecessors* of each department, i.e. all departments that must be placed on the left of the given department.
- *Relation constraints:* Similarly to ordering constraints, relation constraints impose a relative ordering between a pair of departments. In this case, however, the two departments are required to be adjacent in the ordering. The function $previous : N \to N \cup \{0\}$ maps departments to the department that must be placed right before, or to 0 if there is no such constraint.

3 Mixed-Integer Programming Model

In this section, we integrate the constraints of the cSRFLP to the MIP model for the SRFLP, used within the branch-and-cut framework of [4]. This model uses betweenness variables ζ_{ijk} which describe the relative ordering of departments $i, j, k \in N$ in an ordering π:

$$\zeta^{\pi}_{ijk} = \begin{cases} 1, & \text{if } \pi(i) < \pi(k) < \pi(j) \text{ or } \pi(j) < \pi(k) < \pi(i) \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Using those variables, the objective function can be formulated as follows:

$$SRFLP(\zeta) = \sum_{i \in N} \sum_{\substack{j \in N \\ i<j}} c_{ij} \sum_{k \in N} \zeta_{ijk} l_k + K \tag{4}$$

and is to be minimized under the following constraints:

$$\zeta_{ijk} = \zeta_{jik} \qquad \forall \{i,j,k \mid i<j\} \subseteq N \tag{5}$$

$$\zeta_{ijk} + \zeta_{ikj} + \zeta_{jki} = 1 \qquad \forall \{i,j,k\} \subseteq N \tag{6}$$

$$\zeta_{ijd} + \zeta_{jkd} - \zeta_{ikd} \geq 0 \qquad \forall \{i,j,k,d\} \subseteq N \tag{7}$$

$$\zeta_{ijd} + \zeta_{jkd} + \zeta_{ikd} \leq 2 \qquad \forall \{i,j,k,d\} \subseteq N. \tag{8}$$

Equation (5) follows from the definition of the betweenness variables in Equation (3). Equation (6) states that only one department among i, j, k lies between the two others. Finally, Equations (7) and (8) express the fact that when a department d is placed between departments i and k, then the department d must either lie between departments (a) i and j or (b) j and k, but not both (a) and (b).

We now present how the constraints of the cSRFLP can be integrated in the model. A solution to the original model specifies a relative ordering of the departments. Yet, it does not impose one extremity to the left of the arrangement. As we will need this information in the constraints presented in Section 2.2, we solve this issue by adding two dummy departments L and R. For the cSRFLP, the set of departments is thus defined as $N = \{1, \ldots, n\} \cup \{L, R\}$ and departments L and R also obey Equations (3)–(8). We set $l_L = l_R = 0$ and $c_{Li} = c_{iL} = c_{Ri} = c_{iR} = 0, \forall i \in N$ so that the dummy departments have no impact on the objective function. Department L and R are respectively forced on the left and right side of the arrangement by adding the constraints:

$$\zeta_{LRi} = 1 \qquad \forall i \in N \setminus \{L, R\} \tag{9}$$

$$\zeta_{ijL} = 0 \qquad \forall i, j \in N \tag{10}$$

$$\zeta_{ijR} = 0 \qquad \forall i, j \in N. \tag{11}$$

Equation (9) imposes that all other departments are placed between departments L and R. Inversely, Equations (10) and (11) ensure that departments L and R are not placed between any two departments.

We can now write the additional constraints of the model for the cSRFLP:

$$\sum_{k=1}^{n} \zeta_{Lik} = j - 1 \qquad \forall i \in N, position(i) = j \neq 0 \tag{12}$$

$$\sum_{k=1}^{n} \zeta_{iRk} = n - j \qquad \forall i \in N, position(i) = j \neq 0 \tag{13}$$

$$\zeta_{Lij} = 0 \qquad \forall i, j \in N, i \in predecessors(j) \lor i = previous(j) \tag{14}$$

$$\zeta_{Lji} = 1 \qquad \forall i, j \in N, i \in predecessors(j) \lor i = previous(j) \tag{15}$$

$$\zeta_{iRj} = 1 \qquad \forall i, j \in N, i \in predecessors(j) \lor i = previous(j) \tag{16}$$

$$\zeta_{jRi} = 0 \qquad \forall i, j \in N, i \in predecessors(j) \lor i = previous(j) \tag{17}$$

$$\zeta_{ijk} = 0 \qquad \forall i, j \in N, i = previous(j), k \in N \setminus \{i, j\}. \tag{18}$$

Equations (12) and (13) ensure that $j-1$ departments are located on the left of department i and $n-j$ on the right, given that i must be placed at the j-th position. Equations (14)–(17) impose that i is placed between L and j and that j is placed between i and R, when either i is a predecessor of j or i must be placed right before j. Finally, Equation (18) is added for relation constraints to avoid having any departments placed between the two departments involved in the constraint.

4 Dynamic Programming Model

Dynamic programming is a different technique to tackle this problem. Section 4.1 presents an efficient DP model introduced in [50]. We then show in Section 4.2 how the constraints can be incorporated in this model. As a whole, this formulation will be the starting point for our DD-based approach.

4.1 SRFLP

Let us first reformulate the cost function:

$$SRFLP(\pi) = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i<j}}^{n} c_{ij} \tilde{d}_{ij}^{\pi} + K = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ \pi(i)<\pi(j)}}^{n} c_{ij} \tilde{d}_{ij}^{\pi} + K \tag{19}$$

$$= \sum_{i=1}^{n} \sum_{\substack{j=1 \\ \pi(i)<\pi(j)}}^{n} c_{ij} \sum_{\substack{k=1 \\ \pi(i)<\pi(k)<\pi(j)}}^{n} l_k + K \tag{20}$$

$$= \sum_{k=1}^{n} l_k \sum_{\substack{i=1 \\ \pi(i)<\pi(k)}}^{n} \sum_{\substack{j=1 \\ \pi(k)<\pi(j)}}^{n} c_{ij} + K. \tag{21}$$

In Equation (19), we use the bijection π to sum over unique pairs of positions instead of unique pairs of departments. We then develop the end-to-start distances $\tilde{d}_{ij}^{\pi}$ in Equation (20), which are equal to the sum of the lengths of departments between i and j in the ordering π. Finally, we reorder the summations in Equation (21). This allows reading the cost function differently: for each department k, we add its length l_k to the distance between pairs of departments (i, j) lying on opposite sides of k and multiply it by the corresponding traffic intensity c_{ij}.

The idea of the DP model is to place the departments one by one on the line from left to right. From Equation (21), it is clear that the individual cost of placing department k at position $\pi(k)$ only depends on the side on which all other departments are located with respect to k. If the state of the DP model is the subset of departments which remain to be placed – called *free* departments from now on, as opposed to *fixed* departments – we can compute this individual cost and recursively find the optimal ordering of each subset of N. Formally, the components of the DP model are:

- The *control variables* $x_j \in D_j$ with $j \in \{0, \dots, n-1\}$. Variable x_j represents the department placed at position $j+1$ on the line. All variables have the same domain $D_j = N$ since departments can appear anywhere in the ordering.
- The *state space* S which contains all subsets of N. It includes a *root state* $\hat{r} = N$, a *terminal state* $\hat{t} = \emptyset$ and an *infeasible state* $\hat{0}$. The state space is partitioned into the sets $S_0, \dots, S_n$ where S_j contains all states with j variables assigned.
- The set of *transition functions* $t_j : S_j \times D_j \to S_{j+1}$ for $j = 0, \dots, n-1$ which rule the transition between the states of consecutive stages:

$$t_j\left(s^j, x_j\right) = \begin{cases} s^j \setminus \{x_j\}, & \text{if } x_j \in s^j \\ \hat{0}, & \text{otherwise.} \end{cases} \tag{22}$$

- The set of *transition cost functions* $h_j : S_j \times D_j \to \mathbb{R}$ for $j = 0, \dots, n-1$ which associate a value to each transition:

$$h_j\left(s^j, x_j\right) = \begin{cases} l_{x_j} \sum_{i \in \overline{s}^j} \sum_{k \in s^j \setminus \{x_j\}} c_{ik}, & \text{if } x_j \in s^j \\ 0, & \text{otherwise.} \end{cases} \tag{23}$$

  This formula immediately follows from Equation (21) since $\overline{s}^j$ – the complement of s^j – contains fixed departments placed before position j and $s^j \setminus \{x_j\}$ contains free departments, which will be placed after position j.
- The *root value* $v_r = K$ from Equation (2).

To solve an instance of the SRFLP using this DP model, one needs to apply the following recurrence:

$$\begin{aligned} &\min\ \hat{f}(x) = v_r + \sum_{j=0}^{n-1} h_j\left(s^j, x_j\right) \\ &\text{subject to } s^{j+1} = t_j\left(s^j, x_j\right), \text{ for all } x_j \in D_j, j = 0, \ldots, n-1 \\ &\qquad s^j \in S_j, j = 0, \ldots, n. \end{aligned} \tag{24}$$

Speeding up the computation of transition costs. We also store in the states an array containing the *cut values* of each free department: the sum of all traffic intensities from the fixed departments and each free department. It allows to reduce the computational complexity of the transition costs from $\mathcal{O}\left(n^2\right)$ to $\mathcal{O}(n)$ and will also be useful when designing a lower bound in Section 6.2. For a state s^j and each department $i \in N$, we define:

$$s^j_{cut}[i] = \begin{cases} \sum_{j \in \overline{s}^j} c_{ij}, & \text{if } i \in s^j \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

which can be updated in $\mathcal{O}(n)$ during a transition $t_j\left(s^j, x_j\right)$:

$$s^{j+1}_{cut}[i] = \begin{cases} s^j_{cut}[i] + c_{ix_j}, & \text{if } i \in s^j \setminus \{x_j\} \\ 0, & \text{otherwise} \end{cases} \tag{26}$$

and the transition costs become:

$$h_j\left(s^j, x_j\right) = \begin{cases} l_{x_j} \sum_{i \in s^j \setminus \{x_j\}} s^j_{cut}[i], & \text{if } x_j \in s^j \\ 0, & \text{otherwise.} \end{cases} \tag{27}$$

▶ **Example 2.** Considering the instance shown on Figure 1, we compute the cut values for the state $s = \{3, 4\}$. We have that $s_{cut}[1] = s_{cut}[2] = 0$ since departments 1 and 2 are already placed. For the free departments, we apply Equation (25) and obtain: $s_{cut}[3] = c_{13} + c_{23} = 3 + 1 = 4$ and $s_{cut}[4] = c_{14} + c_{24} = 5 + 4 = 9$.

4.2 cSRFLP

Adding constraints to the DP model is done through the predicates $valid_j : S_j \times D_j \to \{true, false\}$ for $j = 0, \ldots, n-1$. They are used in the transition functions to filter out infeasible solutions:

$$t_j\left(s^j, x_j\right) = \begin{cases} s^j \setminus \{x_j\}, & \text{if } x_j \in s^j \wedge valid_j(s^j, x_j) \\ \hat{0}, & \text{otherwise.} \end{cases} \tag{28}$$

For clarity, we split the predicates $valid_j$ into several conditions, corresponding each to a specific constraint:

$$valid_j(s^j, x_j) = p_j(s^j, x_j) \wedge o_j(s^j, x_j) \wedge r_j(s^j, x_j) \tag{29}$$

with p_j, o_j and r_j concerning respectively positioning, ordering and relation constraints:

$$p_j(s^j, x_j) = (position(x_j) = 0 \wedge department(j+1) = 0) \vee position(x_j) = j + 1 \tag{30}$$

$$o_j(s^j, x_j) = predecessors(x_j) \subseteq \overline{s}^j \tag{31}$$

$$r_j(s^j, x_j) = (previous(x_j) = 0 \wedge \nexists k \in s^j : previous(k) \in \overline{s}^j) \vee previous(x_j) \in \overline{s}^j. \tag{32}$$

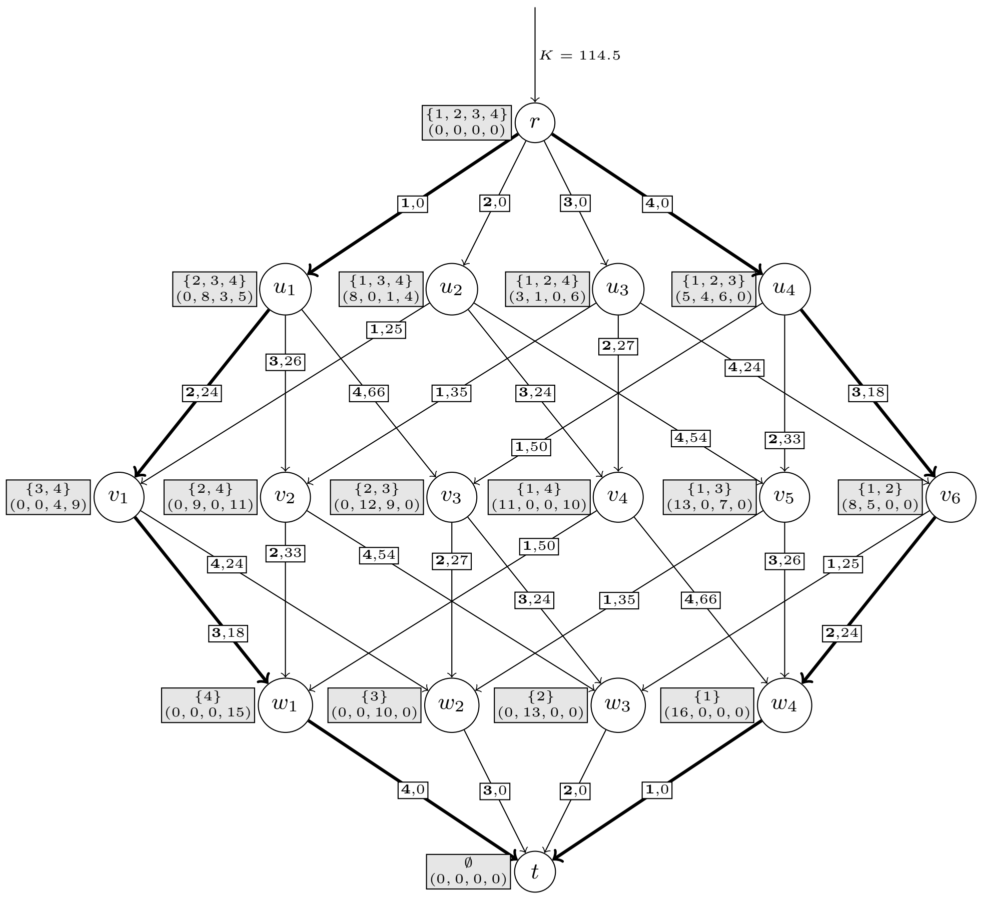

Figure 2 The exact DD associated with the instance shown on Figure 1. Arcs are annotated with their label (in bold) and cost. Next to each node, a gray box contains the corresponding state: the set of free departments and the cut values. Arcs in bold are part of an optimal solution.

In Equation (30), p_j checks that either department x_j and position $j + 1$ are both unconstrained, or that department x_j is constrained to be at position $j + 1$. As explained previously, the j-th transition decides which department is placed at position $j + 1$. For ordering constraints, Equation (31) verifies that all predecessors of department x_j have already been placed. The predicates r_j for relation constraints are slightly more complicated. Either x_j has no relation constraint, then it can only be placed if no other free department has a relation constraint with a fixed department, or x_j has a relation constraint and $previous(x_j)$ must be a fixed department.

5 Decision Diagram Representation

This section explains how a DP model can be used to derive DDs. A *weighted decision diagram* is a graphical structure which encodes a set of solutions to a discrete optimization problem $\mathcal{P}$. Formally, it is represented by a layered directed acyclic graph $B = (U, A, d, v, \sigma)$ where U is the set of nodes, A is the set of arcs. The set of nodes is partitioned into layers $L_0, \ldots, L_n$. In particular, layers L_0 and L_n contain only one node, respectively the root node r and the terminal node t. Each node is mapped to a state by the function σ. An arc

$a \in A$ connects a node in a layer L_j to a node in the next layer L_{j+1}. Its label $d(a) \in D_j$ represents the assignment of value $d(a)$ to variable x_j and $v(a)$ denotes its length. As a result, each path $p = \left(a^{(0)}, \ldots, a^{(n-1)}\right)$ from r to t is a complete assignment of the variables, with $x_j = d\left(a^{(j)}\right)$, and has a total length of $v(p) = v_r + \sum_{j=0}^{n-1} v\left(a^{(j)}\right)$. The set of all $r-t$ paths of B encodes the set of possible assignments $\mathrm{Sol}(B)$. In an *exact decision diagram*, the length of each $r-t$ path is equal to the objective function value of the corresponding assignment and $\mathrm{Sol}(B) = \mathrm{Sol}(\mathcal{P})$. Thus, the resolution of discrete optimization problems is reduced to a shortest-path problem on a directed acyclic graph $f\left(x^*\right) = v^*(B)$.

The *size* $|B|$ of a decision diagram is the number of nodes it contains in all layers. Its *width* is given by $\max_j |L_j|$, where $|L_j|$ is the *width* of layer j. Arcs leaving a same node always have different labels, so every node $u \in L_j$ has a maximum out-degree of $|D_j|$. A *binary decision diagram* encodes binary variables only, as opposed to *multi-valued decision diagrams* in the general case [38].

Using a DP model, an exact DD can be built layer by layer starting with the first layer L_0 containing the root node r associated to the root state $\hat{r}$. From a layer L_j, we then fill L_{j+1} with all nodes corresponding to *distinct* feasible states which can be reached from any state in L_j. For each of these transitions, we add an arc from the node in L_j to the one in L_{j+1} and its length is given by the transition cost.

▶ **Example 3.** The exact DD for the instance shown in Figure 1 is illustrated in Figure 2. The size of this DD is 16 and its width is 6. On the left side of the DD, the path in bold is an optimal solution. It corresponds to the ordering displayed in Figure 1 and its length is equal to $114.5 + 0 + 24 + 18 + 0 = 156.5$ as computed in Example 1.

6 Branch-and-Bound

Although DP formulations tend to represent problems in a compact manner, it is usually intractable to generate exact DDs for combinatorial problems as the size of the state space can grow exponentially with the number of variables. An adapted branch-and-bound algorithm exploiting DDs (B&B-DD) was presented in [13] with the potential to solve larger instances to optimality. The algorithm successively explores subproblems corresponding to nodes in the exact DD of the problem. As in classical branch-and-bound, two ingredients are used: a primal upper bound heuristic to discover good feasible solutions, and a lower bound procedure allowing to prune the nodes with a lower bound larger than the best so far solution. The major idea of B&B-DD is to limit the width of the DDs to obtain these two ingredients. The primal heuristic is obtained by discarding nodes of the DD to respect the width limit while the lower bound procedure consists in discovering the best path in a relaxed DD obtained by merging nodes. The nodes of B&B-DD are expanded using a classical best-first-search.

Preliminary experiments convinced us to slightly deviate from the generic B&B-DD framework and specialize it for the SRFLP in order to be competitive with state-of-the-art approaches. We use a problem specific lower bound rather than the state-merging procedure, as well as a breadth-first search allowing to better exploit the recursive structure of the problem. The lower bound and the custom search are detailed in the next sections.

6.1 Primal Upper Bound Heuristic

As explained in Section 6, we rely on restricted DDs to generate good feasible solutions starting from a given node of the exact DD. To obtain a restricted DD, it is sufficient to remove nodes of a layer when its width exceeds a given maximum width. The nodes and

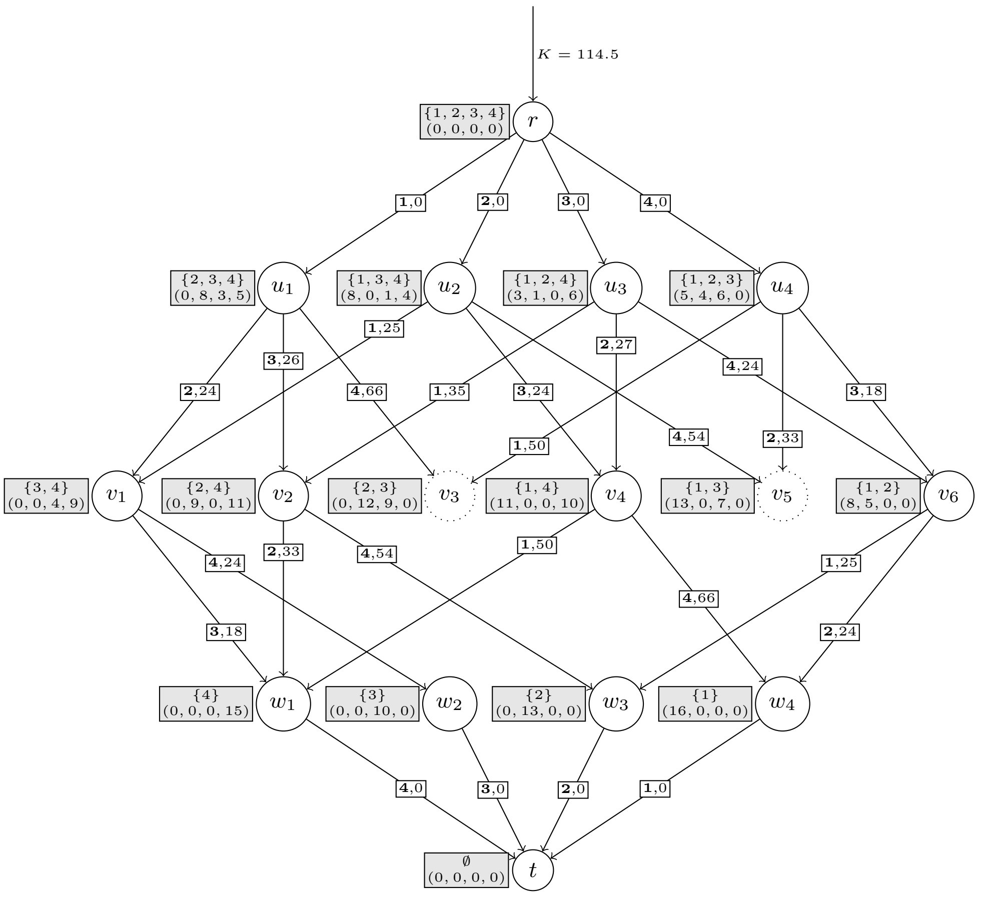

Figure 3 A restricted DD associated with the instance shown on Figure 1 and built with a maximum width of 4. Nodes in dotted circles have been removed from the layer.

arcs which remain in the DD are not modified and thus correspond to feasible solutions. A heuristic is used to select nodes to remove from a layer and attempts to identify nodes leading to the poorer quality solutions. Restricted DDs also allow to retrieve the set of subproblems which need to be explored next. In this paper, this set is computed as the set of direct successors of the initial node of the restricted DD.

▶ **Example 4.** Figure 3 shows a restricted DD built for the instance displayed in Figure 1 with a maximum width of 4. The third layer exceeded the maximum width so the nodes v_3 and v_5 have been removed.

6.2 Lower Bound

In [13], a relaxed DD is used to compute a single lower bound at a given node. Based on this lower bound, we decide whether to enqueue or prune the open subproblems. Recently, [22] suggested that we could attach a different lower bound to each node to be added to the branch-and-bound queue. In our approach, this lower bound is based on a heuristic *rough lower bound* (RLB) which can be computed swiftly for any node. As described in [22], the RLB can also be used to skip nodes during the compilation of restricted DDs.

In order to derive the RLB from a node u, the next theorem shows that the cost to optimally complete the partial solution of node u can be decomposed in two terms: one solely involving the free departments and the other one involving the cost between free and fixed departments.

▶ **Theorem 5.** *Given a node u and its state $\sigma(u) = s$, let $\pi^*|_u$ be the best ordering one can obtain when crossing node u. For conciseness, we set $\pi = \pi^*|_u$. We have the equivalence:*

$$SRFLP(\pi) - v^*(u) = \underbrace{\sum_{i \in s} \sum_{\substack{j \in s \\ \pi(i) < \pi(j)}} c_{ij} \sum_{\substack{k \in s \\ \pi(i) < \pi(k) < \pi(j)}} l_k}_{\text{free departments layout cost}} + \underbrace{\sum_{j \in s} s_{cut}[j] \sum_{\substack{k \in s \\ \pi(k) < \pi(j)}} l_k}_{\text{cost w.r.t. fixed departments}} . \tag{33}$$

Those two terms of Equation (33) cannot be evaluated exactly in a cheap way as this would be as difficult as solving the original problem. Nonetheless one can compute an efficient lower bound for each term independently. For a node u, the value of the RLB is given by:

$$RLB(u) = \begin{cases} \infty, & \text{if } \sigma(u) = \hat{0} \\ LB_{edge}(u) + LB_{cut}(u), & \text{otherwise,} \end{cases} \tag{34}$$

where $LB_{edge}(u)$ is a lower bound on the free departments layout cost and $LB_{cut}(u)$ is a lower bound on the cost induced by the cut values of free departments.

6.2.1 Free departments layout cost

The first lower bound LB_{edge} is an under-approximation of the internal layout cost of free departments. Given a subset of departments, we compute a lower bound on the cost of its optimal layout by multiplying each pairwise traffic intensity by an optimistic distance. If we must place n departments on a line, $n - k$ pairs of departments will have $k - 1$ departments between them (see Figure 1). In order to under-approximate the layout cost, we greedily multiply the highest traffic intensities by the smallest distance possible. Since we cannot assume any particular ordering of the free departments, the distances between pairs of free departments are unknown. Still, we can compute lower bounds on those distances if we sort the free departments by increasing length and assume that a separation of k departments will be formed by the k shortest departments. This lower bound can be seen as a generalization of the Edges method [49] designed for the MinLA.

In practice, a list containing all pairwise traffic intensities in decreasing weight order is precomputed, as stated by the precondition of Algorithm 1. The same is done for the department lengths. We then only need to traverse those lists and multiply each traffic intensity value by the adequate cumulative length. The complexity of the algorithm is $\mathcal{O}\left(n^2\right)$ since there are $\frac{n(n-1)}{2}$ pairs in total.

▶ **Example 6.** Let us illustrate the computation of this lower bound on the root node of the DD in Figure 2. We first create the list of traffic intensities sorted decreasingly: $edge = [c_{12} = 8, c_{34} = 6, c_{14} = 5, c_{24} = 4, c_{13} = 3, c_{23} = 1]$ and the list of free department lengths sorted increasingly: $length = [l_3 = 2, l_2 = 3, l_1 = 5, l_4 = 6]$. There are 3 pairs of departments with 0 departments in between, 2 pairs with 1 department in between and 1 pair with 2 departments in between.

$$\begin{aligned} LB_{edge}(r) &= 0 \cdot c_{12} + 0 \cdot c_{34} + 0 \cdot c_{14} + l_3 c_{24} + l_3 c_{13} + (l_3 + l_2) c_{23} \\ &= 0 \cdot 8 + 0 \cdot 6 + 0 \cdot 5 + 2 \cdot 4 + 2 \cdot 3 + (2 + 3) \cdot 1 = 19 \end{aligned}$$

Algorithm 1 Computation of $LB_{edge}(u)$.

```
Require: $edge = sorted_{\geq}(\{\langle c : c_{ij}, dep1 : i, dep2 : j\rangle \mid 1 \leq i < j \leq n\})$
    and $length = sorted_{\leq}(\{\langle l : l_i, dep : i\rangle \mid 1 \leq i \leq n\})$
 1: $s \leftarrow \sigma(u), lb \leftarrow 0, cumul\_l \leftarrow 0, i \leftarrow 1, j \leftarrow 1$
 2: for $k \leftarrow 1$ to $|s| - 1$ do
 3:     for $l \leftarrow 1$ to $k$ do
 4:         while $edge[i].dep1 \notin s \vee edge[i].dep2 \notin s$ do
 5:             $i \leftarrow i + 1$
 6:         $lb \leftarrow lb + cumul\_l \cdot edge[i].c$
 7:         $i \leftarrow i + 1$
 8:     while $length[j].dep \notin s$ do
 9:         $j \leftarrow j + 1$
10:     $cumul\_l \leftarrow cumul\_l + length[j].l$
11:     $j \leftarrow j + 1$
12: return $lb$
```

6.2.2 Cost with respect to fixed departments

The second term of the RLB is related to the cut values of free departments and a lower bound is given by the *first-generation bound* described in [52]. Given a department i placed first on the line, the minimum total cost with respect to i is defined as:

$$MTC(i) = \min_{\pi} \sum_{\substack{j=1 \\ i \neq j}}^{n} c_{ij} \sum_{\substack{k=1 \\ \pi(k) < \pi(j)}}^{n} l_k \tag{35}$$

and Lemma 7 tells us how to find the optimal arrangement π.

▶ **Lemma 7.** *Suppose that department i is placed in first position on the line. For every other department j compute the cost-to-length ratio $r_j = \frac{c_{ij}}{l_j}$. The optimal arrangement, which yields $MTC(i)$ is obtained by ordering the departments according to decreasing values of this ratio r_j, the department with the greatest r_j being adjacent to i.*

This lower bound can also be used when several departments are placed in the leftmost positions on the line. We only need to consider all fixed departments as a single department connected to free departments with traffic intensities given by the respective cut values, exactly as in the second term of Equation (33). As the free departments need to be sorted by decreasing cut-to-length ratios, the time complexity of this lower bound is $\mathcal{O}(n \log(n))$.

▶ **Example 8.** We compute the lower bound for the node u_1 of the DD shown in Figure 2, with $\sigma(u_1) = s$. The departments are first sorted as follows:

$$order = \left[\frac{s_{cut}[2]}{l_2} = \frac{8}{3}, \frac{s_{cut}[3]}{l_3} = \frac{3}{2}, \frac{s_{cut}[4]}{l_4} = \frac{5}{6}\right].$$

We then compute the lower bound as the total cost with respect to all fixed departments:

$$\begin{aligned} LB_{cut}(u_1) &= 0 \cdot s_{cut}[2] + l_2 s_{cut}[3] + (l_2 + l_3) s_{cut}[4] \\ &= 0 \cdot 8 + 3 \cdot 3 + (3 + 2) \cdot 5 = 34. \end{aligned}$$

6.2.3 Refining the lower bound

In Section 6.1, we mentioned that for a node $u^{j-1} \in L_{j-1}$, its direct successors are added to the branch-and-bound queue. During the compilation of a restricted DD, not only we generate these successors in layer L_j but we also create all nodes which we can reach in layer L_{j+1}. As a result, we can compute a tighter lower bound for each node of L_j by taking advantage of the RLB values of its successors:

$$LB\left(u^j\right) = v^*(u^j) + \min_{x_j \in D_j} \left(h_j\left(\sigma\left(u^j\right), x_j\right) + RLB\left(t_j\left(\sigma\left(u^j\right), x_j\right)\right)\right). \tag{36}$$

▶ **Example 9.** Given the restricted DD shown in Figure 3, the local lower bound of node u_1 is computed as follows: $LB(u_1) = 0 + \min(24 + RLB(v_1), 26 + RLB(v_2), 66 + RLB(v_3))$.

6.3 A Breadth-First Branch-and-Bound

In the DP model of the SRFLP, a state s^j at level j is the successor of exactly j different states. More generally, it can be reached by as many as $j!$ different paths since any permutation of the departments could be a valid solution. The classical branch-and-bound algorithm always explores the most promising node first with a best-first-search strategy i.e. the one with the lowest lower bound or lowest shortest-path length. In the context of B&B-DD, nodes with a same state could be enqueued and explored multiple times during the algorithm. Preliminary experiments showed that this was often the case for the cSRFLP. This can be avoided by only exploring a complete layer before considering the next one. Therefore we suggest exploring the most promising node of what we call the *lowest active layer* (LAL) – the layer containing nodes of the queue with the least variables assigned. By doing so, all ancestors of the chosen node must have already been explored. It also ensures that at most one node associated with any state of the model will be inserted in the queue. The only adjustment to make is to maintain an additional data structure keeping track of all nodes in the queue. In that data structure, exactly like in any layer of a DD, we identify nodes by their state and keep in memory the path with shortest length to each state. We then only add one node to the queue for each state, and otherwise update the shortest-path leading to it. Our strategy is thus equivalent to a breadth-first-search in the exact DD but enhanced by pruning mechanisms.

This whole procedure is described by Algorithm 2. The index of the LAL is denoted l and increases throughout the execution of the algorithm. The branch-and-bound queue is split between Q_l and Q_{l+1} which respectively contain open nodes of layers l and $l + 1$. For each layer L_j, M_j is a map containing the node with the shortest path to each state of the level j. It is used in lines 17-26 to avoid adding multiple nodes in the branch-and-bound queue for the same state. The loop of line 8 can be parallelized, which is a key asset of B&B-DD [12, 21]. Each thread is responsible for developing a different restricted DD at line 13 and synchronization happens when queues, maps and the incumbent solution need to be updated.

7 Computational Experiments

In this section, we draw a comparison between the existing techniques to solve the cSRFLP to optimality. Namely, the MIP model from [44], the MIP model introduced in [4] and extended in Section 3 and the DD-based approach presented throughout the rest of the paper. In the following, they are respectively referred to as Liu, Amaral and DD. The MIP models were implemented and evaluated using Gurobi version 9.1.2 [24]. Concerning the DD approach, it was implemented in C++ and the code was largely based on DDO [21], a Rust library for DD-based discrete optimization. The heuristics selected are the following:

- *Maximum width:* We use fixed-width DDs for all experiments. To that end, we experimentally determined that a narrow maximum width of 3 leads to the best performance. It may seem very small but as explained in Section 6, the lower bound of a node is exclusively based on RLB values of its child nodes. As a result, the quality of the lower bounds does not depend on the maximum width of the DDs. Moreover, we observed that we were able to find very good solutions early in the search anyway.

Algorithm 2 The breadth-first branch-and-bound algorithm. *select_node* is a heuristic used to select the most promising node of the queue.

```
1: $v(r) \leftarrow v_r$ // root node value
2: $Q_0 \leftarrow \{r\}$ // queue for layer 0
3: $M_0 \leftarrow \{\sigma(r) : r\}$ // map for layer 0
4: $UB \leftarrow \infty$
5: for $l = 0$ to $n-1$ do // $l$ is the lowest active layer
6:     $Q_{l+1} \leftarrow \emptyset$ // queue for layer $l+1$
7:     $M_{l+1} \leftarrow \emptyset$ // map for layer $l+1$
8:     while $Q_l \neq \emptyset$ do
9:         $u \leftarrow select\_node(Q_l)$
10:        $Q_l \leftarrow Q_l \setminus \{u\}$
11:        if $LB(u) \geq UB$ then
12:            continue
13:        $\overline{B} \leftarrow Restricted(u)$
14:        for all $u' \in L_n$ of $\overline{B}$ do // update best solution
15:            if $v(u') < UB$ then
16:                $UB \leftarrow v(u')$
17:        for all $u' \in L_{l+1}$ of $\overline{B}$ do // enqueue successors of $u$ and update $M_{l+1}$
18:            if $LB(u') < UB$ then
19:                if $M_{l+1}.contains(\sigma(u'))$ then // this state is already in the queue and map
20:                    if $v(u') < v(M_{l+1}[\sigma(u')])$ then // update only if the value is improved
21:                        $Q_{l+1} \leftarrow Q_{l+1} \setminus \{M_{l+1}[\sigma(u')]\}$
22:                        $Q_{l+1} \leftarrow Q_{l+1} \cup \{u'\}$
23:                        $M_{l+1}[\sigma(u')] \leftarrow u'$
24:                else // this state is not in the queue and map
25:                    $M_{l+1}[\sigma(u')] \leftarrow u'$
26:                    $Q_{l+1} \leftarrow Q_{l+1} \cup \{u'\}$
27: return $UB$
```

- *Variable ordering:* The vertices must be placed from left to right on the line in the DP model so it is imposed for this formulation of the problem.
- *Search node selection:* Nodes with the smallest lower bound in the branch-and-bound queue are explored first in the branch-and-bound.
- *Node selection for restriction:* When the size of a layer exceeds the maximum width of the DD, we delete the nodes with the largest RLB values.

The instances used in the experiments are classical SRFLP instances taken from [2, 3, 8, 31, 52] with up to 25 departments. We then created admissible sets of constraints for each problem size:
- constraint sets with $2, 4, 6, 8$ and 10 positioning, ordering or relation constraints.
- constraint sets with $0, 2, 4, 6, 8$ and 10 constraints of each type.

For each of these scenarios, 5 different random sets of constraints were generated, except for the case with no constraints. Note that an instance with n departments can not have more than n positioning constraints, and that similar limits exist for the other types of constraints, we thus have up to 101 sets of constraints for each problem size. A link to the source code along with all the benchmark instances is given in the supplementary material. All experiments were performed on a machine with two Intel Xeon E5-2640 (2.6GHz) processors.

The three algorithms were executed on all combinations of instances and constraints with a time limit of 5 hours for each. The first row of Figure 4 shows the cumulative number of instances solved by each algorithm over time while the second row shows the mean ratio between the runtimes of each instance and its corresponding unconstrained instance, with respect to the number of constraints of each given type. Our first observation is that the two models presented in this paper clearly outperform the one from [44], which fails to solve most of the instances under the time limit regardless of the type of constraints

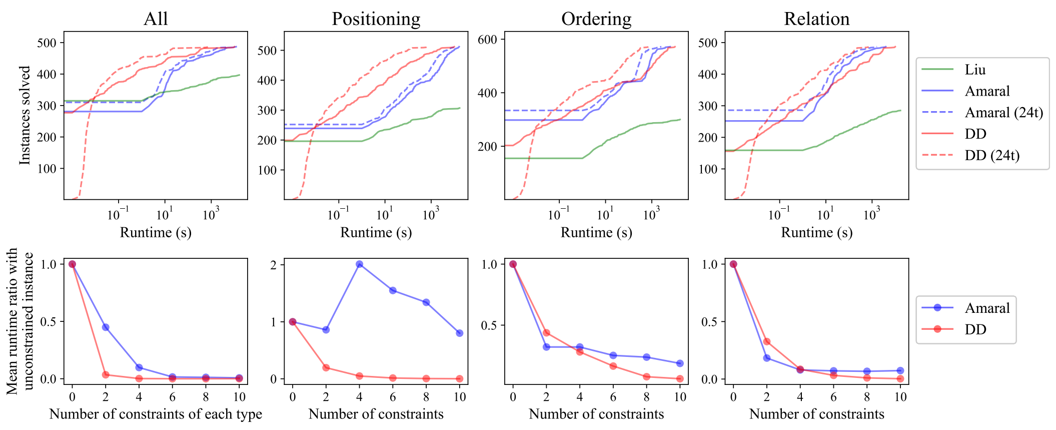

Figure 4 Number of instances solved by each algorithm for the different types of constraints, and mean ratio between the runtime of each constrained instance and the runtime of the corresponding unconstrained instance, with respect to the number of constraints of each type.

applied. Next, even if Amaral and DD both succeed in solving all instances, they have different behaviors depending on the type of scenario. From the graphs of the second row, we notice that the more constraints we add, the faster the DD approach gets. The constraints in the DD formulation are indeed handled very efficiently because all infeasible solutions are automatically pruned in the transition functions, which results in a smaller DP graph to explore. The same cannot be said about Amaral, since instances with between 4 and 8 positioning constraints take more time to solve than their corresponding unconstrained instance on average. This is probably because positioning constraints are modeled with a sum of n variables on the left side of an equality. On the contrary, ordering and relation constraints are modeled very naturally in Amaral because it uses relative ordering variables. Adding these types of constraints thus tightens the model and reduces the execution time. It allows Amaral to solve hard instances with ordering and relation constraints slightly faster than DD. However, DD is the first to solve all instances when using 24 threads and seems to benefit the most from parallelization.

8 Conclusion

In this paper, two novel exact models for the cSRFLP have been presented: an extension of the MIP model from [4] for the SRFLP and a DD-based approach starting from the DP model of [50]. The computational experiments have shown that they greatly improve on the performance of the only MIP model introduced in the literature to the best of our knowledge. Both models have their benefits, the DD approach incorporates the three types of constraints very efficiently, especially positioning constraints, and parallelizes better. On the other hand, the MIP model integrates ordering and relation constraints very well and can be easily implemented with any MIP solver. The DD approach can surely be improved in the future, for instance by taking the constraints into account within the lower bounds. It would also be interesting to combine the strengths of our two approaches.

References

1 Sheldon B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27(06):509–516, 1978.

2 André R. S. Amaral. On the exact solution of a facility layout problem. *European Journal of Operational Research*, 173(2):508–518, 2006.

3 André R. S. Amaral. An exact approach to the one-dimensional facility layout problem. *Operations Research*, 56(4):1026–1033, 2008.

4 André R. S. Amaral. A new lower bound for the single row facility layout problem. *Discrete Applied Mathematics*, 157(1):183–190, 2009.

5 André R. S. Amaral and Adam N. Letchford. A polyhedral approach to the single row facility layout problem. *Mathematical programming*, 141(1-2):453–477, 2013.

6 Henrik R. Andersen, Tarik Hadžić, John N. Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 118–132. Springer, 2007.

7 Miguel F. Anjos, Andrew Kennings, and Anthony Vannelli. A semidefinite optimization approach for the single-row layout problem with unequal dimensions. *Discrete Optimization*, 2(2):113–122, 2005.

8 Miguel F. Anjos and Anthony Vannelli. Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. *INFORMS Journal on Computing*, 20(4):611–617, 2008.

9 Miguel F. Anjos and Ginger Yen. Provably near-optimal solutions for very large single-row facility layout problems. *Optimization Methods & Software*, 24(4-5):805–817, 2009.

10 Bernd Becker, Markus Behle, Friedrich Eisenbrand, and Ralf Wimmer. Bdds in a branch and cut framework. In *International Workshop on Experimental and Efficient Algorithms*, pages 452–463. Springer, 2005.

11 David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John N. Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2):253–268, 2014.

12 David Bergman, Andre A. Cire, Ashish Sabharwal, Horst Samulowitz, Vijay Saraswat, and Willem-Jan van Hoeve. Parallel combinatorial optimization with decision diagrams. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 351–367. Springer, 2014.

13 David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016.

14 David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and Tallys Yunes. Bdd-based heuristics for binary optimization. *Journal of Heuristics*, 20(2):211–234, 2014.

15 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

16 Margarita P. Castro, Andre A. Cire, and J. Christopher Beck. An mdd-based lagrangian approach to the multicommodity pickup-and-delivery tsp. *INFORMS Journal on Computing*, 32(2):263–278, 2020.

17 Andre A. Cire and Willem-Jan van Hoeve. Multivalued decision diagrams for sequencing problems. *Operations Research*, 61(6):1411–1428, 2013.

18 Dilip Datta, André R. S. Amaral, and José R. Figueira. Single row facility layout problem using a permutation-based genetic algorithm. *European Journal of Operational Research*, 213(2):388–394, 2011.

19 Zvi Drezner. A heuristic procedure for the layout of a large number of facilities. *Management Science*, 33(7):907–915, 1987.

20 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Books in mathematical series. W. H. Freeman, 1979.

21 Xavier Gillard, Pierre Schaus, and Vianney Coppé. Ddo, a generic and efficient framework for mdd-based optimization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, pages 5243–5245, 2020.

22 Xavier Gillard, Pierre Schaus, Vianney Coppé, and André A. Cire. Improving the filtering of branch-and-bound mdd solver. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2021.

23 Jian Guan and Geng Lin. Hybridizing variable neighborhood search with ant colony optimization for solving the single row facility layout problem. *European Journal of Operational Research*, 248(3):899–909, 2016.

24 LLC Gurobi Optimization. Gurobi optimizer reference manual, 2022. URL: `https://www.gurobi.com`.

25 Gary D. Hachtel and Fabio Somenzi. A symbolic algorithms for maximum flow in 0-1 networks. *Formal Methods in System Design*, 10(2):207–219, 1997.

26 Tarik Hadžić and John N. Hooker. Postoptimality analysis for integer programming using binary decision diagrams. Technical report, Carnegie Mellon University, 2006.

27 Tarik Hadžić and John N. Hooker. Cost-bounded binary decision diagrams for 0-1 programming. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 84–98. Springer, 2007.

28 Kenneth M. Hall. An r-dimensional quadratic placement algorithm. *Management science*, 17(3):219–229, 1970.

29 Sunderesh S. Heragu and Attahiru Sule Alfa. Experimental analysis of simulated annealing based algorithms for the layout problem. *European Journal of Operational Research*, 57(2):190–202, 1992.

30 Sunderesh S. Heragu and Andrew Kusiak. Machine layout problem in flexible manufacturing systems. *Operations research*, 36(2):258–268, 1988.

31 Sunderesh S. Heragu and Andrew Kusiak. Efficient models for the facility layout problem. *European Journal of Operational Research*, 53(1):1–13, 1991.

32 Samid Hoda, Willem-Jan van Hoeve, and John N. Hooker. A systematic approach to mdd-based constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 266–280. Springer, 2010.

33 John N. Hooker. Improved job sequencing bounds from decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 268–283. Springer, 2019.

34 Alan J. Hu. *Techniques for efficient formal verification using binary decision diagrams*. PhD thesis, Stanford University, Department of Computer Science, 1995.

35 Philipp Hungerländer and Franz Rendl. A computational study and survey of methods for the single-row facility layout problem. *Computational Optimization and Applications*, 55(1):1–20, 2013.

36 Philipp Hungerländer and Franz Rendl. Semidefinite relaxations of ordering problems. *Mathematical Programming*, 140(1):77–97, 2013.

37 Zahnupriya Kalita and Dilip Datta. A constrained single-row facility layout problem. *The international journal of advanced manufacturing technology*, 98(5):2173–2184, 2018.

38 Timothy Kam. Multi-valued decision diagrams: Theory and applications. *Multiple-Valued Logic*, 4(1):9–62, 1998.

39 Richard M. Karp and Michael Held. Finite-state processes and dynamic programming. *SIAM Journal on Applied Mathematics*, 15(3):693–718, 1967.

40 Ravi Kothari and Diptesh Ghosh. An efficient genetic algorithm for single row facility layout. *Optimization Letters*, 8(2):679–690, 2014.

41 K. Ravi Kumar, George C. Hadjinicola, and Ting-li Lin. A heuristic procedure for the single-row facility layout problem. *European Journal of Operational Research*, 87(1):65–73, 1995.

42 Yung-Te Lai, Massoud Pedram, and Sarma B. K. Vrudhula. EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(8):959–975, 1994.

43 C.-Y. Lee. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999, 1959.

44 Silu Liu, Zeqiang Zhang, Chao Guan, Lixia Zhu, Min Zhang, and Peng Guo. An improved fireworks algorithm for the constrained single-row facility layout problem. *International Journal of Production Research*, 59(8):2309–2327, 2021.

45 Elsa Loekito, James Bailey, and Jian Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowledge and Information Systems*, 24(2):235–268, 2010.

46 Robert Love and Jsun Wong. On solving a one-dimensional space allocation problem with integer programming. *INFOR: Information Systems and Operational Research*, 14(2):139–143, 1976.

47 Gintaras Palubeckis. Single row facility layout using multi-start simulated annealing. *Computers & Industrial Engineering*, 103:1–16, 2017.

48 Guillaume Perez and Jean-Charles Régin. Efficient operations on mdds for building constraint programming models. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 374–380, 2015.

49 Jordi Petit. Experiments on the minimum linear arrangement problem. *Journal of Experimental Algorithmics*, 8, 2003.

50 Jean-Claude Picard and Maurice Queyranne. On the one-dimensional space allocation problem. *Operations Research*, 29(2):371–391, 1981.

51 Hamed Samarghandi and Kourosh Eshghi. An efficient tabu algorithm for the single row facility layout problem. *European Journal of Operational Research*, 205(1):98–105, 2010.

52 Donald M. Simmons. One-dimensional space allocation: an ordering algorithm. *Operations Research*, 17(5):812–826, 1969.

53 J. K. Suryanarayanan, Bruce L. Golden, and Qi Wang. A new heuristic for the linear placement problem. *Computers & Operations Research*, 18(3):255–262, 1991.

54 Willem-Jan van Hoeve. Graph coloring lower bounds from decision diagrams. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 405–418. Springer, 2020.

55 Hélène Verhaeghe, Christophe Lecoutre, and Pierre Schaus. Compact-mdd: Efficiently filtering (s) mdd constraints with reversible sparse bit-sets. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, pages 1383–1389, 2018.

56 Ingo Wegener. Branching programs and binary decision diagrams: Theory and applications. *Discrete Applied Mathematics*, 2000.