A Framework for Handling and Exploiting Symmetry in Benders' Decomposition

Christopher Hojny and Cédric Roy

Eindhoven University of Technology
Eindhoven, The Netherlands
email {*c.hojny, c.j.roy*}@*tue.nl*

November 25, 2025

Abstract

Benders' decomposition (BD) is a framework for solving optimization problems by removing some variables and modeling their contribution to the original problem via so-called Benders cuts. While many advanced optimization techniques can be applied in a BD framework, one central technique has not been applied systematically in BD: symmetry handling. The main reason for this is that Benders cuts are not known explicitly but only generated via a separation oracle.

In this work, we close this gap by developing a theory of symmetry detection within the BD framework. To this end, we introduce a tailored family of graphs that capture the symmetry information of both the Benders master problem and the Benders oracles. Once symmetries of these graphs are known, which can be found by established techniques, classical symmetry handling approaches become available to accelerate BD. We complement these approaches by devising techniques for the separation and aggregation of symmetric Benders cuts by means of tailored separation routines and extended formulations. Both substantially reduce the number of executions of the separation oracles. In a numerical study, we show the effect of both symmetry handling and cut aggregation for bin packing and scheduling problems.

1 Introduction

We consider mixed-integer programs (MIP)

$$\min\{c^\top z + d^\top y : Az \leq b,\ Cz + Dy \leq f,\ z \in K(p,n),\ y \in K(q,m)\}, \tag{1}$$

where A, C, D are matrices and b, c, d, f are vectors of suitable dimensions, and denoting $K(r,s) \coloneqq \mathbb{Z}^r \times \mathbb{R}^{s-r}$ for integers $0 \leq r \leq s$. In many applications like healthcare delivery [12], sports scheduling [32], or facility location [9], the matrix D has a block structure. That is, when the z-variables are fixed, the MIP decomposes into several independent smaller MIPs.

Benders' Decomposition (BD) [3] exploits this structure by splitting the MIP into a master and subproblem. The master problem can be considered an abstract problem (2), where the function $\varphi(z) = \min\{d^\top y : Dy \leq f - Cz,\ y \in K(q,m)\}$ models the optimal objective value of a y-solution for a given z-solution. Variable x then models the contribution of the y-variables to the objective and is linked to the z-variables by (2c).

$$\begin{aligned} \min_{\substack{z \in K(p,n)\\ x \in \mathbb{R}}} \quad & c^\top z + x && \text{(2a)}\\ & Az \leq b && \text{(2b)}\\ & \varphi(z) \leq x && \text{(2c)} \end{aligned}$$

$$\begin{aligned} \min_{\substack{z \in K(p,n)\\ x \in \mathbb{R}}} \quad & c^\top z + x && && \text{(3a)}\\ & Az \leq b && && \text{(3b)}\\ & \alpha^\top z + \beta \leq x, && (\alpha,\beta) \in \mathcal{I} && \text{(3c)} \end{aligned}$$

Since φ does not admit a closed algebraic description, (2) cannot be solved by standard techniques. Instead, (2c) is linearized by a family of linear constraints (3c), where $\mathcal{I}$ is some (usually finite) subset of $\mathbb{R}^n \times \mathbb{R}$. To solve the original MIP, BD removes (3c) from the master problem and adds, if needed, violated inequalities by calling a separation oracle. The original BD [3] showed how to implement a separation oracle for (3c) when $\varphi(z)$ is a linear program; logic-based BD extends these ideas to general MIPs [14, 15]. In particular, when D has a block structure, each block can have an independent oracle, which can reduce the running time of each oracle substantially.

Once a separation oracle has been implemented, Problem (3) can be solved by standard branch-and-cut algorithms. BD therefore also benefits from sophisticated MIP techniques (presolving, cutting planes, etc.), which are readily available in MIP software, with one important exception: symmetry handling. The reason is that MIP software needs to have access to the full problem (3) to detect symmetries, whereas Inequalities (3c) are only available through their oracles. Many researchers [1, 8, 10] therefore add some symmetry handling inequalities by hand, but do not make use of advanced techniques such as orbital branching [23] or orbitopal fixing [4, 33, 16] to handle symmetries. This motivates the following two questions: (Q1) *Can we develop a theory for detecting symmetries in BD?* (Q2) *Can we exploit symmetries in the generation of Benders cuts?* Our main contributions are:

1. We develop a practical framework for detecting symmetries in BD by using the recently introduced concept of symmetry detection graphs (SDG) [13] to provide the master problem (3) symmetry information of the Benders oracles. Detecting automorphisms of SDGs then allows to compute symmetries of (3) and to apply established symmetry handling methods. (Sec. 2)

2. We leverage symmetry information to enhance the separation of Benders cuts: Let π be a symmetry of (3). If $\alpha^\top z + \beta \leq x$ is a Benders cut (3c), Sec. 3 shows that also $\pi^{-1}(\alpha)^\top z + \beta \leq x$ is a Benders cut. A single Benders cut thus gives access to an entire family of symmetric Benders cuts. For actions of the symmetric group, which arise in many applications, we show that this family of symmetric cuts can be separated very efficiently, thus avoiding to call potentially expensive separation oracles. Moreover, we devise an extended formulation to express an exponentially large family of Benders cuts by polynomially many inequalities. That is, separating symmetric cuts becomes obsolete once one member of the family is known. (Secs. 3 and 4)

3. In a numerical study, we demonstrate that exploiting symmetry in BD can reduce the running time of BD by orders of magnitude. (Sec. 5)

2 Benders Symmetry

In this section, we introduce our framework for detecting symmetries in BD. We start by defining our notion of symmetries. For a set A, let $\mathcal{S}_A$ be the set of all permutations of A, the so-called *symmetric group* of A. If $A = [n] := \{1, \dots, n\}$ for some positive integer n, we write $\mathcal{S}_n$ instead of $\mathcal{S}_{[n]}$. For $\pi \in \mathcal{S}_n$ and $x \in \mathbb{R}^n$, let $\pi(x) = (x_{\pi^{-1}(1)}, \dots, x_{\pi^{-1}(n)})$, i.e., π acts on x by permuting its coordinates. Following the MIP literature [18, 21], a *symmetry* of $\min\left\{c^\top x : Ax \leq b, x \in K(p,n)\right\}$ is a permutation $\pi \in \mathcal{S}_n$ such that $c^\top x = c^\top \pi(x)$ for all $x \in \mathbb{R}^n$, and x is feasible if and only if $\pi(x)$ is feasible. As deciding if $\pi \in \mathcal{S}_n$ is a symmetry is NP-hard [21], one usually considers symmetries that keep a specific MIP formulation invariant. Assuming A has m rows, a permutation $\pi \in \mathcal{S}_n$ is a *formulation symmetry* if there is $\rho \in \mathcal{S}_m$ such that $\pi^{-1}(c) = c$, $\rho(b) = b$, and $A_{\rho(i),\pi^{-1}(j)} = A_{i,j}$ for all $(i,j) \in [n] \times [m]$.

To detect formulation symmetries, one usually translates the problem into the language of graph automorphisms. To this end, one constructs a colored graph $G = (V, E)$ with the following properties: (i) there exists an injective map of the MIP's variables to a subset V' of V, the *variable nodes*; (ii) for every color-preserving automorphism $\pi \colon V \to V$ of G, the restriction of π to V' corresponds to a symmetry of the MIP. We refer to such a graph as a *symmetry detection graph (SDG)*. For MIPs, [27] suggests a concrete construction of an SDG by introducing a node for each constraint i and variable j. These nodes are connected by an edge whose color corresponds to $A_{i,j}$. The constraint nodes i are assigned a color corresponding to their right-hand side b_i. Similarly,

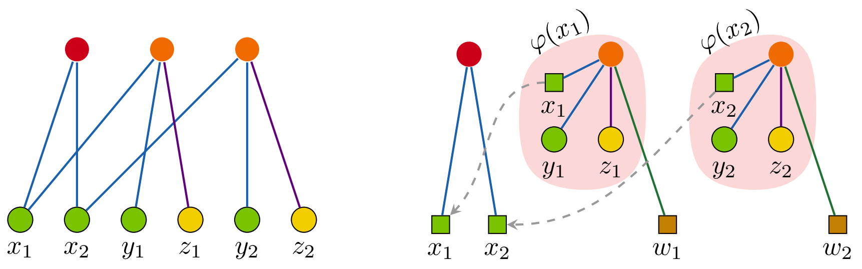

Figure 1: Symmetry detection graph for both formulations of (4). On the left, SDG of the original formulation. On the right, example on how the SDG of the BD can be recovered. We can join The SDG of the subproblems $\varphi(x_i)$, in the pink areas, to the master problem by identifying the nodes x_i. Variables of the master problem are depicted as squares nodes.

the variable nodes j are assigned a color encoding their objective coefficient c_j and their type (integer or continuous). In the context of BD, however, this method is not applicable as the list of constraints (3c) if often too large or not explicitly given.

Recently, [13] introduced a framework for constructing an SDG for a problem by combing SDGs of its building blocks. A central concept for this construction are anchors. An SDG $G = (V, E)$ with variable nodes V' is called *anchored*, if there exists a node $a \in V \setminus V'$, the *anchor*, such that, for every $v \in V$, there exists an a-v-path p in G such that no interior point of p belongs to V'.

Theorem 1 (cf. [13]). *Let $P = \min\{c^\top x : Ax \leq b,\ Cx \leq d,\ x \in K(p,n)\}$ be a MIP. Let G_1 and G_2 be anchored SDGs of $\min\{c^\top x : Ax \leq b,\ x \in K(p,n)\}$ and $\min\{c^\top x : Cx \leq d,\ x \in K(p,n)\}$, respectively. Then, an SDG for P is given by the disjoint union of G_1 and G_2, and identifying the variable nodes of G_1 and G_2 with each other.*

One can thus find an SDG for (2) by merging anchored SDGs for (2b) and (2c) (or (3b) and (3c)). This theorem forms the core of our framework for detecting symmetries in BD, which we illustrate for two different cases of BD.

Classical BD For illustration purposes, consider the linear program (LP)

$$\begin{aligned} \min\ & 5x_1 + 5x_2 + 5y_1 + 5y_2 + 3z_1 + 3z_2 \\ & x_1 + x_2 = 1, && \text{(4a)} \\ & x_1 + y_1 + 2z_1 = 2, && \text{(4b)} \\ & x_2 + y_2 + 2z_2 = 2. && \text{(4c)} \end{aligned}$$

If we keep only the x-variables in the master problem, the master problem is given by

$$\min\{5x_1 + 5x_2 + w_1 + w_2 : x_1 + x_2 = 1,\ \varphi(x_1) \leq w_1,\ \varphi(x_2) \leq w_2\},$$

where $\varphi(x_i) = \min\{5y_i + 3z_i : y_i + 2z_i = 2 - x_i\}$ and $i \in \{1,2\}$. By Theorem 1, an SDG for the BD master problem is given by merging anchored SDGs for $\min\{5x_1 + 5x_2 + w_1 + w_2 : x_1 + x_2 = 1\}$ and $\min\{5x_1 + 5x_2 + w_1 + w_2 : \varphi(x_i) \leq w_i\}$ for $i \in \{1,2\}$, see Fig. 1 for an illustration. Here, the dashed arcs correspond to the identification of variable nodes. Since (3c) is derived from (2c), the SDG constructed this way is also an SDG for (3).

Of course, this construction can be generalized to the classical BD framework, in which the cut generation subproblem is an LP: the SDG for (2c) can be chosen as the anchored SDG for the subproblem (using the construction from [27]).

No-Good Cuts Theorem 1 can also be used to find an SDG for a Benders master problem if the subproblem is not an LP. This is often the case for assignment problems, where we are given a set N of items and a set M of slots, and the task is to assign each item to exactly one slot while minimizing some cost $c \in \mathbb{R}^{M \times N}$. The Benders master problem with Benders cuts (3c) can then be formulated as

$$\min_{z \in \{0,1\}^{M \times N}} c^\top z \;:\quad \sum_{i \in M} z_{ij} = 1, \qquad j \in N, \tag{5a}$$
$$\sum_{j \in C} z_{ij} \leq |C| - 1, \qquad (C, i) \in \mathcal{I}. \tag{5b}$$

The variable z_{ij} indicates whether item j is assigned to slot i, and (5a) ensures that each item j is assigned to exactly one slot. Constraint (5b) contains an inequality for every tuple (C, i), where $C \subseteq N$ is a set of items that cannot simultaneously be assigned to slot $i \in M$. These inequalities are called *no-good cuts*. Depending on the application, the set $\mathcal{I}$ is defined differently.

One example of the assignment problem is the Multiple Knapsack problem (MKP). Here, each item $j \in N$ has a weight a_j and each slot $i \in M$ has a capacity β_i. Given a cost c_{ij} for assigning item j to slot i, the task is to solve

$$\min_{x \in \{0,1\}^{M \times N}} \Big\{ c^\top x : \sum_{i \in M} x_{ij} = 1 \text{ for all } j \in N \text{ and } \sum_{j \in N} a_j x_{ij} \leq \beta_i \text{ for all } i \in M \Big\}.$$

In a BD (5) of the MKP, one initially discards all no-good cuts, i.e., $\mathcal{I} = \emptyset$. The no-good cuts are then generated by repeatedly solving (5) and checking if the obtained solution $\hat{z}$ corresponds to a solution of the MKP. That is, the Benders oracles test if the sum of weights $\sum_{j \in N} a_j \hat{z}_{ij}$ exceeds the capacity β_i for some $i \in M$. In this case, the no-good cut (C, i) for $C = \{j \in N : \hat{z}_{ij} = 1\}$ is added to $\mathcal{I}$; otherwise, the BD terminates and returns $\hat{z}$ as optimal solution.

As the Benders subproblem decomposes into problems $P_i(z) = \min\{0 : \sum_{j \in N} a_j z_j \leq \beta_i\}$ for all $i \in M$, an SDG for (5) is given by merging an SDG for $\min\{c^\top z : z \text{ satisfies (5a)}, z \in \{0,1\}^{M \times N}\}$ and SDGs for $P_i(z)$, $i \in M$. The SDG for $P_i(z)$ has a single constraint node connected to all variables nodes for z_{ij}, $j \in N$, with edge weight a_j.

3 Symmetry Exploitation

Once symmetries of (3) are known, a solver can make use of built-in symmetry handling methods. Next to solving (3) by branch-and-bound, a computational bottleneck is the separation of Benders cuts (3c). We therefore present two ways to exploit symmetries to generate Benders cuts. The main idea is that whenever $\pi \in \mathcal{S}_n$ is a symmetry of (3) and $(\alpha, \beta) \in \mathcal{I}$ is a Benders cut, then also $\pi^{-1}(\alpha)^\top z + \beta \leq x$ is a Benders cut. Indeed, for every feasible (z, x) of (3) and symmetry π, we have $x \geq \alpha^\top \pi(z) + \beta = \pi^{-1}(\alpha)^\top z + \beta$, where the last equality holds since permutations are orthogonal maps. Every Benders cut (α, β) thus gives rise to an entire family of symmetric Benders cuts $\{(\pi^{-1}(\alpha), \beta) : \pi \text{ symmetry of (3)}\}$. One can thus potentially avoid solving an expensive separation problem for (3c) by first separating symmetric Benders cuts of already separated cuts.

In general, this problem is difficult, but for the MKP, separating symmetric Benders cuts is easy by Prop. 2: items j with identical weight a_j form groups $\mathcal{G}$ of variables that can be exchanged arbitrarily, i.e., $\mathcal{S}_G$ acts on group $G \in \mathcal{G}$ of identical items. These symmetries arise in many applications and separating symmetric Benders cuts can greatly improve the performance of BD, see Sect. 5. For some applications, this idea has been applied before [7, 11], but it has not been described from a general perspective.

Proposition 2. *Let Π be a symmetry group of (3) and $(\alpha, \beta) \in \mathcal{I}$ a Benders cut. If Π is isomorphic to the Cartesian product $\bigotimes_{G \in \mathcal{G}} \mathcal{S}_G$ for some partition $\mathcal{G}$ of $[n]$, then $\{(\pi^{-1}(\alpha), \beta) : \pi \in \Pi\} \subseteq \mathcal{I}$ can be separated in $O(n\log(n))$ time.*

We refer to $\{(\pi^{-1}(\alpha), \beta) : \pi \in \Pi\}$ as a *symmetric cut pool* derived from (α, β).

Proof. Let $(\hat{z}, \hat{x}) \in \mathbb{R}^{n+1}$. Solving the separation problem is equivalent to deciding whether $\max_{\pi \in \Pi} \pi^{-1}(\alpha)^\top \hat{z} > \hat{x} - \beta$. Since Π is isomorphic to $\bigotimes_{G \in \mathcal{G}} \mathcal{S}_G$, finding a maximizer π is the same as finding a permutation that matches the k-th largest value of $\{\alpha_j : j \in G\}$ to the k-th largest value of $\{\hat{z}_j : j \in G\}$ for all $G \in \mathcal{G}$, $k \in [|G|]$, which can be done by sorting both α and $\hat{z}$ in $\mathcal{O}(n\log(n))$ time. □ □

Prop. 2 allows to possibly accelerate the separation of (3c). A natural question is if the separation of symmetric Benders cuts can be avoided at all. If all z-variables are binary and the symmetries have the structure as in Prop. 2, this question can be answered affirmatively. Extending ideas of [26], we introduce for each group $G \in \mathcal{G}$ variables $\zeta_G^k \in \{0,1\}$ and constraints that enforce $\zeta_G^k = 1$ if and only if $\sum_{j \in G} z_j \geq k$, see Appendix A for details. Variables $\zeta_G^1, \dots, \zeta_G^{|G|}$ can thus be considered a non-increasing reordering of $\{z_j : j \in G\}$. By denoting the k-th largest value in $\{\alpha_j : j \in G\}$ by α_G^k, the most violated cut constructed in the proof of Prop. 2 can be phrased in terms of the ζ-variables as

$$\sum\nolimits_{G \in \mathcal{G}} \sum\nolimits_{k=1}^{|G|} \alpha_G^k \zeta_G^k + \beta \leq x. \tag{6}$$

We thus immediately obtain the following theorem.

Theorem 3. *Assume all z-variables in (3) are binary. Let $(\alpha, \beta) \in \mathcal{I}$ and symmetry group Π adhere to the conditions in Prop. 2. Then, $(\hat{z}, \hat{x}) \in \{0,1\}^n \times \mathbb{R}$ satisfies all equivalent inequalities in $\{(\pi^{-1}(\alpha), \beta) : \pi \in \Pi\}$ if and only if $(\hat{z}, \hat{\zeta}, \hat{x})$ in the extended model satisfies (6).*

Application to No-Good Cuts We illustrate both techniques for the MKP. Let $\mathcal{G}$ be a partition of the set of items N such that, for every $G \in \mathcal{G}$ and $j_1, j_2 \in G$, we have both $a_{j_1} = a_{j_2}$ as well as $c_{i,j_1} = c_{i,j_2}$ for all $i \in M$, i.e., items j_1 and j_2 have the same weight and objective value. Then, a symmetry group of the MKP is $\Pi = \bigotimes_{G \in \mathcal{G}} \mathcal{S}_G$, where $\pi \in \Pi$ acts on a solution z of (5) by permuting the indices of items, i.e., $\pi(z)_{i,j} = z_{i,\pi^{-1}(j)}$. Note that Prop. 2 is not directly applicable, because this action of Π reorders entire columns of the solution matrix z rather than individual entries. Nevertheless, we can use Prop. 2 since the Benders cuts (5b) only contain variables from a single row i of z.

Let $(C, i) \in \mathcal{I}$ be an index of a cut from (5b). Then, the family of symmetric cuts is the set $\{(\pi^{-1}(C), i) : \pi \in \Pi\}$, and it is uniquely determined by the number of elements selected per group G. We can thus characterize the entire family by a representative vector $R(C) \in \mathbb{Z}_+^{\mathcal{G}}$, where $R_G(C) = |C \cap G|$ for all $G \in \mathcal{G}$. The cuts of the family derived from (C, i) are then given by $\sum_{G \in \mathcal{G}} \sum_{j \in G \cap C'} z_{ij} \leq |C'| - 1$ for all $C' \subseteq N$ with $R(C') = R(C)$.

In practice, to find a violated cut for a solution $\hat{z}$, one can iterate through all $i \in M$, compute $N_i = \{j \in N : \hat{z}_{ij} = 1\}$, and check if there is (C, i) in the list of known cuts that has $R(N_i) = R(C)$. If such a pair exists, we can immediately derive a symmetric Benders cut separating $\hat{z}$ without solving the subproblem. This idea can be enhanced by observing that, if $(C, i) \in \mathcal{I}$, also $(C', i) \in \mathcal{I}$ for all C' with $C \subseteq C' \subseteq N$. Hence, instead of checking $R(N_i) = R(C)$, we can check whether there is $(C, i) \in \mathcal{I}$ with $R(N_i) \geq R(C)$ to find a violated symmetric cut.

By introducing ζ-variables for each slot $i \in M$, Inequality (6) for a no-good cut (C, i) can be easily phrased as $\sum_{G \in \mathcal{G}} \sum_{k=1}^{|G \cap C|} \zeta_{iG}^k \leq |C| - 1$. Since this inequality is violated if and only if $\zeta_{iG}^{|G \cap C|} = 1$ for all $G \in \mathcal{G}$ (recall $\zeta_{iG}^k = 1$ if and only if $\sum_{j \in G} z_{ij} \geq k$), the inequality can be simplified to $\sum_{G \in \mathcal{G}} \zeta_{iG}^{|C \cap G|} \leq |\mathcal{G}| - 1$.

4 Applications

We applied our techniques to three problems with varying difficulty of the cut generation subproblem: sphere packing (SP), rectangle packing (RP), and machine scheduling (MS). In the following, we discuss how the ideas explained in Secs. 2 and 3 can be realized for these examples.

2D Bin Packing Given a set of items N and bins B, the bin packing problem is to find the least number of bins that can hold all the items. We assume that all bins $i \in B$ are rectangles of width W_i and height H_i, and consider two different types of items: spheres and rectangles. For both variants, we use the same master problem (7). Note that, as this is an assignment problem, its formulation stems from the generic one in (5).

$$
\begin{aligned}
\min \sum_{i \in B} u_i & & \\
\sum_{i \in B} z_{ij} = 1, & \qquad j \in N, & \text{(7a)}\\
\sum_{j \in C} z_{ij} \leq |C| - 1, & \qquad (C,i) \in \mathcal{I}, & \text{(7b)}\\
\sum_{j \in N} a_j z_{ij} \leq W_i H_i u_i, & \qquad i \in B, & \text{(7c)}\\
z_{ij} \leq u_i, & \qquad j \in N,\ i \in B. & \text{(7d)}
\end{aligned}
$$

In addition to the constraints from (5), the master problem adds auxiliary variables $u_i \in \{0,1\}$, $i \in B$, indicating whether the corresponding bin i has assigned items. The coefficients a_j, $j \in N$, correspond to the area of item j. Thus, Equation (7c) strengthens the master problem (5) by excluding all solutions that assign a single bin $i \in B$ items whose total area exceeds the bin's area $W_i H_i$.

Given a solution $\hat{z}$, the Benders subproblem is to decide, for each bin $i \in B$, if the selected items $\{j \in N : \hat{z}_{ij} = 1\}$ fit into the bin. For SP, this problem can be solved by deciding feasibility of the following system, cf. [17]:

$$
\begin{aligned}
(x_j - x_k)^2 + (y_j - y_k)^2 \geq (\hat{z}_{ij} + \hat{z}_{ik} - 1)(r_j + r_k)^2, & \qquad j,k \in N,\ j \neq k, & \text{(8a)}\\
r_j \leq x_j \leq W_i - r_j, & \qquad j \in N, & \text{(8b)}\\
r_j \leq y_j \leq H_i - r_j, & \qquad j \in N. & \text{(8c)}
\end{aligned}
$$

The variable pair (x_j, y_j) acts as the position of the center of sphere j in the bin. Observe that Equation (8a) enforces that two spheres do not overlap only if they are both assigned to bin i ($\hat{z}_{ij} = \hat{z}_{ik} = 1$). To build an SDG for Subproblem (8), one can either use rules for general MINLPs, see [18], or capture the symmetries by a more compact tailored graph, see Remark 4 in Appendix B.1.

The subproblem of RP can be modeled as a MIP, c.f. [25], and an SDG can be derived from this MIP, see Appendix B.1 for further details.

Machine Scheduling With Setup Times We consider a machine scheduling problem with setup times between different jobs. We are given a set of machines M, set of jobs N, processing time p_{ij} for job j on machine i, and setup times s_{ijk} for processing job k directly after job j on machine i that satisfy the triangle inequality $s_{ijk} + s_{ikl} \geq s_{ijl}$. The objective is to find the minimum required time to have all jobs processed, called the *makespan*, given that each machine can only process one job at a time. Using the BD approach of [31], we add to (5) a variable $T \geq 0$ representing the makespan, and set the objective to minimize it. To link T with the remaining variables, [31] introduces Benders cuts

$$
T \geq T_i(C) - \sum_{j \in C} (1 - z_{ij})\theta_{ij}, \quad (C,i) \in \mathcal{I}, \tag{9}
$$

where $C \subseteq N$, $i \in M$, and $T_i(C)$ denotes the minimum makespan for processing all jobs in C on machine i. The coefficients $\theta_{ij} := p_{ij} + \max\{s_{ikj} : k \in C\}$ represent an upper bound on the time saved when not assigning job j to that machine. Next to (9), they strengthen the model by introducing auxiliary continuous variables. Since the details are not relevant for the following discussion, we refer to Appendix B.2 for details.

Given a solution $\hat{z} \in \{0,1\}^{M \times N}$, the subproblem is to determine, for all $i \in M$, the value $T_i(N_i)$, where $N_i = \{j \in N : \hat{z}_{ij} = 1\}$. Note that for any subset of jobs C, the minimum makespan is given by $T_i(C) = \sum_{j \in C} p_{ij} + S_i(C)$, where $S_i(C)$ is the minimum total setup time of jobs in C on machine i. Since the setup times depend on the ordering, finding $S_i(C)$ can be reduced to solving a traveling salesperson problem (TSP), see [31]. Note that the classical subtour elimination formulation of the TSP has exponentially many constraints. To detect symmetries, standard approaches list all these constraints explicitly, which is impractical. Theorem1, however, allows to define a compact SDG for the subproblem, see Fig. 2 for an illustration.

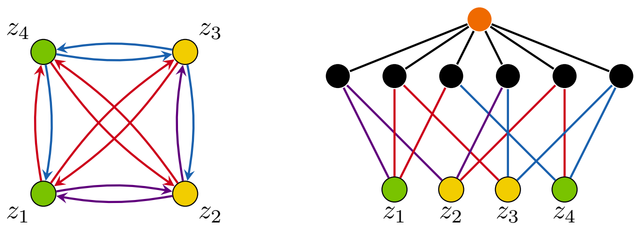

Figure 2: Example of compact SDG for TSP. Left: exemplary instance of the MS subproblem with four selected jobs; node colors represent processing times, and arc colors the setup times. Right: corresponding SDG. One only needs to add a dummy node for each job pair and the orange anchor node, where $z_1, \dots, z_4$ are variable nodes.

Cut Pool and Extended Formulation Since solving the TSP subproblem can be costly, the symmetric cut pool can potentially save a lot of time. When scanning the cut pool, we need to decide if the set of jobs C used in a previously derived cut (C, i) is symmetric to the current assignment $N_i = \{j \in N : \hat{z}_{ij} = 1\}$, $i \in M$. To this end, we define for each machine $i \in M$ an equivalence relation $\sim_i$ defining the groups of identical jobs. We say that two jobs $j_1, j_2 \in N$ satisfy

$$j_1 \sim_i j_2 \iff \begin{cases} p_{ij_1} &= p_{ij_2}, \\ s_{ij_1k} &= s_{ij_2k}, \text{ for all } k \in N \setminus \{j_1, j_2\}, \\ s_{ikj_1} &= s_{ikj_2}, \text{ for all } k \in N \setminus \{j_1, j_2\}, \\ s_{ij_1j_2} &= s_{ij_2j_1}. \end{cases}$$

This relation induces job groups $G \in \mathcal{G}_i$ that depend on machine $i \in M$, and again, we can define a representative vector $R(N_i) \in \mathbb{Z}_+^{\mathcal{G}_i}$, $R_G(N_i) = |N_i \cap G|$, to decide whether a symmetric violated cut exists in the pool. Moreover, by introducing ζ-variables, all cuts symmetric to (9), can be represented by

$$T \geq T_i(C) - \sum_{G \in \mathcal{G}_i} \sum_{h=1}^{|G \cap C|} (1 - \zeta_{iG}^h)\theta_{iG}, \quad (C, i) \in \mathcal{I} \tag{10}$$

where $\theta_{iG} := p_{iG} + \max\{s_{iHG} : H \in \mathcal{G}_i,\ H \cap C \neq \emptyset\}$, $p_{iG} = p_{ij}$ if $j \in G$, $s_{iGH} = s_{ijk}$ if $j \in G$ and $k \in H$ for groups $G, H \in \mathcal{G}_i$.

5 Numerical Results

Sec. 3 has introduced different means to exploit symmetry in BD. This section's goal is to empirically evaluate the impact of these methods. We are particularly interested in the impact of state-of-the-art symmetry handling methods when applied within BD, and if the cut pool or extended formulation has a stronger impact on the performance of BD. The former has not been investigated yet because symmetry detection for BD was not available; regarding the latter, although cut pools and extended formulations have been discussed before [7, 11, 26], to the best of our knowledge, they have not been compared with each other.

Implementation Details We have implemented BD schemes for the RP, SP, and MS problem using the solver SCIP [5]. For each problem, we implemented a so-called constraint handler (CH) to separate Benders cuts. To enable automatic symmetry detection for BD in SCIP, we implemented callbacks of the respective CHs that add symmetry information of the Benders oracles to SCIP's SDG.

Symmetric cut pools are implemented by maintaining a list of abstract Benders cuts that have already been separated. An *abstract cut* does not make use of explicit variables (items/jobs in our applications), but only counts how many variables of which symmetry class $G \in \mathcal{G}$ have been involved in the cut via representative vectors $R(C)$. Abstract cuts thus correspond to families of cuts. Whenever a solution needs to be separated, we iterate through the abstract cuts in our list.

Table 1: Statistics on different approaches for solving RP, SP, and MS problems.

	with symmetry handling				without symmetry handling			
setting	#solved	time	cut-time	#sepa	#solved	time	cut-time	#sepa
RP:								
plain	24	2678.0	2613.6	894.4	24	2704.8	2642.1	1022.0
pool	35	1159.9	395.0	2600.1	26	1960.4	498.5	6981.4
EFrow	42	1059.5	935.7	979.1	34	1451.0	1286.6	1480.8
EFcons	59	232.9	118.4	103.7	41	793.3	233.7	199.0
SP:								
plain	46	287.4	287.1	8.8	42	357.9	357.8	11.6
pool	57	109.2	109.0	10.1	55	127.8	113.3	24.3
EFrow	55	202.7	202.4	7.1	46	250.0	249.8	9.0
EFcons	55	173.4	173.1	5.7	53	187.4	187.0	6.6
MS:								
plain	7	6801.9	6274.9	230 070.3	8	6790.8	6263.8	229 783.5
pool	45	3318.7	29.7	2 018 389.5	46	3286.2	30.2	2 039 326.9
EFrow	84	561.7	6.4	143 804.3	83	588.6	6.8	158 353.7
EFcons	120	29.9	1.0	126.7	119	30.5	1.1	145.5

For each abstract cut, we check whether a member of its associated family of cuts is violated. Only when no violated cut is found in the list, the separation oracles are triggered to possibly generate a new cut, which is then stored in our list of cuts.

Extended formulations are implemented by adding the auxiliary ζ-variables to the initial Benders master problem. The separation of cuts then follows the same framework as for the cut pool, but cuts are added in terms of the ζ-variables. For both the cut pool and extended formulation, we only separate integer solutions to avoid too many calls of the separation oracles.

Settings We compare the performance of a plain BD not exploiting symmetry information (setting `plain`) with BD making use of symmetric cut pools (`pool`) and extended formulations. For extended formulations, we investigate the effect of adding Benders cuts as constraints to the problem (`EFcons`) or as rows to the LP relaxation (`EFrow`). Constraints remain in the problem, whereas rows can be removed again. Using rows, the LP relaxation has potentially less constraints and a weaker bound, but can be solved potentially faster. We also investigate the effect of enabling/disabling SCIP's built-in default symmetry handling methods.

All experiments use a time limit of 2 h. Mean numbers are reported in shifted geometric mean $\prod_{i=1}^{n}(t_i+s)^{1/n}-s$ for numbers $t_1,\dots,t_n$. We use $s=10$ for nodes of the branch-and-bound tree, and $s=1$ for all remaining quantities. For detailed hard- and software specifications, see Appendix C.1.

Numerical Results We have randomly generated 80 RP, 80 SP, and 120 MS instances, see Appendix C.2 for details. The results are summarized in Table 1, where column "#solved" reports on the number of solved instances, "time" and "cut-time" give the mean solving time and time needed to separate Benders cuts, respectively; "#sepa" is the mean number of solutions that have been separated.

In the introduction, we have posed two questions, which we now answer in turn. Question (Q1) concerned whether symmetries can be automatically detected when using BD. Using the framework of SDGs, this is indeed possible and Table 1 shows that we can greatly benefit from handling symmetries in BD when using SCIP's default symmetry handling methods. For the RP problem, we can solve considerably more instances when handling symmetries and substantially reduce the running time (between 27 % and 71 % for `pool`, `EFrow`, and `EFcons`). Using `plain`, however, the effect is moderate. This indicates that symmetry handling only becomes effective when it is combined with exploiting symmetries in generating Benders cuts.

The effect of symmetry handling for the SP problem is less pronounced with running time improvements between 8 % and 19 %. To explain this behavior, we compared the size of the branch-and-bound trees for `pool`, `EFrow`, and `EFcons` for the instances that could be solved by all three methods within the time limit. The mean number of nodes in these trees is 11.5–16.5, see Appendix C.4, i.e., the effect of symmetry handling is minor due to the rather small trees.

Finally, symmetry handling seems to have almost no effect on the MS problem, which can be explained by the symmetry handling methods used by SCIP. Since all master problem variables for the RP and SP problems are binary, SCIP handles symmetries by the methods orbital fixing [23, 24] and lexicographic reduction [33]. The MS problem, however, has predominantly continuous variables in the master problem, and SCIP decides that it is more important to handle symmetries on the continuous variables using SST cuts [19, 28]. SST cuts handle only a small amount of symmetries and their structure is comparable to inequalities that one would add by hand. We therefore also tested whether the more sophisticated techniques lexicographic reduction and orbital reduction [33] yield a performance improvement for MS problems, see Appendix C.3 for details. Indeed, using these techniques, `plain` solves 61 more instances and reduces the running time by more than 92 %, and also `pool`, `EFrow`, and `EFcons` substantially improve their performance. In particular, the most competitive setting `EFcons` reduces its running time by another 78 %. We thus conclude that automatic symmetry detection and handling in BD is very important, because it allows solvers to make use of sophisticated (already built-in) symmetry handling techniques that users can not easily implement on their own.

Question (Q2) asked whether the approach using a cut pool or extended formulation is better suited to exploit symmetries in BD. In the following discussion, we focus on the results for BD that handles symmetries of the master problem. From Table 1 it is apparent that both approaches substantially improve `plain` BD, reducing the mean running times for RP, SP, and MS up to 91 %, 62 %, and 99 %, respectively. In particular, while `plain` could only solve 7 of the 120 MS instances, `EFcons` solves all instances.

For the RP and MS problem, `EFcons` clearly dominates `pool` and `EFrow` when separating solutions. A possible explanation is that adding Benders cuts as constraints in an extended formulation guarantees that no symmetric solutions can be computed anymore. The search space for `EFcons` is thus considerably smaller than for `EFrow` and `pool`. This hypothesis can be confirmed by comparing the number of nodes in the branch-and-bound trees of the instances that are solved by all three settings, see Appendix C.4: For the three test sets, the mean trees for `EFcons` are 30–95 % smaller than for `EFrow` (resp. 22–99 % for `pool`).

For SP problems, however, `EFcons` is considerably slower than `pool`, which has two explanations. On the one hand, many of our instances admit an optimal solution that uses two or three bins. That is, a few Benders cuts are sufficient to achieve a matching dual bound. On the other hand, we observed that the time needed by `EFcons` to find an optimal solution is much higher than for `pool`. We explain the latter by the very expensive separation problem of Benders cuts, which requires to solve a sphere packing problem. Since, as argued above, `EFcons` will never encounter symmetric infeasible solutions, every separation problem is very time consuming. Setting `EFrow` and `pool`, however, can explore symmetric parts of the branch-and-bound tree, which arguably increases the chance that heuristics find good feasible solution, while symmetric solutions can be separated very easily by making use of a symmetric cut pool. Table 1 confirms that the mean time needed to separate a cut using `pool` and `EFrow` is much smaller than for `EFcons` when comparing the time spent for cut separation and the number of separated solutions.

To answer (Q2), the results show that problems in which a small number of Benders cuts suffices to derive a good dual bound benefit from `pool`; problems that require a lot of Benders cuts to prove optimality benefit from `EFcons` (e.g., `EFcons` reduces the number of separated solutions for MS from more than two millions to less than 200). This suggests that a hybrid strategy could be beneficial that first uses the `pool` approach to explore the branch-and-bound tree to find good solutions, and then switches to `EFcons` to keep the tree small.

Conclusion We have addressed the topic of detecting and exploiting symmetries in BD. While automatic symmetry handling became a powerful component of modern MIP solvers [24], BD cannot benefit from these techniques because MIP solvers are not aware of the symmetries of the cuts that are separated. We demonstrated that symmetry detection for BD is, in principle, possible by creating an SDG that captures the symmetries of Benders cuts, and that it can be easily implemented within the solver SCIP. This allowed us to leverage state-of-the-art symmetry handling methods for MIP to BD to achieve substantial performance improvements. Moreover, we systematically compared (to the best of our knowledge) for the first time the cut pool and extended formulation approach to enhance the separation of symmetric Benders cuts. Based on our numerical study, we have seen that neither approach is dominant and suggested a hybrid approach that aims to create synergies between these approaches.

These findings provide fundamental insights into how symmetries should be exploited in BD. This is particularly relevant for automatic BD schemes for generating Benders cuts that exist, e.g., for the solvers CPLEX [6], SAS [29], and SCIP [20]. Our long term goal is to incorporate our findings into such an automatic BD scheme, including a fully automatic symmetry detection algorithm for generic BD and different strategies for exploiting symmetries: next to the separation of Benders cuts, we aim to use symmetry information for strengthening Benders cuts by means of lifting and sparsification

References

[1] Yossiri Adulyasak, Jean-François Cordeau, and Raf Jans. Benders decomposition for production routing under demand uncertainty. *Operations Research*, 63(4):851–867, 2015.

[2] Markus Anders, Pascal Schweitzer, and Julian Stieß. Engineering a preprocessor for symmetry detection. *CoRR*, abs/2302.06351, 2023.

[3] Jacques F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[4] Pascale Bendotti, Pierre Fouilhoux, and Cécile Rottner. Orbitopal fixing for the full (sub-)orbitope and application to the unit commitment problem. *Mathematical Programming*, 186:337–372, 2021.

[5] Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP Optimization Suite 9.0, 2024.

[6] Pierre Bonami, Domenico Salvagnin, and Andrea Tramontani. Implementing automatic Benders decomposition in a modern MIP solver. In Daniel Bienstock and Giacomo Zambelli, editors, *Integer Programming and Combinatorial Optimization*, pages 78–90, Cham, 2020. Springer International Publishing.

[7] Maryam Daryalal, Hamed Pouya, and Marc Antoine DeSantis. Network migration problem: A hybrid logic-based Benders decomposition approach. *INFORMS Journal on Computing*, 35(3):593–613, 2023.

[8] Maxence Delorme, Manuel Iori, and Silvano Martello. Logic based Benders' decomposition for orthogonal stock cutting problems. *Computers & Operations Research*, 78:290–298, 2017.

[9] Mohammad M Fazel-Zarandi and J Christopher Beck. Using logic-based Benders decomposition to solve the capacity- and distance-constrained plant location problem. *INFORMS Journal on Computing*, 24(3):387–398, 2012.

[10] Cheng Guo, Merve Bodur, Dionne M. Aleman, and David R. Urbach. Logic-based Benders decomposition and binary decision diagram based approaches for stochastic distributed operating room scheduling. *INFORMS Journal on Computing*, 33(4):1551–1569, 2021.

[11] Peiran Han, Lingyun Meng, Xiaojie Luan, Nikola Bešinović, Jianrui Miao, Yihui Wang, and Zhengwen Liao. Integrated optimization of train makeup problem and resource scheduling in railway marshalling yards: A hybrid MILP-CP approach with logic-based Benders decomposition. *Transportation Research Part B: Methodological*, 200:103306, 2025.

[12] Aliza Heching, John N Hooker, and Ryo Kimura. A logic-based Benders approach to home healthcare delivery. *Transportation Science*, 53(2):510–522, 2019.

[13] Christopher Hojny. Detecting and handling reflection symmetries in mixed-integer (nonlinear) programming and beyond. *Mathematical Programming Computation*, pages 1–48, 2025.

[14] John Hooker. *Logic-Based Benders Decomposition.* Synthesis Lectures on Operations Research and Applications. Springer, Cham, 1 edition, 2024.

[15] John N Hooker and Greger Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[16] Volker Kaibel, Matthias Peinhardt, and Marc E. Pfetsch. Orbitopal fixing. *Discrete Optimization*, 8(4):595–610, 2011.

[17] Aida Khajavirad. The circle packing problem: A theoretical comparison of various convexification techniques. *Operations Research Letters*, 57:107197, 2024.

[18] Leo Liberti. Reformulations in mathematical programming: automatic symmetry detection and exploitation. *Mathematical Programming*, 131(1):273–304, 2012.

[19] Leo Liberti and James Ostrowski. Stabilizer-based symmetry breaking constraints for mathematical programs. *Journal of Global Optimization*, 60:183–194, 2014.

[20] Stephen J. Maher. Implementing the branch-and-cut approach for a general purpose Benders' decomposition framework. *European Journal of Operational Research*, 290(2):479–498, 2021.

[21] François Margot. Symmetry in integer linear programming. *50 years of integer programming 1958–2008: from the early years to the state-of-the-art*, pages 647–686, 2009.

[22] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014.

[23] James Ostrowski, Jeff Linderoth, Fabrizio Rossi, and Stefano Smriglio. Orbital branching. *Mathematical Programming*, 126(1):147–178, 2011.

[24] Marc E Pfetsch and Thomas Rehn. A computational comparison of symmetry handling methods for mixed integer programs. *Mathematical Programming Computation*, 11(1):37–93, 2019.

[25] David Pisinger and Mikkel Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing*, 19(1):36–51, 2007.

[26] Atle Riise, Carlo Mannino, and Leonardo Lamorgese. Recursive logic-based Benders' decomposition for multi-mode outpatient scheduling. *European Journal of Operational Research*, 255(3):719–728, 2016.

[27] Domenico Salvagnin. A dominance procedure for integer programming. *Master's thesis, University of Padova, Padova, Italy*, 2005.

[28] Domenico Salvagnin. Symmetry breaking inequalities from the Schreier-Sims table. In Willem-Jan van Hoeve, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 521–529. Springer International Publishing, 2018.

[29] SAS Institute Inc. *SAS Programming Documentation.* https://documentation.sas.com/doc/en/pgmsascdc/v_068/casmopt/casmopt_benders_overview.htm.

[30] Christopher S Tang and Eric V Denardo. Models arising from a flexible manufacturing machine, Part I: minimization of the number of tool switches. *Operations research*, 36(5):767–777, 1988.

[31] Tony T Tran, Arthur Araujo, and J Christopher Beck. Decomposition methods for the parallel machine scheduling problem with setups. *INFORMS Journal on Computing*, 28(1):83–95, 2016.

[32] David Van Bulck and Dries Goossens. A traditional Benders' approach to sports timetabling. *European Journal of Operational Research*, 307(2):813–826, 2023.

[33] Jasper van Doornmalen and Christopher Hojny. A unified framework for symmetry handling. *Mathematical Programming*, 212:217–271, 2025.

[34] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.

A Omitted Technical Details

Let Π be a symmetry group of a BD problem (3) such that Π is isomorphic to $\bigotimes_{G\in\mathcal{G}} \mathcal{S}_G$ for some partition $\mathcal{G}$ of $[n]$. In Section 3, we have introduced auxiliary binary variables ζ to express an entire family

$$\sum_{j=1}^{n} \pi^{-1}(\alpha)_j z_j + \beta \leq x, \qquad\qquad \pi \in \Pi,$$

of symmetric Benders cuts by a single constraint

$$\sum_{G\in\mathcal{G}} \sum_{k=1}^{|G|} \alpha_G^k \zeta_G^k + \beta \leq x.$$

Assuming that all z-variables involved in the Benders cut are binary, this construction required that $\zeta_G^k = 1$ if and only if $\sum_{j\in G} z_j \geq k$ for all $G \in \mathcal{G}$ and $k \in [|G|]$. Since the main part did not discuss how this linking of the z- and ζ-variables can be achieved, we provide the missing details next. To this end, we introduce the following constraints

$$\begin{aligned} &\sum_{j\in G} z_j \leq k - 1 + (|G| - (k-1))\zeta_G^k, && G \in \mathcal{G},\ k \in [|G|], \\ &k \cdot \zeta_G^k \leq \sum_{j\in G} z_j, && G \in \mathcal{G},\ k \in [|G|]. \end{aligned}$$

Indeed, the first set of constraints enforces $\sum_{j\in G} z_j \leq k-1$ whenever $\zeta_G^k = 0$, whereas the second set of constraints implies $\sum_{j\in G} z_j \geq k$ if $\zeta_G^k = 1$.

B Full Models for Tested Benders Decompositions

This appendix provides the full details for the master problems and subproblems used in our implementation of BD for the rectangle packing and scheduling problem. The presented models are also the models that we have used in our implementation.

B.1 Rectangle Packing

In the rectangle packing problem, we are given a set N of rectangles and a set B of rectangular bins. Rectangle $j \in N$ has width w_j and height h_j, whereas bin $i \in B$ has width W_i and height H_i. The task is to find an assignment of the rectangles N to the bins B such that the rectangles assigned to the same bin i can be placed inside bin i without overlap. Among all such assignments, one is looking for one that minimizes the number of bins with an assigned rectangle.

The master problem follows the blueprint problem (5) and introduces the usual binary variables $z \in \{0,1\}^{B\times N}$ to indicate whether rectangle $j \in N$ is assigned to bin i. Additional variables $u_i \in \{0,1\}$, $i \in B$, are introduced to indicate whether bin i is non-empty. Instead of deciding feasibility of an assignment purely based on solving Benders subproblems, we enhance the master problem by ruling out assignments of rectangles whose total area exceeds the area of the bin. The master problem is then given by

$$\min \sum_{i \in B} u_i$$

$$\sum_{i \in B} z_{ij} = 1, \qquad j \in N, \tag{11a}$$

$$\sum_{j \in C} z_{ij} \leq |C| - 1, \qquad (C, i) \in \mathcal{I}, \tag{11b}$$

$$\sum_{j \in N} (w_j \cdot h_j) z_{ij} \leq W_i \cdot H_i, \qquad i \in B, \tag{11c}$$

$$z_{ij} \leq u_i, \qquad i \in B,\ j \in N, \tag{11d}$$

$$z_{ij} \in \{0, 1\}, \qquad i \in B,\ j \in N, \tag{11e}$$

$$u_i \in \{0, 1\}, \qquad i \in B. \tag{11f}$$

The family of Benders cuts $\sum_{j \in C} z_{ij} \leq |C| - 1$ for $(C, i) \in \mathcal{I}$ can then be separated by taking a solution $\hat{z} \in \{0, 1\}^{B \times N}$ and checking whether the items assigned to bin $i \in B$ fit into the bin. Following [25], given a bin i and solution $\hat{z}$, the subproblem for deciding feasibility of $\hat{z}$ can be formulated as

$$\max 0$$

$$x_j + \hat{z}_{ij} w_j + l_{jk} W_i \leq x_k + W_i, \qquad j \neq k \in N, \tag{12a}$$

$$y_j + \hat{z}_{ij} h_j + b_{jk} H_i \leq y_k + H_i, \qquad j \neq k \in N, \tag{12b}$$

$$l_{jk} + l_{kj} + b_{jk} + b_{kj} \geq 1, \qquad j < k \in N, \tag{12c}$$

$$l_{jk} + l_{kj} \leq 1, \qquad j < k \in N \tag{12d}$$

$$b_{jk} + b_{kj} \leq 1, \qquad j < k \in N \tag{12e}$$

$$l_{jk}, b_{jk} \in \{0, 1\}, \qquad j \neq k \in N, \tag{12f}$$

$$0 \leq x_j \leq \hat{z}_{ij}(W - w_j), \qquad j \in N, \tag{12g}$$

$$0 \leq y_j \leq \hat{z}_{ij}(H - h_j), \qquad j \in N, \tag{12h}$$

where the pair (x_j, y_j) acts as the bottom left corner of the j-th rectangle. Observe that if $\hat{z}_{ij} = 0$ for some $j \in N$, then the rectangle j is forced to be placed at $(0, 0)$ and its width and height are ignored. In practice, one would remove the rectangle j from the subproblem. For the sake of exposition, however, we decided to keep all z-variables in the subproblem because it allows to define an SDG for the subproblem by identifying $\hat{z}$ with the corresponding master variables.

The binary variable l_{jk} indicates whether rectangles j and k do not overlap horizontally, and j is to the left of k. Indeed, if $l_{jk} = 1$, Equation (12b) reduces to $x_j + w_j \leq x_k$. Otherwise, if $l_{jk} = 0$, the constraint is trivially always satisfied. Analogously, the variable b_{jk} is 1 only if rectangles j and k do not overlap vertically, and j is below k. This means that for two rectangles to overlap, they necessarily would have $l_{kj} = l_{kj} = b_{jk} = b_{kj} = 0$, which is prevented by Equation (12c). Moreover, we add (12d) and (12e) to the model to directly encode that neither rectangle j is simultaneously to the left and right of rectangle k nor above/below it.

To build an SDG for the RP problem, we follow Theorem 1 and create an SDG for (11) and (12). Afterwards, these two SDGs are merged into a single SDG by identifying variable nodes with each other. Since both problems are MIPs, these SDGs can be created following the standard rules for building SDGs for MIPs [27]. By exploiting knowledge of the subproblem, however, we can create a more compact SDG $G = (V, E)$ for the subproblem, which is expected to reduce the time needed for detecting automorphisms of the SDG.

To this end, recall that the SDG only needs to capture the symmetries of the z-variables, because the x- and y-variables are not present in the master problem. For a given bin $i \in B$, it is therefore sufficient to introduce variable nodes for z_{ij}, $j \in N$, in the SDG. More precisely, we define the node set

$$V = \{a\} \cup \bigcup_{j \in N} \{v_j, w_j, h_j\},$$

where a is the anchor of the graph, v_j is a variable node representing variable z_{ij}, and w_j and h_j are nodes representing the width and height of item j, respectively. The anchor a receives a color that uniquely characterizes the dimensions (W_i, H_i) of the respective bin i, the variable nodes are colored according to their objective coefficient and type in the master problem, and the nodes w_j and h_j receive a unique color that indicates them as width-node and height-node, respectively. Moreover, we define the edge set

$$E = \bigcup_{j \in N} \{\{a, w_j\}, \{w_j, h_j\}, \{h_j, v_j\}\},$$

where $\{a, w_j\}$ receives a color corresponding to the width w_j, $\{w_j, h_j\}$ receives a color based on the height h_j, and edge $\{h_j, v_j\}$ remains uncolored. Due to this construction, an automorphism of G can only exchange variable nodes v_j and $v_{j'}$ if $w_j = w_{j'}$ and $h_j = h_{j'}$. Thus, the SDG G captures the symmetries of the subproblem.

Remark 4. *An analogous construction can be used to capture the symmetries of the SP problem. Instead of introducing a node w_j for the width and h_j for the height of item j, it is sufficient to introduce a node representing the radius of sphere j. The edge connecting the anchor with the radius-node is then colored according to the radius of sphere j.*

B.2 Machine Scheduling with Setup Times

Let M be a set of machines and N be a set of jobs. For each job $j \in N$, its processing time on machine $i \in M$ is p_{ij}. Moreover, if job k is processed immediately after job j on machine i, a setup time of s_{ijk} is needed. The machine scheduling problem, requires to assign each job to exactly one machine. The makespan of a machine is then determined by finding an ordering of the assigned jobs on that machine such that the total setup time is minimized; the makespan is then this minimum total setup time plus the total processing time of the assigned jobs. The goal of the machine scheduling problem is to find an assignment of jobs to machines such that the maximum makespan of a machine is minimized.

Master Problem In our implementation of the master problem for the machine scheduling problem, we use an enhanced formulation as proposed by [31]. To this end, recall from Section 4 that finding the optimal setup time of the jobs assigned to a machine requires to solve a TSP instance. The idea of [31] is therefore to embed a relaxation of the TSP into the master problem, which allows to encode a lower bound on the optimal setup time into the model. To this end, the set of jobs N is augmented by an artificial job 0 that serves as the starting and finishing job on each machine; denote $N_0 = N \cup \{0\}$. We assume that both the processing time p_0 and the setup times s_{ij0} from any job $j \in N$ to job 0 is 0. We do not impose $s_{i0k} = 0$ though to allow for modeling start up times of machines. Additionally, the setup times need to respect the triangle inequality, i.e., $s_{ijk} + s_{ikl} \geq s_{ijl}$ for all $j, k, l \in N$.

The model of [31] then introduces a variable T to model the maximum makespan. Moreover, it has continuous variables y_{ijk} to model whether job j is the direct predecessor of job k on machine i. These variables can be considered as a relaxation of the variables of the subtour elimination formulation of the TSP on machine i. The model of [31] then reads as follows:

$$\begin{aligned}
&\min T \\
&\sum_{i\in M} z_{ij} = 1, && j \in N, && \text{(13a)}\\
&z_{i0} = 1, && i \in M, && \text{(13b)}\\
&z_{ij} = \sum_{k\in N_0} y_{ijk}, && j \in N_0,\ i \in M, && \text{(13c)}\\
&z_{ij} = \sum_{j\in N_0} y_{ikj}, && j \in N_0,\ i \in M, && \text{(13d)}\\
&\xi_i = \sum_{j\in N_0, k\in N_0} y_{ijk} s_{ijk}, && i \in M, && \text{(13e)}\\
&\sum_{j\in N} z_{ij} p_{ij} + \xi_i \le T, && i \in M, && \text{(13f)}\\
&T_i(C) - \sum_{j\in C} (1 - z_{ij})\theta_{ij} \le T, && (C, i) \in \mathcal{I}, && \text{(13g)}\\
&z_{ij} \in \{0, 1\}, && j \in N,\ i \in M, && \text{(13h)}\\
&y_{ijk} \in [0, 1], && j, k \in N_0,\ i \in M, && \text{(13i)}\\
&\xi_i \in \mathbb{R}_{\ge 0}, && i \in M. && \text{(13j)}
\end{aligned}$$

The variables z_{ij} model whether job j is assigned to machine i, and (13a) enforces that each job is assigned to exactly one machine. Moreover, the auxiliary job 0 is assigned to every machine via (13b). The y-variables are linked with the z-variables via (13c) and (13d). These constraints enforce that, if a job j is assigned to machine i, then the corresponding node in a relaxed TSP solution is entered and left exactly once (by the possibly fractional TSP solution y). The relaxed TSP solution y then allows to derive a lower bound on the optimal setup time on machine i via (13e). Constraint (13f) then links this lower bounds and the processing times of jobs assigned to machine i, to derive a lower bound on the makespan T. Finally, (13g) are the Benders cuts that provide the correct linking of the makespan variable T with the z-variables.

Subproblem The Benders cuts are derived as follows. Given a solution $\hat{z}$ of the master problem, the subproblem determines the minimum required time to process all these jobs per machine. As the solution revolves around finding an optimal ordering, we can use a slight variation of the subtour elimination formulation of the TSP:

$$\begin{aligned}
&\min \sum_{j\in N_0} \hat{z}_{ij} p_{ij} + \sum_{j\in N_0} \sum_{k\in N_0\setminus\{j\}} s_{ijk} x_{jk} \\
&\sum_{j\in N_0} x_{jk} = 1, && k \in N_0, && \text{(14a)}\\
&\sum_{k\in N_0} x_{jk} = 1, && j \in N_0, && \text{(14b)}\\
&\sum_{j\in S} \sum_{k\in S\setminus\{j\}} x_{jk} \le \sum_{j\in S} \hat{z}_{ij} - 1, && S \subsetneq N_0, |S| \ge 2, && \text{(14c)}\\
&x_{jj} = 1 - \hat{z}_{ij}, && j \in N_0, && \text{(14d)}\\
&x_{jk} \in \{0, 1\}, && j, k \in N_0. && \text{(14e)}
\end{aligned}$$

Formulation (14) represents each job as a node of a graph, and uses variable x_{jk} to indicate whether there is a directed edge from j to k. Equations (14a) and (14b) ensure that exactly one edge is coming in and going out of each node, and Constraint (14c) guarantees that there cannot be subcycles on the set of selected jobs. Because Constraint (14d) forces the selected jobs to not be isolated, the variables x_{jk} necessarily form a cycle over the selected jobs.

C Additional Information on the Numerical Experiments

C.1 Hardware and Software Specifications

All experiments have been conducted on a Linux Cluster with Intel Xeon E5-1620 v4 3.5 GHz quad core processors and 32 GB memory. The code was executed single-threaded with a time limit of 2 h. We use SCIP 9.2.3 [5] as branch-and-bound framework; LP relaxations are solved using SoPlex 7.1.5 and nonlinear problems are solved by Ipopt 3.14.20 [34]. Symmetries of SDGs are detected by sassy 1.1 [2] and Nauty 2.8.8 [22].

C.2 Details About Random Instances

We have generated random instances of the RP, SP, and MS problem as follows.

Sphere Packing All instances make use of 5 rectangular bins of width and height $W = 15$ and $H = 10$ respectively. The instances differ, however, in the number of spheres and their radii. To guarantee that the instances have symmetries, we create spheres in batches of identical radii. Each batch has size 5, and for each batch we sample its radius uniformly at random from the interval $[1, 3]$ and round it to one decimal. We have created our instances by either sampling two or three batches, i.e., our instances have either 10 or 15 spheres.

For each number of batches, we have created 40 random instances, thus yielding 80 instances in total.

Rectangle Packing The instances of the rectangle packing problem are created similarly. Instead of making use of 5 bins, however, we create one bin per item, and bins have width $W = 15$ and height $H = 15$. Moreover, for each instance, we create five batches consisting of either 4 or 5 items. The width and height of each batch of items is sampled uniformly at random from the interval $[1, 10]$ and is rounded to one decimal.

For each number of batches, we have created 40 random instances, thus yielding 80 instances in total.

Machine Scheduling The instances of the machine scheduling problem have been generated similarly, except for the setup times. Each instance consists of three identical machines. We divided the instances in three batches each consisting of 5, 6, or 7 jobs. By adding the artificial job 0, this amounts in instances with 16, 19, or 22 jobs. The processing time of each batch of jobs is sampled uniformly at random from the interval $[1, 10]$ and is rounded to one decimal.

In order to generate setup times between job pairs that follow the triangle inequality, we took inspiration from the tool switching problem [30]. For each job j, we select a set C_j of tools, and the setup time between jobs j and j' is computed as the cardinality of $(C_j \setminus C_{j'}) \cup (C_{j'} \setminus C_j)$, i.e., the symmetric difference of C_j and $C_{j'}$. This notion of setup time automatically satisfies the triangle inequality. In our implementation, the sets of tools are randomly selected subsets $C \subseteq \{1, \ldots, 10\}$ with $|C| \leq 5$. Finally, we force the setup times $s_{ij0} = 0$ for all jobs j and all machines i.

For each type listed above, we generated batches of 40 instances, thus yielding 120 instances in total.

C.3 Numerical Results for Scheduling with Advanced Symmetry Handling

In this appendix, we provide numerical results for the MS problem when using different symmetry handling methods. We compare SCIP's default symmetry handling methods (SST cuts) with the more advanced techniques of lexicographic and orbital reduction [33]. For the latter, we set the parameter `misc/usesymmetry` of SCIP to 3. Table 2 summarizes our results, where the columns corresponding to "default symmetry handling" correspond to SST cuts, and "advanced symmetry handling" refer to the results for lexicographic and orbital reduction.

We can observe that the running time significantly improves when using the advanced symmetry handling methods. The `plain` setting solves 61 more instances and reduces the running time

Table 2: Statistics on different approaches for solving MS problems with default and advanced symmetry handling methods.

	default symmetry handling				advanced symmetry handling			
setting	#solved	time	cut-time	#sepa	#solved	time	cut-time	#sepa
plain	7	6801.9	6274.9	230 070.3	68	511.6	455.8	16 226.3
pool	45	3318.7	29.7	2 018 389.5	76	160.6	7.9	56 406.8
EFrow	84	561.7	6.4	143 804.3	99	65.8	3.6	11 828.9
EFcons	120	29.9	1.0	126.7	120	6.5	1.0	86.1

Table 3: Detailed statistics for RP instances that are solved by all Benders approaches.

	running time			#nodes			#sepa		
instance	pool	EFrow	EFcons	pool	EFrow	EFcons	pool	EFrow	EFcons
2DRP_20N5grp_0	1101.7	5577.6	304.4	6104	12 814	1575	4993	5081	231
2DRP_20N5grp_1	5.0	80.2	747.6	1	39	95	5	41	76
2DRP_20N5grp_10	36.5	33.0	6.1	23 620	701	200	15 254	661	93
2DRP_20N5grp_11	13.1	48.6	9.9	1552	1617	620	1498	985	121
2DRP_20N5grp_13	3.1	48.6	10.8	25	719	128	61	589	136
2DRP_20N5grp_17	16.5	7.8	5.7	93	86	52	221	67	43
2DRP_20N5grp_18	16.2	39.4	46.0	28	36	46	74	51	58
2DRP_20N5grp_19	113.9	9.7	6.6	13	47	22	57	84	48
2DRP_20N5grp_21	1662.7	6436.9	6446.2	20	11	11	57	22	22
2DRP_20N5grp_23	521.5	2.5	2.5	9	8	8	24	12	12
2DRP_20N5grp_28	37.1	52.0	33.0	508	387	264	759	211	164
2DRP_20N5grp_29	60.1	1088.2	14.8	1377	633 909	375	1886	8383	189
2DRP_20N5grp_3	3729.4	368.4	369.0	1	1	1	5	7	7
2DRP_20N5grp_30	148.5	101.6	101.3	1	1	1	5	9	9
2DRP_20N5grp_31	3180.0	7.0	7.0	1	1	1	5	5	5
2DRP_20N5grp_35	40.8	62.6	62.9	1	1	1	18	5	5
2DRP_20N5grp_37	17.8	242.5	16.5	1766	2598	206	1802	2230	186
2DRP_20N5grp_5	32.8	983.6	44.2	350	7811	264	486	4040	135
2DRP_20N5grp_6	29.3	3549.8	28.3	886	40 638	388	856	14 927	157
2DRP_20N5grp_7	9.1	19.2	19.2	1	1	1	5	7	7
2DRP_20N5grp_8	117.7	1204.0	91.2	6333	6841	5326	7711	3474	225
2DRP_20N5grp_9	661.5	265.6	265.1	3	1	1	35	10	10
2DRP_25N5grp_0	423.3	1868.5	47.8	1270	1373	266	1653	800	136
2DRP_25N5grp_1	38.9	204.9	204.8	1	42	42	19	15	15
2DRP_25N5grp_17	29.1	813.9	45.7	340	6266	547	446	3240	145
2DRP_25N5grp_19	5975.0	12.4	13.2	97	34	34	196	40	40
2DRP_25N5grp_28	13.2	223.8	68.0	57	869	781	165	627	420
2DRP_25N5grp_31	16.3	84.9	85.1	1	6	6	10	11	11
2DRP_25N5grp_35	30.2	1.8	1.8	1	1	1	14	7	7
mean (29 inst.)	80.5	122.5	41.3	98.4	221.9	76.5	131.5	138.6	46.5

by more than 92 %, and also the settings exploiting symmetries in the generation of Benders cuts substantially improve their performance. In particular, the most competitive setting `EFcons` reduces its running time by another 78 %.

Next to the faster running time, we also observe that symmetry handling substantially reduces number of solutions that need to be separated. This effect is most pronounced for `pool`, where the mean number of separated solutions reduces from more than two million to approximately 56.4 thousand.

C.4 Detailed Numerical Results

This appendix provides detailed numerical results for the settings `pool`, `EFrow`, and `EFcons` (with symmetry handling enabled) for all instances of the test sets RP (Table 3), SP (Table 4), and MS (Table 5) that could be solved by all three settings within the time limit. For each instance, these tables provide the running time in seconds (running time), the number of nodes in the branch-and-bound tree (#nodes), and the number of times a solution was separated (#sepa). The last line of each table provides the shifted geometric mean of the respective columns.

Table 4: Detailed statistics for SP instances that are solved by all Benders approaches.

	running time			#nodes			#sepa		
instance	pool	EFrow	EFcons	pool	EFrow	EFcons	pool	EFrow	EFcons
2DSP_10N2grp_0	112.9	357.3	358.1	1	1	1	16	6	6
2DSP_10N2grp_1	1.3	1.3	1.3	1	1	1	1	1	1
2DSP_10N2grp_10	12.2	40.4	40.2	1	5	5	15	19	19
2DSP_10N2grp_12	0.9	0.8	0.9	1	1	1	1	1	1
2DSP_10N2grp_13	16.4	46.0	49.4	1	1	1	8	6	6
2DSP_10N2grp_14	0.0	0.1	0.0	1	1	1	1	1	1
2DSP_10N2grp_15	225.6	755.2	806.0	1	1	1	11	7	7
2DSP_10N2grp_16	13.8	21.9	22.0	1	1	1	10	7	7
2DSP_10N2grp_17	0.2	0.2	0.2	1	1	1	1	1	1
2DSP_10N2grp_19	0.2	0.2	0.2	1	1	1	1	1	1
2DSP_10N2grp_2	44.7	132.2	131.5	1	1	1	9	6	6
2DSP_10N2grp_21	0.0	0.0	0.0	1	1	1	1	1	1
2DSP_10N2grp_22	0.2	0.2	0.2	1	1	1	1	1	1
2DSP_10N2grp_23	19.6	46.7	47.0	4	1	1	43	15	15
2DSP_10N2grp_24	0.2	0.2	0.2	1	1	1	1	1	1
2DSP_10N2grp_25	0.0	0.0	0.1	1	1	1	1	1	1
2DSP_10N2grp_26	3.4	9.3	7.9	23	34	19	54	32	23
2DSP_10N2grp_27	0.2	0.2	0.2	1	1	1	1	1	1
2DSP_10N2grp_28	440.9	2839.9	2851.9	3	3	3	28	10	10
2DSP_10N2grp_29	7.5	195.1	194.0	1	1	1	10	11	11
2DSP_10N2grp_3	0.9	0.9	1.0	1	1	1	1	1	1
2DSP_10N2grp_30	0.8	0.8	0.8	1	1	1	1	1	1
2DSP_10N2grp_31	0.1	0.1	0.1	1	1	1	1	1	1
2DSP_10N2grp_32	3.6	147.0	147.0	1	1	1	8	7	7
2DSP_10N2grp_33	28.3	147.1	147.0	1	2	2	10	18	18
2DSP_10N2grp_34	70.1	96.8	96.7	1	1	1	17	8	8
2DSP_10N2grp_35	1.1	1.1	1.1	1	1	1	1	1	1
2DSP_10N2grp_36	2.2	25.4	25.5	1	1	1	8	10	10
2DSP_10N2grp_37	4.7	23.6	22.3	29	32	25	93	31	29
2DSP_10N2grp_38	0.2	0.2	0.2	1	1	1	1	1	1
2DSP_10N2grp_39	8.8	49.0	34.8	35	52	33	96	33	27
2DSP_10N2grp_4	21.7	20.7	20.7	1	2	2	18	19	19
2DSP_10N2grp_5	211.8	1683.5	1567.8	4	19	18	27	73	67
2DSP_10N2grp_6	0.2	0.2	0.2	1	1	1	1	1	1
2DSP_10N2grp_8	7.5	21.0	18.1	27	19	17	95	14	12
2DSP_10N2grp_9	205.9	2338.0	2553.4	4	13	13	37	66	66
2DSP_15N3grp_1	1.9	1.9	1.9	1	1	1	1	1	1
2DSP_15N3grp_10	78.6	482.2	357.6	53	85	30	99	78	59
2DSP_15N3grp_13	85.8	1846.7	144.6	1603	1139	229	2395	985	104
2DSP_15N3grp_15	640.1	2375.4	851.4	145	40	21	339	30	16
2DSP_15N3grp_2	680.2	4333.1	1602.8	199	176	60	305	122	43
2DSP_15N3grp_21	1.5	1.5	1.5	1	1	1	1	1	1
2DSP_15N3grp_22	1.5	1.5	1.5	1	1	1	1	1	1
2DSP_15N3grp_26	1792.9	5489.1	5146.1	107	129	66	132	72	45
2DSP_15N3grp_29	290.4	562.8	404.4	27	137	45	63	134	41
2DSP_15N3grp_31	1.4	1.5	1.5	1	1	1	1	1	1
2DSP_15N3grp_32	427.1	1541.7	541.7	1270	1041	463	1925	752	122
2DSP_15N3grp_33	120.0	604.6	213.4	131	137	42	76	134	50
2DSP_15N3grp_35	1.4	1.4	1.4	1	1	1	1	1	1
2DSP_15N3grp_36	21.4	126.6	65.4	1126	52	26	1105	87	48
2DSP_15N3grp_37	42.8	1215.0	835.2	693	1089	404	1410	449	76
2DSP_15N3grp_39	42.8	1595.5	161.0	2106	5405	2719	4415	1009	110
2DSP_15N3grp_8	1267.2	4352.5	2097.0	266	176	115	324	78	40
mean (53 inst.)	12.6	34.8	27.7	16.4	16.5	11.5	15.9	11.0	8.1

Table 5: Detailed statistics for MS instances that are solved by all Benders approaches.

	running time			#nodes			#sepa		
instance	pool	EFrow	EFcons	pool	EFrow	EFcons	pool	EFrow	EFcons
MSPS_M3_16J4_0	208.1	21.7	5.6	334 963	8098	1537	154 053	1279	80
MSPS_M3_16J4_1	745.8	469.6	61.1	1 411 173	745 902	60 954	641 905	35 281	151
MSPS_M3_16J4_10	188.2	20.9	4.5	251 892	4108	791	127 872	795	49
MSPS_M3_16J4_11	744.0	153.8	16.2	1 498 201	176 372	6418	658 155	23 631	70
MSPS_M3_16J4_12	1430.1	60.3	9.7	2 589 118	55 353	1912	1 305 023	21 638	183
MSPS_M3_16J4_13	496.1	48.4	11.3	888 909	42 583	2131	380 819	16 082	51
MSPS_M3_16J4_14	2366.7	134.8	26.1	4 102 236	168 881	4468	1 674 821	56 647	143
MSPS_M3_16J4_15	700.9	64.4	18.5	1 399 138	80 900	4990	727 886	29 817	75
MSPS_M3_16J4_16	152.8	15.9	2.1	225 335	9492	444	100 422	3857	89
MSPS_M3_16J4_17	2961.8	127.8	19.5	5 085 046	140 884	3072	2 564 053	63 479	80
MSPS_M3_16J4_18	1772.2	312.1	11.6	2 972 314	493 631	2410	1 478 175	242 596	294
MSPS_M3_16J4_19	2382.8	241.4	24.0	4 312 101	326 184	5301	2 146 882	138 837	293
MSPS_M3_16J4_2	2367.6	495.0	12.7	3 962 974	688 217	2092	1 934 204	312 764	222
MSPS_M3_16J4_20	715.1	20.4	4.5	1 302 010	8942	1231	618 990	2650	58
MSPS_M3_16J4_21	635.2	36.9	11.0	1 066 923	30 551	1595	500 595	6964	107
MSPS_M3_16J4_22	960.4	100.4	20.5	1 883 006	115 854	4575	942 525	41 231	127
MSPS_M3_16J4_23	503.5	36.5	8.9	991 814	21 650	2049	492 281	6835	56
MSPS_M3_16J4_24	1031.4	38.0	14.3	1 833 236	24 079	2076	906 884	8125	100
MSPS_M3_16J4_25	620.2	28.9	2.7	1 053 714	11 180	724	492 034	3310	50
MSPS_M3_16J4_26	761.2	42.1	15.9	1 319 166	15 211	3147	688 770	2945	76
MSPS_M3_16J4_27	561.8	45.2	12.2	1 153 721	39 555	2459	590 868	1918	53
MSPS_M3_16J4_28	872.2	72.7	25.0	1 813 662	62 549	11 210	743 536	16 238	46
MSPS_M3_16J4_29	232.7	16.9	2.1	382 419	4349	303	183 254	907	56
MSPS_M3_16J4_3	1513.7	84.0	13.4	2 736 777	57 898	2261	1 235 881	6475	80
MSPS_M3_16J4_30	1111.2	160.7	15.5	2 235 635	232 802	4278	1 104 867	97 622	294
MSPS_M3_16J4_31	131.4	24.2	2.9	198 873	15 556	880	82 820	1593	42
MSPS_M3_16J4_32	230.8	19.5	4.6	341 899	7219	956	177 159	2148	58
MSPS_M3_16J4_33	614.9	49.4	9.2	1 179 530	34 978	1818	579 800	4883	91
MSPS_M3_16J4_34	629.9	22.4	7.3	1 030 701	7275	902	488 129	2561	219
MSPS_M3_16J4_35	951.8	341.1	23.9	1 819 960	520 265	6926	841 374	117 639	72
MSPS_M3_16J4_36	905.1	41.8	8.0	1 571 586	31 310	1702	702 296	6251	168
MSPS_M3_16J4_37	349.5	57.2	9.5	622 188	63 987	1958	318 018	22 120	54
MSPS_M3_16J4_38	2069.0	152.6	15.0	3 325 297	179 661	2816	1 631 977	78 653	161
MSPS_M3_16J4_39	93.6	17.0	2.7	136 274	8520	385	58 457	3638	67
MSPS_M3_16J4_4	1348.2	37.1	14.5	2 553 764	21 753	3077	1 250 382	6591	119
MSPS_M3_16J4_5	1436.2	488.1	30.2	2 852 482	662 332	9764	1 361 762	287 749	163
MSPS_M3_16J4_6	1410.5	32.1	9.5	2 212 063	16 124	1488	1 101 449	6457	90
MSPS_M3_16J4_7	776.2	283.2	31.6	1 477 659	410 993	14 588	700 793	54 494	98
MSPS_M3_16J4_8	1241.8	178.3	20.9	2 525 524	254 203	7324	1 244 443	93 452	135
MSPS_M3_16J4_9	1230.6	337.6	20.9	2 462 718	514 077	4626	1 203 918	203 483	75
MSPS_M3_19J4_10	4482.8	283.2	14.2	5 458 847	284 089	5499	2 021 252	32 560	109
MSPS_M3_19J4_16	6113.0	24.9	8.8	6 813 189	8062	1084	2 820 143	1536	59
MSPS_M3_19J4_29	5236.9	45.2	14.5	6 664 102	20 042	2634	2 986 595	4093	87
MSPS_M3_19J4_31	3745.1	150.9	74.4	5 047 269	120 995	34 202	1 805 159	5402	140
MSPS_M3_19J4_32	5506.0	135.0	15.0	6 776 613	101 483	2179	3 215 502	36 872	90
mean (45 inst.)	912.3	72.7	12.0	1 550 407.5	55 351.8	2649.2	726 246.4	13 587.5	95.6