INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 28 (2021) 2633–2690
DOI: 10.1111/itor.12911

Selective arc-ng pricing for vehicle routing

Luciano Costa[a], Claudio Contardo[b,*], Guy Desaulniers[a] and Diego Pecin[c]

[a] *Department of Applied Mathematics and Industrial Engineering & GERAD, Polytechnique Montréal, Montréal, Québec, Canada H3C 3A7*

[b] *Department of Analytics, Operations and Information Technologies, ESG UQAM, CIRRELT & GERAD, Montréal, Québec, Canada H2X 3X2*

[c] *Department of Econometrics, Erasmus School of Economics (ESE), Erasmus University Rotterdam, Rotterdam 3000 DR, The Netherlands*

E-mail: luciano.costa@polymtl.ca [Costa]; claudio.contardo@gerad.ca [Contardo]; guy.desaulniers@gerad.ca [Desaulniers]; galindopecin@ese.eur.nl [Pecin]

Received 24 January 2020; received in revised form 14 November 2020; accepted 16 November 2020

Abstract

Column generation algorithms for solving vehicle routing problems often rely on a relaxed pricing subproblem where routes may be nonelementary and which is solved by a labeling algorithm. This pricing algorithm is said to be selective if it can discard nonelementary paths that may be Pareto-optimal but guarantees finding at least one (not necessarily elementary) path of negative reduced cost when there exists at least one elementary path of negative reduced cost. In this paper, we propose a selective pricing mechanism for a recently introduced variant of the *ng*-route relaxation, in which the neighborhoods are associated with arcs instead of nodes. We extend a state-of-the-art set-based dominance criterion for this problem and show, by means of an exhaustive computational campaign, that the resulting mechanism is effective at reducing the number of nondominated labels as compared to the original pricing algorithm for the same problem. As a result, typically shorter computing times are required to compute similar lower bounds when the proposed mechanism is embedded within a column generation solver.

Keywords: column generation; shortest path problem with resource constraints; selective pricing; vehicle routing

1. Introduction

The vehicle routing problem (VRP) is a classic optimization problem arising in many applications in logistics. Since the seminal work of Dantzig and Ramser (1959), many researchers and practitioners have attempted to tackle this problem as efficiently as possible. Unfortunately, solving the VRP is $\mathcal{NP}$-hard as the traveling salesman problem (TSP, Dantzig et al., 1954) can be reduced to a

* Corresponding author.

VRP in polynomial time. Over the years, several variants of the VRP have been proposed to model the different attributes associated with real-life problems, namely: heterogeneous fleet, multiple depots, combined pickup and delivery, split services, and vehicles with special capabilities (e.g., electric vehicles), among others. For recent compendiums and a broad view of the recent trends in vehicle routing applications, the reader is referred to Toth and Vigo (2014) and Vidal et al. (2020).

Nowadays, the dominant methodology for solving many variants of the VRP is branch-price-and-cut applied to a set-partitioning-type formulation (Costa et al., 2019), which can be described generically as follows. Let N^+ be a set of customers that need to be serviced by exactly one vehicle each. Let Ω be the set of feasible vehicle routes, where the feasibility of a route depends on the VRP variant considered. With each route $r \in \Omega$, associate a cost c_r and, for each customer $i \in N^+$, a binary parameter a_i^r that indicates whether customer i is visited in route r. Furthermore, define a binary variable ω_r, which is equal to 1 if route r is selected in the solution, or 0 otherwise. With this notation, a pure set-partitioning formulation for a VRP is as follows:

$$\min \sum_{r \in \Omega} c_r \omega_r \tag{1}$$

$$\text{s.t.} \sum_{r \in \Omega} a_i^r \omega_r = 1 \qquad \forall i \in N^+ \tag{2}$$

$$\omega_r \in \{0, 1\} \qquad \forall r \in \Omega. \tag{3}$$

The objective function (1) aims at minimizing the total routing cost. Constraints (2) impose a single visit to each customer $i \in N^+$. Finally, constraints (3) define the domain of the variables. For certain problem variants, additional constraints and variables may be required, yielding a set-partitioning-type model. Note also that, under certain conditions, models (1) and (2) can be replaced by a set-covering model, obtained by replacing the equality in (2) by a greater-than-or-equal relation.

Formulations like (1)–(3) have an exponential number of variables with respect to the number of customers. Handling these variables all at once may be computationally prohibitive. This drawback can be overcome using branch-and-price, that is a branch-and-bound algorithm applying column generation to solve the linear relaxations encountered throughout the search tree. Column generation is an iterative algorithm that solves at each iteration a restricted master problem (RMP) and a pricing subproblem. The RMP is defined as the linear relaxation of (1)–(3) considering only a subset of its variables. It is solved by a linear programming solver to yield a pair of primal and dual solutions. The pricing subproblem looks for negative reduced cost variables to add to the RMP. For most VRPs, it is cast as an elementary shortest path problem with resource constraints (ESPPRC) defined over an application-dependent network and solved by a labeling algorithm. When negative reduced cost columns are identified, they are added to the RMP before starting a new iteration. Otherwise, the column generation algorithm stops and the current RMP solution yields a lower bound. To strengthen the linear relaxations, cutting planes can be added dynamically to yield a branch-price-and-cut algorithm.

As mentioned above, the pricing subproblem is often an ESPPRC, which is strongly $\mathcal{NP}$-hard (Dror, 1994). At the opposite, the subproblem arising from completely dropping the elementarity constraints, called the shortest path problem with resource constraints (SPPRC), is solvable

in pseudo-polynomial time (Desrosiers et al., 1984), but proves to be weak when the resource constraints are not sufficiently tight. To overcome this issue, several authors have focused their attention on strengthening the bounds by partially imposing path elementarity. Two main frameworks can be found in the literature to deal with the issue of partially addressing the elementarity constraints in an efficient manner: decremental state-space relaxation (DSSR, Boland et al., 2006; Righini and Salani, 2006); and cycle elimination (Desrochers et al., 1992; Irnich and Villeneuve, 2006; Baldacci et al., 2011; Bulhões et al., 2018).

In DSSR, the elementarity constraints are imposed gradually. Starting from a pure SPPRC, the path with the least reduced cost is checked for feasibility. If it is, an optimal solution has been found; otherwise, a cycle is detected and the elementarity requirements are strengthened for the subsequent iterations. Cycle elimination techniques, in turn, are closely related to the notion of route relaxations. In the seminal article of Desrochers et al. (1992), the authors presented the first successful application of column generation for VRPs. They considered the vehicle routing problem with times windows (VRPTW, Desaulniers et al., 2014) and solved by branch-and-price. The authors recognized the difficulty of the underlying pricing subproblem and described for the first time the SPPRC without cycles of length 2 (*2-cyc*-SPPRC). Unlike the ESPPRC, this problem can be solved in pseudo-polynomial time. Later, Irnich and Villeneuve (2006) studied the *k-cyc*-SPPRC, for values of k that range between 3 and 5, which provides bounds that are much stronger than those obtained by the *2-cyc*-SPPRC, yet, in longer computing times. Baldacci et al. (2011) introduced the *ng*-route relaxation (*ng*-SPPRC), which relies on a memory mechanism to remember nodes that have been visited recently within a route. Thus, a feasible path may still contain cycles, but those cycles should be long and have a limited negative impact on the resulting bounds. Some hybrid strategies have been proposed by Martinelli et al. (2014), who successfully achieved improved results by adapting DSSR to tackle the *ng*-SPPRC, and Contardo et al. (2015), who showed that elementary paths can be obtained by imposing partial elementarity in an escalated way within a column-generation-based solver for routing problems. More recently, Bulhões et al. (2018) proposed the *arc-ng*-SPPRC, an extension of the classical *ng*-SPPRC, in which the neighborhoods are associated with arcs instead of nodes. Arc-based neighborhoods allow better management of the undesirable cycles, although, when embedded into a DSSR framework, they usually induce many more iterations compared to a traditional node-based *ng*-SPPRC. Moreover, the authors also devised a novel set-based dominance mechanism that enables to anticipate that certain label extensions may become dominated without the need of performing them.

Desaulniers et al. (2019) introduced the concept of *selective* pricing when the elementarity constraints are dropped at least partially. A pricing algorithm (associated with a column generation pricing subproblem) is said to be *selective* if it can discard nonelementary paths that may be Pareto-optimal but guarantees finding at least one path (possibly with cycles) of negative reduced cost when there exists at least one elementary path with a negative reduced cost. Such an algorithm can stop when it proves that no such elementary paths exist, even if there are still some nonelementary paths with a negative reduced cost. A selective pricing strategy can be used to derive valid dual bounds, and those bounds have the potential of being stronger than those that would be achieved using a nonselective pricing strategy. Yet, Desaulniers et al. (2019) pointed out that the gains in the bounds are only theoretical. They observed in their experiments that the differences between the bounds produced by the selective algorithm and the standard pricing algorithm are not significant. The authors illustrate the selective pricing

strategy on the *ng*-SPPRC, but the algorithm is quite demanding and requires the addition of several data structures and sub-procedures, whose overhead is not always beneficial in the long run.

In this article, we extend the set-based dominance criterion of Bulhões et al. (2018) by making it selective. Unlike the original approach of using additional data structures to keep track of the non-dominated labels (Desaulniers et al., 2019), the proposed mechanism requires minimal modifications of the labeling algorithm and is therefore easy to implement. When applied to *arc-ng*-SPPRC, the proposed selective pricing strategy proves to be faster in practice than the original (nonselective) *arc-ng*-SPPRC when embedded within a column generation algorithm for solving benchmark instances of the VRPTW.

The remainder of this article is organized as follows: Section 2 presents some of the main route relaxations proposed in the literature, with an emphasis on the *arc-ng*-SPPRC. In Section 3, we describe the selective *arc-ng*-SPPRC, provide some theoretical proofs of its correctness, and discuss how this new framework extends the traditional *arc-ng*-SPPRC. In Section 4, we report the results of our computational experiments to show the impact of considering the new framework when tackling some hard VRPTW instances. Finally, some conclusions are drawn in Section 5.

2. Route relaxations

As discussed in the previous section, the natural way of modeling pricing subproblems arising from most VRP applications is by considering an ESPPRC. However, due to the high computational complexity of this problem, route relaxations have been preferred to design efficient column-generation-based algorithms. In this section, we discuss some of the main route relaxations proposed in the literature, namely: the SPPRC, which is obtained when all elementarity requirements are relaxed; and the *ng*-SPPRC and *arc-ng*-SPPRC, which tend to avoid the formation of cycles containing nodes that might be located close to each other. To simplify the exposition, we consider that the pricing subproblem is defined over a digraph $G = (N, A)$, where $N = \{0, \ldots, n, n+1\}$ is its set of nodes and $A \subseteq N \times N$ its set of arcs. Nodes 0 (source) and $n+1$ (sink) represent the origin and the destination of a path, respectively, whereas $N^+ = N \setminus \{0, n+1\}$ is the set of customer nodes. The following discussion can be easily extended to other networks.

2.1. The SPPRC

The SPPRC can be formally described as follows. Consider network G and let $\mathcal{R}$ be a set of resources, which are used to model some route feasibility rules for the problem at hand, such as vehicle capacity, time windows, or precedence, among others (Irnich and Desaulniers, 2005). The objective of the SPPRC is to find a least-cost path, not necessarily elementary, from node 0 to node $n+1$, in such a way that the path respects the following resource constraints.

With each node $v \in N$ and resource $r \in \mathcal{R}$, define resource windows $[e_v^r, l_v^r]$, with $0 \leq e_v^r \leq l_v^r$, representing the allowed limits on the consumption of each resource r by a path ending at node v. Specifically, it is assumed that $l_0^r = 0$ and $l_{n+1}^r = H^r$, where H^r represents the maximum consumption allowed for resource r when arriving at the sink.

With each arc $(u, v) \in A$, we associate a modified routing cost $\bar{c}_{uv} \in \mathbb{R}$ that depends on the RMP dual values such that the sum of the modified costs of the arcs composing a path between nodes 0 and $n+1$ is equal to the reduced cost of the corresponding route variable. For example, if we denote by α_i, $i \in N^+$, the dual variables associated with constraints (2), then the modified cost of arc $(u, v) \in A$ is given by

$$\bar{c}_{uv} = \begin{cases} c_{uv} - \alpha_u & \text{if } u \in N^+ \\ c_{uv} & \text{otherwise (i.e., } u = 0), \end{cases} \tag{4}$$

where c_{uv} is the routing cost along arc (u, v). With each arc (u, v), we also associate a vector of resource consumptions $(\gamma^r_{uv})_{r\in\mathcal{R}} \in \mathbb{N}_+^{|\mathcal{R}|}$ (e.g., travel time and load delivered). In many classical SPPRC variants, it is assumed that, if a path ending at node v uses less than e^r_v for some resource $r \in \mathcal{R}$, it is still possible to consume the extra amount of resource, so as to respect the lower bound e^r_v. In other words, when a path is extended along an arc (u, v), assuming the resource consumptions at node u given by ξ^r_u, $\forall r \in \mathcal{R}$, they are updated at node v with the resource extension functions $\xi^r_v \leftarrow \max\{e^r_v, \xi^r_u + \gamma^r_{uv}\}$, $\forall r \in \mathcal{R}$. The extension is deemed *resource-feasible* (i.e., it satisfies the resource constraints) if $e^r_v \le \xi^r_v \le l^r_v$, $\forall r \in \mathcal{R}$, otherwise it is deemed infeasible. Note that, depending on the problem at hand, the resources may vary along the paths according to more complex resource extension functions (Irnich, 2008). For the sake of simplicity and without loss of generality, we limit our discussion to the above resource extension functions.

2.2. The ng-SPPRC of Baldacci et al. (2011)

Currently, the state-of-the-art—and widely adopted—framework to eliminate cycles is the *ng*-route relaxation (*ng*-SPPRC) proposed by Baldacci et al. (2011). The *ng*-SPPRC differs from the *2-cyc*-SPPRC and *k-cyc*-SPPRC because the notion of length of a cycle does not depend on the number of arcs (or edges) traversed along the cycle, but rather on the notion of proximity between the nodes that belong to it. In the *ng*-SPPRC, cycles are forbidden whenever they contain customers that are close to each other. For every customer $v \in N^+$, one considers a *ng*-set (neighborhood) $\mathcal{N}_v \subseteq N^+$. Because the goal is to prevent short cycles, it is common to define the *ng*-set $\mathcal{N}_v$ as containing the Δ geographically closest customers to v, and the vertex v itself, where $\Delta > 0$ is a predefined parameter. Let $P = \{v_1, \ldots, v_h = v, \ldots, v_{h'} = v, \ldots, v_\ell\}$ be a path generated while solving the *ng*-SPPRC. The cycle $H = \{v_h = v, \ldots, v_{h'} = v\}$ is feasible if and only if $v \notin \cap_{i=h+1}^{h'-1} \mathcal{N}_{v_i}$, that is, v does not belong to the *ng*-set of at least one intermediate node in the cycle. The theoretical complexity of the *ng*-SPPRC is also pseudo-polynomial if $|\mathcal{N}_v| \le \Delta + 1$ for every $v \in N^+$. In this case, a labeling algorithm can solve the *ng*-SPPRC in pseudo-polynomial time with a multiplying constant that depends on $2^{2\Delta}$. The *ng*-SPPRC normally provides much tighter bounds than those provided by the *k-cyc*-SPPRC in comparable CPU times. In fact, experiments carried out by Poggi and Uchoa (2014) using some hard capacitated VRP instances have shown that *ng*-sets with $\Delta = 8$ yield bounds similar to those obtained by the *5-cyc*-SPPRC, but spending a time comparable to the *4-cyc*-SPPRC. For this reason, the *ng*-SPPRC has become the route relaxation of preference ever since.

2.3. The arc-ng-SPPRC of Bulhões et al. (2018)

In a recent article by Bulhões et al. (2018), the *ng*-SPPRC is extended to consider neighborhoods associated with arcs instead of nodes. We call this problem the *ng*-SPPRC with arc-based neighborhoods, and denote it *arc-ng*-SPPRC. The *arc-ng*-SPPRC can be defined as follows. With every arc $a = (u, v) \in A$, we associate a *ng*-set $\mathcal{N}_a \subseteq N^+$ that corresponds to the set of customer nodes that can be remembered by arc a. The *ng*-sets $\mathcal{N}_a$ may be defined according to any suitable criterion. Most successful implementations use some notion of distance to favor neighborhoods containing nodes that are close to each other. An example is to consider the set of nodes that are close to both extremities of the arcs, that is, $\mathcal{N}_a \subseteq \mathcal{N}_u \cap \mathcal{N}_v$, with $\mathcal{N}_u$ and $\mathcal{N}_v$ defined as for the *ng*-SPPRC. Let $P = (a_0, a_1, \ldots, a_p)$ be a partial path, composed of arcs $a_0 = (0, v_1)$, $a_1 = (v_1, v_2)$, $\ldots$, $a_p = (v_p, v_{p+1})$ and $\Pi(v_p)$ be the memory of this path upon arrival at node v_p, which is a subset of nodes. An extension using arc $a = (v_p, v_{p+1})$ is deemed *arc-ng-feasible* only if $v_{p+1} \notin \Pi(v_p)$. The memory upon arrival to v_{p+1} is updated according to the formula $\Pi(v_{p+1}) \leftarrow (\Pi(v_p) \cap \mathcal{N}_a) \cup \{v_{p+1}\}$. If $\mathcal{N}_a = \mathcal{N}_u \cap \mathcal{N}_v$, the *arc-ng*-SPPRC coincides with the classical *ng*-SPPRC, and, therefore, it is more general.

By considering the notions presented in Section 2.1, a path $P' = (v_1 = 0, \ldots, v_{p+1} = n + 1)$ is said to be feasible only if every prefix $P'_i = (v_1, \ldots, v_i)$, $i \in \{2, \ldots, p + 1\}$ is both resource- and *ng*-feasible. The cost of P' is given by $c(P') = \sum_{1 \leq q \leq p} c_{v_q, v_{q+1}}$. The *arc-ng*-SPPRC consists of finding a feasible path P' of minimum total cost. Not only this mechanism allows a better management of the undesired cycles, but also yields a sharper dominance rule.

In trying to enhance the performance of their algorithm, Bulhões et al. (2018) proposed a novel set-based dominance criterion (multiple dominance) that extends the traditional dominance, which only allows pairwise comparisons to be made. This new strategy was embedded into the algorithm used to solve the *arc-ng*-SPPRC. We provide further details about this new dominance relation in Section 2.3.2.

2.3.1. Labeling algorithm

A general description of a labeling algorithm similar to the one adopted by Bulhões et al. (2018) to solve the *arc-ng*-SPPRC is as follows. A label L encodes a partial path $(v_1 = 0, \ldots, v_p)$ from the source until reaching a node v_p through the following data structures: a terminal node $v(L) = v_p$; resource consumptions $\xi^r(L)$, $r \in \mathcal{R}$; a (reduced) cost $\bar{c}(L)$; a memory $\Pi(L)$; and a pointer $pred(L)$, which indicates the predecessor of label L'.

An extension to a node w over an arc $a = (v_p, w)$ yields a label L' with the associated data structures computed as follows:

$$v(L') \leftarrow w \tag{5}$$

$$\xi^r(L') \leftarrow \max\{e^r_w, \xi^r(L) + \gamma^r_a\} \qquad \forall r \in \mathcal{R} \tag{6}$$

$$\bar{c}(L') \leftarrow \bar{c}(L) + \bar{c}_{v_p w} \tag{7}$$

$$\Pi(L') \leftarrow (\Pi(L) \cap \mathcal{N}_a) \cup \{w\} \tag{8}$$

$$pred(L') \leftarrow L. \tag{9}$$

Algorithm 1. Labeling algorithm

1: $\mathcal{V} \leftarrow \{L_0\}$, $L_{best} \leftarrow nil$, $\bar{c}_{min} \leftarrow +\infty$,
2: $\mathcal{B}(v, \xi^{r^*}) \leftarrow \emptyset, \forall v \in N, \forall \xi^{r^*} \in \{0, 1, \ldots, H^{r^*}\}$
3: **while** $\mathcal{V} \neq \emptyset$ **do**
4: Let L be the first label in $\mathcal{V}$
5: Set $\mathcal{V} \leftarrow \mathcal{V} \setminus \{L\}$
6: Set $\mathcal{B}(v(L), \xi^{r^*}(L)) \leftarrow \mathcal{B}(v(L), \xi^{r^*}(L)) \cup \{L\}$
7: **for all** $a \in \mathcal{A}(L)$ **do**
8: Let L' be the label resulting from extending L along arc a using (5)–(9)
9: **if** $v(L') = n + 1$ **then**
10: **if** $\bar{c}(L') < \bar{c}_{min}$ **then**
11: Set $L_{best} \leftarrow L'$, $\bar{c}_{min} \leftarrow \bar{c}(L')$
12: **else if** there exists no label $L'' \in \mathcal{B}(v(L'), \xi^{r^*}(L'))$ such that $L'' \prec L'$ **then**
13: Set $\mathcal{B}(v(L'), \xi^{r^*}(L')) \leftarrow \mathcal{B}(v(L'), \xi^{r^*}(L')) \cup \{L'\}$
14: Remove from $\mathcal{V}$ all labels L'' such that $L' \prec L''$
15: **return** L_{best}

Label L' is deemed feasible if and only if it is both resource- and *ng*-feasible. For the sake of conciseness, we denote from now on an extension of label L to a node w by $L \oplus w$.

A generic representation of a labeling algorithm is outlined in Algorithm 1. We point out that this pseudocode is only used for illustration purposes. We do not intend to provide an exact description of Bulhões et al.'s (2018) algorithm. Several implementation details, among which bi-directional search (Righini and Salani, 2006) and acceleration tricks were intentionally left out. The interested reader is referred to Bulhões et al. (2018) and Costa et al. (2019) for further details on the implementation of labeling algorithms, in particular of the *arc-ng*-SPPRC. Similarly to what is done in most of the recent implementations (e.g., Pecin et al., 2017a,b), Bulhões et al. adopt label buckets to store nondominated labels. This data structure allows a more efficient application of dominance rules. For each node $v \in N$ and possible consumption ξ^{r^*} of a given resource $r^* \in \mathcal{R}$, one defines a bucket $\mathcal{B}(v, \xi^{r^*})$ that stores all the nondominated labels associated with node v and having consumption ξ^{r^*} for resource r^*. It is initialized as empty and enlarged dynamically. The choice for the resource defining the buckets may vary according to the problem at hand (e.g., load or time). In Algorithm 1, the set $\mathcal{V}$ keeps the unextended labels waiting in a queue. This set is kept sorted at all times in lexicographic order with regard to the consumption of resource r^*, that is, the labels with lowest consumptions of r^* are positioned before those with larger consumptions (Desrochers and Soumis, 1988). As mentioned before, the maximum consumption allowed for resource r^* is H^{r^*}. The set $\mathcal{A}(L) = \{a = (u, v) \in A : u = v(L), v \in N \setminus \Pi(L)$ and $L \oplus v$ is resource-feasible$\}$ contains the arcs that can be used to extend L. For two labels L and L', $L \prec L'$ means that L dominates L' (according to the rules defined in Sections 2.3.2 and 3). Let L_0 be the label encoding an empty path of cost zero, with zero resource consumptions, empty memory, and such that $pred(L_0) = nil$. Algorithm 1 returns the label L_{best} representing a path with lowest cost. This labeling algorithm extends nondominated labels to create new labels (line 8), coupled with a dominance rule to discard nonpromising labels (lines 12 and 14). The dominance procedure prevents a combinatorial explosion and, therefore, keeps the computational burden within reasonable limits.

2.3.2. Multiple partial label dominance

Dominance in Bulhões et al. (2018) is not only performed pairwise. The authors use a sharper *multiple partial label dominance rule* similar to that introduced by Irnich and Villeneuve (2006) for the *k-cyc*-SPPRC. To formalize this idea in the context of the *arc-ng*-SPPRC, we use the notion of *weak* and *strong* dominance described in Irnich and Villeneuve (2006). Let L and L' be two feasible labels. Label L is said to *weakly dominate* L' if all the following conditions hold:

$$v(L) = v(L'), \tag{10}$$

$$\xi^r(L) \leq \xi^r(L'), \qquad \forall r \in \mathcal{R} \tag{11}$$

$$\bar{c}(L) \leq \bar{c}(L'). \tag{12}$$

Conditions (10)–(12), however, do not ensure a proper dominance of L' by L. In fact, depending on the label memories $\Pi(L)$ and $\Pi(L')$, it may still be possible for L' to be extended to a node w that would not be an *ng*-feasible extension of L. This would happen for every node $w \in \Pi(L) \setminus \Pi(L')$. The additional condition

$$\Pi(L) \subseteq \Pi(L') \tag{13}$$

ensures the correctness of the pairwise dominance.

Bulhões et al. (2018) realized that condition (13) may be unnecessarily too restrictive. In fact, if a label L weakly dominates another label L', one may restrict the extensions of L' only to the nodes $w \in \Pi(L) \setminus \Pi(L') \cup \{j \in N \setminus \Pi(L') \mid \mathcal{N}_{(v(L),j)} \cap (\Pi(L) \setminus \Pi(L')) \neq \emptyset\}$, that is, the nodes where L' can be directly extended but not L or the nodes that remember at least one such node. This brings out the following set-based dominance criterion. Let $\mathcal{L}$ be a collection of labels $(L_i)_{i \in \mathcal{L}}$. It is said to *strongly dominate* a label $L' \notin \mathcal{L}$ if the two following conditions hold:

i. L_i weakly dominates L' for all $i \in \mathcal{L}$.
ii. For every *ng*-feasible extension of L' to a node w, there exists $i \in \mathcal{L}$ such that L_i can also be extended to w and $\Pi(L_i \oplus w) \subseteq \Pi(L' \oplus w)$.

Conditions (i) and (ii) combined assure that any path obtained by feasibly extending label L' would be dominated. Hence, L' can be discarded. Moreover, the authors also demonstrate that, whenever a weakly dominating label L_i is identified, one can immediately restrict feasible extensions of L', by discarding those that would be dominated by path $L_i \oplus w$ for all $w \in N \setminus \Pi(L_i)$ such that $L_i \oplus w$ is resource-feasible. As a result, not only full dominance reduces the number of labels, but also partial dominance contributes to avoiding certain extensions. On the other hand, their computational experiments showed that it may be computationally expensive to store the set of dominated extensions for each label. Consequently, as a compromise, instead of representing this set explicitly, they rely on an implicit representation that requires to only extend the memory $\Pi(L')$ of a dominated label L' along a dominated extension, thus, avoiding the extension of the label cost and resource components.

In a framework similar to Algorithm 1, partial dominance using implicit representation of dominated extensions is applied as follows. Let $L, L' \in \mathcal{B}(u, \xi^{r^*})$ be two labels for which conditions (10)–(12) hold, but not (13). When L, L' were extended over the same arc (u, v), resource

consumptions and the reduced cost for both labels increase by the same amount (assuming that nonrobust cuts are not employed). Consequently, (10)–(12) remain valid for the resulting labels $L \oplus v$ and $L' \oplus v$. Hence, the only condition that needs to be verified to determine if $L \oplus v$ dominates $L' \oplus v$ is (13). To apply partial dominance using implicit representation, one assigns to each bucket $\mathcal{B}(u, \xi^{r^*})$, $\forall u \in N$, $\forall \xi^{r^*} \in \{0, 1, \ldots, H^{r^*}\}$, and arc $(u, v) \in A$, a list for storing the sets $\Pi(\cdot)$ associated with labels in $\mathcal{B}(u, \xi^{r^*})$ that have been extended over (u, v). In the case of the label L' mentioned above, to avoid its complete extension over (u, v), it suffices to check if $\Pi(L' \oplus v)$ is a subset of some set $\Pi(\cdot)$ stored in the list of memories associated with $\mathcal{B}(u, \xi^{r^*})$ and (u, v), and which was obtained from a label L such that (10)–(12) hold. If it is the case, the extension can be discarded. Note that, as in Bulhões et al. (2018), when there is a single resource in $\mathcal{R}$ and the labels in a bucket are kept in a nonincreasing order of their reduced cost, all stored sets $\Pi(\cdot)$ are obtained from labels L such that (10)–(12) hold. This is, however, not the case when there are multiple resources in $\mathcal{R}$.

When $|\mathcal{R}| > 1$, it may be necessary to adopt a more sophisticated data structure to verify (10)–(12) efficiently. For example, when $|\mathcal{R}| = 2$ as for the VRPTW, lists of sets $\Pi(\cdot)$ can be stored for each possible value of the resource $r \neq r^*$. For larger values of $|\mathcal{R}|$, one may need to consider additional structures to check when dominance relations for the resources that are not used in the definition of the bucket structure are respected. The overhead caused by the extra tests may impact negatively on the overall performance of the algorithm.

3. Selective *arc-ng*-SPPRC

In this section, we redefine strong dominance. For the sake of readability, we present the mechanism in two steps. First of all, we show that the pairwise comparison criterion (13) can be replaced by

$$\Pi(L) \subseteq \Pi(pred(L')) \cup \{v(L')\}. \tag{14}$$

Proposition 1 below formalizes the correctness of this criterion. Its proof relies on the following lemma.

Lemma 1. *A labeling algorithm using conditions (10)–(12), (14) to establish pairwise dominance produces, for every label E representing an elementary partial path P, at least one nondominated label L such that $v(L) = v(E)$, $\xi^r(L) \leq \xi^r(E)$, $\forall r \in R$, $\bar{c}(L) \leq \bar{c}(E)$, and $\Pi(L) \subseteq V(E)$, where $V(E)$ denotes the set of customer nodes visited by P.*

Proof. The proof works by induction on the size p of $P = \{v_1 = 0, \ldots, v_p\}$. If $p = 1$, then E is nondominated and $L = E$. If $p \geq 2$, let us assume that E is obtained by extending another label E' over an arc $(v_{p-1}, v_p) \in A$, with $v_{p-1} = v(E')$ and $v_p \notin V(E')$. Using the induction hypothesis, the labeling algorithm generates a nondominated label L' such that $v(L') = v(E')$, $\xi^r(L') \leq \xi^r(E')$, $\forall r \in R$, $c(L') \leq c(E')$, $\Pi(L') \subseteq V(E')$. Denote by L'' the label produced by extending L' along arc (v_{p-1}, v_p). It is easy to see that L'' is a feasible extension of L', and that $v(L'') = v(E)$, $c(L'') \leq c(E)$ and $\xi^r(L'') \leq \xi^r(E)$, $\forall r \in R$, hold. In addition, $\Pi(L'') \subseteq \Pi(L') \cup \{v_p\} \subseteq V(E') \cup \{v_p\} = V(E)$. If L'' is not dominated, then $L = L''$. Otherwise, if L'' is dominated by a nondominated label L''', then L''' satisfies $v(L''') = v(E)$, $c(L''') \leq c(L'') \leq c(E)$, $\xi^r(L''') \leq \xi^r(L'') \leq \xi^r(E)$, $\forall r \in R$, and $\Pi(L''') \subseteq \Pi(pred(L'')) \cup \{v_p\} = \Pi(L') \cup \{v_p\} \subseteq V(E') \cup \{v_p\} = V(E)$. In this case, $L = L'''$. □

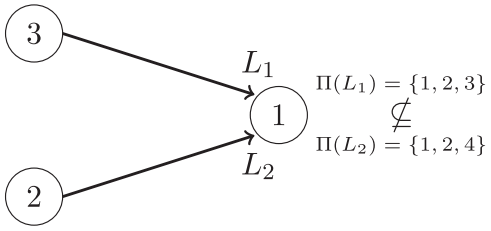

Fig. 1. Standard dominance.

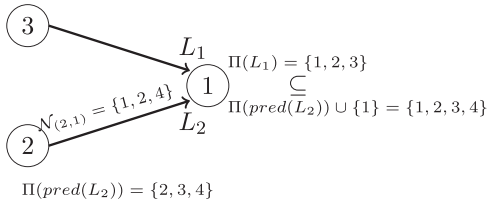

Fig. 2. Selective dominance.

From this lemma, we deduce the following result.

Proposition 1. *If there exists an elementary path P from* 0 *to* $n + 1$ *with a negative reduced cost, then a labeling algorithm based on the dominance rule (10)–(12) and (14) finds at least one (possibly nonelementary) path* P' *from* 0 *to* $n + 1$ *with a negative reduced cost.*

Proof. Assume that there exists such a path P which is represented by a label E. If E is not dominated, then $P' = P$. Otherwise, according to Lemma 1, there exists a nondominated label L such that $v(L) = v(E)$ and $\bar{c}(L) \leq \bar{c}(E)$. This label represents path P'. □

To illustrate the strength of this new dominance rule, let us consider the following example. Let L_1 and L_2 be two labels such that $v(L_1) = v(L_2) = 1$, $c(L_1) \leq c(L_2)$, $\xi^r(L_1) \leq \xi^r(L_2)$, $\forall r \in R$, and let us assume that $\Pi(L_1) = \{1, 2, 3\}$ and $\Pi(L_2) = \{1, 2, 4\}$. Although L_1 weakly dominates L_2, the latter cannot be discarded according to (10)–(13) because $\Pi(L_1) \not\subseteq \Pi(L_2)$ (Fig. 1). Now, let us assume that label L_2 was extended from label $pred(L_2)$ with $v(pred(L_2)) = 2$, over the arc $a = (2, 1)$, and such that $\mathcal{N}_a = \{1, 2, 4\}$ and $\Pi(pred(L_2)) = \{2, 3, 4\}$. We now have that $\Pi(L_1) = \{1, 2, 3\} \subseteq \{1, 2, 3, 4\} = \Pi(pred(L_2)) \cup \{1\}$, and label L_2 can be safely discarded according to condition (14) (Fig. 2).

Observe that, in this example, it might be feasible to extend label L_2 to node 3 and that this extension might subsequently yield nondominated *ng*-paths between 0 and $n + 1$, possibly an optimal one. Nevertheless, when label L_2 is dominated according to conditions (10)–(12) and (14), this extension is not possible and the algorithm remains valid because the discarded *ng*-paths are not elementary (they visit node 3 more than once). The pricing is, thus, selective.

The second step in the presentation of our selective set-based dominance rule is stated in a corollary of the pairwise dominance condition, and extends the strong dominance criterion given by conditions (i) and (ii).

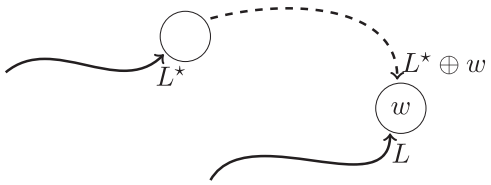

Fig. 3. Selective set-based comparison.

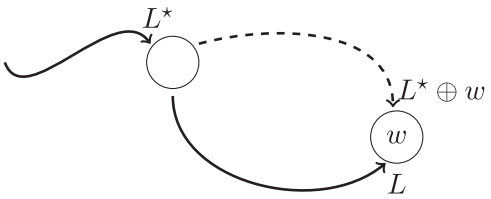

Fig. 4. Bulhões et al.'s mechanism.

Corollary 1. *For a given label $L^\star$, an extension to a node $w \in N^+ \setminus \Pi(L^\star)$ can be safely omitted if there exists a label L such that all the following conditions hold:*

$$v(L) = v(L^\star \oplus w) = w, \tag{15}$$

$$\bar{c}(L) \leq \bar{c}(L^\star \oplus w), \tag{16}$$

$$\xi^r(L) \leq \xi^r(L^\star \oplus w), \qquad \forall r \in R \tag{17}$$

$$\Pi(L) \subseteq \Pi(L^\star) \cup \{w\}. \tag{18}$$

In fact, one can observe that (15)–(18) correspond to (10)–(12) and (14) when setting $L' = L^\star \oplus w$, showing that, if L exists and $L^\star$ was extended along arc $(v(L^\star), w)$, the resulting label $L' = L^\star \oplus w$ would be dominated by L.

Similarly to the idea proposed by Bulhões et al. (2018), the dominance rule given by conditions (15)–(18) intends to consider some available information to identify dominated label extensions. Yet, the dominance rule described in Corollary 1 is stronger than the one proposed by Bulhões et al. (2018) because it uses the selection mechanism given by condition (18), which is stronger than condition (13) (with $L' = L^\star \oplus w$). Another difference between the two algorithms is that our set-based dominance rule does not assume L and $L^\star \oplus w$ to be extended from the same node (see Fig. 3), unlike that of Bulhões et al. (2018) (see Fig. 4). In these cases, label $L^\star \oplus w$ can be identified as dominated by L without extending the cost and resource components of $L^\star$ in the algorithm of Bulhões et al. (2018) and without extending the memory of $L^\star$ in our selective algorithm. Therefore, if the average memory size is much larger than the number of cost and resource components, the selective dominance rule proposed in our algorithm can yield an additional speedup.

One way to reinforce dominance relations (14) and (18) is by adding to the set Π of the dominating label, the unreachable nodes, that is, the nodes to which extending the path would not be resource-feasible (equivalently, it is possible to do the same for relation (13)). Assuming that the resource consumptions γ_{ij}^r, $(i, j) \in A$ satisfy the triangle inequality for each resource $r \in R$, the set

of unreachable nodes of a label L is defined as $\mathcal{U}(L) = \{w \in N^+ | \ \exists r \in R$ such that $\xi^r(L) + \gamma^r_{v(L),w} > l^r_w\}$. Given that L and $L^\star \oplus w$ cannot be feasibly extended to any node in $\mathcal{U}(L)$ and $\mathcal{U}(L^\star \oplus w)$, respectively, these sets can be included in the right-hand side of their respective dominance conditions (14) and (18), which can be rewritten as

$$\Pi(L) \subseteq \Pi(pred(L')) \cup \{v(L')\} \cup \mathcal{U}(L') \tag{19}$$

$$\Pi(L) \subseteq \Pi(L^\star) \cup \{w\} \cup \mathcal{U}(L^\star \oplus w). \tag{20}$$

To avoid computing the unreachable set of label $L^* \oplus w$ before testing dominance condition (20), the following slightly weaker condition can be used:

$$\Pi(L) \subseteq \Pi(L^\star) \cup \mathcal{U}(L^\star) \cup \{w\}. \tag{21}$$

Finally, we mention that, even if Lemma 1 is valid for the case when ng-sets are associated with nodes, adopting relation (14) to devise an algorithm to solve the traditional *ng*-SPPRC does not have any impact on the solution process. This issue is further discussed in Appendix A.

4. Computational experiments

To assess the effectiveness of the proposed selective dominance rules, we performed a series of computational experiments on well-known benchmark instances of the VRPTW that can be defined as follows. Consider the network $G = (N, A)$ described in Section 2.1, where N is the set of nodes, A is the set of arcs, and N^+ is the set of customer nodes. Each customer $i \in N^+$ is associated with a demand q_i, a service time s_i, and a time window $[e_i, l_i]$ within which the service must start. Moreover, we set $q_0 = s_0 = e_0 = q_{n+1} = s_{n+1} = e_{n+1} = 0$ and $l_0 = l_{n+1} = H$, where H is the planning horizon duration. Each arc $(i, j) \in A$ is associated with a routing cost c_{ij} and a traveling time t_{ij}. An unlimited fleet of homogeneous vehicles with capacity Q is available at a single depot. The VRPTW consists of finding feasible routes visiting all customers $i \in N^+$ exactly once such that the total routing cost is minimized. A route is deemed feasible if: it is elementary; all visited customers are served within their time windows; the total demand of these customers does not exceed the vehicle capacity Q. Furthermore, the cost of a route is computed as the sum of the costs of the arcs traversed by the route.

For the VRPTW, the pricing subproblem involves a set R of two resources, namely, time and load. At a node $i \in N$, the resource windows $[e^r_i, l^r_i]$, $r \in R$, are defined by their time window $[e_i, l_i]$ and by the load window $[0, Q]$. Along an arc $(i, j) \in A$, the resource consumptions γ^r_{ij}, $r \in R$, are given by $t_{ij} + s_i$ for the time resource and by q_j for the load resource. The choice of the VRPTW as the test problem comes from the fact that, despite being simple to understand, it remains challenging. This aspect allows us to assess the performance of our selective algorithm when combined with different enhancing techniques that are typically required to solve the VRPTW efficiently. We detail some of these techniques in Section 4.1.

In this section, we provide some details on how we conducted our computational experiments and report the results obtained. The performance of the selective *arc-ng*-SPPRC pricing is compared against the default *arc-ng*-SPPRC pricing. Both of these settings are implemented over the

same column-and-cut generation framework, which is described in Section 4.1. In Section 4.2, we provide details on the experiments design. Finally, in Section 4.3, we report and discuss the computational results obtained.

4.1. Column-and-cut-generation framework

In our experiments, we consider a column-and-cut-generation framework very similar to the one developed by Pecin et al. (2017a) for the VRPTW. The algorithm includes several refinements that enhance its performance. The labeling algorithm used to solve the pricing subproblem relies on a bidirectional DSSR procedure (without completion bounds), having the time as the critical resource (Righini and Salani, 2006). The maximum size of *ng*-sets in the algorithm varies according to the instance being solved as discussed below. Furthermore, three fast heuristic labeling algorithms are always executed before any execution of the exact pricing algorithm. The first heuristic keeps only the least-cost label associated with a given time arriving at each customer node. The second and third heuristics rely on a reduced version of the network G, where around 7 and 12 (for the second and third, respectively) arcs entering and leaving each customer node are kept (see Desaulniers et al., 2008, for details on how these arcs are selected). At each column generation iteration, these three heuristic labeling algorithms are called in sequence until one of them finds negative reduced cost columns. If they all fail, the cut separation routines (see below) are invoked. When cuts are found, they are added to the RMP and column generation is re-started. Otherwise, the exact pricing algorithm is called. If it finds negative reduced cost columns, the process starts over again. Otherwise, the algorithm stops.

Because we wanted to make the implementation of our column generation algorithm as simple as possible, we decided not to consider the dynamic mechanism described by Bulhões et al. to increase and reduce the size of the *ng*-sets when needed. We judged that this procedure would make the implementation of the algorithm very cumbersome. Hence, we decided to rely on Martinelli et al.'s (2014) framework to handle the size of the *ng*-sets. In our implementation, *ng*-sets are initially empty and are updated throughout the solution process. Two strategies were considered to update the arc *ng*-sets: (i) `only-cycle`, where only *ng*-sets associated with arcs appearing in an infeasible cycle are increased; and (ii) `all-arcs`, where all the arcs induced by the customer nodes in an infeasible cycle have their *ng*-sets increased. The strategy `only-cycle` is the same considered by Bulhões et al. (2018). As pointed out by the authors, and also as we have observed in our preliminary experiments, the `only-cycle` strategy produces a higher number of DSSR iterations. However, the computational complexity of each such iteration remains lower (this is more notorious toward the end).

The baseline algorithm also includes the separation of several cuts, namely: rounded capacity cuts (RCCs, Laporte et al., 1985), limited-memory rank-1 cuts (Pecin et al., 2017a, 2017b), and elementary cuts (Pecin et al., 2017a). Except for the RCCs, all the other cuts are nonrobust, that is, their dual variables cannot be considered directly in the modified arc costs when solving the pricing subproblem. The explicit handling of these dual variables increases the difficulty of solving it. Consequently, the solution process is divided in two phases, called the *robust* and *nonrobust* phases. In the robust phase, the search for violated cuts is restricted to the RCCs. This phase ends with a call to the exact pricing algorithm that yielded no new columns and the application of

variable arc fixing (Irnich et al., 2010). The nonrobust phase then starts by searching for nonrobust violated cuts. If some are found, the solution process is re-started. This phase also ends with a final unsuccessful call to the exact pricing algorithm, ensuring that the value of the current RMP optimal solution provides a valid lower bound.

4.2. Experiments design

Our tests were performed on two datasets: (1) the 14 hardest 100-customer VRPTW instances from Solomon (1987); and (2) the 200-customer instances of Gehring and Homberger (2001) that are reported to be solved in less than five hours by Sadykov et al. (2017). These instances are called hereafter the S and the GH instances. Both datasets contain instances whose node locations are: chosen at random, clustered, and mixed (random and clustered). This is denoted in the S and GH instances, respectively, as: C, R, and RC; and C1, C2, R1, R2, RC1 and RC2. For the GH instances, specifically, instances in sets C1, R1, and RC1 have tight time windows and vehicle capacity, and typically admit optimal solutions with short vehicle routes. On the other hand, sets C2, R2, and RC2 have large time windows and loose vehicle capacity and, typically, admit optimal solutions with fewer but longer vehicle routes.

Similarly to what is done in Pecin et al. (2017a), for all S instances except R208, and all C1, RC1, and R1 GH instances, the cardinality of *ng*-sets $\mathcal{N}_i$, $i \in N^+$ are limited to 10. For the other instances, where allowing more cycles can be very harmful to the lower bound, we consider $|\mathcal{N}_i| = 20$. These node *ng*-sets are used to generate the arc *ng*-sets $\mathcal{N}_a$, $a = (u, v) \in A$, by using the formula $\mathcal{N}_a \leftarrow \mathcal{N}_u \cap \mathcal{N}_v$. Furthermore, for all instances except the C1, RC1, and R1 GH instances, we observed that the capacity constraint is not binding. Therefore, to alleviate the solution of the pricing subproblem, we did not consider this constraint, which is later enforced using RCCs whenever needed.

Given that our goal is not to solve to optimality the considered instances but rather to illustrate the impact of applying the selective *arc-ng*-SPPRC pricing, we focus only on the root node results. The algorithm described previously was coded in C++, and IBM ILOG CPLEX Optimizer 12.8 was used as the LP solver. The experiments were carried out on an Intel Xeon ES-2637 3.5GHz with 128GB RAM, running Linux Oracle Server 7.6. A time limit of 48 hours was set to solve all instances.

In our experiments, we consider three versions of the selective *arc-ng*-SPPRC pricing, which differ by the dominance rule employed. We compare these variants against the *arc-ng*-SPPRC considering a default dominance rule and the multiple partial label dominance. The latter consists of the default *arc-ng*-SPPRC with partial dominance being applied. We adopt the implicit representation of dominated extensions described in Section 2.3.2. All settings considered in our study are detailed below:

1. `Default`: The baseline algorithm that considers relations (10)–(13) as dominance rule. Relation (13) is reinforced by including the set $\mathcal{U}(L')$ of unreachable nodes in its right-hand side (which becomes $\Pi(L') \cup \mathcal{U}(L')$).
2. `PartialDom`: It is the same as `Default`, but considers the partial dominance using implicit representation of dominated extensions described in Section 2.3.2.

Table 1
Aggregated results using the `only-cycle` DSSR strategy before adding nonrobust cuts

Classes		T(s)					#labels				
		`Default`	`PartialDom`	`SetBased`	`Pairwise`	`SetPair`	`Default`	`PartialDom`	`SetBased`	`Pairwise`	`SetPair`
S-C	Average	17,534	12,272	8524	10,637	14,315	49,247	57,038	47,188	52,392	35,872
	Median	17,534	12,272	8524	10,637	14,315	49,247	57,038	47,188	52,392	35,872
S-R	Average	1621	1713	1232	1345	1366	69,557	72,606	55,929	58,002	56,907
	Median	649	665	533	520	542	40,523	36,038	32,088	33,020	31,437
S-RC	Average	6772	8297	4576	4335	4726	433,549	451,283	337,074	318,183	365,446
	Median	3043	3167	2363	2280	2516	589,868	583,749	452,280	419,077	441,036
GH-C1	Average	578	643	475	554	491	317,915	306,198	267,573	244,796	255,764
	Median	535	594	472	548	474	282,538	298,233	253,558	233,975	254,303
GH-C2	Average	5763	6564	4745	4509	5223	20,185	20,002	17,992	15,704	16,630
	Median	6158	7352	4811	4483	5458	14,186	14,377	13,307	11,640	13,866
GH-R1	Average	176	199	146	176	146	202,795	215,188	199,190	186,380	190,431
	Median	152	165	139	157	142	201,404	182,425	181,273	188,565	194,376
GH-R	Average	4807	4969	4370	4293	4334	15,255	16,182	12,289	12,290	12,687
	Median	4140	4379	4003	4076	4049	13,263	11,681	10,589	10,192	10,103
GH-RC	Average	206	227	167	192	167	436,372	439,793	443,949	402,631	371,387
	Median	143	153	127	140	127	273,251	270,804	260,652	237,269	221,346
GH-RC2	Average	6799	6694	5577	5688	5551	46,331	49,874	38,017	36,721	37,847
	Median	6769	6863	5441	5725	5567	44,015	42,940	35,663	33,906	35,051
# Best		0/45	0/45	18/45	13/45	14/45	3/45	1/45	9/45	21/45	16/45
# OPD		–	8/45	42/45	39/45	44/45	–	20/45	35/45	39/45	40/45
# OPP		37/45	–	44/45	44/45	44/45	24/45	–	36/45	39/45	39/45

3. `SetBased`: Consider relations (10)–(12) and the set-based relation (21) as dominance rule.
4. `Pairwise`: Consider relations (10)–(12) and the pairwise relation (19) as dominance rule.
5. `SetPair`: Consider relations (10)–(12), both pairwise (19) and set-based (21) relations as dominance rule.

Additionally, each pricing setting is executed twice, each time considering a different DSSR strategy, that is, `only-cycle` or `all-arcs`. Hence, in total, we test 10 distinct configurations.

For each instance, we retrieve the following information: lower bound (**lb**), number of column generation iterations performed by each algorithm (**#iters**), CPU time in seconds (**T(s)**), average number of DSSR iterations per column generation iteration performed by each algorithm (**#DSSR**), and average number of labels generated per iteration (**#labels**). This information is collected before (robust phase) and after (nonrobust phase) the addition of nonrobust cuts. Yet, because the main indicators for evaluating the efficiency of the selective algorithm are computational time and average number of labels generated per iteration, in Section 4.3, we focus our analysis on these two measures. Detailed results for all indicators are presented in Appendix B.

4.3. Computational results

In this section, we discuss the results obtained during our experiments. For the sake of conciseness, we only report summary results (see Appendix B for complete results). In Tables 1–4, we

Table 2
Aggregated results using the `all-arcs` DSSR strategy before adding nonrobust cuts

Classes		T(s)					#labels				
		`Default`	`PartialDom`	`SetBased`	`Pairwise`	`SetPair`	`Default`	`PartialDom`	`SetBased`	`Pairwise`	`SetPair`
S-C	Average	8744	7194	10,703	7399	10,495	32,462	36,091	31,839	22,682	29,243
	Median	8744	7194	10,703	7399	10,495	32,462	36,091	31,839	22,682	29,243
S-R	Average	815	857	813	755	789	41,340	47,478	42,687	40,071	41,562
	Median	368	419	373	360	362	23,608	30,685	24,467	25,960	25,301
S-RC	Average	2000	2532	1490	1531	1462	173,236	183,334	133,280	157,321	128,918
	Median	842	1036	761	759	765	223,594	240,585	160,031	199,839	155,315
GH-C1	Average	354	429	326	357	327	170,597	170,982	142,030	151,625	145,292
	Median	353	433	323	356	325	131,610	125,898	111,027	121,422	114,389
GH-C2	Average	4325	3311	2966	2781	2774	13,135	13,500	12,554	11,508	12,963
	Median	4014	3639	3476	2985	3097	10,609	11,426	9659	9440	12,609
GH-R1	Average	138	149	126	140	122	131,318	133,633	126,176	123,834	127,299
	Median	131	144	120	132	116	152,863	126,463	115,984	117,794	138,251
GH-R2	Average	3523	3649	3445	3435	3398	10,362	11,054	9731	9610	9959
	Median	3298	3543	3371	3272	3416	8490	9481	8318	8243	8129
GH-RC1	Average	134	141	117	127	117	212,577	221,608	244,069	229,380	216,139
	Median	117	126	109	113	108	161,529	157,078	193,357	172,433	164,655
GH-RC2	Average	3746	3916	3542	3593	3342	25,103	25,060	25,165	22,525	22,744
	Median	3482	3645	3247	3221	3111	23,951	26,726	24,889	21,480	24,995
# Best		3/45	1/45	10/45	8/45	23/45	10/45	3/45	9/45	17/45	11/45
# OPD		–	5/45	34/45	32/45	38/45	–	18/45	25/45	30/45	29/45
# OPP		40/45	–	42/45	43/45	44/45	26/45	–	28/45	35/45	34/45

provide the average and the median for **T(s)** and **#labels** per instance class, where an instance class is preceded by S or GH to identify if they correspond to S or GH instances. In each table, we compare the performance of settings `Default`, `PartialDom`, `SetBased`, `Pairwise`, and `SetPair`. We analyze separately the performance of the algorithms with respect to the DSSR strategy employed (`only-cycle` and `all-arcs`). We provide distinct tables for the algorithms before and after separating nonrobust cuts. Finally, line **#Best** shows the number of times that each setting provides the best results for each indicator, and lines **#OPD** and **#OPP** display the number of times that the corresponding setting outperforms the `Default` and `PartialDom` settings, respectively. Note that, in the case where both algorithms produce equal results for a given instance, this instance is counted for both algorithms. As a consequence, the sum of the **#Best** values can exceed the number of instances.

In Tables 1 and 2, we present the summary results produced by the algorithms before the separation of nonrobust cuts. These results show a clear advantage of the selective settings over both default algorithms (`Default` and `PartialDom`) in terms of running time and the number of labels produced. This statement is supported by the values of **#OPD** and **#OPP**. By considering the values of **#Best**, among the three selective variants, settings `SetBased` and `SetPair` are the ones yielding the best gains in terms of running time for `only-cycle` and `all-arcs`, respectively. Regarding the number of labels, for both DSSR strategies, the setting `Pairwise` is the most effective at reducing the number of nondominated labels kept for both DSSR strategies. We point out that the best performance of the selective algorithms become more evident when `only-cycle` strategy is adopted. Moreover, as one can see from detailed results in Appendix B, during the robust phase of

Table 3
Aggregated results using the `only-cycle` DSSR strategy after adding nonrobust cuts

Classes		T(s)					#labels				
		`Default`	`PartialDom`	`SetBased`	`Pairwise`	`SetPair`	`Default`	`PartialDom`	`SetBased`	`Pairwise`	`SetPair`
S-C	Average	17,535	12,273	8525	10,637	14,315	49,247	57,038	47,188	52,392	35,872
	Median	17,535	12,273	8525	10,637	14,315	49,247	57,038	47,188	52,392	35,872
S-R	Average	28,394	22,927	23,543	23,739	23,217	173,658	135,068	165,085	154,845	133,561
	Median	2854	2796	2348	2572	2644	32,796	30,983	31,385	28,540	29,710
S-RC	Average	8371	10,096	6274	5904	6492	259,595	257,033	203,106	187,660	195,340
	Median	5054	5232	4157	3864	4309	358,790	346,185	256,083	246,708	246,296
GH-C1	Average	44,147	44,099	43,933	44,657	43,825	260,127	255,879	231,525	232,520	227,062
	Median	1132	1341	997	1122	1052	291,258	286,609	263,977	268,100	246,535
GH-C2	Average	8914	9450	7292	7828	8644	15,744	15,437	14,587	12,807	12,439
	Median	8523	8425	5572	6271	7544	10,098	9613	8921	8643	9316
GH-R1	Average	63,024	65,315	61,748	62,681	64,377	200,621	196,682	200,899	197,020	195,001
	Median	26,455	28,930	22,703	28,276	30,800	94,172	93,539	96,043	88,250	91,813
GH-R2	Average	9940	9521	8726	8427	8481	6091	6313	5758	5636	5475
	Median	6913	7216	6841	6254	6547	5796	5899	5016	5033	4904
GH-RC1	Average	77,254	78,590	75,859	81,790	87,095	348,714	356,581	367,944	324,192	336,289
	Median	65,555	65,340	61,909	73,183	82,086	143,555	141,917	145,699	139,747	139,481
GH-RC2	Average	26,932	23,606	22,929	26,219	18,461	14,999	16,885	12,949	11,926	12,394
	Median	15,307	12,941	12,889	18,543	15,025	13,961	14,485	12,514	12,812	12,798
# Best		3/45	2/45	24/45	7/45	9/45	1/45	5/45	9/45	17/45	13/45
# OPD		–	17/45	40/45	32/45	29/45	–	24/45	35/45	39/45	41/45
# OPP		28/45	–	39/45	37/45	36/45	21/45	–	30/45	38/45	38/45

the algorithm, the selective algorithms allow better bounds to be achieved, although the differences in the bounds compared with the standard algorithms are not significant. The selective strategies also tend to require less DSSR iterations. When comparing the performance of the selective algorithms employing different DSSR strategies, the results confirm what has already been hinted by Bulhões et al. (2018), that is, the algorithms employing `only-cycle` produce a larger number of DSSR iterations. Also, they tend to keep more nondominated labels.

Tables 3 and 4 show the results obtained once the algorithms have completed their nonrobust phase. In the presence of nonrobust cuts, the selective algorithms can become more time-consuming. Nevertheless, they are still faster than the `Default` and `PartialDom` for the majority of the instances. The better performance of the selective algorithms is confirmed when we analyze the performance of the algorithms in the pairwise comparisons of Appendix B. Regarding the performance of the settings, the behavior of the algorithms during the nonrobust phase is similar to that observed during the robust phase. `SetBased` is the fastest selective setting and `Pairwise` is the most effective in reducing the number of nondominated labels kept. The less impressive advantage of the selective algorithms in the nonrobust phase may be explained by the impact that the nonrobust cuts have on the complexity of the labeling algorithms. Even if the selective algorithms rely on less restrictive dominance rules associated with *arc-ng*-SPPRC memory sets, as nonrobust cuts are incorporated into the RMP, the complexity of the labeling algorithms increase.

Finally, we present time profiles (Dolan and More, 2002) to compare the performance of the algorithms, which are generated as follows. Let $\mathcal{A}$ be a set of algorithms, namely, $\mathcal{A} = \{$`Default`,

Table 4
Aggregated results using the `all-arcs` DSSR strategy after adding nonrobust cuts

Classes		T(s)					#labels				
		`Default`	`PartialDom`	`SetBased`	`Pairwise`	`SetPair`	`Default`	`PartialDom`	`SetBased`	`Pairwise`	`SetPair`
S-C	Average	8744	7195	10,704	7399	10,495	32,462	36,091	31,839	22,682	29,243
	Median	8744	7195	10,704	7399	10,495	32,462	36,091	31,839	22,682	29,243
S-R	Average	22,253	23,550	22,447	23,984	23,742	127,362	114,715	124,953	133,732	110,891
	Median	1947	2186	2139	1868	1979	23,670	24,935	27,022	25,738	23,459
S-RC	Average	3173	4054	2745	2496	2808	104,460	106,102	83,246	91,149	84,988
	Median	2116	2521	1930	1493	2111	143,464	146,739	97,395	116,605	98,136
GH-C1	Average	44,288	43,750	43,663	43,978	43,997	161,729	156,948	151,531	147,506	147,533
	Median	659	865	667	686	714	159,911	158,767	144,722	140,424	143,434
GH-C2	Average	6607	6037	5956	5646	5287	10,286	10,357	10,007	9173	9796
	Median	5128	4770	5002	3959	4147	7082	6671	7360	7878	7873
GH-R1	Average	60,841	62,291	60,474	60,520	62,658	172,083	164,023	169,285	164,297	163,595
	Median	25,055	23,514	26,704	22,911	27,965	80,401	80,352	83,754	79,696	82,338
GH-R2	Average	7704	7927	7626	7486	7218	5248	5278	4970	4711	4972
	Median	6074	6130	5944	5714	6143	4570	4768	4597	4442	4360
GH-RC1	Average	74,751	73,509	70,540	77,213	81,305	208,928	211,301	216,317	203,785	201,337
	Median	56,572	57,263	52,908	65,389	73,765	124,358	124,054	123,466	123,456	122,347
GH-RC2	Average	22,709	22,449	22,851	19,354	15,725	10,049	11,085	11,575	9579	10,337
	Median	12,754	12,792	9052	11,967	8319	9879	11,266	10,536	9181	9966
# Best		11/45	5/45	14/45	7/45	8/45	2/45	8/45	6/45	19/45	10/45
# OPD		–	17/45	23/45	25/45	24/45	–	19/45	21/45	32/45	32/45
# OPP		28/45	–	35/45	30/45	31/45	26/45	–	22/45	33/45	33/45

`PartialDom`, `SetBased`, `Pairwise`, `SetPair`} in our case. Let I be a set of instances, namely, the 45 instances considered in our experiments. For each algorithm $A \in \mathcal{A}$ and each instance $i \in I$, denote by T_A^i the time spent by A to solve instance i and let $T_{\text{Best}}^i = \min_{A \in \mathcal{A}}\{T_A^i\}$ be the best time achieved by an algorithm in $\mathcal{A}$ to solve this instance. For an algorithm $A \in \mathcal{A}$, a time profile is a function $\rho_A(\tau)$ of a ratio $\tau \geq 1$ that is equal to the percentage of instances such that $\frac{T_A^i}{T_{\text{Best}}^i} \leq \tau$, that is,

$$\rho_A(\tau) = 100\,\frac{\left|\left\{i \in I \mid \frac{T_A^i}{T_{\text{Best}}^i} \leq \tau\right\}\right|}{|I|}. \tag{22}$$

In particular, $\rho_A(1)$ represents the percentage of instances for which algorithm A was the fastest. Figures 5 and 6 display the time profiles of the five algorithms for the robust and the nonrobust phases, respectively. From these profiles, one can see that the selective algorithms yield considerable speed-ups when compared to `Default` and `PartialDom`. The acceleration is more notable when comparing algorithms employing the `only-cycle` DSSR strategy. The time profiles confirm what has been observed in Tables 1–4, that is, `SetBased` seems to be the most time-efficient setting. `SetPair`, on the other hand, is the most time-consuming among the selective settings, to the point where it is slower than `Default` for 70% of the instances during the nonrobust phase when `only-cycle` DSSR strategy is employed. In the presence of nonrobust cuts, the overhead of using both pairwise (19) and set-based (21) dominance relations may not pay off.

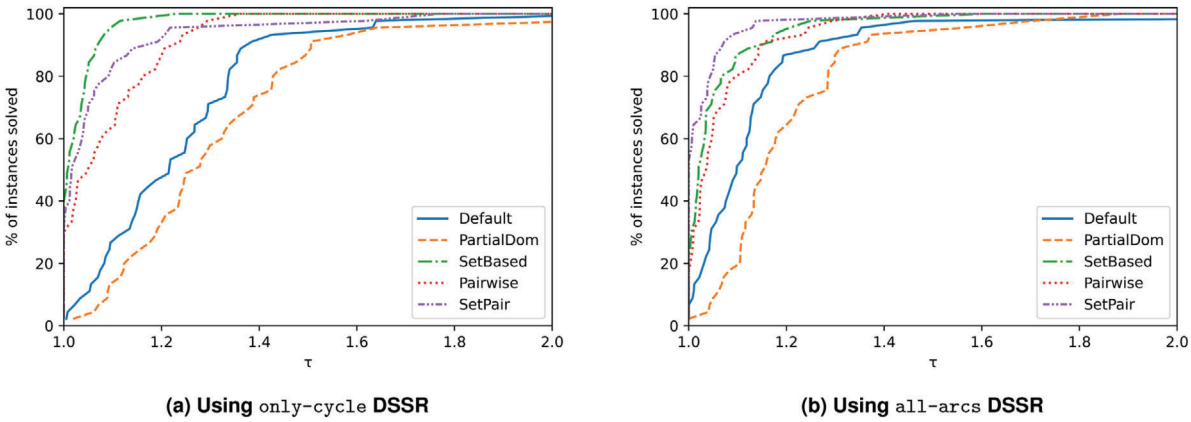

Fig. 5. Time profiles for all instances before adding nonrobust cuts.

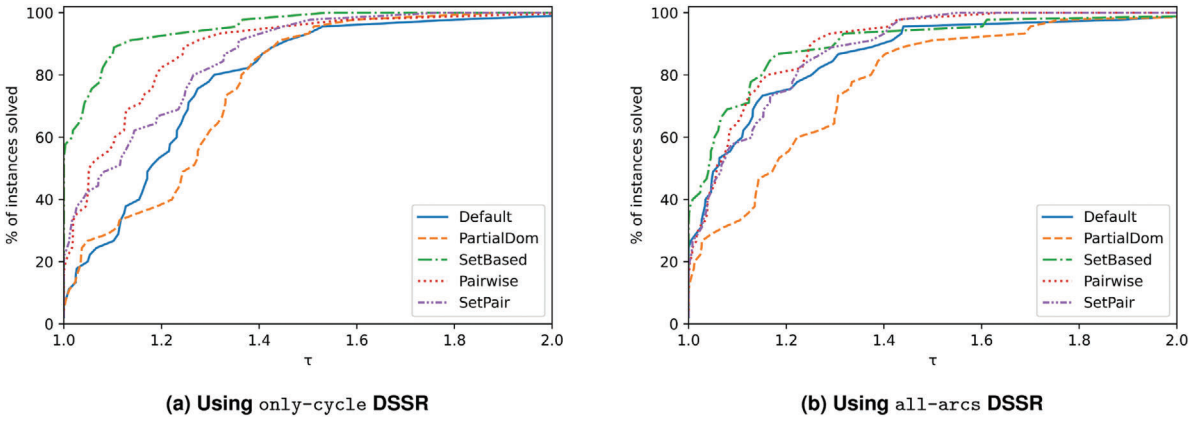

Fig. 6. Time profiles for all instances after adding nonrobust cuts.

5. Concluding remarks

In this paper, we have presented two new selective dominance rules for the *arc-ng*-SPPRC, namely, one that applies pairwise label comparison and one that extends the set-based rule of Bulhões et al. (2018). The latter rule is stronger than the one proposed by Bulhões et al. as it considers the selective mechanism expressed by condition (18), which increases the chances that a label L dominates a label $L' = L^\star \oplus w$. Furthermore, an additional speedup may be observed if the average memory size is much larger than the number of cost and resource components in a label. Our computational experiments on VRPTW benchmark instances showed that, in general, the new mechanism allows a reduction in terms of the number of treated labels and, consequently, of the computational time. Note also that one of the main interests for using this mechanism is that it does not rely on complex data structures like the ones described by Bulhões et al. (2018, Section 4.2) and Desaulniers et al. (2019, Section 3.2) to work satisfactorily.

An extension of our idea would be to investigate the impact of generalizing relation (14) to consider other predecessors further down in the path. Theoretically, this generalization would strengthen even more the dominance rule. However, it seems that an elaborated data structure would be required to address the overhead incurred by the new relations.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada [Grants 2017-05683 and 2020-06311]. This support is gratefully acknowledged.

References

Baldacci, R., Mingozzi, A., Roberti, R., 2011. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research* 59, 1269–1283.

Boland, N., Dethridge, J., Dumitrescu, I., 2006. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters* 34, 58–68.

Bulhões, T., Sadykov, R., Uchoa, E., 2018. A branch-and-price algorithm for the minimum latency problem. *Computers & Operations Research* 93, 66–78.

Contardo, C., Desaulniers, G., Lessard, F., 2015. Reaching the elementary lower bound in the vehicle routing problem with time windows. *Networks* 65, 1, 88–99.

Costa, L., Contardo, C., Desaulniers, G., 2019. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science* 53, 4, 946–985.

Dantzig, G.B., Fulkerson, D.R., Johnson, S.M., 1954. Solution of a large-scale traveling salesman problem. *Operations Research* 2, 393–410.

Dantzig, G.B., Ramser, J.H., 1959. The truck dispatching problem. *Management Science* 6, 80–91.

Desaulniers, G., Lessard, F., Hadjar, A., 2008. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science* 42, 3, 387–404.

Desaulniers, G., Madsen, O.B.G., Ropke, S., 2014. The vehicle routing problem with time windows. In Toth, P., Vigo, D. (eds) *Vehicle Routing: Problems, Methods and Applications* (2nd edn). Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 119–159.

Desaulniers, G., Pecin, D., Contardo, C., 2019. Selective pricing in branch-price-and-cut algorithms for vehicle routing. *EURO Journal on Transportation and Logistics* 8, 2, 147–168.

Desrochers, M., Desrosiers, J., Solomon, M., 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* 40, 342–354.

Desrochers, M., Soumis, F., 1988. A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR: Information Systems and Operational Research* 26, 3, 191–212.

Desrosiers, J., Soumis, F., Desrochers, M., 1984. Routing with time windows by column generation. *Networks* 14, 545–565.

Dolan, E.D., More, J.J., 2002. Benchmarking optimization software with performance profiles. *Mathematical Programming A* 91, 201–213.

Dror, M., 1994. Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research* 42, 977–978.

Gehring, H., Homberger, J., 2001. A parallel two-phase metaheuristic for routing problems with time windows. *Asia-Pacific Journal of Operational Research* 18, 1, 35.

Irnich, S., 2008. Resource extension functions: properties, inversion and generalization to segments. *OR Spectrum* 30, 1, 113–148.

Irnich, S., Desaulniers, G., 2005. Shortest path problems with resource constraints. In Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds) *Column Generation* (1st edn). Springer, Boston, MA, pp. 33–65.

Irnich, S., Desaulniers, G., Desrosiers, J., Hadjar, A., 2010. Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing* 22, 2, 297–313.

Irnich, S., Villeneuve, D., 2006. The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing* 18, 391–406.

Laporte, G., Nobert, Y., Desrochers, M., 1985. Optimal routing under capacity and distance restrictions. *Operations Research* 33, 1050–1073.

Martinelli, R., Pecin, D., Poggi, M., 2014. Efficient elementary and restricted non-elementary route pricing. *European Journal of Operational Research* 239, 1, 102–111.

Pecin, D., Contardo, C., Desaulniers, G., Uchoa, E., 2017a. New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing* 29, 3, 489–502.

Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., 2017b. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation* 9, 61–100.

Poggi, M., Uchoa, E., 2014. New exact algorithms for the capacitated vehicle routing problem. In Toth, P., Vigo, D. (eds) *Vehicle Routing: Problems, Methods and Applications* (2nd edn). Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 59–86.

Righini, G., Salani, M., 2006. Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization* 3, 255–273.

Sadykov, R., Uchoa, E., Pessoa, A.A., 2017. A bucket graph based labeling algorithm with application to vehicle routing. Research Report Cadernos do LOGIS 2017/7, Universidade Federal Fluminense.

Solomon, M.M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* 35, 2, 254–265.

Toth, P., Vigo, D., 2014. *Vehicle routing: problems, methods and applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Vidal, T., Laporte, G., Matl, P., 2020. A concise guide to existing and emerging vehicle routing problem variants. *European Journal of Operational Research* 286, 401–416.

Appendix A: Effect of applying the newly pairwise comparison over the *ng*-SPPRC

In this section, we demonstrate that the selective dominance proposed in this article has no effect in reducing the number of nondominated labels when applied to a traditional *ng*-SPPRC, and becomes a useless overhead when compared to the traditional dominance. For a label L, let $\Pi(L)$ be the nodes currently in the memory of label L. For every node $v \in N^+$, denote by $\mathcal{N}_v$ its neighborhood. When extending a label L to a node $v \in N^+ \setminus \Pi(L)$, the memory of the new label (that we denote L') is $\Pi(L') \leftarrow \{\Pi(L) \cap \mathcal{N}_v\} \cup \{v\}$.

Lemma 2. *For every pair of labels L_1, L_2 with the same terminal node k, $\Pi(L_1) \cap [\Pi(pred(L_2)) \cup \{k\}] = \Pi(L_1) \cap \Pi(L_2)$.*

Proof.

$$\begin{aligned}
\Pi(L_1) \cap [\Pi(pred(L_2)) \cup \{k\}] &= ([\Pi(pred(L_1) \cap \mathcal{N}_k] \cup \{k\}) \cap [\Pi(pred(L_2)) \cup \{k\}] \\
&= (\Pi(pred(L_1)) \cap \Pi(pred(L_2)) \cap \mathcal{N}_k) \cup \{k\} \\
&= ([\Pi(pred(L_1)) \cap \mathcal{N}_k] \cup \{k\}) \cap ([\Pi(pred(L_2)) \cap \mathcal{N}_k] \cup \{k\}) \\
&= \Pi(L_1) \cap \Pi(L_2).
\end{aligned}$$

□

Proposition 2. *For every pair of labels L_1, L_2 with the same terminal node k, $\Pi(L_1) \subseteq \Pi(L_2)$ if and only if $\Pi(L_1) \subseteq \Pi(pred(L_2)) \cup \{k\}$.*

Proof. We have that

$$\begin{aligned}\Pi(L_1) \subseteq \Pi(L_2) &\Leftrightarrow \Pi(L_1) \cap \Pi(L_2) = \Pi(L_1)\\ &\Leftrightarrow \Pi(L_1) \cap [\Pi(pred(L_2)) \cup \{k\}] = \Pi(L_1) \qquad \text{(from Lemma 2)}\\ &\Leftrightarrow \Pi(L_1) \subseteq [\Pi(pred(L_2)) \cup \{k\}].\end{aligned}$$

□

Appendix B: Detailed results

In this section, we report detailed results for all instances solved by settings `Default`, `SetBased`, `Pairwise`, and `SetPair`. We provide pairwise comparisons between the `Default` algorithm and each of the selective settings. Sections B.1 and B.2 present results for S and GH instances, respectively. For each instance, we report: the lower bound (**lb**) achieved, the number of column generation iterations performed by each algorithm (**#iters**), the CPU time in seconds (**T(s)**), the average number of DSSR iterations per column generation iteration performed by each algorithm (**#DSSR**), and the average number of labels generated per iteration (**#labels**). When comparing the performance of two algorithms, we highlight in boldface the best results for each indicator. Line **#Best** shows the total number of instances for which each algorithm was capable of achieving the best values. Note that, in the case where both algorithms produce equal results for a given instance, this instance is counted for both algorithms.

B.1. Solomon's instances

Table B1
Detailed results for comparing `Default` versus `SetBased` settings using `only-cycle` DSSR before adding nonrobust cuts

	Default					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3423	406	1315	21,380	**5,887.0**	**2400**	**375**	**1029**	**14,139**
C204	**5,881.0**	31,645	1553	3492	**77,113**	**5,881.0**	**14,649**	**676**	**2182**	80,236
R202	**10,164.8**	284	139	547	**8564**	**10,164.8**	**249**	**106**	**445**	8975
R203	8,597.0	761	172	852	30,072	**8,597.6**	**625**	**136**	**670**	**29,180**
R204	**7,202.0**	2162	**182**	1189	100,223	**7,202.0**	**1638**	184	**874**	**66,260**
R206	**8,600.1**	608	116	797	34,964	**8,600.1**	**511**	**102**	**691**	**32,088**
R207	7,805.0	1217	159	921	55,715	**7,805.6**	**983**	**125**	**760**	**54,710**
R208	**6,910.9**	8133	285	2167	263,264	**6,910.9**	**5872**	**256**	**1741**	**203,856**
R209	**8,371.8**	346	97	609	24,750	**8,371.8**	**304**	**86**	**535**	**22,607**
R210	8,813.2	426	**78**	746	40,523	**8,813.8**	**376**	83	**626**	**30,214**
R211	7,311.0	649	**127**	867	67,939	**7,311.2**	**533**	128	**743**	**55,474**
RC204	**7,764.9**	16,472	320	3264	618,355	**7,764.9**	**10,746**	**275**	**2895**	**452,280**
RC207	**9,415.8**	802	**102**	1596	92,424	**9,415.8**	**619**	**102**	**1393**	**68,303**
RC208	**7,636.4**	3043	167	3166	589,868	**7,636.4**	**2363**	**136**	**2958**	**490,639**
#Best	10/14	0/14	4/14	0/14	2/14	14/14	14/14	11/14	14/14	12/14

Table B2
Detailed results for comparing `Default` versus `SetBased` settings using `only-cycle` DSSR after adding nonrobust cuts

	Default					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3424	406	1315	21,380	**5,887.0**	**2400**	**375**	**1029**	**14,139**
C204	**5,881.0**	31,645	1553	3492	**77,113**	**5,881.0**	**14,649**	**676**	**2182**	80,236
R202	**10,296.0**	966	815	1329	3948	**10,296.0**	**866**	**709**	**1140**	**3792**
R203	**8,708.0**	1232	642	1375	11,972	**8,708.0**	**1052**	**578**	**1175**	**11,568**
R204	**7,313.0**	5512	**661**	1882	68,200	**7,313.0**	**5463**	684	**1654**	**61,430**
R206	**8,759.0**	1559	514	1324	20,537	**8,759.0**	**1332**	**479**	**1198**	**18,776**
R207	**7,940.0**	2854	606	1515	32,796	**7,940.0**	**2348**	**529**	**1357**	**31,385**
R208	6,998.0	229,080	**653**	3209	1,194,476	**7,001.9**	**185,708**	680	**3094**	**1,129,873**
R209	**8,548.0**	2594	603	1420	28,561	**8,548.0**	**2171**	**596**	**1357**	**27,691**
R210	**9,005.0**	4538	**620**	1549	37,429	**9,005.0**	**4061**	661	**1433**	**33,007**
R211	**7,467.0**	**7215**	**373**	1583	**165,002**	**7,467.0**	8884	374	**1466**	168,246
RC204	**7,835.0**	18,239	**560**	3558	381,153	**7,835.0**	**13,049**	561	**3263**	**256,083**
RC207	**9,629.0**	1822	**461**	2173	38,842	**9,629.0**	**1617**	493	**1989**	**30,905**
RC208	**7,761.0**	5054	396	3667	358,790	**7,761.0**	**4157**	**341**	**3508**	**322,332**
#Best	13/14	1/14	6/14	0/14	2/14	14/14	13/14	8/14	14/14	12/14

Table B3
Detailed results for comparing `Default` versus `Pairwise` settings using `only-cycle` DSSR before adding nonrobust cuts

	Default					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3423	406	1315	21,380	**5,887.0**	**3060**	**360**	**1143**	**18,735**
C204	**5,881.0**	31,645	1553	3492	**77,113**	**5,881.0**	**18,214**	**787**	**2089**	86,049
R202	**10,164.8**	284	139	547	**8564**	**10,164.8**	**252**	**99**	**461**	9175
R203	**8,597.0**	761	172	852	30,072	**8,597.0**	**691**	**145**	**734**	**26,536**
R204	**7,202.0**	2162	182	1189	100,223	**7,202.0**	**1753**	**166**	**944**	**80,729**
R206	**8,600.1**	608	116	797	34,964	**8,600.1**	**486**	**111**	**647**	**27,500**
R207	**7,805.0**	1217	159	921	55,715	**7,805.0**	**943**	**126**	**726**	**48,319**
R208	**6,910.9**	8133	285	2167	263,264	**6,910.9**	**6789**	**272**	**1920**	**218,370**
R209	**8,371.8**	346	97	609	**24,750**	**8,371.8**	**305**	**74**	**521**	24,866
R210	8,813.2	426	78	746	40,523	**8,814.4**	**363**	**72**	**632**	**33,020**
R211	7,311.0	649	127	867	67,939	**7,311.4**	**520**	**125**	**729**	**53,503**
RC204	**7,764.9**	16,472	320	3264	618,355	**7,764.9**	**10,090**	**282**	**2968**	**419,077**
RC207	**9,415.8**	802	102	1596	92,424	**9,415.8**	**635**	**94**	**1379**	**75,578**
RC208	**7,636.4**	3043	167	3166	589,868	**7,636.4**	**2280**	**135**	**2937**	**459,893**
#Best	12/14	0/14	0/14	0/14	3/14	14/14	14/14	14/14	14/14	11/14

Table B4
Detailed results for comparing `Default` versus `Pairwise` settings using `only-cycle` DSSR after adding nonrobust cuts

	Default					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3424	406	1315	21,380	**5,887.0**	**3060**	**360**	**1143**	**18,735**
C204	**5,881.0**	31,645	1553	3492	**77,113**	**5,881.0**	**18,214**	**787**	**2089**	86,049
R202	**10,296.0**	966	815	1329	3948	**10,296.0**	**860**	**802**	**1235**	**3368**
R203	**8,708.0**	1232	642	1375	11,972	**8,708.0**	**1148**	**635**	**1269**	**10,019**
R204	**7,313.0**	**5512**	661	1882	68,200	**7,313.0**	6206	**657**	**1776**	**66,589**
R206	**8,759.0**	1559	514	1324	20,537	**8,759.0**	**1395**	**482**	**1150**	**16,487**
R207	**7,940.0**	2854	606	1515	32,796	**7,940.0**	**2477**	**579**	**1330**	**28,540**
R208	6,998.0	229,080	**653**	3209	1,194,476	**7,001.9**	**185,645**	703	**3152**	**1,046,169**
R209	**8,548.0**	2594	**603**	1420	28,561	**8,548.0**	**2572**	614	**1387**	**26,942**
R210	**9,005.0**	4538	620	1549	37,429	**9,005.0**	**3842**	**598**	**1401**	**33,483**
R211	**7,467.0**	**7215**	**373**	1583	165,002	**7,467.0**	9510	388	**1461**	**162,013**
RC204	**7,835.0**	18,239	560	3558	381,153	**7,835.0**	**12,150**	**555**	**3311**	**246,708**
RC207	**9,629.0**	1822	461	2173	38,842	**9,629.0**	**1698**	**432**	**1942**	**34,256**
RC208	**7,761.0**	5054	396	3667	358,790	**7,761.0**	**3864**	**358**	**3470**	**282,017**
#Best	13/14	2/14	3/14	0/14	1/14	14/14	12/14	11/14	14/14	13/14

Table B5
Detailed results for comparing `Default` versus `SetPair` settings using `only-cycle` DSSR before adding nonrobust cuts

	Default					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3423	**406**	1315	21,380	**5,887.0**	**2737**	440	**1147**	**13,784**
C204	**5,881.0**	31,645	**1553**	3492	77,113	**5,881.0**	**25,892**	1576	**3019**	**57,959**
R202	**10,164.8**	284	139	547	**8564**	**10,164.8**	**246**	**98**	**480**	9754
R203	**8,597.0**	761	172	852	30,072	**8,597.0**	**650**	**155**	**704**	**24,398**
R204	**7,202.0**	2162	182	1189	100,223	**7,202.0**	**1587**	**173**	**934**	**74,028**
R206	**8,600.1**	608	**116**	797	34,964	**8,600.1**	**533**	**116**	**711**	**27,342**
R207	**7,805.0**	1217	159	921	55,715	**7,805.1**	**954**	**149**	**727**	**44,896**
R208	**6,910.9**	8133	285	2167	263,264	**6,910.9**	**7087**	**277**	**1954**	**219,155**
R209	**8,371.8**	346	97	609	**24,750**	**8,371.8**	**316**	**80**	**560**	25,636
R210	**8,813.2**	426	78	746	40,523	**8,813.2**	**382**	**77**	**650**	**31,437**
R211	**7,311.0**	649	127	867	67,939	**7,311.0**	**542**	**124**	**739**	**55,515**
RC204	**7,764.9**	16,472	320	3264	618,355	**7,764.9**	**11,012**	**271**	**2815**	**441,036**
RC207	**9,415.8**	802	102	1596	92,424	**9,415.8**	**650**	**97**	**1366**	**73,623**
RC208	**7,636.4**	3043	167	3166	589,868	**7,636.4**	**2516**	**119**	**2952**	**581,679**
#Best	14/14	0/14	3/14	0/14	2/14	14/14	14/14	12/14	14/14	12/14

Table B6
Detailed results for comparing `Default` versus `SetPair` settings using `only-cycle` DSSR after adding nonrobust cuts

	Default					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3424	**406**	1315	21,380	**5,887.0**	**2738**	440	**1147**	**13,784**
C204	**5,881.0**	31,645	**1553**	3492	77,113	**5,881.0**	**25,893**	1576	**3019**	**57,959**
R202	**10,296.0**	966	815	1329	3948	**10,296.0**	**762**	**659**	**1118**	**3654**
R203	**8,708.0**	1232	642	1375	11,972	**8,708.0**	**1080**	**600**	**1203**	**10,074**
R204	**7,313.0**	5512	661	1882	68,200	**7,313.0**	**5255**	**611**	**1621**	**62,939**
R206	**8,759.0**	**1559**	514	1324	20,537	**8,759.0**	1586	**498**	**1203**	**19,174**
R207	**7,940.0**	2854	606	1515	32,796	**7,940.0**	**2644**	**594**	**1367**	**29,710**
R208	**6,998.0**	229,080	653	3209	1,194,476	6,996.4	**179,708**	**642**	**2979**	**855,593**
R209	**8,548.0**	2594	603	1420	28,561	**8,548.0**	**2324**	**587**	**1370**	**25,630**
R210	**9,005.0**	**4538**	**620**	1549	37,429	**9,005.0**	4742	**620**	**1404**	**30,323**
R211	**7,467.0**	**7215**	**373**	1583	165,002	**7,467.0**	10,856	409	**1477**	**164,952**
RC204	**7,835.0**	18,239	560	3558	381,153	**7,835.0**	**13,148**	**549**	**3162**	**246,296**
RC207	**9,629.0**	**1822**	461	2173	38,842	**9,629.0**	2021	**445**	**1953**	**35,634**
RC208	**7,761.0**	5054	396	3667	358,790	**7,761.0**	**4309**	**353**	**3462**	**304,091**
#Best	14/14	4/14	4/14	0/14	0/14	13/14	10/14	11/14	14/14	14/14

Table B7
Detailed results for comparing `Default` versus `SetBased` settings using `all-arcs` DSSR before adding nonrobust cuts

	Default					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	1747	**319**	639	11,902	**5,887.0**	**1517**	324	**564**	**10,446**
C204	**5,881.0**	**15,741**	**1107**	**1557**	**53,022**	**5,881.0**	19,889	1368	1808	53,233
R202	**10,164.8**	**216**	**104**	366	7822	**10,164.8**	219	116	**338**	**6198**
R203	**8,597.0**	531	139	525	22,779	**8,597.0**	**478**	**130**	**462**	**21,196**
R204	**7,201.6**	1174	167	602	**55,959**	**7,201.6**	**1110**	**139**	**523**	57,509
R206	**8,600.1**	**368**	92	420	**23,608**	**8,600.1**	373	**87**	**418**	24,467
R207	**7,805.0**	708	115	476	**40,871**	**7,805.0**	**653**	**97**	**453**	43,833
R208	**6,910.9**	**3441**	**203**	**792**	146,423	**6,910.9**	3608	211	808	**145,786**
R209	**8,371.8**	**243**	90	**362**	**15,426**	**8,371.8**	245	**80**	364	17,652
R210	**8,813.2**	308	84	475	**22,313**	**8,813.2**	**282**	**75**	**420**	24,123
R211	**7,311.0**	**344**	112	**426**	**36,857**	**7,311.0**	348	**97**	435	43,420
RC204	**7,764.9**	4843	215	946	252,430	**7,764.9**	**3413**	**213**	**829**	**160,031**
RC207	**9,415.8**	316	73	580	43,685	**9,415.8**	**295**	**68**	**521**	**38,995**
RC208	**7,636.4**	842	105	867	223,594	**7,636.4**	**761**	**97**	**830**	**200,815**
#Best	14/14	6/14	4/14	4/14	7/14	14/14	8/14	10/14	10/14	7/14

Table B8
Detailed results for comparing `Default` versus `SetBased` settings using `all-arcs` DSSR after adding nonrobust cuts

	Default					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	1747	**319**	639	11,902	**5,887.0**	**1517**	324	**564**	**10,446**
C204	**5,881.0**	**15,741**	**1107**	**1557**	**53,022**	**5,881.0**	19,890	1368	1808	53,233
R202	**10,296.0**	**717**	677	973	**3232**	**10,296.0**	736	**660**	**912**	3494
R203	**8,708.0**	1140	746	1156	**8708**	**8,708.0**	**830**	**502**	**864**	9819
R204	**7,313.0**	**3146**	566	1048	**42,611**	**7,313.0**	3642	**554**	**997**	44,628
R206	**8,759.0**	**1086**	**414**	**779**	15,333	**8,759.0**	1099	453	826	**14,891**
R207	**7,940.0**	**1947**	529	924	**23,670**	**7,940.0**	2083	**505**	**907**	27,022
R208	**7,009.7**	**182,430**	581	1287	911,617	7,006.1	183,931	**558**	**1246**	**873,352**
R209	**8,548.0**	**1638**	**545**	**903**	**18,992**	**8,548.0**	2139	596	958	22,394
R210	**9,005.0**	4378	634	1082	**29,615**	**9,005.0**	**3580**	**576**	**993**	30,062
R211	**7,467.0**	**3793**	325	768	**92,478**	**7,467.0**	3985	**300**	**759**	98,913
RC204	**7,835.0**	6464	**474**	1221	143,464	**7,835.0**	**5250**	493	**1127**	**97,395**
RC207	**9,629.0**	**940**	**354**	**917**	21,801	**9,629.0**	1056	403	922	**20,609**
RC208	**7,761.0**	2116	**326**	1125	148,114	**7,761.0**	**1930**	331	**1109**	**131,734**
#Best	14/14	9/14	7/14	4/14	8/14	13/14	5/14	7/14	10/14	6/14

Table B9
Detailed results for comparing `Default` versus `Pairwise` settings using `all-arcs` DSSR before adding nonrobust cuts

	Default					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	**1747**	**319**	**639**	11,902	**5,887.0**	1889	491	679	**8099**
C204	**5,881.0**	15,741	1107	1557	53,022	**5,881.0**	**12,909**	**1091**	**1466**	**37,265**
R202	**10,164.8**	216	**104**	366	7822	**10,164.8**	**210**	**104**	**334**	**6575**
R203	**8,597.0**	531	139	525	22,779	**8,597.0**	**479**	**133**	**473**	**21,009**
R204	**7,201.6**	1174	167	602	55,959	**7,201.6**	**1126**	**151**	**546**	**52,530**
R206	**8,600.1**	368	**92**	**420**	23,608	**8,600.1**	**360**	98	**420**	**19,897**
R207	**7,805.0**	708	115	476	**40,871**	**7,805.0**	**660**	**92**	**457**	47,529
R208	**6,910.9**	3441	203	792	146,423	**6,910.9**	**3077**	**199**	**742**	**130,870**
R209	**8,371.8**	**243**	90	**362**	**15,426**	**8,371.8**	248	**79**	365	17,136
R210	**8,813.2**	308	84	475	**22,313**	**8,813.2**	**277**	**63**	**416**	25,960
R211	**7,311.0**	**344**	112	**426**	**36,857**	**7,311.0**	355	**110**	453	39,132
RC204	**7,764.9**	4843	215	946	252,430	**7,764.9**	**3547**	**191**	**865**	**199,839**
RC207	**9,415.8**	316	**73**	580	43,685	**9,415.8**	**288**	78	**513**	**33,072**
RC208	**7,636.4**	842	105	867	**223,594**	7,636.0	**759**	**81**	**820**	239,051
#Best	14/14	3/14	4/14	4/14	5/14	13/14	11/14	11/14	11/14	9/14

Table B10
Detailed results for comparing `Default` versus `Pairwise` settings using `all-arcs` DSSR after adding nonrobust cuts

	Default					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	**1747**	**319**	**639**	11,902	**5,887.0**	1889	491	679	**8099**
C204	**5,881.0**	15,741	1107	1557	53,022	**5,881.0**	**12,909**	**1091**	**1466**	**37,265**
R202	**10,296.0**	717	677	973	3232	**10,296.0**	**699**	**622**	**891**	**3161**
R203	**8,708.0**	1140	746	1156	**8708**	**8,708.0**	**933**	**575**	**934**	9083
R204	**7,313.0**	**3146**	566	1048	**42,611**	**7,313.0**	3887	**548**	**991**	49,736
R206	**8,759.0**	**1086**	**414**	**779**	15,333	**8,759.0**	1088	444	796	**14,029**
R207	**7,940.0**	1947	529	924	**23,670**	**7,940.0**	**1868**	**468**	**874**	25,738
R208	**7,009.7**	**182,430**	581	1287	**911,617**	7,008.4	197,603	**555**	**1195**	962,739
R209	**8,548.0**	**1638**	**545**	**903**	**18,992**	**8,548.0**	1709	565	929	19,346
R210	**9,005.0**	4378	634	1082	29,615	**9,005.0**	**3718**	**587**	**1010**	**28,225**
R211	**7,467.0**	**3793**	**325**	**768**	92,478	**7,467.0**	4350	329	792	**91,532**
RC204	**7,835.0**	6464	474	1221	143,464	**7,835.0**	**4949**	**425**	**1115**	**116,605**
RC207	**9,629.0**	**940**	**354**	917	21,801	**9,629.0**	1046	410	**907**	**19,866**
RC208	**7,761.0**	2116	326	1125	148,114	**7,761.0**	**1493**	**267**	**1041**	**136,976**
#Best	14/14	7/14	5/14	4/14	5/14	13/14	7/14	9/14	10/14	9/14

Table B11
Detailed results for comparing `Default` versus `SetPair` settings using `all-arcs` DSSR before adding nonrobust cuts

	Default					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	1747	**319**	639	**11,902**	**5,887.0**	**1726**	321	**637**	12,275
C204	**5,881.0**	**15,741**	**1107**	**1557**	53,022	**5,881.0**	19,263	1526	1934	**46,212**
R202	**10,164.8**	216	104	366	7822	**10,164.8**	**208**	**102**	**332**	**6754**
R203	8,597.0	531	**139**	525	22,779	**8,598.4**	**469**	159	**493**	**18,365**
R204	7,201.6	1174	167	602	55,959	**7,202.0**	**1042**	**140**	**504**	**51,514**
R206	**8,600.1**	368	**92**	420	23,608	**8,600.1**	**356**	97	**405**	**20,186**
R207	**7,805.0**	708	115	476	40,871	**7,805.0**	**645**	**100**	**451**	**39,837**
R208	**6,910.9**	**3441**	203	792	**146,423**	**6,910.9**	3477	**175**	**770**	153,871
R209	**8,371.8**	**243**	90	362	**15,426**	**8,371.8**	244	**70**	**358**	18,608
R210	**8,813.2**	308	84	475	**22,313**	**8,813.2**	**295**	**70**	**448**	25,301
R211	**7,311.0**	**344**	112	**426**	**36,857**	**7,311.0**	362	**108**	455	39,622
RC204	**7,764.9**	4843	215	946	252,430	**7,764.9**	**3306**	**201**	**807**	**155,315**
RC207	**9,415.8**	316	**73**	580	43,685	**9,415.8**	**314**	78	**535**	**36,044**
RC208	**7,636.4**	842	105	867	223,594	7,636.0	**765**	**92**	**797**	**195,395**
#Best	12/14	4/14	5/14	2/14	5/14	13/14	10/14	9/14	12/14	9/14

Table B12
Detailed results for comparing `Default` versus `SetPair` settings using `all-arcs` DSSR after adding nonrobust cuts

	Default					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	1747	**319**	639	**11,902**	**5,887.0**	**1727**	321	**637**	12,275
C204	**5,881.0**	**15,741**	**1107**	**1557**	53,022	**5,881.0**	19,263	1526	1934	**46,212**
R202	**10,296.0**	**717**	677	973	3232	**10,296.0**	724	**618**	**874**	**3135**
R203	**8,708.0**	1140	746	1156	8708	**8,708.0**	**896**	**650**	**999**	**8447**
R204	**7,313.0**	**3146**	566	1048	**42,611**	**7,313.0**	3916	**544**	**959**	45,432
R206	**8,759.0**	1086	**414**	779	15,333	**8,759.0**	**1051**	**414**	**757**	**14,440**
R207	**7,940.0**	**1947**	529	924	23,670	**7,940.0**	1979	**505**	**885**	**23,459**
R208	**7,009.7**	**182,430**	581	1287	911,617	7,005.3	194,490	**506**	**1170**	**764,528**
R209	**8,548.0**	**1638**	545	903	**18,992**	**8,548.0**	1911	**520**	**886**	19,705
R210	**9,005.0**	4378	634	1082	29,615	**9,005.0**	**4340**	**600**	**1054**	**28,787**
R211	**7,467.0**	**3793**	325	**768**	92,478	**7,467.0**	4373	**320**	779	**90,083**
RC204	**7,835.0**	6464	474	1221	143,464	**7,835.0**	**5174**	**450**	**1078**	**98,136**
RC207	**9,629.0**	**940**	**354**	**917**	21,801	**9,629.0**	1139	398	920	**20,313**
RC208	**7,761.0**	2116	326	1125	148,114	**7,761.0**	**2111**	**297**	**1063**	**136,516**
#Best	14/14	8/14	4/14	3/14	3/14	13/14	6/14	11/14	11/14	11/14

Table B13
Detailed results for comparing `PartialDom` versus `SetBased` settings using `only-cycle` DSSR before adding nonrobust cuts

	PartialDom					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3427	422	1332	19,401	**5,887.0**	**2400**	**375**	**1029**	**14,139**
C204	**5,881.0**	21,117	813	2623	94,674	**5,881.0**	**14,649**	**676**	**2182**	**80,236**
R202	**10,164.8**	304	130	558	9265	**10,164.8**	**249**	**106**	**445**	**8975**
R203	8,597.0	836	178	881	**28,547**	**8,597.6**	**625**	**136**	**670**	29,180
R204	7,201.6	2203	**184**	1159	96,974	**7,202.0**	**1638**	**184**	**874**	**66,260**
R206	**8,600.1**	646	123	828	36,038	**8,600.1**	**511**	**102**	**691**	**32,088**
R207	7,805.0	1175	153	896	57,911	**7,805.6**	**983**	**125**	**760**	**54,710**
R208	**6,910.9**	8774	268	2256	294,003	**6,910.9**	**5872**	**256**	**1741**	**203,856**
R209	**8,371.8**	358	93	592	24,458	**8,371.8**	**304**	**86**	**535**	**22,607**
R210	8,813.2	454	91	758	33,615	**8,813.8**	**376**	**83**	**626**	**30,214**
R211	7,311.0	665	**120**	862	72,642	**7,311.2**	**533**	128	**743**	**55,474**
RC204	**7,764.9**	20,875	320	3198	664,362	**7,764.9**	**10,746**	**275**	**2895**	**452,280**
RC207	**9,415.8**	850	**97**	1580	105,739	**9,415.8**	**619**	102	**1393**	**68,303**
RC208	**7,636.4**	3167	156	3164	583,749	**7,636.4**	**2363**	**136**	**2958**	**490,639**
#Best	9/14	0/14	3/14	0/14	1/14	14/14	14/14	12/14	14/14	13/14

Table B14
Detailed results for comparing `PartialDom` versus `SetBased` settings using `only-cycle` DSSR after adding nonrobust cuts

	PartialDom					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3428	422	1332	19,401	**5,887.0**	**2400**	**375**	**1029**	**14,139**
C204	**5,881.0**	21,118	813	2623	94,674	**5,881.0**	**14,649**	**676**	**2182**	**80,236**
R202	**10,296.0**	1012	860	1369	**3783**	**10,296.0**	**866**	**709**	**1140**	3792
R203	**8,708.0**	1292	640	1396	11,821	**8,708.0**	**1052**	**578**	**1175**	**11,568**
R204	**7,313.0**	**5288**	**654**	1868	63,573	**7,313.0**	5463	684	**1654**	**61,430**
R206	**8,759.0**	1648	488	1301	20,682	**8,759.0**	**1332**	**479**	**1198**	**18,776**
R207	**7,940.0**	2796	619	1533	**30,983**	**7,940.0**	**2348**	**529**	**1357**	31,385
R208	6,993.1	**176,704**	**625**	3186	**845,641**	**7,001.9**	185,708	680	**3094**	1,129,873
R209	**8,548.0**	2766	604	1433	**27,355**	**8,548.0**	**2171**	**596**	**1357**	27,691
R210	**9,005.0**	5233	**625**	1519	34,038	**9,005.0**	**4061**	661	**1433**	**33,007**
R211	**7,467.0**	9605	**340**	1519	177,736	**7,467.0**	**8884**	374	**1466**	**168,246**
RC204	**7,835.0**	22,986	593	3523	383,749	**7,835.0**	**13,049**	**561**	**3263**	**256,083**
RC207	**9,629.0**	2071	**453**	2121	41,165	**9,629.0**	**1617**	493	**1989**	**30,905**
RC208	**7,761.0**	5232	376	3646	346,185	**7,761.0**	**4157**	**341**	**3508**	**322,332**
#Best	13/14	2/14	5/14	0/14	4/14	14/14	12/14	9/14	14/14	10/14

Table B15
Detailed results for comparing `PartialDom` versus `Pairwise` settings using `only-cycle` DSSR before adding nonrobust cuts

	PartialDom					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3427	422	1332	19,401	**5,887.0**	**3060**	**360**	**1143**	**18,735**
C204	**5,881.0**	21,117	813	2623	94,674	**5,881.0**	**18,214**	**787**	**2089**	**86,049**
R202	**10,164.8**	304	130	558	9265	**10,164.8**	**252**	**99**	**461**	**9175**
R203	**8,597.0**	836	178	881	28,547	**8,597.0**	**691**	**145**	**734**	**26,536**
R204	7,201.6	2203	184	1159	96,974	**7,202.0**	**1753**	**166**	**944**	**80,729**
R206	**8,600.1**	646	123	828	36,038	**8,600.1**	**486**	**111**	**647**	**27,500**
R207	**7,805.0**	1175	153	896	57,911	**7,805.0**	**943**	**126**	**726**	**48,319**
R208	**6,910.9**	8774	**268**	2256	294,003	**6,910.9**	**6789**	272	**1920**	**218,370**
R209	**8,371.8**	358	93	592	**24,458**	**8,371.8**	**305**	**74**	**521**	24,866
R210	8,813.2	454	91	758	33,615	**8,814.4**	**363**	**72**	**632**	**33,020**
R211	7,311.0	665	**120**	862	72,642	**7,311.4**	**520**	125	**729**	**53,503**
RC204	**7,764.9**	20,875	320	3198	664,362	**7,764.9**	**10,090**	**282**	**2968**	**419,077**
RC207	**9,415.8**	850	97	1580	105,739	**9,415.8**	**635**	**94**	**1379**	**75,578**
RC208	**7,636.4**	3167	156	3164	583,749	**7,636.4**	**2280**	**135**	**2937**	**459,893**
#Best	11/14	0/14	2/14	0/14	1/14	14/14	14/14	12/14	14/14	13/14

Table B16
Detailed results for comparing `PartialDom` versus `Pairwise` settings using `only-cycle` DSSR after adding nonrobust cuts

	PartialDom					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3428	422	1332	19,401	**5,887.0**	**3060**	**360**	**1143**	**18,735**
C204	**5,881.0**	21,118	813	2623	94,674	**5,881.0**	**18,214**	**787**	**2089**	**86,049**
R202	**10,296.0**	1012	860	1369	3783	**10,296.0**	**860**	**802**	**1235**	**3368**
R203	**8,708.0**	1292	640	1396	11,821	**8,708.0**	**1148**	**635**	**1269**	**10,019**
R204	**7,313.0**	**5288**	**654**	1868	**63,573**	**7,313.0**	6206	657	**1776**	66,589
R206	**8,759.0**	1648	488	1301	20,682	**8,759.0**	**1395**	**482**	**1150**	**16,487**
R207	**7,940.0**	2796	619	1533	30,983	**7,940.0**	**2477**	**579**	**1330**	**28,540**
R208	6,993.1	**176,704**	**625**	3186	**845,641**	**7,001.9**	185,645	703	**3152**	1,046,169
R209	**8,548.0**	2766	**604**	1433	27,355	**8,548.0**	**2572**	614	**1387**	**26,942**
R210	**9,005.0**	5233	625	1519	34,038	**9,005.0**	**3842**	**598**	**1401**	**33,483**
R211	**7,467.0**	9605	**340**	1519	177,736	**7,467.0**	**9510**	388	**1461**	**162,013**
RC204	**7,835.0**	22,986	593	3523	383,749	**7,835.0**	**12,150**	**555**	**3311**	**246,708**
RC207	**9,629.0**	2071	453	2121	41,165	**9,629.0**	**1698**	**432**	**1942**	**34,256**
RC208	**7,761.0**	5232	376	3646	346,185	**7,761.0**	**3864**	**358**	**3470**	**282,017**
#Best	13/14	2/14	4/14	0/14	2/14	14/14	12/14	10/14	14/14	12/14

Table B17
Detailed results for comparing `PartialDom` versus `SetPair` settings using `only-cycle` DSSR before adding nonrobust cuts

	PartialDom					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3427	**422**	1332	19,401	**5,887.0**	**2737**	440	**1147**	**13,784**
C204	**5,881.0**	**21,117**	**813**	**2623**	94,674	**5,881.0**	25,892	1576	3019	**57,959**
R202	**10,164.8**	304	130	558	**9265**	**10,164.8**	**246**	**98**	**480**	9754
R203	**8,597.0**	836	178	881	28,547	**8,597.0**	**650**	**155**	**704**	**24,398**
R204	7,201.6	2203	184	1159	96,974	**7,202.0**	**1587**	**173**	**934**	**74,028**
R206	**8,600.1**	646	123	828	36,038	**8,600.1**	**533**	**116**	**711**	**27,342**
R207	**7,805.0**	1175	153	896	57,911	**7,805.1**	**954**	**149**	**727**	**44,896**
R208	**6,910.9**	8774	**268**	2256	294,003	**6,910.9**	**7087**	277	**1954**	**219,155**
R209	**8,371.8**	358	93	592	**24,458**	**8,371.8**	**316**	**80**	**560**	25,636
R210	**8,813.2**	454	91	758	33,615	**8,813.2**	**382**	**77**	**650**	**31,437**
R211	**7,311.0**	665	**120**	862	72,642	**7,311.0**	**542**	124	**739**	**55,515**
RC204	**7,764.9**	20,875	320	3198	664,362	**7,764.9**	**11,012**	**271**	**2815**	**441,036**
RC207	**9,415.8**	850	**97**	1580	105,739	**9,415.8**	**650**	**97**	**1366**	**73,623**
RC208	**7,636.4**	3167	156	3164	583,749	**7,636.4**	**2516**	**119**	**2952**	**581,679**
#Best	13/14	1/14	5/14	1/14	2/14	14/14	13/14	10/14	13/14	12/14

Table B18
Detailed results for comparing `PartialDom` versus `SetPair` settings using `only-cycle` DSSR after adding nonrobust cuts

	PartialDom					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	3428	**422**	1332	19,401	**5,887.0**	**2738**	440	**1147**	**13,784**
C204	**5,881.0**	**21,118**	**813**	**2623**	94,674	**5,881.0**	25,893	1576	3019	**57,959**
R202	**10,296.0**	1012	860	1369	3783	**10,296.0**	**762**	**659**	**1118**	**3654**
R203	**8,708.0**	1292	640	1396	11,821	**8,708.0**	**1080**	**600**	**1203**	**10,074**
R204	**7,313.0**	5288	654	1868	63,573	**7,313.0**	**5255**	**611**	**1621**	**62,939**
R206	**8,759.0**	1648	**488**	1301	20,682	**8,759.0**	**1586**	498	**1203**	**19,174**
R207	**7,940.0**	2796	619	1533	30,983	**7,940.0**	**2644**	**594**	**1367**	**29,710**
R208	6,993.1	**176,704**	**625**	3186	**845,641**	**6,996.4**	179,708	642	**2979**	855,593
R209	**8,548.0**	2766	604	1433	27,355	**8,548.0**	**2324**	**587**	**1370**	**25,630**
R210	**9,005.0**	5233	625	1519	34,038	**9,005.0**	**4742**	**620**	**1404**	**30,323**
R211	**7,467.0**	**9605**	**340**	1519	177,736	**7,467.0**	10,856	409	**1477**	**164,952**
RC204	**7,835.0**	22,986	593	3523	383,749	**7,835.0**	**13,148**	**549**	**3162**	**246,296**
RC207	**9,629.0**	2071	453	2121	41,165	**9,629.0**	**2021**	**445**	**1953**	**35,634**
RC208	**7,761.0**	5232	376	3646	346,185	**7,761.0**	**4309**	**353**	**3462**	**304,091**
#Best	13/14	3/14	5/14	1/14	1/14	14/14	11/14	9/14	13/14	13/14

Table B19
Detailed results for comparing `PartialDom` versus `SetBased` settings using `all-arcs` DSSR before adding nonrobust cuts

	PartialDom					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	1970	**307**	686	14,256	**5,887.0**	**1517**	324	**564**	**10,446**
C204	**5,881.0**	**12,418**	**757**	**1206**	57,927	**5,881.0**	19,889	1368	1808	**53,233**
R202	**10,164.8**	236	**115**	365	6995	**10,164.8**	**219**	116	**338**	**6198**
R203	**8,597.0**	538	146	511	**20,678**	**8,597.0**	**478**	**130**	**462**	21,196
R204	**7,202.0**	1338	170	610	**55,653**	7,201.6	**1110**	**139**	**523**	57,509
R206	**8,600.1**	419	98	445	**23,433**	**8,600.1**	**373**	**87**	**418**	24,467
R207	**7,805.0**	714	103	468	45,511	**7,805.0**	**653**	**97**	**453**	**43,833**
R208	**6,910.9**	**3486**	**164**	**777**	171,881	**6,910.9**	3608	211	808	**145,786**
R209	**8,371.8**	260	**66**	370	20,844	**8,371.8**	**245**	80	**364**	**17,652**
R210	**8,813.2**	314	**62**	458	30,685	**8,813.2**	**282**	75	**420**	**24,123**
R211	**7,311.0**	407	**95**	472	51,620	**7,311.0**	**348**	97	**435**	**43,420**
RC204	**7,764.9**	6220	**182**	936	273,583	**7,764.9**	**3413**	213	**829**	**160,031**
RC207	**9,415.8**	339	85	555	**35,834**	**9,415.8**	**295**	**68**	**521**	38,995
RC208	**7,636.4**	1036	103	888	240,585	**7,636.4**	**761**	**97**	**830**	**200,815**
#Best	14/14	2/14	8/14	2/14	4/14	13/14	12/14	6/14	12/14	10/14

Table B20
Detailed results for comparing `PartialDom` versus `SetBased` settings using `all-arcs` DSSR after adding nonrobust cuts

	PartialDom					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	1971	**307**	686	14,256	**5,887.0**	**1517**	324	**564**	**10,446**
C204	**5,881.0**	**12,419**	**757**	**1206**	57,927	**5,881.0**	19,890	1368	1808	**53,233**
R202	**10,296.0**	854	713	993	**3272**	**10,296.0**	**736**	**660**	**912**	3494
R203	**8,708.0**	948	563	943	**9147**	**8,708.0**	**830**	**502**	**864**	9819
R204	**7,313.0**	4712	564	1055	50,449	**7,313.0**	**3642**	**554**	**997**	**44,628**
R206	**8,759.0**	1194	**426**	**800**	15,427	**8,759.0**	**1099**	453	826	**14,891**
R207	**7,940.0**	**1854**	**475**	**893**	**24,935**	**7,940.0**	2083	505	907	27,022
R208	7,003.4	187,409	**459**	**1156**	**749,700**	**7,006.1**	**183,931**	558	1246	873,352
R209	**8,548.0**	2186	**533**	**931**	**20,620**	**8,548.0**	**2139**	596	958	22,394
R210	**9,005.0**	3623	**564**	1031	**28,315**	**9,005.0**	**3580**	576	**993**	30,062
R211	**7,467.0**	9167	**300**	799	130,566	**7,467.0**	**3985**	**300**	**759**	**98,913**
RC204	**7,835.0**	8414	**441**	1211	146,739	**7,835.0**	**5250**	493	**1127**	**97,395**
RC207	**9,629.0**	1226	**373**	**914**	21,959	**9,629.0**	**1056**	403	922	**20,609**
RC208	**7,761.0**	2521	**327**	1166	149,609	**7,761.0**	**1930**	331	**1109**	**131,734**
#Best	13/14	2/14	11/14	6/14	6/14	14/14	12/14	4/14	8/14	8/14

Table B21
Detailed results for comparing `PartialDom` versus `Pairwise` settings using `all-arcs` DSSR before adding nonrobust cuts

	PartialDom					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	1970	**307**	686	14,256	**5,887.0**	**1889**	491	**679**	**8099**
C204	**5,881.0**	**12,418**	**757**	**1206**	57,927	**5,881.0**	12,909	1091	1466	**37,265**
R202	**10,164.8**	236	115	365	6995	**10,164.8**	**210**	**104**	**334**	**6575**
R203	**8,597.0**	538	146	511	**20,678**	**8,597.0**	**479**	**133**	**473**	21,009
R204	**7,202.0**	1338	170	610	55,653	7,201.6	**1126**	**151**	**546**	**52,530**
R206	**8,600.1**	419	**98**	445	23,433	**8,600.1**	**360**	**98**	**420**	**19,897**
R207	**7,805.0**	714	103	468	**45,511**	**7,805.0**	**660**	**92**	**457**	47,529
R208	**6,910.9**	3486	**164**	777	171,881	**6,910.9**	**3077**	199	**742**	**130,870**
R209	**8,371.8**	260	**66**	370	20,844	**8,371.8**	**248**	79	**365**	**17,136**
R210	**8,813.2**	314	**62**	458	30,685	**8,813.2**	**277**	63	**416**	**25,960**
R211	**7,311.0**	407	**95**	472	51,620	**7,311.0**	**355**	110	**453**	**39,132**
RC204	**7,764.9**	6220	**182**	936	273,583	**7,764.9**	**3547**	191	**865**	**199,839**
RC207	**9,415.8**	339	85	555	35,834	**9,415.8**	**288**	**78**	**513**	**33,072**
RC208	**7,636.4**	1036	103	888	240,585	7,636.0	**759**	**81**	**820**	**239,051**
#Best	14/14	1/14	8/14	1/14	2/14	12/14	13/14	7/14	13/14	12/14

Table B22
Detailed results for comparing `PartialDom` versus `Pairwise` settings using `all-arcs` DSSR after adding nonrobust cuts

	PartialDom					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	1971	**307**	686	14,256	**5,887.0**	**1889**	491	**679**	**8099**
C204	**5,881.0**	**12,419**	**757**	**1206**	57,927	**5,881.0**	12,909	1091	1466	**37,265**
R202	**10,296.0**	854	713	993	3272	**10,296.0**	**699**	**622**	**891**	**3161**
R203	**8,708.0**	948	**563**	943	9147	**8,708.0**	**933**	575	**934**	**9083**
R204	**7,313.0**	4712	564	1055	50,449	**7,313.0**	**3887**	**548**	**991**	**49,736**
R206	**8,759.0**	1194	**426**	800	15,427	**8,759.0**	**1088**	444	**796**	**14,029**
R207	**7,940.0**	**1854**	475	893	**24,935**	**7,940.0**	1868	**468**	**874**	25,738
R208	7,003.4	**187,409**	**459**	**1156**	**749,700**	**7,008.4**	197,603	555	1195	962,739
R209	**8,548.0**	2186	**533**	931	20,620	**8,548.0**	**1709**	565	**929**	**19,346**
R210	**9,005.0**	**3623**	**564**	1031	28,315	**9,005.0**	3718	587	**1010**	**28,225**
R211	**7,467.0**	9167	**300**	799	130,566	**7,467.0**	**4350**	329	**792**	**91,532**
RC204	**7,835.0**	8414	441	1211	146,739	**7,835.0**	**4949**	**425**	**1115**	**116,605**
RC207	**9,629.0**	1226	**373**	914	21,959	**9,629.0**	**1046**	410	**907**	**19,866**
RC208	**7,761.0**	2521	327	1166	149,609	**7,761.0**	**1493**	**267**	**1041**	**136,976**
#Best	13/14	4/14	9/14	2/14	2/14	14/14	10/14	5/14	12/14	12/14

Table B23
Detailed results for comparing `PartialDom` versus `SetPair` settings using `all-arcs` DSSR before adding nonrobust cuts

	PartialDom					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	1970	**307**	686	14,256	**5,887.0**	**1726**	321	**637**	**12,275**
C204	**5,881.0**	**12,418**	**757**	**1206**	57,927	**5,881.0**	19,263	1526	1934	**46,212**
R202	**10,164.8**	236	115	365	6995	**10,164.8**	**208**	**102**	**332**	**6754**
R203	8,597.0	538	**146**	511	20,678	**8,598.4**	**469**	159	**493**	**18,365**
R204	**7,202.0**	1338	170	610	55,653	**7,202.0**	**1042**	**140**	**504**	**51,514**
R206	**8,600.1**	419	98	445	23,433	**8,600.1**	**356**	**97**	**405**	**20,186**
R207	**7,805.0**	714	103	468	45,511	**7,805.0**	**645**	**100**	**451**	**39,837**
R208	**6,910.9**	3486	**164**	777	171,881	**6,910.9**	**3477**	175	**770**	**153,871**
R209	**8,371.8**	260	**66**	370	20,844	**8,371.8**	**244**	70	**358**	**18,608**
R210	**8,813.2**	314	**62**	458	30,685	**8,813.2**	**295**	70	**448**	**25,301**
R211	**7,311.0**	407	**95**	472	51,620	**7,311.0**	**362**	108	**455**	**39,622**
RC204	**7,764.9**	6220	**182**	936	273,583	**7,764.9**	**3306**	201	**807**	**155,315**
RC207	**9,415.8**	339	85	555	**35,834**	**9,415.8**	**314**	**78**	**535**	36,044
RC208	**7,636.4**	1036	103	888	240,585	7,636.0	**765**	**92**	**797**	**195,395**
#Best	13/14	1/14	8/14	1/14	1/14	13/14	13/14	6/14	13/14	13/14

Table B24
Detailed results for comparing `PartialDom` versus `SetPair` settings using `all-arcs` DSSR after adding nonrobust cuts

	PartialDom					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	**5,887.0**	1971	**307**	686	14,256	**5,887.0**	**1727**	321	**637**	**12,275**
C204	**5,881.0**	**12,419**	**757**	**1206**	57,927	**5,881.0**	19,263	1526	1934	**46,212**
R202	**10,296.0**	854	713	993	3272	**10,296.0**	**724**	**618**	**874**	**3135**
R203	**8,708.0**	948	**563**	**943**	9147	**8,708.0**	**896**	650	999	**8447**
R204	**7,313.0**	4712	564	1055	50,449	**7,313.0**	**3916**	**544**	**959**	**45,432**
R206	**8,759.0**	1194	426	800	15,427	**8,759.0**	**1051**	**414**	**757**	**14,440**
R207	**7,940.0**	**1854**	**475**	893	24,935	**7,940.0**	1979	505	**885**	**23,459**
R208	7,003.4	**187,409**	**459**	**1156**	**749,700**	**7,005.3**	194,490	506	1170	764,528
R209	**8,548.0**	2186	533	931	20,620	**8,548.0**	**1911**	**520**	**886**	**19,705**
R210	**9,005.0**	**3623**	**564**	**1031**	**28,315**	**9,005.0**	4340	600	1054	28,787
R211	**7,467.0**	9167	**300**	799	130,566	**7,467.0**	**4373**	320	**779**	**90,083**
RC204	**7,835.0**	8414	**441**	1211	146,739	**7,835.0**	**5174**	450	**1078**	**98,136**
RC207	**9,629.0**	1226	**373**	**914**	21,959	**9,629.0**	**1139**	398	920	**20,313**
RC208	**7,761.0**	2521	327	1166	149,609	**7,761.0**	**2111**	**297**	**1063**	**136,516**
#Best	13/14	4/14	9/14	5/14	2/14	14/14	10/14	5/14	9/14	12/14

B.2. Gehring and Homberger's instances

Table B25
Detailed results for comparing `Default` versus `SetBased` settings using `only-cycle` DSSR before adding nonrobust cuts

	Default					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	427	**120**	1381	174,713	**26,597.4**	**319**	124	**1219**	**136,113**
C1_2_4	**26,197.7**	999	200	2834	390,363	**26,197.7**	**737**	**181**	**2582**	**371,003**
C1_2_9	**26,396.0**	245	**54**	748	126,697	**26,396.0**	**220**	56	**638**	**109,625**
C1_210	**26,197.3**	644	**71**	2201	579,885	**26,197.3**	**624**	85	**1961**	**453,549**
C2_2_1	**19,150.3**	348	**7**	**7**	**435**	**19,150.3**	**347**	**7**	**7**	**435**
C2_2_2	18,395.5	**6158**	**583**	1517	9693	**18,396.0**	6366	708	**1464**	**7354**
C2_2_5	**18,607.1**	**1346**	**141**	630	5369	**18,607.1**	1423	148	**565**	**4885**
C2_2_6	**18,406.7**	4696	377	1779	14,186	**18,406.7**	**4104**	**342**	**1528**	**13,307**
C2_2_7	18,356.4	7349	513	2698	22,335	**18,365.4**	**4811**	**331**	**1792**	**20,601**
C2_2_8	**18,082.4**	7841	434	2819	**31,722**	**18,082.4**	**5854**	**314**	**2206**	33,208
C2_2_9	**18,011.9**	12,604	**453**	3088	57,553	**18,011.9**	**10,311**	488	**2750**	**46,156**
R1_2_4	**30,009.9**	295	55	591	**299,338**	**30,009.9**	**232**	**46**	**522**	311,214
R1_2_5	**40,068.1**	42	**11**	**23**	**10,792**	**40,068.1**	**41**	**11**	**23**	10,864
R1_2_6	**34,930.6**	100	**44**	181	**46,927**	**34,930.6**	**97**	46	**180**	47,956
R1_2_7	**31,000.4**	271	50	451	201,404	30,999.1	**211**	**49**	**398**	**181,273**
R1_2_8	**29,032.3**	307	34	664	582,132	**29,032.3**	**246**	**31**	**595**	**559,334**
R1_2_9	**36,787.8**	64	**16**	126	56,508	**36,787.8**	**60**	**16**	**119**	**53,852**
R1_210	**32,451.1**	152	25	427	**222,464**	**32,451.1**	**139**	**23**	**399**	229,834
R2_2_1	34,625.1	**1724**	**66**	226	1610	**34,625.2**	1748	69	**212**	**1441**
R2_2_2	**30,006.1**	5304	**270**	1454	13,263	**30,006.1**	**4964**	288	**1244**	**10,589**
R2_2_5	**30,290.3**	2838	**148**	689	5854	**30,290.3**	**2755**	174	**658**	**4709**
R2_2_6	26,455.6	10,031	435	2601	41,857	**26,456.2**	**8379**	**397**	**2141**	**33,639**
R2_2_9	28,206.0	4140	227	1220	13,690	**28,206.4**	**4003**	**214**	**1009**	**11,066**
RC1_2_1	**34,595.9**	57	18	133	**44,460**	**34,595.9**	**53**	**17**	**131**	46,746
RC1_2_2	**31,774.7**	136	**37**	347	124,573	**31,774.7**	**123**	38	**300**	**110,289**
RC1_2_6	**32,554.4**	150	**19**	683	421,929	**32,554.4**	**131**	**19**	**645**	**411,015**
RC1_2_8	**30,216.1**	479	35	1504	**1,154,526**	**30,216.1**	**359**	**31**	**1439**	1,207,744
RC2_2_1	27,891.0	2018	**125**	707	5248	**27,891.3**	**1859**	131	**643**	**4500**
RC2_2_2	24,588.6	11,640	582	3940	37,446	**24,589.8**	**9569**	**568**	**3218**	**28,827**
RC2_2_5	**24,788.5**	7731	519	4326	92,047	**24,788.6**	**6100**	**444**	**3622**	**76,239**
RC2_2_6	**24,724.4**	5808	414	3467	50,584	**24,724.4**	**4782**	**372**	**2965**	**42,500**
#Best	24/31	3/31	16/31	2/31	8/31	30/31	28/31	19/31	31/31	24/31

Table B26
Detailed results for comparing `Default` versus `SetBased` settings using `only-cycle` DSSR after adding nonrobust cuts

	Default					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,752.7**	174,077	1903	4274	278,173	**26,752.6**	**173,517**	**1884**	**4137**	**262,970**
C1_2_4	**26,256.0**	1403	**348**	3605	331,293	**26,256.0**	**1152**	358	**3376**	**288,520**
C1_2_9	**26,396.0**	245	**54**	748	126,697	**26,396.0**	**221**	56	**638**	**109,625**
C1_210	**26,247.0**	861	**196**	2995	304,343	**26,247.0**	**843**	214	**2629**	**264,985**
C2_2_1	**19,221.0**	1007	**414**	**414**	1405	**19,221.0**	**914**	479	479	**1033**
C2_2_2	**18,514.0**	13,187	1830	2979	7847	**18,514.0**	**11,811**	**1558**	**2420**	**7697**
C2_2_5	**18,696.0**	2463	717	1269	**3294**	**18,696.0**	**2292**	**615**	**1073**	3539
C2_2_6	**18,448.0**	5770	**716**	2277	10,098	**18,448.0**	**5180**	737	**2008**	**8921**
C2_2_7	18,422.0	8523	875	3117	15,455	**18,423.3**	**5572**	**572**	**2114**	**14,329**
C2_2_8	**18,137.0**	9446	800	3348	22,370	**18,137.0**	**7534**	**706**	**2764**	**21,520**
C2_2_9	**18,150.0**	22,001	1049	4311	49,739	**18,150.0**	**17,741**	**1010**	**3810**	**45,067**
R1_2_4	**30,381.8**	178,343	928	2520	**467,945**	30,381.2	**177,114**	**903**	**2504**	478,440
R1_2_5	**40,471.9**	2034	694	706	**10,679**	40,471.6	**1782**	**628**	**640**	10,706
R1_2_6	**35,486.2**	26,455	**1209**	**1706**	80,659	35,485.7	**22,703**	1220	1711	**79,309**
R1_2_7	**31,404.5**	40,594	1061	2223	216,549	31,404.4	**38,044**	**1036**	**2189**	**212,635**
R1_2_8	29,345.9	**173,240**	**1104**	**3018**	506,789	**29,346.2**	173,559	1165	3086	**502,407**
R1_2_9	**37,284.5**	4118	778	983	27,555	37,283.6	**2992**	**665**	**864**	**26,756**
R1_210	**32,812.4**	16,387	1086	1963	**94,172**	32,812.1	**16,045**	**1073**	**1951**	96,043
R2_2_1	**34,680.0**	**1851**	**215**	**386**	860	**34,680.0**	1880	272	430	**743**
R2_2_2	**30,082.0**	6677	686	1955	6538	**30,082.0**	**5896**	**599**	**1580**	**6130**
R2_2_5	**30,611.0**	7700	1864	2583	**3892**	**30,611.0**	**6898**	**1640**	**2270**	4224
R2_2_6	**26,750.5**	26,558	2995	5367	13,371	**26,750.5**	**22,113**	**2473**	**4426**	**12,679**
R2_2_9	**28,433.0**	6913	**1261**	2334	5796	**28,433.0**	**6841**	1269	**2166**	**5016**
RC1_2_1	**35,017.8**	4855	938	1080	20,427	35,016.8	**3331**	**770**	**909**	**19,631**
RC1_2_2	32,087.2	93,147	**1114**	2025	141,613	**32,087.7**	**90,927**	1139	**1980**	**140,173**
RC1_2_6	**32,960.2**	37,964	1269	2740	**145,496**	**32,960.2**	**32,890**	**1250**	**2715**	151,225
RC1_2_8	30,508.3	**173,052**	701	**4572**	**1,087,318**	**30,508.7**	176,286	**686**	4591	1,160,747
RC2_2_1	**27,974.0**	2789	1239	1959	**1448**	**27,974.0**	**2483**	**913**	**1549**	1690
RC2_2_2	**24,780.0**	74,326	7382	11,203	9576	**24,780.0**	**63,458**	**6212**	**9261**	**9275**
RC2_2_5	**24,914.0**	19,218	3954	8621	30,624	**24,914.0**	**13,672**	**3099**	**6510**	**25,079**
RC2_2_6	**24,951.0**	**11,396**	**2406**	6109	18,346	**24,951.0**	12,105	2737	**5862**	**15,752**
#Best	27/31	4/31	11/31	5/31	8/31	24/31	27/31	20/31	26/31	23/31

Table B27
Detailed results for comparing `Default` versus `Pairwise` settings using `only-cycle` DSSR before adding nonrobust cuts

	Default					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	427	**120**	1381	174,713	**26,597.4**	**395**	126	**1284**	**149,825**
C1_2_4	**26,197.7**	999	200	2834	390,363	**26,197.7**	**889**	**199**	**2534**	**318,126**
C1_2_9	**26,396.0**	245	**54**	748	126,697	**26,396.0**	**230**	56	**629**	**101,539**
C1_210	**26,197.3**	**644**	**71**	2201	579,885	**26,197.3**	701	87	**1956**	**409,693**
C2_2_1	**19,150.3**	348	**7**	**7**	**435**	**19,150.3**	**346**	**7**	**7**	**435**
C2_2_2	18,395.5	6158	**583**	1517	9693	**18,396.0**	**5180**	593	**1187**	**6541**
C2_2_5	**18,607.1**	**1346**	**141**	630	5369	**18,607.1**	1514	168	**594**	**4328**
C2_2_6	**18,406.7**	4696	377	1779	14,186	**18,406.7**	**4340**	**363**	**1504**	**11,640**
C2_2_7	**18,356.4**	7349	513	2698	22,335	**18,356.4**	**4483**	**285**	**1846**	**21,371**
C2_2_8	**18,082.4**	7841	434	2819	31,722	**18,082.4**	**6293**	**380**	**2164**	**24,979**
C2_2_9	**18,011.9**	12,604	453	3088	57,553	**18,011.9**	**9410**	**421**	**2482**	**40,634**
R1_2_4	**30,009.9**	295	55	591	299,338	**30,009.9**	**287**	**49**	**539**	**292,363**
R1_2_5	**40,068.1**	42	**11**	**23**	**10,792**	**40,068.1**	**42**	**11**	**23**	**10,792**
R1_2_6	**34,930.6**	**100**	**44**	181	46,927	**34,930.6**	106	47	**172**	**44,706**
R1_2_7	**31,000.4**	271	50	451	201,404	31,000.0	**242**	**46**	**406**	**188,565**
R1_2_8	**29,032.3**	**307**	**34**	664	582,132	**29,032.3**	334	36	**619**	**506,166**
R1_2_9	**36,787.8**	**64**	**16**	126	56,508	**36,787.8**	64	**16**	**125**	**56,068**
R1_210	**32,451.1**	**152**	**25**	427	222,464	**32,451.1**	157	**25**	**397**	**206,001**
R2_2_1	34,625.1	1724	66	226	1610	**34,625.2**	**1711**	**63**	**209**	**1530**
R2_2_2	**30,006.1**	5304	**270**	1454	13,263	**30,006.1**	**4585**	276	**1223**	**10,192**
R2_2_5	**30,290.3**	2838	148	689	5854	**30,290.3**	**2621**	**133**	**590**	**5236**
R2_2_6	26,455.6	10,031	435	2601	41,857	**26,456.0**	**8471**	**384**	**2066**	**32,993**
R2_2_9	**28,206.0**	4140	227	1220	13,690	**28,206.0**	**4076**	**201**	**1015**	**11,496**
RC1_2_1	**34,595.9**	57	**18**	**133**	**44,460**	**34,595.9**	**56**	**18**	**133**	**44,460**
RC1_2_2	**31,774.7**	136	**37**	347	124,573	**31,774.7**	**132**	41	**325**	**103,418**
RC1_2_6	**32,554.4**	150	**19**	683	421,929	**32,554.4**	**149**	20	**636**	**371,121**
RC1_2_8	**30,216.1**	479	35	1504	1,154,526	**30,216.1**	**430**	**33**	**1435**	**1,091,526**
RC2_2_1	27,891.0	2018	125	707	5248	**27,891.3**	**1972**	**124**	**601**	**4258**
RC2_2_2	24,588.6	11,640	582	3940	37,446	**24,590.6**	**9328**	**516**	**3210**	**29,730**
RC2_2_5	**24,788.5**	7731	519	4326	92,047	**24,788.6**	**6530**	**480**	**3813**	**74,813**
RC2_2_6	**24,724.4**	5808	414	3467	50,584	**24,724.4**	**4921**	**393**	**2956**	**38,082**
#Best	26/31	6/31	15/31	3/31	3/31	30/31	25/31	21/31	31/31	31/31

Table B28
Detailed results for comparing `Default` versus `Pairwise` settings using `only-cycle` DSSR after adding nonrobust cuts

	Default					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,752.7**	**174,077**	1903	4274	278,173	26,752.2	176,153	**1843**	**4017**	**270,184**
C1_2_4	**26,256.0**	1403	348	3605	331,293	**26,256.0**	**1294**	**343**	**3336**	**292,342**
C1_2_9	**26,396.0**	245	**54**	748	126,697	**26,396.0**	**231**	56	**629**	**101,539**
C1_210	**26,247.0**	**861**	**196**	2995	304,343	**26,247.0**	950	206	**2708**	**266,016**
C2_2_1	**19,221.0**	**1007**	**414**	**414**	**1405**	**19,221.0**	1009	**414**	**414**	**1405**
C2_2_2	**18,514.0**	**13,187**	1830	2979	**7847**	**18,514.0**	13,283	**1667**	**2415**	7983
C2_2_5	**18,696.0**	2463	717	1269	3294	**18,696.0**	**2231**	**572**	**1034**	**2913**
C2_2_6	**18,448.0**	5770	716	2277	10,098	**18,448.0**	**5322**	**653**	**1897**	**8643**
C2_2_7	**18,422.0**	8523	875	3117	15,455	**18,422.0**	**6271**	**874**	**2625**	**10,538**
C2_2_8	**18,137.0**	9446	**800**	3348	22,370	**18,137.0**	**8065**	832	**2832**	**17,545**
C2_2_9	**18,150.0**	22,001	**1049**	4311	49,739	**18,150.0**	**18,612**	1087	**3794**	**40,620**
R1_2_4	**30,381.8**	178,343	928	2520	467,945	**30,381.8**	**174,763**	**903**	**2485**	**464,605**
R1_2_5	**40,471.9**	2034	**694**	**706**	**10,679**	**40,471.9**	**1981**	**694**	**706**	**10,679**
R1_2_6	**35,486.2**	**26,455**	**1209**	**1706**	80,659	35,485.3	28,276	1317	1814	**80,153**
R1_2_7	**31,404.5**	**40,594**	**1061**	**2223**	216,549	31,404.4	45,569	1131	2263	**206,701**
R1_2_8	**29,345.9**	**173,240**	**1104**	**3018**	506,789	29,345.6	173,387	1106	3034	**501,744**
R1_2_9	**37,284.5**	4118	778	983	27,555	37,283.4	**3049**	**659**	**872**	**27,008**
R1_210	**32,812.4**	16,387	1086	1963	94,172	32,811.1	**11,744**	**937**	**1766**	**88,250**
R2_2_1	**34,680.0**	1851	**215**	386	860	**34,680.0**	**1837**	225	**382**	**800**
R2_2_2	**30,082.0**	6677	686	1955	6538	**30,082.0**	**5421**	**580**	**1562**	**5808**
R2_2_5	**30,611.0**	7700	1864	2583	3892	**30,611.0**	**6254**	**1461**	**2089**	**3566**
R2_2_6	**26,750.5**	26,558	2995	5367	13,371	**26,750.5**	**21,719**	**2291**	**4223**	**12,972**
R2_2_9	**28,433.0**	6913	1261	2334	5796	**28,433.0**	**6903**	**1236**	**2141**	**5033**
RC1_2_1	**35,017.8**	4855	**938**	**1080**	**20,427**	**35,017.8**	**4825**	**938**	**1080**	**20,427**
RC1_2_2	32,087.2	**93,147**	**1114**	2025	141,613	**32,087.6**	108,209	1140	**1986**	**137,733**
RC1_2_6	32,960.2	**37,964**	**1269**	**2740**	145,496	**32,960.4**	38,157	**1269**	2745	**141,760**
RC1_2_8	30,508.3	**173,052**	701	4572	1,087,318	**30,509.0**	175,969	**697**	**4543**	**996,846**
RC2_2_1	**27,974.0**	2789	1239	1959	**1448**	**27,974.0**	**2735**	**1205**	**1875**	1464
RC2_2_2	**24,780.0**	74,326	7382	11,203	9576	**24,780.0**	**65,053**	**6954**	**10,101**	**8688**
RC2_2_5	**24,914.0**	**19,218**	**3954**	**8621**	30,624	**24,914.0**	28,143	6011	10,046	**20,619**
RC2_2_6	**24,951.0**	11,396	2406	6109	18,346	**24,951.0**	**8944**	**1944**	**5107**	**16,935**
#Best	28/31	11/31	14/31	8/31	5/31	25/31	20/31	21/31	26/31	29/31

Table B29
Detailed results for comparing `Default` versus `SetPair` settings using `only-cycle` DSSR before adding nonrobust cuts

	Default					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	427	**120**	1381	174,713	**26,597.4**	**360**	133	**1237**	**128,396**
C1_2_4	**26,197.7**	999	**200**	**2834**	390,363	**26,197.7**	**791**	**200**	2945	**380,210**
C1_2_9	**26,396.0**	245	**54**	748	126,697	**26,396.0**	**223**	58	**663**	**102,973**
C1_210	**26,197.3**	644	**71**	2201	579,885	**26,197.3**	**588**	85	**1917**	**411,479**
C2_2_1	**19,150.3**	348	**7**	**7**	**435**	**19,150.3**	**347**	**7**	**7**	**435**
C2_2_2	18,395.5	**6158**	**583**	**1517**	9693	**18,396.0**	8386	724	1664	**9571**
C2_2_5	**18,607.1**	1346	141	630	5369	**18,607.1**	**1276**	**133**	**529**	**4859**
C2_2_6	**18,406.7**	4696	377	1779	14,186	**18,406.7**	**4084**	**295**	**1455**	**13,866**
C2_2_7	18,356.4	7349	513	2698	22,335	**18,370.4**	**5458**	**393**	**2034**	**18,636**
C2_2_8	**18,082.4**	7841	434	2819	31,722	**18,082.4**	**5788**	**380**	**2244**	**25,504**
C2_2_9	**18,011.9**	12,604	**453**	3088	57,553	**18,011.9**	**11,220**	469	**2672**	**43,541**
R1_2_4	**30,009.9**	295	55	591	**299,338**	**30,009.9**	**222**	**42**	**498**	315,121
R1_2_5	**40,068.1**	42	11	23	**10,792**	**40,068.1**	**41**	**10**	**22**	11,161
R1_2_6	**34,930.6**	100	**44**	181	46,927	**34,930.6**	**95**	46	**171**	**44,276**
R1_2_7	**31,000.4**	271	50	451	201,404	30,998.9	**209**	**46**	**415**	**194,376**
R1_2_8	**29,032.3**	307	**34**	664	582,132	**29,032.3**	**255**	**34**	**593**	**508,686**
R1_2_9	**36,787.8**	64	**16**	126	56,508	**36,787.8**	**59**	**16**	**119**	**52,895**
R1_210	**32,451.1**	152	**25**	427	222,464	**32,451.1**	**142**	**25**	**398**	**206,501**
R2_2_1	34,625.1	1724	66	226	1610	**34,625.2**	**1717**	**65**	**211**	**1505**
R2_2_2	**30,006.1**	5304	**270**	1454	13,263	**30,006.1**	**4871**	282	**1214**	**10,282**
R2_2_5	**30,290.3**	2838	**148**	689	5854	**30,290.3**	**2783**	173	**623**	**4274**
R2_2_6	**26,455.6**	10,031	435	2601	41,857	**26,455.6**	**8248**	**365**	**2054**	**37,270**
R2_2_9	28,206.0	4140	**227**	1220	13,690	**28,206.2**	**4049**	229	**1012**	**10,103**
RC1_2_1	**34,595.9**	57	**18**	**133**	44,460	**34,595.9**	**54**	**18**	**133**	**44,324**
RC1_2_2	**31,774.7**	136	**37**	347	124,573	**31,774.7**	**119**	43	**315**	**99,413**
RC1_2_6	32,554.4	150	**19**	683	421,929	**32,554.6**	**136**	22	**651**	**343,279**
RC1_2_8	**30,216.1**	479	35	1504	1,154,526	**30,216.1**	**359**	**34**	**1390**	**998,532**
RC2_2_1	27,891.0	2018	125	707	5248	**27,891.3**	**1890**	**118**	**618**	**4553**
RC2_2_2	24,588.6	11,640	582	3940	37,446	**24,590.6**	**9181**	**511**	**3102**	**29,002**
RC2_2_5	**24,788.5**	7731	519	4326	92,047	**24,788.5**	**6277**	**448**	**3744**	**76,733**
RC2_2_6	**24,724.4**	5808	414	3467	50,584	**24,724.4**	**4856**	**367**	**2929**	**41,101**
#Best	24/31	1/31	17/31	4/31	3/31	30/31	30/31	20/31	29/31	29/31

Table B30
Detailed results for comparing `Default` versus `SetPair` settings using `only-cycle` DSSR after adding nonrobust cuts

	Default					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,752.7**	174,077	1903	4274	278,173	26,751.3	**172,971**	**1841**	**3991**	**242,010**
C1_2_4	**26,256.0**	1403	**348**	**3605**	331,293	**26,256.0**	**1287**	359	3703	**312,205**
C1_2_9	**26,396.0**	245	**54**	748	126,697	**26,396.0**	**224**	58	**663**	**102,973**
C1_210	**26,247.0**	861	**196**	2995	304,343	**26,247.0**	**818**	213	**2686**	**251,061**
C2_2_1	**19,221.0**	**1007**	**414**	**414**	1405	**19,221.0**	1086	566	566	**1147**
C2_2_2	**18,514.0**	**13,187**	**1830**	2979	**7847**	**18,514.0**	14,946	1838	**2841**	7905
C2_2_5	**18,696.0**	2463	717	1269	3294	**18,696.0**	**2123**	**555**	**984**	**3258**
C2_2_6	**18,448.0**	5770	716	2277	10,098	**18,448.0**	**5215**	**635**	**1953**	**9316**
C2_2_7	**18,422.0**	8523	**875**	3117	15,455	**18,422.0**	**7570**	912	**2930**	**13,018**
C2_2_8	**18,137.0**	9446	**800**	3348	22,370	**18,137.0**	**7544**	802	**2879**	**17,826**
C2_2_9	**18,150.0**	**22,001**	**1049**	4311	49,739	**18,150.0**	22,022	1389	**4258**	**34,605**
R1_2_4	**30,381.8**	178,343	928	2520	467,945	30,378.9	**174,117**	**782**	**2329**	**451,245**
R1_2_5	**40,471.9**	**2034**	694	706	**10,679**	40,471.0	2040	**684**	**697**	10,716
R1_2_6	**35,486.2**	**26,455**	**1209**	**1706**	80,659	35,485.3	30,800	1298	1802	**79,940**
R1_2_7	**31,404.5**	**40,594**	1061	2223	**216,549**	**31,404.6**	50,470	**1015**	**2161**	219,204
R1_2_8	**29,345.9**	**173,240**	1104	3018	506,789	29,345.5	175,173	**1073**	**2980**	**485,150**
R1_2_9	**37,284.5**	4118	778	983	27,555	37,284.1	**3390**	**711**	**914**	**26,937**
R1_210	**32,812.4**	16,387	1086	1963	94,172	32,811.6	**14,648**	**1001**	**1870**	**91,813**
R2_2_1	**34,680.0**	1851	215	386	860	**34,680.0**	**1804**	**189**	**338**	**844**
R2_2_2	**30,082.0**	6677	686	1955	6538	**30,082.0**	**5806**	**592**	**1540**	**5887**
R2_2_5	**30,611.0**	7700	1864	2583	3892	**30,611.0**	**6547**	**1633**	**2205**	**3392**
R2_2_6	**26,750.5**	26,558	2995	5367	13,371	**26,750.5**	**21,322**	**2457**	**4378**	**12,350**
R2_2_9	**28,433.0**	**6913**	**1261**	2334	5796	**28,433.0**	6925	1279	**2178**	**4904**
RC1_2_1	**35,017.8**	4855	938	1080	20,427	35,017.3	**4427**	**831**	**969**	**20,095**
RC1_2_2	32,087.2	**93,147**	**1114**	2025	141,613	**32,088.0**	118,036	1186	**2005**	**135,460**
RC1_2_6	32,960.2	**37,964**	**1269**	**2740**	145,496	**32,960.6**	46,136	1289	2757	**143,502**
RC1_2_8	**30,508.3**	**173,052**	701	4572	1,087,318	30,505.7	179,781	**623**	**4417**	**1,046,100**
RC2_2_1	**27,974.0**	2789	1239	1959	**1448**	**27,974.0**	**2542**	**976**	**1631**	1501
RC2_2_2	**24,780.0**	74,326	7382	11,203	9576	**24,780.0**	**41,251**	**4738**	**7679**	**8973**
RC2_2_5	**24,914.0**	**19,218**	**3954**	8621	30,624	**24,914.0**	19,812	4388	**8265**	**22,480**
RC2_2_6	**24,951.0**	11,396	2406	6109	18,346	**24,951.0**	**10,238**	**2196**	**5446**	**16,623**
#Best	29/31	12/31	13/31	4/31	4/31	22/31	19/31	18/31	27/31	27/31

Table B31
Detailed results for comparing `Default` versus `SetBased` settings using `all-arcs` DSSR before adding nonrobust cuts

	Default					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	249	**106**	478	71,318	**26,597.4**	**215**	**106**	**452**	**65,229**
C1_2_4	**26,197.7**	484	**162**	**902**	169,325	**26,197.7**	**431**	182	906	**134,003**
C1_2_9	**26,396.0**	225	**55**	559	93,896	**26,396.0**	**207**	57	**522**	**88,052**
C1_210	26,197.3	457	**64**	1189	347,848	**26,197.4**	**452**	78	**1122**	**280,835**
C2_2_1	**19,150.3**	358	**7**	**7**	**435**	**19,150.3**	**353**	**7**	**7**	**435**
C2_2_2	**18,396.0**	12,186	1176	1406	6656	**18,396.0**	**3346**	**432**	**619**	**5484**
C2_2_5	**18,607.1**	1328	153	538	**4423**	**18,607.1**	**1277**	**137**	**493**	4644
C2_2_6	**18,406.7**	**2901**	**266**	997	10,609	**18,406.7**	3476	328	**993**	**9659**
C2_2_7	**18,356.4**	4014	401	1262	**12,577**	**18,356.4**	**3598**	**324**	**1090**	14,090
C2_2_8	**18,082.4**	4019	319	1297	**19,148**	**18,082.4**	**3516**	**273**	**1152**	20,758
C2_2_9	**18,011.9**	5466	**284**	1315	38,099	**18,011.9**	**5194**	318	**1207**	**32,809**
R1_2_4	**30,009.9**	210	**41**	320	210,946	**30,009.9**	**177**	46	**296**	**170,241**
R1_2_5	**40,068.1**	42	**11**	**23**	**10,792**	**40,068.1**	**42**	**11**	**23**	10,864
R1_2_6	**34,930.6**	**94**	**39**	**134**	40,197	**34,930.6**	94	47	151	**39,320**
R1_2_7	**31,000.8**	206	**42**	295	152,863	31,000.5	**194**	52	**282**	**115,984**
R1_2_8	**29,032.3**	218	39	**344**	**263,303**	**29,032.2**	**200**	**33**	360	330,244
R1_2_9	**36,787.8**	62	**16**	113	49,861	**36,787.8**	**58**	17	**110**	**45,953**
R1_210	**32,451.1**	131	**21**	311	191,263	**32,451.1**	**120**	23	**302**	**170,625**
R2_2_1	**34,625.1**	1668	61	199	1518	**34,625.1**	**1625**	**58**	**185**	**1499**
R2_2_2	**30,006.1**	4211	287	927	**8767**	**30,006.1**	**4085**	**248**	**825**	8971
R2_2_5	**30,290.3**	2655	165	527	**4055**	**30,290.3**	**2473**	**140**	**460**	4139
R2_2_6	**26,455.5**	5783	**288**	1203	28,978	**26,455.5**	**5669**	**288**	**1076**	**25,726**
R2_2_9	**28,206.0**	**3298**	201	697	8490	**28,206.0**	3371	**175**	**617**	**8318**
RC1_2_1	**34,595.9**	56	18	133	**44,460**	**34,595.9**	**54**	**17**	**131**	46,746
RC1_2_2	**31,774.8**	122	41	**263**	**90,794**	**31,774.7**	**117**	**37**	266	101,657
RC1_2_6	**32,554.4**	112	22	442	**232,263**	**32,554.4**	**101**	**18**	**424**	285,058
RC1_2_8	**30,216.1**	246	33	580	**482,789**	**30,216.1**	**196**	**29**	**572**	542,817
RC2_2_1	**27,891.0**	**1751**	131	540	3854	**27,891.0**	1867	**129**	**475**	**3534**
RC2_2_2	24,588.6	6270	472	1763	**20,783**	**24,589.4**	**5806**	**369**	**1524**	23,857
RC2_2_5	**24,788.5**	3723	329	1464	48,659	**24,788.5**	**3245**	**264**	**1305**	**47,348**
RC2_2_6	**24,724.4**	**3241**	**278**	1315	27,118	**24,724.4**	3249	**278**	**1277**	**25,921**
#Best	29/31	5/31	15/31	6/31	13/31	30/31	26/31	21/31	27/31	19/31

Table B32
Detailed results for comparing `Default` versus `SetBased` settings using `all-arcs` DSSR after adding nonrobust cuts

	Default					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	26,752.8	175,608	**1865**	**2526**	233,199	**26,753.2**	**173,111**	1985	2635	**228,628**
C1_2_4	**26,256.0**	**708**	**295**	**1162**	146,396	**26,256.0**	709	350	1205	**120,837**
C1_2_9	**26,396.0**	226	**55**	559	93,896	**26,396.0**	**208**	57	**522**	**88,052**
C1_210	**26,247.0**	**610**	**188**	1593	173,425	**26,247.0**	625	198	**1505**	**168,608**
C2_2_1	**19,221.0**	1037	**414**	**414**	1405	**19,221.0**	**935**	479	479	**1033**
C2_2_2	**18,514.0**	18,036	1937	2186	**7081**	**18,514.0**	**8511**	**1296**	**1505**	7360
C2_2_5	**18,696.0**	2259	692	1125	**2962**	**18,696.0**	**2031**	**556**	**947**	3272
C2_2_6	**18,448.0**	**3725**	**576**	**1346**	**7082**	**18,448.0**	4766	795	1498	7195
C2_2_7	**18,422.0**	**5128**	**795**	**1679**	8776	**18,422.0**	9523	1994	2891	**8236**
C2_2_8	**18,137.0**	5213	**676**	1677	14,250	**18,137.0**	**5002**	705	**1654**	**13,938**
C2_2_9	**18,150.0**	**10,850**	**855**	2056	30,447	**18,150.0**	10,928	934	**1969**	**29,016**
R1_2_4	**30,382.9**	178,950	995	1620	392,300	30,382.5	**174,279**	**932**	**1501**	**375,703**
R1_2_5	**40,471.9**	2027	694	706	**10,679**	40,471.6	**1730**	**628**	**640**	10,706
R1_2_6	35,485.6	**25,055**	**1295**	1576	**71,917**	**35,485.8**	26,704	1306	**1574**	75,130
R1_2_7	**31,404.6**	**29,284**	**989**	1515	**172,649**	31,404.3	30,365	1002	**1493**	173,444
R1_2_8	29,346.5	174,477	**1215**	**1904**	449,750	**29,346.8**	**174,211**	1311	1988	**439,039**
R1_2_9	**37,284.2**	3658	729	912	**26,890**	37,283.9	**3026**	**643**	**817**	27,223
R1_210	32,811.1	**12,433**	**996**	**1514**	**80,401**	**32,811.5**	13,001	1030	1551	83,754
R2_2_1	**34,680.0**	**1769**	**194**	**337**	829	**34,680.0**	1811	294	430	**725**
R2_2_2	**30,082.0**	**4859**	**508**	**1158**	5904	**30,082.0**	5343	706	1294	**4597**
R2_2_5	**30,611.0**	**6339**	**1629**	**2029**	**3579**	**30,611.0**	6727	1709	2106	3886
R2_2_6	**26,750.5**	19,476	2620	3582	11,358	**26,750.5**	**18,306**	**2363**	**3232**	**10,879**
R2_2_9	**28,433.0**	6074	1251	1794	**4570**	**28,433.0**	**5944**	**1113**	**1605**	4762
RC1_2_1	**35,017.8**	4874	938	1080	20,427	35,016.8	**3385**	**770**	**909**	**19,631**
RC1_2_2	32,087.7	82,456	**1092**	**1586**	123,907	**32,087.8**	**78,701**	1132	1628	**120,445**
RC1_2_6	32,959.7	30,687	1196	2086	**124,809**	**32,960.2**	**27,114**	**1193**	**2044**	126,487
RC1_2_8	30,514.2	180,988	898	2108	**566,568**	**30,514.7**	**172,960**	**897**	**2072**	598,704
RC2_2_1	**27,974.0**	**2230**	**873**	**1335**	**1447**	**27,974.0**	2507	940	1382	1619
RC2_2_2	**24,780.0**	**63,098**	**7402**	**8837**	**8189**	**24,780.0**	70,794	7815	9078	9037
RC2_2_5	**24,914.0**	16,895	4447	5684	**18,991**	**24,914.0**	**10,548**	**2593**	**3713**	23,611
RC2_2_6	**24,951.0**	8612	2473	3561	**11,569**	**24,951.0**	**7557**	**1932**	**2976**	12,034
#Best	24/31	13/31	19/31	13/31	16/31	26/31	18/31	12/31	18/31	15/31

Table B33
Detailed results for comparing `Default` versus `Pairwise` settings using `all-arcs` DSSR before adding nonrobust cuts

	Default					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	**249**	**106**	478	71,318	**26,597.4**	259	112	**451**	**61,612**
C1_2_4	**26,197.7**	**484**	**162**	**902**	169,325	**26,197.7**	492	179	951	**157,224**
C1_2_9	**26,396.0**	225	**55**	559	93,896	**26,396.0**	**223**	**55**	**516**	**85,621**
C1_210	**26,197.3**	457	**64**	1189	347,848	**26,197.3**	**454**	70	**1156**	**302,044**
C2_2_1	**19,150.3**	358	**7**	**7**	**435**	**19,150.3**	**346**	**7**	**7**	**435**
C2_2_2	**18,396.0**	12,186	1176	1406	6656	18,394.0	**3258**	**416**	**612**	**5208**
C2_2_5	**18,607.1**	1328	153	538	**4423**	**18,607.1**	**1281**	**128**	**490**	4733
C2_2_6	**18,406.7**	2901	266	997	10,609	**18,406.7**	**2809**	**263**	**935**	**9440**
C2_2_7	**18,356.4**	4014	401	1262	**12,577**	**18,356.4**	**2985**	**265**	**998**	13,361
C2_2_8	**18,082.4**	4019	319	1297	19,148	**18,082.4**	**3533**	**297**	**1146**	**16,726**
C2_2_9	**18,011.9**	5466	**284**	1315	38,099	18,011.8	**5259**	311	**1234**	**30,655**
R1_2_4	**30,009.9**	**210**	**41**	320	210,946	**30,009.9**	229	44	**308**	**189,214**
R1_2_5	**40,068.1**	**42**	**11**	**23**	**10,792**	**40,068.1**	42	**11**	**23**	**10,792**
R1_2_6	**34,930.6**	**94**	**39**	**134**	40,197	34,930.5	97	46	146	**36,742**
R1_2_7	**31,000.8**	206	**42**	295	152,863	31,000.5	**199**	49	**271**	**117,794**
R1_2_8	**29,032.3**	**218**	39	344	**263,303**	**29,032.4**	226	**35**	**343**	290,122
R1_2_9	**36,787.8**	62	**16**	113	49,861	**36,787.8**	**60**	**16**	**109**	**47,384**
R1_210	**32,451.1**	**131**	**21**	311	191,263	**32,451.1**	132	22	**302**	**174,792**
R2_2_1	**34,625.1**	**1668**	**61**	**199**	1518	**34,625.1**	1719	76	210	**1263**
R2_2_2	**30,006.1**	4211	287	927	**8767**	**30,006.1**	**3917**	**242**	**824**	8778
R2_2_5	**30,290.3**	**2655**	165	527	4055	**30,290.3**	2665	**158**	**490**	**3761**
R2_2_6	**26,455.5**	5783	288	1203	28,978	**26,455.5**	**5601**	**280**	**1099**	**26,008**
R2_2_9	**28,206.0**	3298	201	697	8490	**28,206.0**	**3272**	**171**	**622**	**8243**
RC1_2_1	**34,595.9**	56	**18**	**133**	**44,460**	**34,595.9**	**56**	**18**	**133**	**44,460**
RC1_2_2	**31,774.8**	122	41	263	**90,794**	**31,774.7**	**117**	**39**	**257**	90,986
RC1_2_6	**32,554.4**	112	22	442	**232,263**	**32,554.4**	**109**	**19**	**423**	253,881
RC1_2_8	**30,216.1**	246	33	580	**482,789**	**30,216.1**	**226**	**28**	**571**	528,193
RC2_2_1	27,891.0	**1751**	131	540	3854	**27,891.3**	1841	**125**	**477**	**3532**
RC2_2_2	24,588.6	6270	472	1763	20,783	**24,588.8**	**6091**	**421**	**1561**	**19,566**
RC2_2_5	**24,788.5**	3723	329	1464	48,659	**24,788.5**	**3408**	**286**	**1343**	**43,608**
RC2_2_6	**24,724.4**	3241	278	1315	27,118	**24,724.4**	**3035**	**269**	**1196**	**23,394**
#Best	29/31	10/31	14/31	6/31	10/31	27/31	21/31	22/31	28/31	24/31

Table B34
Detailed results for comparing `Default` versus `Pairwise` settings using `all-arcs` DSSR after adding nonrobust cuts

	Default					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,752.8**	175,608	**1865**	**2526**	233,199	**26,752.8**	**174,314**	1951	2601	**223,555**
C1_2_4	**26,256.0**	**708**	**295**	**1162**	146,396	**26,256.0**	750	334	1209	**128,992**
C1_2_9	**26,396.0**	226	**55**	559	93,896	**26,396.0**	**224**	**55**	**516**	**85,621**
C1_210	**26,247.0**	**610**	**188**	1593	173,425	**26,247.0**	623	205	**1547**	**151,855**
C2_2_1	**19,221.0**	1037	**414**	**414**	**1405**	**19,221.0**	**1011**	**414**	**414**	**1405**
C2_2_2	**18,514.0**	18,036	1937	2186	**7081**	**18,514.0**	**12,010**	**1754**	**1965**	7878
C2_2_5	**18,696.0**	2259	692	1125	2962	**18,696.0**	**2108**	**602**	**995**	**2823**
C2_2_6	**18,448.0**	**3725**	**576**	**1346**	7082	**18,448.0**	3823	630	**1346**	**6806**
C2_2_7	**18,422.0**	5128	795	1679	8776	**18,422.0**	**3959**	**614**	**1390**	**8432**
C2_2_8	**18,137.0**	5213	**676**	1677	14,250	**18,137.0**	**4882**	712	**1615**	**11,765**
C2_2_9	**18,150.0**	**10,850**	**855**	**2056**	30,447	**18,150.0**	11,728	1066	2166	**25,098**
R1_2_4	**30,382.9**	178,950	995	1620	392,300	30,382.1	**177,048**	**916**	**1479**	**372,593**
R1_2_5	**40,471.9**	**2027**	**694**	**706**	**10,679**	**40,471.9**	2039	**694**	**706**	**10,679**
R1_2_6	**35,485.6**	25,055	1295	1576	**71,917**	**35,485.6**	**22,911**	**1211**	**1480**	72,105
R1_2_7	31,404.6	**29,284**	**989**	1515	**172,649**	**31,404.7**	31,416	1033	**1507**	172,905
R1_2_8	29,346.5	174,477	**1215**	**1904**	449,750	**29,346.8**	**174,130**	1286	1969	**415,519**
R1_2_9	**37,284.2**	3658	729	912	26,890	37,284.0	**3451**	**701**	**866**	**26,581**
R1_210	32,811.1	**12,433**	**996**	**1514**	80,401	**32,811.5**	12,644	1009	1530	**79,696**
R2_2_1	**34,680.0**	**1769**	**194**	**337**	829	**34,680.0**	1915	353	494	**678**
R2_2_2	**30,082.0**	**4859**	**508**	**1158**	5904	**30,082.0**	4902	596	1190	**4794**
R2_2_5	**30,611.0**	6339	1629	2029	**3579**	**30,611.0**	**5967**	**1405**	**1778**	3636
R2_2_6	**26,750.5**	19,476	2620	3582	11,358	**26,750.5**	**18,934**	**2493**	**3392**	**10,006**
R2_2_9	**28,433.0**	6074	1251	1794	4570	**28,433.0**	**5714**	**1089**	**1566**	**4442**
RC1_2_1	**35,017.8**	4874	**938**	**1080**	**20,427**	**35,017.8**	**4832**	**938**	**1080**	**20,427**
RC1_2_2	**32,087.7**	**82,456**	**1092**	**1586**	**123,907**	32,087.6	97,015	1156	1627	124,469
RC1_2_6	32,959.7	**30,687**	**1196**	**2086**	124,809	**32,960.2**	33,762	1229	2092	**122,444**
RC1_2_8	**30,514.2**	180,988	898	2108	566,568	**30,514.1**	**173,242**	**874**	**2085**	**547,801**
RC2_2_1	**27,974.0**	**2230**	**873**	**1335**	1447	**27,974.0**	2468	1022	1442	**1380**
RC2_2_2	**24,780.0**	63,098	7402	8837	8189	**24,780.0**	**51,013**	**6063**	**7308**	**7725**
RC2_2_5	**24,914.0**	16,895	4447	5684	18,991	**24,914.0**	**11,536**	**2890**	**3996**	**18,575**
RC2_2_6	**24,951.0**	**8612**	**2473**	**3561**	11,569	**24,951.0**	12,398	3136	4117	**10,638**
#Best	27/31	13/31	19/31	15/31	8/31	28/31	18/31	16/31	20/31	26/31

Table B35
Detailed results for comparing `Default` versus `SetPair` settings using `all-arcs` DSSR before adding non-robust cuts

	Default					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	249	106	478	**71,318**	**26,597.4**	**184**	**104**	**468**	71,359
C1_2_4	**26,197.7**	484	**162**	902	169,325	**26,197.7**	**435**	171	**816**	**137,328**
C1_2_9	**26,396.0**	225	55	559	93,896	**26,396.0**	**215**	**52**	**523**	**91,451**
C1_210	**26,197.3**	**457**	**64**	1189	347,848	**26,197.3**	476	69	**1053**	**281,031**
C2_2_1	**19,150.3**	358	**7**	**7**	**435**	**19,150.3**	**348**	**7**	**7**	**435**
C2_2_2	**18,396.0**	12,186	1176	1406	6656	**18,396.0**	**3495**	**440**	**695**	**6377**
C2_2_5	**18,607.1**	1328	153	538	**4423**	**18,607.1**	**1234**	**130**	**498**	4742
C2_2_6	**18,406.7**	2901	266	997	**10,609**	**18,406.7**	**2781**	**199**	**931**	12,609
C2_2_7	**18,356.4**	4014	401	1262	**12,577**	**18,356.4**	**3097**	**234**	**979**	15,181
C2_2_8	**18,082.4**	4019	319	1297	19,148	**18,082.4**	**3449**	**264**	**1138**	**19,031**
C2_2_9	**18,011.9**	5466	**284**	1315	38,099	18,011.8	**5014**	289	**1212**	**32,366**
R1_2_4	**30,009.9**	210	**41**	320	210,946	**30,009.9**	**185**	44	**313**	**188,885**
R1_2_5	**40,068.1**	42	11	23	**10,792**	**40,068.1**	**41**	**10**	**22**	11,161
R1_2_6	**34,930.6**	94	**39**	**134**	40,197	**34,930.6**	**93**	46	159	**39,948**
R1_2_7	**31,000.8**	206	**42**	295	152,863	31,000.2	**177**	44	**286**	**138,251**
R1_2_8	**29,032.3**	218	39	**344**	**263,303**	**29,032.3**	**183**	**35**	345	275,634
R1_2_9	**36,787.8**	62	**16**	113	49,861	**36,787.8**	**58**	**16**	**109**	**47,399**
R1_210	**32,451.1**	131	21	311	191,263	**32,451.1**	**116**	**19**	**284**	**189,812**
R2_2_1	**34,625.1**	1668	**61**	199	1518	**34,625.1**	**1666**	70	**197**	**1290**
R2_2_2	**30,006.1**	4211	287	927	**8767**	**30,006.1**	**3895**	**222**	**846**	9565
R2_2_5	**30,290.3**	2655	**165**	527	4055	**30,290.3**	**2536**	182	**514**	**3457**
R2_2_6	**26,455.5**	5783	288	1203	28,978	**26,455.5**	**5477**	**273**	**1092**	**27,355**
R2_2_9	**28,206.0**	**3298**	201	697	8490	**28,206.0**	3416	**176**	**633**	**8129**
RC1_2_1	**34,595.9**	56	**18**	**133**	44,460	**34,595.9**	**54**	**18**	**133**	**44,324**
RC1_2_2	**31,774.8**	122	**41**	263	90,794	**31,774.7**	**116**	43	**262**	**84,050**
RC1_2_6	**32,554.4**	112	22	442	**232,263**	**32,554.4**	**100**	**20**	**423**	245,260
RC1_2_8	**30,216.1**	246	33	580	**482,789**	**30,216.1**	**196**	**31**	**572**	490,921
RC2_2_1	27,891.0	**1751**	**131**	540	3854	**27,892.5**	1818	**131**	**487**	**3448**
RC2_2_2	24,588.6	6270	472	1763	20,783	**24,588.8**	**5330**	**379**	**1492**	**20,620**
RC2_2_5	**24,788.5**	3723	**329**	1464	48,659	**24,788.5**	**3324**	330	**1394**	**37,539**
RC2_2_6	**24,724.4**	3241	278	1315	**27,118**	**24,724.4**	**2898**	**207**	**1163**	29,369
#Best	29/31	3/31	14/31	4/31	11/31	29/31	28/31	21/31	29/31	21/31

Table B36
Detailed results for comparing `Default` versus `SetPair` settings using `all-arcs` DSSR after adding nonrobust cuts

	Default					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,752.8**	175,608	1865	2526	233,199	26,751.9	**174,346**	**1821**	**2460**	**211,811**
C1_2_4	**26,256.0**	**708**	**295**	1162	146,396	**26,256.0**	800	343	**1153**	**129,352**
C1_2_9	**26,396.0**	226	55	559	93,896	**26,396.0**	**216**	**52**	**523**	**91,451**
C1_210	**26,247.0**	**610**	**188**	1593	173,425	**26,247.0**	628	192	**1483**	**157,517**
C2_2_1	**19,221.0**	**1037**	**414**	**414**	1405	**19,221.0**	1091	566	566	**1147**
C2_2_2	**18,514.0**	18,036	1937	2186	**7081**	**18,514.0**	**10,819**	**1366**	**1633**	7385
C2_2_5	**18,696.0**	2259	692	1125	2962	**18,696.0**	**2211**	**688**	**1104**	**2863**
C2_2_6	**18,448.0**	3725	576	1346	**7082**	**18,448.0**	**3617**	**498**	**1252**	7873
C2_2_7	**18,422.0**	5128	795	1679	8776	**18,422.0**	**4147**	**620**	**1381**	**8749**
C2_2_8	**18,137.0**	5213	676	1677	14,250	**18,137.0**	**4637**	**593**	**1523**	**13,588**
C2_2_9	**18,150.0**	10,850	**855**	2056	30,447	**18,150.0**	**10,488**	899	**1967**	**26,966**
R1_2_4	**30,382.9**	**178,950**	995	1620	392,300	30,380.0	179,044	**842**	**1441**	**366,007**
R1_2_5	**40,471.9**	2027	694	706	**10,679**	40,471.0	**1986**	**684**	**697**	10,716
R1_2_6	**35,485.6**	**25,055**	1295	1576	**71,917**	35,485.3	27,965	**1205**	**1505**	74,688
R1_2_7	**31,404.6**	**29,284**	989	1515	172,649	**31,404.6**	33,773	**968**	**1461**	**168,974**
R1_2_8	**29,346.5**	**174,477**	1215	1904	449,750	29,346.1	174,610	**1147**	**1846**	**416,161**
R1_2_9	**37,284.2**	3658	729	912	26,890	37,283.2	**3421**	**698**	**863**	**26,282**
R1_210	32,811.1	**12,433**	**996**	**1514**	**80,401**	**32,812.3**	17,806	1146	1665	82,338
R2_2_1	**34,680.0**	**1769**	**194**	**337**	829	**34,680.0**	1783	218	354	**756**
R2_2_2	**30,082.0**	4859	508	1158	5904	**30,082.0**	**4646**	**495**	**1132**	**5412**
R2_2_5	**30,611.0**	6339	**1629**	**2029**	**3579**	**30,611.0**	**6300**	1652	2040	3690
R2_2_6	**26,750.5**	19,476	2620	3582	11,358	**26,750.5**	**17,219**	**2235**	**3139**	**10,640**
R2_2_9	**28,433.0**	**6074**	1251	1794	4570	**28,433.0**	6143	**1236**	**1726**	**4360**
RC1_2_1	**35,017.8**	4874	938	1080	20,427	35,017.3	**4376**	**831**	**969**	**20,095**
RC1_2_2	**32,087.7**	**82,456**	**1092**	**1586**	123,907	**32,087.6**	110,251	1132	1618	**118,878**
RC1_2_6	32,959.7	**30,687**	**1196**	2086	**124,809**	**32,960.3**	37,278	1203	**2053**	125,815
RC1_2_8	**30,514.2**	180,988	898	2108	566,568	30,512.5	**173,314**	**830**	**1976**	**540,561**
RC2_2_1	**27,974.0**	**2230**	873	1335	**1447**	**27,974.0**	2314	**839**	**1279**	1464
RC2_2_2	**24,780.0**	63,098	7402	8837	8189	**24,780.0**	**43,946**	**5701**	**6928**	**8103**
RC2_2_5	**24,914.0**	16,895	4447	5684	**18,991**	**24,914.0**	**8939**	**2407**	**3533**	19,953
RC2_2_6	**24,951.0**	8612	2473	3561	**11,569**	**24,951.0**	**7700**	**2189**	**3232**	11,829
#Best	29/31	13/31	9/31	5/31	10/31	23/31	18/31	22/31	26/31	21/31

Table B37
Detailed results for comparing `PartialDom` versus `SetBased` settings using `only-cycle` DSSR before adding nonrobust cuts

	PartialDom					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	394	**124**	1385	173,802	**26,597.4**	**319**	**124**	**1219**	**136,113**
C1_2_4	**26,197.7**	1110	196	2999	422,665	**26,197.7**	**737**	**181**	**2582**	**371,003**
C1_2_9	**26,396.0**	274	62	694	**103,678**	**26,396.0**	**220**	**56**	**638**	109,625
C1_210	**26,197.3**	795	**83**	2249	524,649	**26,197.3**	**624**	85	**1961**	**453,549**
C2_2_1	**19,150.3**	371	**7**	**7**	**435**	**19,150.3**	**347**	**7**	**7**	**435**
C2_2_2	18,394.0	11,347	773	2043	11,156	**18,396.0**	**6366**	**708**	**1464**	**7354**
C2_2_5	**18,607.1**	1542	170	656	**4847**	**18,607.1**	**1423**	**148**	**565**	4885
C2_2_6	**18,406.7**	4451	**327**	1673	14,377	**18,406.7**	**4104**	342	**1528**	**13,307**
C2_2_7	18,356.4	7352	480	2461	21,229	**18,365.4**	**4811**	**331**	**1792**	**20,601**
C2_2_8	**18,082.9**	7479	369	2749	35,935	18,082.4	**5854**	**314**	**2206**	**33,208**
C2_2_9	**18,011.9**	13,406	510	3142	52,033	**18,011.9**	**10,311**	**488**	**2750**	**46,156**
R1_2_4	**30,009.9**	328	**44**	576	360,513	**30,009.9**	**232**	46	**522**	**311,214**
R1_2_5	**40,068.1**	44	**11**	**23**	**10,796**	**40,068.1**	**41**	**11**	**23**	10,864
R1_2_6	34,930.1	106	**43**	**167**	**45,657**	**34,930.6**	**97**	46	180	47,956
R1_2_7	**30,999.1**	298	54	441	182,425	**30,999.1**	**211**	**49**	**398**	**181,273**
R1_2_8	**29,032.4**	388	33	648	595,520	**29,032.3**	**246**	**31**	**595**	**559,334**
R1_2_9	**36,787.8**	66	**16**	123	54,489	**36,787.8**	**60**	**16**	**119**	**53,852**
R1_210	**32,451.1**	165	**21**	414	256,912	**32,451.1**	**139**	23	**399**	**229,834**
R2_2_1	34,625.1	**1743**	71	234	1546	**34,625.2**	1748	**69**	**212**	**1441**
R2_2_2	**30,006.1**	6068	328	1575	11,681	**30,006.1**	**4964**	**288**	**1244**	**10,589**
R2_2_5	**30,290.3**	3124	197	741	4739	**30,290.3**	**2755**	**174**	**658**	**4709**
R2_2_6	26,455.5	9531	**377**	2641	48,567	**26,456.2**	**8379**	397	**2141**	**33,639**
R2_2_9	28,206.0	4379	**203**	1167	14,379	**28,206.4**	**4003**	214	**1009**	**11,066**
RC1_2_1	**34,595.9**	60	18	133	**44,486**	**34,595.9**	**53**	**17**	**131**	46,746
RC1_2_2	**31,774.7**	135	43	353	**109,271**	**31,774.7**	**123**	**38**	**300**	110,289
RC1_2_6	**32,554.4**	171	**19**	703	432,336	**32,554.4**	**131**	**19**	**645**	**411,015**
RC1_2_8	**30,216.1**	540	36	1576	**1,173,078**	**30,216.1**	**359**	**31**	**1439**	1,207,744
RC2_2_1	27,891.0	2024	**123**	720	5359	**27,891.3**	**1859**	131	**643**	**4500**
RC2_2_2	24,588.6	11,026	595	3696	33,713	**24,589.8**	**9569**	**568**	**3218**	**28,827**
RC2_2_5	**24,788.5**	7814	445	4284	108,258	**24,788.6**	**6100**	**444**	**3622**	**76,239**
RC2_2_6	**24,724.4**	5913	396	3411	52,168	**24,724.4**	**4782**	**372**	**2965**	**42,500**
#Best	23/31	1/31	13/31	3/31	8/31	30/31	30/31	23/31	30/31	24/31

Table B38
Detailed results for comparing `PartialDom` versus `SetBased` settings using `only-cycle` DSSR after adding nonrobust cuts

	PartialDom					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	26,751.8	**173,438**	**1861**	**4059**	264,466	**26,752.6**	173,517	1884	4137	**262,970**
C1_2_4	**26,256.0**	1620	**347**	3768	346,619	**26,256.0**	**1152**	358	**3376**	**288,520**
C1_2_9	**26,396.0**	275	62	694	**103,678**	**26,396.0**	**221**	**56**	**638**	109,625
C1_210	**26,247.0**	1063	**203**	3037	308,752	**26,247.0**	**843**	214	**2629**	**264,985**
C2_2_1	**19,221.0**	1116	587	587	**954**	**19,221.0**	**914**	**479**	**479**	1033
C2_2_2	**18,514.0**	16,282	1678	3034	8686	**18,514.0**	**11,811**	**1558**	**2420**	**7697**
C2_2_5	**18,696.0**	2444	675	1239	**2999**	**18,696.0**	**2292**	**615**	**1073**	3539
C2_2_6	**18,448.0**	5433	**661**	2097	9613	**18,448.0**	**5180**	737	**2008**	**8921**
C2_2_7	18,422.0	8425	806	2854	15,121	**18,423.3**	**5572**	**572**	**2114**	**14,329**
C2_2_8	**18,137.0**	9548	813	3447	23,318	**18,137.0**	**7534**	**706**	**2764**	**21,520**
C2_2_9	**18,150.0**	22,898	1146	4461	47,370	**18,150.0**	**17,741**	**1010**	**3810**	**45,067**
R1_2_4	30,379.9	178,129	**844**	**2444**	**468,566**	**30,381.2**	**177,114**	903	2504	478,440
R1_2_5	40,471.0	**1620**	**573**	**585**	**10,317**	**40,471.6**	1782	628	640	10,706
R1_2_6	35,485.2	28,930	**1193**	**1686**	81,141	**35,485.7**	**22,703**	1220	1711	**79,309**
R1_2_7	**31,404.4**	51,878	1111	2243	215,687	**31,404.4**	**38,044**	**1036**	**2189**	**212,635**
R1_2_8	29,344.9	175,303	**1038**	**2954**	**480,037**	**29,346.2**	**173,559**	1165	3086	502,407
R1_2_9	**37,284.2**	3711	670	875	27,485	37,283.6	**2992**	**665**	**864**	**26,756**
R1_210	32,811.9	17,630	1094	1996	**93,539**	**32,812.1**	**16,045**	**1073**	**1951**	96,043
R2_2_1	**34,680.0**	**1871**	**214**	**387**	864	**34,680.0**	1880	272	430	**743**
R2_2_2	**30,082.0**	7216	710	1981	6415	**30,082.0**	**5896**	**599**	**1580**	**6130**
R2_2_5	**30,611.0**	8223	1955	2677	**4004**	**30,611.0**	**6898**	**1640**	**2270**	4224
R2_2_6	**26,750.5**	23,238	2556	5012	14,381	**26,750.5**	**22,113**	**2473**	**4426**	**12,679**
R2_2_9	**28,433.0**	7057	**1144**	2220	5899	**28,433.0**	**6841**	1269	**2166**	**5016**
RC1_2_1	**35,017.2**	4623	868	1007	20,579	35,016.8	**3331**	**770**	**909**	**19,631**
RC1_2_2	**32,087.9**	94,153	**1111**	2012	142,075	32,087.7	**90,927**	1139	**1980**	**140,173**
RC1_2_6	**32,960.3**	36,527	**1245**	**2714**	**141,759**	32,960.2	**32,890**	1250	2715	151,225
RC1_2_8	**30,508.7**	179,057	702	4660	**1,121,911**	**30,508.7**	**176,286**	**686**	**4591**	1,160,747
RC2_2_1	**27,974.0**	2766	1083	1813	**1617**	**27,974.0**	**2483**	**913**	**1549**	1690
RC2_2_2	**24,780.0**	65,777	7220	10,924	9515	**24,780.0**	**63,458**	**6212**	**9261**	**9275**
RC2_2_5	**24,914.0**	14,037	**2384**	6699	36,953	**24,914.0**	**13,672**	3099	**6510**	**25,079**
RC2_2_6	**24,951.0**	**11,845**	**2346**	5953	19,456	**24,951.0**	12,105	2737	**5862**	**15,752**
#Best	24/31	4/31	14/31	7/31	11/31	27/31	27/31	17/31	24/31	20/31

Table B39
Detailed results for comparing `PartialDom` versus `Pairwise` settings using `only-cycle` DSSR before adding nonrobust cuts

	PartialDom					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	**394**	**124**	1385	173,802	**26,597.4**	395	126	**1284**	**149,825**
C1_2_4	**26,197.7**	1110	**196**	2999	422,665	**26,197.7**	**889**	199	**2534**	**318,126**
C1_2_9	**26,396.0**	274	62	694	103,678	**26,396.0**	**230**	**56**	**629**	**101,539**
C1_210	**26,197.3**	795	**83**	2249	524,649	**26,197.3**	**701**	87	**1956**	**409,693**
C2_2_1	**19,150.3**	371	**7**	**7**	**435**	**19,150.3**	**346**	**7**	**7**	**435**
C2_2_2	18,394.0	11,347	773	2043	11,156	**18,396.0**	**5180**	**593**	**1187**	**6541**
C2_2_5	**18,607.1**	1542	170	656	4847	**18,607.1**	**1514**	**168**	**594**	**4328**
C2_2_6	**18,406.7**	4451	**327**	1673	14,377	**18,406.7**	**4340**	363	**1504**	**11,640**
C2_2_7	**18,356.4**	7352	480	2461	**21,229**	**18,356.4**	**4483**	**285**	**1846**	21,371
C2_2_8	**18,082.9**	7479	**369**	2749	35,935	18,082.4	**6293**	380	**2164**	**24,979**
C2_2_9	**18,011.9**	13,406	510	3142	52,033	**18,011.9**	**9410**	**421**	**2482**	**40,634**
R1_2_4	**30,009.9**	328	**44**	576	360,513	**30,009.9**	**287**	49	**539**	**292,363**
R1_2_5	**40,068.1**	44	**11**	**23**	10,796	**40,068.1**	**42**	**11**	**23**	**10,792**
R1_2_6	34,930.1	106	**43**	**167**	45,657	**34,930.6**	**106**	47	172	**44,706**
R1_2_7	30,999.1	298	54	441	**182,425**	**31,000.0**	**242**	**46**	**406**	188,565
R1_2_8	**29,032.4**	388	**33**	648	595,520	**29,032.3**	**334**	36	**619**	**506,166**
R1_2_9	**36,787.8**	66	**16**	**123**	**54,489**	**36,787.8**	**64**	**16**	125	56,068
R1_210	**32,451.1**	165	**21**	414	256,912	**32,451.1**	**157**	25	**397**	**206,001**
R2_2_1	34,625.1	1743	71	234	1546	**34,625.2**	**1711**	**63**	**209**	**1530**
R2_2_2	**30,006.1**	6068	328	1575	11,681	**30,006.1**	**4585**	**276**	**1223**	**10,192**
R2_2_5	**30,290.3**	3124	197	741	**4739**	**30,290.3**	**2621**	**133**	**590**	5236
R2_2_6	26,455.5	9531	**377**	2641	48,567	**26,456.0**	**8471**	384	**2066**	**32,993**
R2_2_9	**28,206.0**	4379	203	1167	14,379	**28,206.0**	**4076**	**201**	**1015**	**11,496**
RC1_2_1	**34,595.9**	60	**18**	**133**	44,486	**34,595.9**	**56**	**18**	**133**	**44,460**
RC1_2_2	**31,774.7**	135	43	353	109,271	**31,774.7**	**132**	**41**	**325**	**103,418**
RC1_2_6	**32,554.4**	171	**19**	703	432,336	**32,554.4**	**149**	20	**636**	**371,121**
RC1_2_8	**30,216.1**	540	36	1576	1,173,078	**30,216.1**	**430**	**33**	**1435**	**1,091,526**
RC2_2_1	27,891.0	2024	**123**	720	5359	**27,891.3**	**1972**	124	**601**	**4258**
RC2_2_2	24,588.6	11,026	595	3696	33,713	**24,590.6**	**9328**	**516**	**3210**	**29,730**
RC2_2_5	**24,788.5**	7814	**445**	4284	108,258	**24,788.6**	**6530**	480	**3813**	**74,813**
RC2_2_6	**24,724.4**	5913	396	3411	52,168	**24,724.4**	**4921**	**393**	**2956**	**38,082**
#Best	24/31	1/31	17/31	5/31	5/31	30/31	30/31	18/31	29/31	27/31

Table B40
Detailed results for comparing `PartialDom` versus `Pairwise` settings using `only-cycle` DSSR after adding nonrobust cuts

	PartialDom					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	26,751.8	**173,438**	1861	4059	**264,466**	**26,752.2**	176,153	**1843**	**4017**	270,184
C1_2_4	**26,256.0**	1620	347	3768	346,619	**26,256.0**	**1294**	**343**	**3336**	**292,342**
C1_2_9	**26,396.0**	275	62	694	103,678	**26,396.0**	**231**	**56**	**629**	**101,539**
C1_210	**26,247.0**	1063	**203**	3037	308,752	**26,247.0**	**950**	206	**2708**	**266,016**
C2_2_1	**19,221.0**	1116	587	587	**954**	**19,221.0**	**1009**	**414**	**414**	1405
C2_2_2	**18,514.0**	16,282	1678	3034	8686	**18,514.0**	**13,283**	**1667**	**2415**	**7983**
C2_2_5	**18,696.0**	2444	675	1239	2999	**18,696.0**	**2231**	**572**	**1034**	**2913**
C2_2_6	**18,448.0**	5433	661	2097	9613	**18,448.0**	**5322**	**653**	**1897**	**8643**
C2_2_7	**18,422.0**	8425	**806**	2854	15,121	**18,422.0**	**6271**	874	**2625**	**10,538**
C2_2_8	**18,137.0**	9548	**813**	3447	23,318	**18,137.0**	**8065**	832	**2832**	**17,545**
C2_2_9	**18,150.0**	22,898	1146	4461	47,370	**18,150.0**	**18,612**	**1087**	**3794**	**40,620**
R1_2_4	30,379.9	178,129	**844**	**2444**	468,566	**30,381.8**	**174,763**	903	2485	**464,605**
R1_2_5	40,471.0	**1620**	**573**	**585**	**10,317**	**40,471.9**	1981	694	706	10,679
R1_2_6	35,485.2	28,930	**1193**	**1686**	81,141	**35,485.3**	**28,276**	1317	1814	**80,153**
R1_2_7	**31,404.4**	51,878	**1111**	**2243**	215,687	**31,404.4**	**45,569**	1131	2263	**206,701**
R1_2_8	29,344.9	175,303	**1038**	**2954**	**480,037**	**29,345.6**	**173,387**	1106	3034	501,744
R1_2_9	**37,284.2**	3711	670	875	27,485	37,283.4	**3049**	**659**	**872**	**27,008**
R1_210	**32,811.9**	17,630	1094	1996	93,539	32,811.1	**11,744**	**937**	**1766**	**88,250**
R2_2_1	**34,680.0**	1871	**214**	387	864	**34,680.0**	**1837**	225	**382**	**800**
R2_2_2	**30,082.0**	7216	710	1981	6415	**30,082.0**	**5421**	**580**	**1562**	**5808**
R2_2_5	**30,611.0**	8223	1955	2677	4004	**30,611.0**	**6254**	**1461**	**2089**	**3566**
R2_2_6	**26,750.5**	23,238	2556	5012	14,381	**26,750.5**	**21,719**	**2291**	**4223**	**12,972**
R2_2_9	**28,433.0**	7057	**1144**	2220	5899	**28,433.0**	**6903**	1236	**2141**	**5033**
RC1_2_1	35,017.2	**4623**	**868**	**1007**	20,579	**35,017.8**	4825	938	1080	**20,427**
RC1_2_2	**32,087.9**	**94,153**	**1111**	2012	142,075	32,087.6	108,209	1140	**1986**	**137,733**
RC1_2_6	**32,960.3**	**36,527**	**1245**	**2714**	**141,759**	**32,960.4**	38,157	1269	2745	141,760
RC1_2_8	30,508.7	179,057	702	4660	1,121,911	**30,509.0**	**175,969**	**697**	**4543**	**996,846**
RC2_2_1	**27,974.0**	2766	**1083**	**1813**	1617	**27,974.0**	**2735**	1205	1875	**1464**
RC2_2_2	**24,780.0**	65,777	7220	10,924	9515	**24,780.0**	**65,053**	**6954**	**10,101**	**8688**
RC2_2_5	**24,914.0**	**14,037**	**2384**	**6699**	36,953	**24,914.0**	28,143	6011	10,046	**20,619**
RC2_2_6	**24,951.0**	11,845	2346	5953	19,456	**24,951.0**	**8944**	**1944**	**5107**	**16,935**
#Best	24/31	6/31	15/31	9/31	5/31	28/31	25/31	16/31	22/31	26/31

Table B41
Detailed results for comparing `PartialDom` versus `SetPair` settings using `only-cycle` DSSR before adding nonrobust cuts

	PartialDom					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	394	**124**	1385	173,802	**26,597.4**	**360**	133	**1237**	**128,396**
C1_2_4	**26,197.7**	1110	**196**	2999	422,665	**26,197.7**	**791**	200	**2945**	**380,210**
C1_2_9	**26,396.0**	274	62	694	103,678	**26,396.0**	**223**	**58**	**663**	**102,973**
C1_210	**26,197.3**	795	**83**	2249	524,649	**26,197.3**	**588**	85	**1917**	**411,479**
C2_2_1	**19,150.3**	371	**7**	**7**	**435**	**19,150.3**	**347**	**7**	**7**	**435**
C2_2_2	18,394.0	11,347	773	2043	11,156	**18,396.0**	**8386**	**724**	**1664**	**9571**
C2_2_5	**18,607.1**	1542	170	656	**4847**	**18,607.1**	**1276**	**133**	**529**	4859
C2_2_6	**18,406.7**	4451	327	1673	14,377	**18,406.7**	**4084**	**295**	**1455**	**13,866**
C2_2_7	18,356.4	7352	480	2461	21,229	**18,370.4**	**5458**	**393**	**2034**	**18,636**
C2_2_8	**18,082.9**	7479	**369**	2749	35,935	18,082.4	**5788**	380	**2244**	**25,504**
C2_2_9	**18,011.9**	13,406	510	3142	52,033	**18,011.9**	**11,220**	**469**	**2672**	**43,541**
R1_2_4	**30,009.9**	328	44	576	360,513	**30,009.9**	**222**	**42**	**498**	**315,121**
R1_2_5	**40,068.1**	44	11	23	**10,796**	**40,068.1**	**41**	**10**	**22**	11,161
R1_2_6	34,930.1	106	**43**	**167**	45,657	**34,930.6**	**95**	46	171	**44,276**
R1_2_7	**30,999.1**	298	54	441	**182,425**	30,998.9	**209**	**46**	**415**	194,376
R1_2_8	**29,032.4**	388	**33**	648	595,520	**29,032.3**	**255**	34	**593**	**508,686**
R1_2_9	**36,787.8**	66	**16**	123	54,489	**36,787.8**	**59**	**16**	**119**	**52,895**
R1_210	**32,451.1**	165	**21**	414	256,912	**32,451.1**	**142**	25	**398**	**206,501**
R2_2_1	34,625.1	1743	71	234	1546	**34,625.2**	**1717**	**65**	**211**	**1505**
R2_2_2	**30,006.1**	6068	328	1575	11,681	**30,006.1**	**4871**	**282**	**1214**	**10,282**
R2_2_5	**30,290.3**	3124	197	741	4739	**30,290.3**	**2783**	**173**	**623**	**4274**
R2_2_6	**26,455.5**	9531	377	2641	48,567	**26,455.6**	**8248**	**365**	**2054**	**37,270**
R2_2_9	28,206.0	4379	**203**	1167	14,379	**28,206.2**	**4049**	229	**1012**	**10,103**
RC1_2_1	**34,595.9**	60	**18**	**133**	44,486	**34,595.9**	**54**	**18**	**133**	**44,324**
RC1_2_2	**31,774.7**	135	**43**	353	109,271	**31,774.7**	**119**	**43**	**315**	**99,413**
RC1_2_6	32,554.4	171	**19**	703	432,336	**32,554.6**	**136**	22	**651**	**343,279**
RC1_2_8	**30,216.1**	540	36	1576	1,173,078	**30,216.1**	**359**	**34**	**1390**	**998,532**
RC2_2_1	27,891.0	2024	123	720	5359	**27,891.3**	**1890**	**118**	**618**	**4553**
RC2_2_2	24,588.6	11,026	595	3696	33,713	**24,590.6**	**9181**	**511**	**3102**	**29,002**
RC2_2_5	**24,788.5**	7814	**445**	4284	108,258	**24,788.5**	**6277**	448	**3744**	**76,733**
RC2_2_6	**24,724.4**	5913	396	3411	52,168	**24,724.4**	**4856**	**367**	**2929**	**41,101**
#Best	23/31	0/31	14/31	3/31	4/31	29/31	31/31	21/31	30/31	28/31

Table B42
Detailed results for comparing `PartialDom` versus `SetPair` settings using `only-cycle` DSSR after adding nonrobust cuts

	PartialDom					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,751.8**	173,438	1861	4059	264,466	26,751.3	**172,971**	**1841**	**3991**	**242,010**
C1_2_4	**26,256.0**	1620	**347**	3768	346,619	**26,256.0**	**1287**	359	**3703**	**312,205**
C1_2_9	**26,396.0**	275	62	694	103,678	**26,396.0**	**224**	**58**	**663**	**102,973**
C1_210	**26,247.0**	1063	**203**	3037	308,752	**26,247.0**	**818**	213	**2686**	**251,061**
C2_2_1	**19,221.0**	1116	587	587	**954**	**19,221.0**	**1086**	**566**	**566**	1147
C2_2_2	**18,514.0**	16,282	**1678**	3034	8686	**18,514.0**	**14,946**	1838	**2841**	**7905**
C2_2_5	**18,696.0**	2444	675	1239	**2999**	**18,696.0**	**2123**	**555**	**984**	3258
C2_2_6	**18,448.0**	5433	661	2097	9613	**18,448.0**	**5215**	**635**	**1953**	**9316**
C2_2_7	**18,422.0**	8425	**806**	**2854**	15,121	**18,422.0**	**7570**	912	2930	**13,018**
C2_2_8	**18,137.0**	9548	813	3447	23,318	**18,137.0**	**7544**	**802**	**2879**	**17,826**
C2_2_9	**18,150.0**	22,898	**1146**	4461	47,370	**18,150.0**	**22,022**	1389	**4258**	**34,605**
R1_2_4	**30,379.9**	178,129	844	2444	468,566	30,378.9	**174,117**	**782**	**2329**	**451,245**
R1_2_5	**40,471.0**	**1620**	**573**	**585**	**10,317**	**40,471.0**	2040	684	697	10,716
R1_2_6	35,485.2	**28,930**	**1193**	**1686**	81,141	**35,485.3**	30,800	1298	1802	**79,940**
R1_2_7	31,404.4	51,878	1111	2243	**215,687**	**31,404.6**	**50,470**	**1015**	**2161**	219,204
R1_2_8	29,344.9	175,303	**1038**	**2954**	**480,037**	**29,345.5**	**175,173**	1073	2980	485,150
R1_2_9	**37,284.2**	3711	**670**	**875**	27,485	37,284.1	**3390**	711	914	**26,937**
R1_210	**32,811.9**	17,630	1094	1996	93,539	32,811.6	**14,648**	**1001**	**1870**	**91,813**
R2_2_1	**34,680.0**	1871	214	387	864	**34,680.0**	**1804**	**189**	**338**	**844**
R2_2_2	**30,082.0**	7216	710	1981	6415	**30,082.0**	**5806**	**592**	**1540**	**5887**
R2_2_5	**30,611.0**	8223	1955	2677	4004	**30,611.0**	**6547**	**1633**	**2205**	**3392**
R2_2_6	**26,750.5**	23,238	2556	5012	14,381	**26,750.5**	**21,322**	**2457**	**4378**	**12,350**
R2_2_9	**28,433.0**	7057	**1144**	2220	5899	**28,433.0**	**6925**	1279	**2178**	**4904**
RC1_2_1	**35,017.2**	4623	868	1007	20,579	**35,017.3**	**4427**	**831**	**969**	**20,095**
RC1_2_2	32,087.9	**94,153**	**1111**	2012	142,075	**32,088.0**	118,036	1186	**2005**	**135,460**
RC1_2_6	32,960.3	**36,527**	**1245**	**2714**	**141,759**	**32,960.6**	46,136	1289	2757	143,502
RC1_2_8	**30,508.7**	**179,057**	702	4660	1,121,911	30,505.7	179,781	**623**	**4417**	**1,046,100**
RC2_2_1	**27,974.0**	2766	1083	1813	1617	**27,974.0**	**2542**	**976**	**1631**	**1501**
RC2_2_2	**24,780.0**	65,777	7220	10,924	9515	**24,780.0**	**41,251**	**4738**	**7679**	**8973**
RC2_2_5	**24,914.0**	**14,037**	**2384**	**6699**	36,953	**24,914.0**	19,812	4388	8265	**22,480**
RC2_2_6	**24,951.0**	11,845	2346	5953	19,456	**24,951.0**	**10,238**	**2196**	**5446**	**16,623**
#Best	26/31	6/31	13/31	7/31	6/31	26/31	25/31	18/31	24/31	25/31

Table B43
Detailed results for comparing `PartialDom` versus `SetBased` settings using `all-arcs` DSSR before adding nonrobust cuts

	PartialDom					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	314	112	509	74,968	**26,597.4**	**215**	**106**	**452**	**65,229**
C1_2_4	**26,197.7**	553	**166**	**883**	157,596	**26,197.7**	**431**	182	906	**134,003**
C1_2_9	**26,396.0**	229	**53**	540	94,199	**26,396.0**	**207**	57	**522**	**88,052**
C1_210	**26,197.3**	620	**71**	1301	357,164	**26,197.4**	**452**	78	**1122**	**280,835**
C2_2_1	**19,150.3**	360	**7**	**7**	**435**	**19,150.3**	**353**	**7**	**7**	**435**
C2_2_2	18,395.5	5091	568	799	5914	**18,396.0**	**3346**	**432**	**619**	**5484**
C2_2_5	**18,607.1**	1367	149	534	**4416**	**18,607.1**	**1277**	**137**	**493**	4644
C2_2_6	**18,406.7**	**3103**	**250**	1032	11,426	**18,406.7**	3476	328	**993**	**9659**
C2_2_7	**18,356.4**	3639	**297**	1169	15,727	**18,356.4**	**3598**	324	**1090**	**14,090**
C2_2_8	**18,082.4**	4018	285	1251	**20,206**	**18,082.4**	**3516**	**273**	**1152**	20,758
C2_2_9	**18,011.9**	5596	**297**	1342	36,377	**18,011.9**	**5194**	318	**1207**	**32,809**
R1_2_4	**30,009.9**	231	**43**	315	193,243	**30,009.9**	**177**	46	**296**	**170,241**
R1_2_5	**40,068.1**	44	**11**	**23**	**10,796**	**40,068.1**	**42**	**11**	**23**	10,864
R1_2_6	34,930.1	107	**46**	**147**	**38,119**	**34,930.6**	**94**	47	151	39,320
R1_2_7	30,999.5	217	**45**	**282**	126,463	**31,000.5**	**194**	52	**282**	**115,984**
R1_2_8	**29,032.4**	237	**31**	**358**	342,000	29,032.2	**200**	33	360	**330,244**
R1_2_9	**36,787.8**	65	**16**	113	49,843	**36,787.8**	**58**	17	**110**	**45,953**
R1_210	**32,451.2**	144	**22**	**301**	174,968	**32,451.1**	**120**	23	302	**170,625**
R2_2_1	**34,625.1**	1706	**51**	193	1779	**34,625.1**	**1625**	58	**185**	**1499**
R2_2_2	**30,006.1**	4414	252	921	9803	**30,006.1**	**4085**	**248**	**825**	**8971**
R2_2_5	**30,290.3**	2869	189	553	**3649**	**30,290.3**	**2473**	**140**	**460**	4139
R2_2_6	**26,455.5**	5711	**267**	1144	30,557	**26,455.5**	**5669**	288	**1076**	**25,726**
R2_2_9	**28,206.0**	3543	177	684	9481	**28,206.0**	**3371**	**175**	**617**	**8318**
RC1_2_1	**34,595.9**	59	18	133	**44,486**	**34,595.9**	**54**	**17**	**131**	46,746
RC1_2_2	**31,774.7**	124	46	271	**79,659**	**31,774.7**	**117**	**37**	**266**	101,657
RC1_2_6	**32,554.4**	128	23	462	**234,497**	**32,554.4**	**101**	**18**	**424**	285,058
RC1_2_8	**30,216.1**	252	30	577	**527,790**	**30,216.1**	**196**	**29**	**572**	542,817
RC2_2_1	**27,891.0**	1976	144	543	**3499**	**27,891.0**	**1867**	**129**	**475**	3534
RC2_2_2	24,588.6	6397	472	1723	**20,948**	**24,589.4**	**5806**	**369**	**1524**	23,857
RC2_2_5	**24,788.5**	3938	356	1487	**43,291**	**24,788.5**	**3245**	**264**	**1305**	47,348
RC2_2_6	**24,724.4**	3353	**228**	1302	32,504	**24,724.4**	**3249**	278	**1277**	**25,921**
#Best	27/31	1/31	17/31	7/31	13/31	30/31	30/31	16/31	27/31	19/31

Table B44
Detailed results for comparing `PartialDom` versus `SetBased` settings using `all-arcs` DSSR after adding nonrobust cuts

	PartialDom					SetBased				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	26,752.3	**173,041**	2005	2678	**216,061**	**26,753.2**	173,111	**1985**	**2635**	228,628
C1_2_4	**26,256.0**	919	**333**	1206	140,079	**26,256.0**	**709**	350	**1205**	**120,837**
C1_2_9	**26,396.0**	230	**53**	540	94,199	**26,396.0**	**208**	57	**522**	**88,052**
C1_210	**26,247.0**	811	199	1670	177,454	**26,247.0**	**625**	**198**	**1505**	**168,608**
C2_2_1	**19,221.0**	1101	587	587	**954**	**19,221.0**	**935**	**479**	**479**	1033
C2_2_2	**18,514.0**	11,757	1681	1943	**6661**	**18,514.0**	**8511**	**1296**	**1505**	7360
C2_2_5	**18,696.0**	2044	**514**	**932**	**3141**	**18,696.0**	**2031**	556	947	3272
C2_2_6	**18,448.0**	**4586**	**787**	1617	**6671**	**18,448.0**	4766	795	**1498**	7195
C2_2_7	**18,422.0**	**4770**	**673**	**1574**	9874	**18,422.0**	9523	1994	2891	**8236**
C2_2_8	**18,137.0**	5268	**644**	1661	**13,761**	**18,137.0**	**5002**	705	**1654**	13,938
C2_2_9	**18,150.0**	12,732	944	2167	31,434	**18,150.0**	**10,928**	**934**	**1969**	**29,016**
R1_2_4	30,382.2	174,746	**908**	1519	**368,626**	**30,382.5**	**174,279**	932	**1501**	375,703
R1_2_5	40,471.0	**1620**	**573**	**585**	**10,317**	**40,471.6**	1730	628	640	10,706
R1_2_6	35,485.0	**23,514**	**1155**	**1441**	**71,429**	**35,485.8**	26,704	1306	1574	75,130
R1_2_7	**31,404.6**	42,064	1053	1548	174,493	31,404.3	**30,365**	**1002**	**1493**	**173,444**
R1_2_8	29,346.5	**173,694**	**1193**	**1887**	**416,710**	**29,346.8**	174,211	1311	1988	439,039
R1_2_9	**37,283.9**	4155	733	903	**26,232**	**37,283.9**	**3026**	**643**	**817**	27,223
R1_210	**32,811.8**	16,242	1092	1612	**80,352**	32,811.5	**13,001**	**1030**	**1551**	83,754
R2_2_1	**34,680.0**	1811	**198**	**349**	851	**34,680.0**	**1811**	294	430	**725**
R2_2_2	**30,082.0**	**5212**	**541**	**1221**	5727	**30,082.0**	5343	706	1294	**4597**
R2_2_5	**30,611.0**	6790	**1603**	**2023**	**3577**	**30,611.0**	**6727**	1709	2106	3886
R2_2_6	**26,750.5**	19,690	2610	3540	11,468	**26,750.5**	**18,306**	**2363**	**3232**	**10,879**
R2_2_9	**28,433.0**	6130	**1094**	1646	4768	**28,433.0**	**5944**	1113	**1605**	**4762**
RC1_2_1	**35,017.2**	4698	868	1007	20,579	35,016.8	**3385**	**770**	**909**	**19,631**
RC1_2_2	32,087.5	82,382	**1080**	**1553**	123,616	**32,087.8**	**78,701**	1132	1628	**120,445**
RC1_2_6	32,960.1	32,145	**1180**	2070	**124,493**	**32,960.2**	**27,114**	1193	**2044**	126,487
RC1_2_8	30,514.2	174,813	**878**	**2067**	**576,518**	**30,514.7**	**172,960**	897	2072	598,704
RC2_2_1	**27,974.0**	2610	962	1449	**1429**	**27,974.0**	**2507**	**940**	**1382**	1619
RC2_2_2	**24,780.0**	**61,601**	**7158**	**8589**	9289	**24,780.0**	70,794	7815	9078	**9037**
RC2_2_5	**24,914.0**	15,738	3752	4964	**20,378**	**24,914.0**	**10,548**	**2593**	**3713**	23,611
RC2_2_6	**24,951.0**	9846	2330	3484	13,243	**24,951.0**	**7557**	**1932**	**2976**	**12,034**
#Best	23/31	8/31	18/31	11/31	17/31	28/31	23/31	13/31	20/31	14/31

Table B45
Detailed results for comparing `PartialDom` versus `Pairwise` settings using `all-arcs` DSSR before adding nonrobust cuts

	PartialDom					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	314	**112**	509	74,968	**26,597.4**	**259**	**112**	**451**	**61,612**
C1_2_4	**26,197.7**	553	**166**	**883**	157,596	**26,197.7**	**492**	179	951	**157,224**
C1_2_9	**26,396.0**	229	**53**	540	94,199	**26,396.0**	**223**	55	**516**	**85,621**
C1_210	**26,197.3**	620	71	1301	357,164	**26,197.3**	**454**	**70**	**1156**	**302,044**
C2_2_1	**19,150.3**	360	**7**	**7**	**435**	**19,150.3**	**346**	**7**	**7**	**435**
C2_2_2	**18,395.5**	5091	568	799	5914	18,394.0	**3258**	**416**	**612**	**5208**
C2_2_5	**18,607.1**	1367	149	534	**4416**	**18,607.1**	**1281**	**128**	**490**	4733
C2_2_6	**18,406.7**	3103	**250**	1032	11,426	**18,406.7**	**2809**	263	**935**	**9440**
C2_2_7	**18,356.4**	3639	297	1169	15,727	**18,356.4**	**2985**	**265**	**998**	**13,361**
C2_2_8	**18,082.4**	4018	**285**	1251	20,206	**18,082.4**	**3533**	297	**1146**	**16,726**
C2_2_9	**18,011.9**	5596	**297**	1342	36,377	18,011.8	**5259**	311	**1234**	**30,655**
R1_2_4	**30,009.9**	231	**43**	315	193,243	**30,009.9**	**229**	44	**308**	**189,214**
R1_2_5	**40,068.1**	44	**11**	**23**	10,796	**40,068.1**	**42**	**11**	**23**	**10,792**
R1_2_6	34,930.1	107	**46**	147	38,119	**34,930.5**	**97**	**46**	**146**	**36,742**
R1_2_7	30,999.5	217	**45**	282	126,463	**31,000.5**	**199**	49	**271**	**117,794**
R1_2_8	**29,032.4**	237	**31**	358	342,000	**29,032.4**	**226**	35	**343**	**290,122**
R1_2_9	**36,787.8**	65	**16**	113	49,843	**36,787.8**	**60**	**16**	**109**	**47,384**
R1_210	**32,451.2**	144	**22**	**301**	174,968	**32,451.1**	**132**	**22**	302	**174,792**
R2_2_1	**34,625.1**	**1706**	**51**	**193**	1779	**34,625.1**	1719	76	210	**1263**
R2_2_2	**30,006.1**	4414	252	921	9803	**30,006.1**	**3917**	**242**	**824**	**8778**
R2_2_5	**30,290.3**	2869	189	553	**3649**	**30,290.3**	**2665**	**158**	**490**	3761
R2_2_6	**26,455.5**	5711	**267**	1144	30,557	**26,455.5**	**5601**	280	**1099**	**26,008**
R2_2_9	**28,206.0**	3543	177	684	9481	**28,206.0**	**3272**	**171**	**622**	**8243**
RC1_2_1	**34,595.9**	59	**18**	**133**	44,486	**34,595.9**	**56**	**18**	**133**	**44,460**
RC1_2_2	**31,774.7**	124	46	271	**79,659**	**31,774.7**	**117**	**39**	**257**	90,986
RC1_2_6	**32,554.4**	128	23	462	**234,497**	**32,554.4**	**109**	**19**	**423**	253,881
RC1_2_8	**30,216.1**	252	30	577	**527,790**	**30,216.1**	**226**	**28**	**571**	528,193
RC2_2_1	27,891.0	1976	144	543	**3499**	**27,891.3**	**1841**	**125**	**477**	3532
RC2_2_2	24,588.6	6397	472	1723	20,948	**24,588.8**	**6091**	**421**	**1561**	**19,566**
RC2_2_5	**24,788.5**	3938	356	1487	**43,291**	**24,788.5**	**3408**	**286**	**1343**	43,608
RC2_2_6	**24,724.4**	3353	**228**	1302	32,504	**24,724.4**	**3035**	269	**1196**	**23,394**
#Best	27/31	1/31	18/31	6/31	8/31	29/31	30/31	20/31	28/31	24/31

Table B46
Detailed results for comparing `PartialDom` versus `Pairwise` settings using `all-arcs` DSSR after adding nonrobust cuts

	PartialDom					Pairwise				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	26,752.3	**173,041**	2005	2678	**216,061**	**26,752.8**	174,314	**1951**	**2601**	223,555
C1_2_4	**26,256.0**	919	**333**	**1206**	140,079	**26,256.0**	**750**	334	1209	**128,992**
C1_2_9	**26,396.0**	230	**53**	540	94,199	**26,396.0**	**224**	55	**516**	**85,621**
C1_210	**26,247.0**	811	**199**	1670	177,454	**26,247.0**	**623**	205	**1547**	**151,855**
C2_2_1	**19,221.0**	1101	587	587	**954**	**19,221.0**	**1011**	**414**	**414**	1405
C2_2_2	**18,514.0**	**11,757**	**1681**	**1943**	**6661**	**18,514.0**	12,010	1754	1965	7878
C2_2_5	**18,696.0**	**2044**	**514**	**932**	3141	**18,696.0**	2108	602	995	**2823**
C2_2_6	**18,448.0**	4586	787	1617	**6671**	**18,448.0**	**3823**	**630**	**1346**	6806
C2_2_7	**18,422.0**	4770	673	1574	9874	**18,422.0**	**3959**	**614**	**1390**	**8432**
C2_2_8	**18,137.0**	5268	**644**	1661	13,761	**18,137.0**	**4882**	712	**1615**	**11,765**
C2_2_9	**18,150.0**	12,732	**944**	2167	31,434	**18,150.0**	**11,728**	1066	**2166**	**25,098**
R1_2_4	**30,382.2**	**174,746**	**908**	1519	**368,626**	30,382.1	177,048	916	**1479**	372,593
R1_2_5	40,471.0	**1620**	**573**	**585**	**10,317**	**40,471.9**	2039	694	706	10,679
R1_2_6	35,485.0	23,514	**1155**	**1441**	**71,429**	**35,485.6**	**22,911**	1211	1480	72,105
R1_2_7	**31,404.6**	42,064	1053	1548	174,493	**31,404.7**	**31,416**	**1033**	**1507**	**172,905**
R1_2_8	29,346.5	**173,694**	**1193**	**1887**	416,710	**29,346.8**	174,130	1286	1969	**415,519**
R1_2_9	37,283.9	4155	733	903	**26,232**	**37,284.0**	**3451**	**701**	**866**	26,581
R1_210	**32,811.8**	16,242	1092	1612	80,352	32,811.5	**12,644**	**1009**	**1530**	**79,696**
R2_2_1	**34,680.0**	**1811**	**198**	**349**	851	**34,680.0**	1915	353	494	**678**
R2_2_2	**30,082.0**	5212	**541**	1221	5727	**30,082.0**	**4902**	596	**1190**	**4794**
R2_2_5	**30,611.0**	6790	1603	2023	**3577**	**30,611.0**	**5967**	**1405**	**1778**	3636
R2_2_6	**26,750.5**	19,690	2610	3540	11,468	**26,750.5**	**18,934**	**2493**	**3392**	**10,006**
R2_2_9	**28,433.0**	6130	1094	1646	4768	**28,433.0**	**5714**	**1089**	**1566**	**4442**
RC1_2_1	35,017.2	**4698**	**868**	**1007**	20,579	**35,017.8**	4832	938	1080	**20,427**
RC1_2_2	**32,087.5**	**82,382**	**1080**	**1553**	**123,616**	**32,087.6**	97,015	1156	1627	124,469
RC1_2_6	32,960.1	**32,145**	**1180**	**2070**	124,493	**32,960.2**	33,762	1229	2092	**122,444**
RC1_2_8	**30,514.2**	174,813	878	**2067**	576,518	**30,514.1**	**173,242**	**874**	2085	**547,801**
RC2_2_1	**27,974.0**	2610	**962**	1449	1429	**27,974.0**	**2468**	1022	**1442**	**1380**
RC2_2_2	**24,780.0**	61,601	7158	8589	9289	**24,780.0**	**51,013**	**6063**	**7308**	**7725**
RC2_2_5	**24,914.0**	15,738	3752	4964	20,378	**24,914.0**	**11,536**	**2890**	**3996**	**18,575**
RC2_2_6	**24,951.0**	**9846**	**2330**	**3484**	13,243	**24,951.0**	12,398	3136	4117	**10,638**
#Best	24/31	11/31	18/31	12/31	10/31	29/31	20/31	13/31	19/31	21/31

Table B47
Detailed results for comparing `PartialDom` versus `SetPair` settings using `all-arcs` DSSR before adding nonrobust cuts

	PartialDom					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,597.4**	314	112	509	74,968	**26,597.4**	**184**	**104**	**468**	**71,359**
C1_2_4	**26,197.7**	553	**166**	883	157,596	**26,197.7**	**435**	171	**816**	**137,328**
C1_2_9	**26,396.0**	229	53	540	94,199	**26,396.0**	**215**	**52**	**523**	**91,451**
C1_210	**26,197.3**	620	71	1301	357,164	**26,197.3**	**476**	**69**	**1053**	**281,031**
C2_2_1	**19,150.3**	360	**7**	**7**	**435**	**19,150.3**	**348**	**7**	**7**	**435**
C2_2_2	18,395.5	5091	568	799	**5914**	**18,396.0**	**3495**	**440**	**695**	6377
C2_2_5	**18,607.1**	1367	149	534	**4416**	**18,607.1**	**1234**	**130**	**498**	4742
C2_2_6	**18,406.7**	3103	250	1032	**11,426**	**18,406.7**	**2781**	**199**	**931**	12,609
C2_2_7	**18,356.4**	3639	297	1169	15,727	**18,356.4**	**3097**	**234**	**979**	**15,181**
C2_2_8	**18,082.4**	4018	285	1251	20,206	**18,082.4**	**3449**	**264**	**1138**	**19,031**
C2_2_9	**18,011.9**	5596	297	1342	36,377	18,011.8	**5014**	**289**	**1212**	**32,366**
R1_2_4	**30,009.9**	231	**43**	315	193,243	**30,009.9**	**185**	44	**313**	**188,885**
R1_2_5	**40,068.1**	44	11	23	**10,796**	**40,068.1**	**41**	**10**	**22**	11,161
R1_2_6	34,930.1	107	**46**	**147**	**38,119**	**34,930.6**	**93**	**46**	159	39,948
R1_2_7	30,999.5	217	45	**282**	**126,463**	**31,000.2**	**177**	**44**	286	138,251
R1_2_8	**29,032.4**	237	**31**	358	342,000	**29,032.3**	**183**	35	**345**	**275,634**
R1_2_9	**36,787.8**	65	**16**	113	49,843	**36,787.8**	**58**	**16**	**109**	**47,399**
R1_210	**32,451.2**	144	22	301	**174,968**	32,451.1	**116**	**19**	**284**	189,812
R2_2_1	**34,625.1**	1706	**51**	**193**	1779	**34,625.1**	**1666**	70	197	**1290**
R2_2_2	**30,006.1**	4414	252	921	9803	**30,006.1**	**3895**	**222**	**846**	**9565**
R2_2_5	**30,290.3**	2869	189	553	3649	**30,290.3**	**2536**	**182**	**514**	**3457**
R2_2_6	**26,455.5**	5711	**267**	1144	30,557	**26,455.5**	**5477**	273	**1092**	**27,355**
R2_2_9	**28,206.0**	3543	177	684	9481	**28,206.0**	**3416**	**176**	**633**	**8129**
RC1_2_1	**34,595.9**	59	**18**	**133**	44,486	**34,595.9**	**54**	**18**	**133**	**44,324**
RC1_2_2	**31,774.7**	124	46	271	**79,659**	**31,774.7**	**116**	**43**	**262**	84,050
RC1_2_6	**32,554.4**	128	23	462	**234,497**	**32,554.4**	**100**	**20**	**423**	245,260
RC1_2_8	**30,216.1**	252	**30**	577	527,790	**30,216.1**	**196**	31	**572**	**490,921**
RC2_2_1	27,891.0	1976	144	543	3499	**27,892.5**	**1818**	**131**	**487**	**3448**
RC2_2_2	24,588.6	6397	472	1723	20,948	**24,588.8**	**5330**	**379**	**1492**	**20,620**
RC2_2_5	**24,788.5**	3938	356	1487	43,291	**24,788.5**	**3324**	**330**	**1394**	**37,539**
RC2_2_6	**24,724.4**	3353	228	1302	32,504	**24,724.4**	**2898**	**207**	**1163**	**29,369**
#Best	26/31	0/31	10/31	5/31	10/31	29/31	31/31	25/31	28/31	22/31

Table B48
Detailed results for comparing `PartialDom` versus `SetPair` settings using `all-arcs` DSSR after adding nonrobust cuts

	PartialDom					SetPair				
Instance	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_2_3	**26,752.3**	**173,041**	2005	2678	216,061	26,751.9	174,346	**1821**	**2460**	**211,811**
C1_2_4	**26,256.0**	919	**333**	1206	140,079	**26,256.0**	**800**	343	**1153**	**129,352**
C1_2_9	**26,396.0**	230	53	540	94,199	**26,396.0**	**216**	**52**	**523**	**91,451**
C1_210	**26,247.0**	811	199	1670	177,454	**26,247.0**	**628**	**192**	**1483**	**157,517**
C2_2_1	**19,221.0**	1101	587	587	**954**	**19,221.0**	**1091**	**566**	**566**	1147
C2_2_2	**18,514.0**	11,757	1681	1943	**6661**	**18,514.0**	**10,819**	**1366**	**1633**	7385
C2_2_5	**18,696.0**	**2044**	**514**	**932**	3141	**18,696.0**	2211	688	1104	**2863**
C2_2_6	**18,448.0**	4586	787	1617	**6671**	**18,448.0**	**3617**	**498**	**1252**	7873
C2_2_7	**18,422.0**	4770	673	1574	9874	**18,422.0**	**4147**	**620**	**1381**	**8749**
C2_2_8	**18,137.0**	5268	644	1661	13,761	**18,137.0**	**4637**	**593**	**1523**	**13,588**
C2_2_9	**18,150.0**	12,732	944	2167	31,434	**18,150.0**	**10,488**	**899**	**1967**	**26,966**
R1_2_4	**30,382.2**	**174,746**	908	1519	368,626	30,380.0	179,044	**842**	**1441**	**366,007**
R1_2_5	**40,471.0**	**1620**	**573**	**585**	**10,317**	**40,471.0**	1986	684	697	10,716
R1_2_6	35,485.0	**23,514**	**1155**	**1441**	**71,429**	**35,485.3**	27,965	1205	1505	74,688
R1_2_7	**31,404.6**	42,064	1053	1548	174,493	**31,404.6**	**33,773**	**968**	**1461**	**168,974**
R1_2_8	**29,346.5**	**173,694**	1193	1887	416,710	29,346.1	174,610	**1147**	**1846**	**416,161**
R1_2_9	**37,283.9**	4155	733	903	**26,232**	37,283.2	**3421**	**698**	**863**	26,282
R1_210	32,811.8	**16,242**	**1092**	**1612**	**80,352**	**32,812.3**	17,806	1146	1665	82,338
R2_2_1	**34,680.0**	1811	**198**	**349**	851	**34,680.0**	**1783**	218	354	**756**
R2_2_2	**30,082.0**	5212	541	1221	5727	**30,082.0**	**4646**	**495**	**1132**	**5412**
R2_2_5	**30,611.0**	6790	**1603**	**2023**	**3577**	**30,611.0**	**6300**	1652	2040	3690
R2_2_6	**26,750.5**	19,690	2610	3540	11,468	**26,750.5**	**17,219**	**2235**	**3139**	**10,640**
R2_2_9	**28,433.0**	**6130**	**1094**	**1646**	4768	**28,433.0**	6143	1236	1726	**4360**
RC1_2_1	**35,017.2**	4698	868	1007	20,579	**35,017.3**	**4376**	**831**	**969**	**20,095**
RC1_2_2	32,087.5	**82,382**	**1080**	**1553**	123,616	**32,087.6**	110,251	1132	1618	**118,878**
RC1_2_6	32,960.1	**32,145**	**1180**	2070	**124,493**	**32,960.3**	37,278	1203	**2053**	125,815
RC1_2_8	**30,514.2**	174,813	878	2067	576,518	30,512.5	**173,314**	**830**	**1976**	**540,561**
RC2_2_1	**27,974.0**	2610	962	1449	**1429**	**27,974.0**	**2314**	**839**	**1279**	1464
RC2_2_2	**24,780.0**	61,601	7158	8589	9289	**24,780.0**	**43,946**	**5701**	**6928**	**8103**
RC2_2_5	**24,914.0**	15,738	3752	4964	20,378	**24,914.0**	**8939**	**2407**	**3533**	**19,953**
RC2_2_6	**24,951.0**	9846	2330	3484	13,243	**24,951.0**	**7700**	**2189**	**3232**	**11,829**
#Best	27/31	10/31	10/31	8/31	10/31	26/31	21/31	21/31	23/31	21/31