

Branch-and-Price for the Length-Constrained Cycle Partition Problem

Mohammed Ghannam^{1,2,4}[0000-0001-9422-7916], Gioni Mexi²[0000-0003-0964-9802], Edward Lam³[0000-0002-4485-5014], and Ambros Gleixner^{1,2}[0000-0003-0391-5903]

¹ HTW Berlin, Germany

² Zuse Institute Berlin, Germany

³ Monash University, Australia

⁴ felmo GmbH, Germany

Abstract. The length-constrained cycle partition problem (LCCP) is a graph optimization problem in which a set of nodes must be partitioned into a minimum number of cycles. Every node is associated with a critical time and the length of every cycle must not exceed the critical time of any node in the cycle. We formulate LCCP as a set partitioning model and solve it using an exact branch-and-price approach. We use a dynamic programming-based pricing algorithm to generate improving cycles, exploiting the particular structure of the pricing problem for efficient bidirectional search and symmetry breaking. Computational results show that the LP relaxation of the set partitioning model produces strong dual bounds and our branch-and-price method improves significantly over the state of the art. It is able to solve closed instances in a fraction of the previously needed time and closes 13 previously unsolved instances, one of which has 76 nodes, a notable improvement over the previous limit of 52 nodes.

Keywords: Branch-and-Price · Column Generation · Cycle Partitioning · Dynamic Programming · Bidirectional Search

1 Introduction

A *cycle partition* of an undirected graph $G = (V, E)$ is a partition of the set of nodes V into disjoint cycles. In the *length-constrained cycle partition problem* (LCCP) introduced in [9], we are additionally given a *critical time* $q_i > 0$ for every node $i \in V$ and a *travel time* $t_{i,j} \geq 0$ for every edge $\{i, j\} \in E$. We call a cycle $C = (i_0, i_1, \dots, i_K = i_0)$ *length-feasible* if it satisfies the length constraint

$$t(C) := t_{i_0, i_1} + t_{i_1, i_2} + \dots + t_{i_{K-1}, i_K} \leq q(C) := \min\{q_{i_0}, q_{i_1}, \dots, q_{i_K}\}, \quad (1)$$

i.e., the total time to traverse the edges of the cycle must not exceed the critical time of any node in the cycle. In other words an agent that continuously travels along the cycle will visit each node i of the cycle with frequency at least q_i .

The LCCP then amounts to computing the smallest number of disjoint, length-feasible cycles that cover all nodes.

The setting of LCCP is found to be useful, e.g., in applications where the objective is to find the smallest number of agents to conduct periodic surveillance or maintenance actions [10]. Similar types of length constraints can be found in the literature on minimizing cycle length in kidney exchange programmes [14], where the length is simply the number of nodes in the cycle.

In [9], Hoppmann-Baum et al. show that the LCCP is NP-hard by reduction from the *traveling salesman problem* (TSP), and that no polynomial-time approximation algorithm exists. They present a compact mixed-integer programming (MIP) formulation based on the Miller, Tucker, and Zemlin (MTZ) [18] formulation of the TSP. This compact model was able to solve instances with up to 29 nodes to proven optimality. In [11], the same authors introduced a MIP formulation based on subtour elimination constraints (SEC), valid inequalities derived from cliques in conflict hypergraphs, and an efficient primal heuristic. A branch-and-cut implementation of the SEC model based on a state-of-the-art commercial MIP solver was able to solve instances with up to 52 nodes. Despite the mentioned improvements, the proposed models suffer from symmetry, leading to a weak linear relaxation and ineffective branching.

The goal of our work is to try and overcome several bottlenecks of the previous approaches by considering a set partitioning formulation based on cycle variables. Let Ω denote the set of all length-feasible cycles, and let $a_i^C \in \{0, 1\}$ be a constant equal to 1 iff node i is contained in cycle C . Introducing binary variables λ_C to indicate whether cycle $C \in \Omega$ is used in the solution, we arrive at the set partitioning formulation

$$\begin{aligned} \min \quad & \sum_{C \in \Omega} \lambda_C \\ \text{s.t.} \quad & \sum_{C \in \Omega} a_i^C \lambda_C = 1 \quad \text{for all } i \in V, \\ & \lambda_C \in \{0, 1\} \quad \text{for all } C \in \Omega. \end{aligned} \tag{2}$$

This reformulation of LCCP is inspired by formulations for the cardinality-constrained multi-cycle problem in kidney exchange [14], where a maximum-value packing of cycles is required, and by exact methods in many logistics applications such as vehicle routing, see, e.g., [4]. As in these applications, it is not viable to solve (2) explicitly due to the exponential number of variables. Instead, we apply *column generation* in order to solve the linear programming (LP) relaxation of (2) dynamically, and integrate this into an exact *branch-and-price* algorithm enabling us to solve LCCP instances to proven optimality, see, e.g., [5, 24] for a general overview of this methodology.

A major computational bottleneck in this approach is the so-called pricing problem which needs to be solved repeatedly in order to dynamically add variables with negative reduced cost to the LP relaxation of (2). Given travel times $t_{i,j} \geq 0$ for $\{i, j\} \in E$ and both critical times $q_i > 0$ and weights $\pi_i \in \mathbb{R}$ for $i \in V$, the pricing problem amounts to computing a cycle C that satisfies the length constraint (1) and maximizes the sum of node weights. We call this the *length-constrained prize-collecting cycle problem* (LCPCCP). Note that problem data

may be non-metric, i.e., travel times $t_{i,j}$ may violate the triangle inequality, and that the node weights π_i , which are derived as dual multipliers associated with the partitioning constraint of the node in the LP relaxation, are not restricted in sign. Even in the metric case and when critical times are assumed to be constant, one can show that LCPCCP is strongly NP-hard by reduction from the Hamiltonian cycle problem.

The main contributions of this paper are as follows. In Sec. 2 we present a new dynamic programming algorithm, also known as label setting, for LCPCCP including dominance rules for reducing the search space and an efficient bidirectional search that allows us to enumerate cycles only half-way. In Sec. 3 we integrate the resulting column generation scheme into an exact branch-and-price algorithm enhanced by symmetry breaking, heuristic pricing, parallel pricing, early branching, and techniques to exploit the triangle inequality for metric input data. In Sec. 4 we describe the results of our computational study to analyze the performance of the new approach on benchmark instances from the literature and to quantify the impact of the individual improvement techniques. The full branch-and-price algorithm is able to solve 13 open instances with up to 76 nodes, and is on average 14.6 times faster than the best previous approach.

2 Dynamic Programming for the Length-Constrained Prize-Collecting Cycle Problem

Column generation is an advanced technique for solving large linear programs and is an essential subroutine in branch-and-price algorithms [5,24]. The method has recently been found to date back to works by Kantorovich and Zalgaller in 1951 [13,23], and was independently developed by Gomory and Gilmore [7].

In order to solve the LP relaxation of (2) by column generation, we first note that the upper bounds $\lambda_C \leq 1$ are implied by the partitioning constraints and can be removed. We call the resulting LP the *master problem* (MP). Second, we replace Ω with a subset of columns $\Omega' \subset \Omega$ to form the *restricted master problem* (RMP). Column generation then repeatedly optimizes the RMP, each time producing a primal-dual pair of solutions (λ, π) and solving the so-called *pricing problem* to check whether cycles C with negative reduced cost

$$\bar{c}(C) = 1 - \sum_{i \in V} a_i^C \pi_i = 1 - \sum_{i \in C} \pi_i \quad (3)$$

exist in $\Omega \setminus \Omega'$. If yes, these are added to Ω' and the RMP is reoptimized; if not, then (λ, π) is optimal for MP. We refer to [5,24] for details on column generation.

As explained above, the pricing problem of minimizing (3) amounts to the length-constrained prize-collecting cycle problem. In the following, suppose we restrict LCPCCP by fixing one node $s \in V$ to be contained in the cycle. Then the resulting problem closely resembles a *resource-constrained shortest path problem* (RCSPP) [12] typically solved in vehicle routing applications on a directed graph to find a path of minimum reduced cost that starts at and returns to a given node such that resources accumulated along arcs lie within a constant resource

interval specified for each node. Similarly, we propose to solve LCPCCP for a fixed start node using dynamic programming as an implicit enumeration scheme over all length-feasible cycles starting from s .

To this end, we form *partial cycles* (paths) and iteratively extend these paths with incident unvisited nodes until s is visited again, essentially closing the cycle. If at any point a partial cycle becomes infeasible (cannot be extended to a length-feasible cycle) or is proved to be dominated (would only lead to cycles with same or larger reduced cost), it is discarded, see Sec. 2.1. This process is repeated until all non-dominated length-feasible cycles are explored. If there are cycles with negative reduced cost, the ones with minimum reduced cost are returned.

Let $P = (i_0, i_1, \dots, i_K)$ be a path in $G = (V, E)$ with nodes $i_0, \dots, i_K \in V$ and edges $\{i_0, i_1\}, \dots, \{i_{K-1}, i_K\} \in E$. The path P can be represented by a so-called *label* $\ell = (\mathcal{N}, v, \bar{c}, t, q)$ where

- $\mathcal{N} = \mathcal{N}(\ell) = \{i_1, \dots, i_K\}$ is the (unordered) set of nodes without start node,
- $v = v(\ell) = i_K$ is the last node visited in P ,
- $\bar{c} = \bar{c}(\ell) = 1 - \sum_{i \in \mathcal{N}} \pi_i$ is the reduced cost of P ,
- $t = t(\ell) = \sum_{k=1}^K t_{i_{k-1}, i_k}$ is the total travel time taken by P , and
- $q = q(\ell) = \min_{k=0,1,\dots,K} q_{i_k}$ is the minimum critical time of all nodes in P .

The initial label for a starting node s is given as $\ell_s = (\{\}, s, 1, 0, q_s)$. During the search, a label $\ell = (\mathcal{N}, v, \bar{c}, t, q)$ can be *extended* using any edge $\{v(\ell), j\}$ from the last node to a new node $j \in V \setminus \mathcal{N}$. This creates a new label ℓ^+ where $\mathcal{N}(\ell^+) = \mathcal{N}(\ell) \cup \{j\}$, $v(\ell^+) = j$, $\bar{c}(\ell^+) = \bar{c}(\ell) - \pi_j$, $t(\ell^+) = t(\ell) + t_{i,j}$, and $q(\ell^+) = \min\{q(\ell), q_j\}$. We call a label $\ell^* \neq \ell_s$ *fully-extended* if it represents a cycle, i.e., if $v(\ell^*) = s$. Note that LCCP allows singleton cycles (loops) as part of the solution; these are included here as label $(\{s\}, s, 1 - \pi_s, 0, q_s)$. In order to recover the cycle corresponding to a fully-extended label, we also keep track of each label's predecessor, i.e., the label from which it was extended.

2.1 Pruning the Search Space by Feasibility and Dominance

Besides the data structures used to represent and extend labels, the efficiency of the resulting dynamic programming method largely depends on how much of the search space needs to be explored in order to terminate with a provably optimal cycle. Clearly, a label ℓ can be discarded if it can be proven to only lead to length-infeasible cycles. This is the case if $t(\ell) > q(\ell)$. Then for any extension ℓ^+ of label ℓ , we have $t(\ell^+) \geq t(\ell) > q(\ell) \geq q(\ell^+)$. By induction, all fully-extended labels ℓ^* obtained by successive extensions of label ℓ satisfy $t(\ell^*) > q(\ell^*)$, hence the corresponding cycles violate the length constraint (1).

An effective technique for pruning the search space further is to exploit dominance relations. A label ℓ_a is said to *dominate* label ℓ_b if they have the same end node $v = v(\ell_a) = v(\ell_b)$, and all cycles reachable from successive extensions of ℓ_a have an equal or lower reduced cost than those reachable from successive extensions of ℓ_b . Then again, by induction, we may discard label ℓ_b in the search, because at least one minimum reduced cost cycle must remain.

Lemma 1. *A label ℓ_a dominates a label ℓ_b if $v(\ell_a) = v(\ell_b)$ and the conditions*

$$\bar{c}(\ell_a) \leq \bar{c}(\ell_b), \quad (4a)$$

$$t(\ell_a) \leq t(\ell_b), \text{ and} \quad (4b)$$

$$\mathcal{N}(\ell_a) \subseteq \mathcal{N}(\ell_b) \quad (4c)$$

hold.

Proof. From (4c) we have that the set of possible extensions of label ℓ_b is a subset of the set of possible extensions of label ℓ_a . From (4c) it also follows that $q(\ell_a) = \min_{i \in \mathcal{N}(\ell_a)} q_i \geq \min_{i \in \mathcal{N}(\ell_b)} q_i = q(\ell_b)$. Together with (4b) this proves that any feasible extension of ℓ_b is also feasible for ℓ_a . Therefore, any cycle reachable from ℓ_b is also reachable from ℓ_a , and from (4a) it has an equal or lower reduced cost. Therefore, ℓ_a dominates ℓ_b .

2.2 Bidirectional Search

Exploring the search space from both forward and backward directions simultaneously is another successful technique to reduce the number of explored labels. For the RCSPP, [21, 22] has shown that bidirectional search can be more efficient than unidirectional search. The efficiency comes from the fact that the number of non-dominated labels can grow exponentially with the length of the paths. Hence, if it is possible to explore forward and backward paths only until a halfway point and merge them to full paths, this can drastically reduce the total number of labels generated.

For LCPCCP, we can apply bidirectional search even more effectively than for the general RCSPP. Since the graph is undirected and the resource consumption by travel times is symmetric, only extensions from one direction are needed. Specifically, in bidirectional search for a fixed start node s , we only extend labels ℓ with $t(\ell) < \frac{q(\ell)}{2}$. We then *merge* non-dominated labels as follows.

Let labels ℓ_a and ℓ_b both start at s , end at $v = v(\ell_a) = v(\ell_b)$, and be otherwise disjoint, i.e., $\mathcal{N}(\ell_a) \cap \mathcal{N}(\ell_b) = \{v\}$. Then we create a cycle by concatenating the path represented by ℓ_a and, in reverse order, the path represented by ℓ_b . We denote the newly merged label by $\ell_a || \ell_b$, given by $\mathcal{N}(\ell_a || \ell_b) = \mathcal{N}(\ell_a) \cup \mathcal{N}(\ell_b) \cup \{s\}$, $v(\ell_a || \ell_b) = s$, $\bar{c}(\ell_a || \ell_b) = \bar{c}(\ell_a) + \bar{c}(\ell_b) - \pi_s + \pi_{v(\ell_a)} - 1$, $t(\ell_a || \ell_b) = t(\ell_a) + t(\ell_b)$, and $q(\ell_a || \ell_b) = \min\{q(\ell_a), q(\ell_b)\}$, see Fig. 1 for an example.

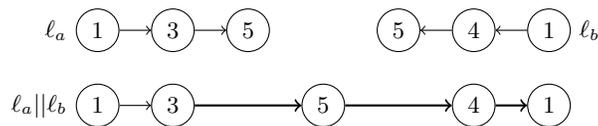


Fig. 1: Example of merging two partial cycles.

Lemma 2. *Dynamic programming with dominance and bidirectional search finds two labels ℓ_a and ℓ_b such that $\ell_a || \ell_b$ represents a length-feasible cycle with minimum reduced cost.*

Proof. Let $C = (i_0, i_1, \dots, i_K)$ be a cycle with minimum reduced cost that is found by some version of full, monodirectional forward search with dominance checks. Suppose $\vec{\mathcal{L}}_C = (\vec{\ell}_0, \vec{\ell}_1, \dots, \vec{\ell}_K)$ is the sequence of labels leading to C in this search, then choose $\vec{\ell}_k$ to be the first label in $\vec{\mathcal{L}}_C$ where $t(\vec{\ell}_k) \geq q(\vec{\ell}_k)/2$ is satisfied. Hence, label $\ell_a := \vec{\ell}_k$ is also explored in bidirectional search. It represents the path (i_0, i_1, \dots, i_k) .

Next, consider a sequence of labels that would generate the same cycle, but in reverse order, i.e., $(i_K, i_{K-1}, \dots, i_0)$. Denote this sequence of labels by $\vec{\mathcal{L}}_C = (\vec{\ell}_K, \vec{\ell}_{K-1}, \dots, \vec{\ell}_0)$, then $\vec{\ell}_k$ represents the path $(i_K, i_{K-1}, \dots, i_k)$. From $t(C) = t(\vec{\ell}_k) + t(\vec{\ell}_k) \leq q(C)$ and $t(\vec{\ell}_k) \geq q(\vec{\ell}_k)/2$ it follows that

$$t(\vec{\ell}_k) = t(C) - t(\vec{\ell}_k) \leq q(C) - \frac{q(\vec{\ell}_k)}{2} \leq q(C) - \frac{q(C)}{2} = \frac{q(C)}{2} \leq \frac{q(\vec{\ell}_k)}{2},$$

hence $\vec{\ell}_k$ is length-feasible and within the halfway point. Therefore, unless it is dominated, $\vec{\ell}_k$ is found by bidirectional search, and we can choose $\ell_b := \vec{\ell}_k$, since $\ell_a || \ell_b$ represents C . If $\vec{\ell}_k$ is discarded due to dominance, then (by transitivity of dominance) bidirectional search must find another label that dominates $\vec{\ell}_k$. By the conditions of Lem. 1 we show that ℓ_b can be chosen as this dominating label.

Labels ℓ_a and ℓ_b trivially start at the same node $s = i_0$, and due to $v(\ell_b) = v(\vec{\ell}_k) = i_k$, they also end at the same node i_k . Due to $\mathcal{N}(\ell_b) \subseteq \mathcal{N}(\vec{\ell}_k)$, we have $\mathcal{N}(\ell_a) \cap \mathcal{N}(\ell_b) \subseteq \mathcal{N}(\ell_a) \cap \mathcal{N}(\vec{\ell}_k) = \{i_k\}$, i.e., they do not overlap otherwise. Hence, they can be merged to $\ell_a || \ell_b$. It remains to show that this merged label is length-feasible and has the same reduced cost as C . This follows from the fact that $\ell_a || \vec{\ell}_k$ is feasible and optimal, and $t(\ell_b) \leq t(\vec{\ell}_k)$, $q(\ell_b) \geq q(\vec{\ell}_k)$, and $\bar{c}(\ell_b) \leq \bar{c}(\vec{\ell}_k)$ hold by domination.

Note that bidirectional search may produce more cycles than monodirectional search. The reason is that many of the labels at the halfway point, could be dominated by labels found later in monodirectional search. Adding these potentially sub-optimal cycles to the RMP can aid the convergence of column generation but it can make re-optimizing much harder. We try to safe-guard against this effect by sorting the cycles by reduced cost and returning at most a fixed number (50 in our implementation) of the ones with lowest reduced cost.

3 Branch-and-Price for the Length-Constrained Cycle Partition Problem

In the following, we discuss different aspects of how we extend the column generation method entailed by Sec. 2 into an exact branch-and-price algorithm.

Branch and Bound. We use a standard branching strategy based on implicit edge variables as described in [4]. We branch on the most used edge variable that does not belong to a singleton cycle. The rationale is that the most used edge would create the biggest disturbance in the subproblem, leading to an early fathoming of the subproblem due to infeasibility or finding a feasible integer solution. For branch-and-bound node selection, we employ the best-estimate rule with plunging described in [1]. After selecting a branch-and-bound node, we solve the LP relaxation using column generation. In each pricing iteration the dynamic programming algorithm from Sec. 2 is called from each node starting node $s \in V$. As these calls are independent they are run in parallel.

Symmetry Breaking. For each call to the dynamic program, i.e., from a fixed starting node s , the search can safely ignore all nodes with an index less than the starting node. Since each cycle contains a unique node with the lowest index, this restriction prevents the generation of an equivalent cycle with a different starting node. A further improvement on this restriction is to sort the node indices by their critical time, such that the pricing problem that explores the most nodes starts from the smallest critical time, allowing earlier pruning due to feasibility.

Heuristic and Exact Pricing. We start each pricing call with a faster, incomplete dynamic programming algorithm that relaxes the subset conditions in the dominance rule (4c). If this fails to find a cycle with negative reduced cost, then exact pricing is performed. From exact pricing we always compute a Lagrangian lower bound [6]. Let z_s^* be the optimal objective value of the pricing problem for a fixed starting node s , and let z_{MP}^* , z_{RMP}^* be the optimal objective value of MP and RMP, respectively, then $z_{MP}^* \geq z_{RMP}^* + \sum_{s \in V} z_s^* =: LB_{lg}$. From integrality of the objective function, we can use $\lceil LB_{lg} \rceil$ as a valid lower bound.

Early Branching. Another acceleration idea based on integrality, inspired by the technique in [17], is to skip pricing in some branch-and-bound node Q by looking at the lower bound $\lceil LB_{lg} \rceil$ of the parent node. Let z_{RMP}^* be the first RMP objective value for Q obtained after removing columns due to branching. If $\lceil z_{RMP}^* \rceil = \lceil LB_{lg} \rceil$, then computing z_{MP}^* exactly will not improve over the lower bound of the parent node. Therefore, we can skip pricing in this subproblem and perform early branching.

Farkas Pricing and RMP Initialization. If the RMP at a subproblem becomes infeasible due to branching and removal of columns, we use Farkas pricing [3, 19] to generate new columns that render the RMP feasible again, or to prove that the MP is infeasible. The root node RMP is always feasible, because we initialize it with the trivially feasible singleton cycles and the cycles in the primal solution generated by the Most-Critical-Vertex-Based Heuristic (MCV) from [11].

Exploiting the Triangle Inequality. The algorithm developed up to this point works for any choice of nonnegative travel times. For many real-world applications, such as in aerial vehicle routing where distances are Euclidean, we know additionally that the triangle inequality holds, i.e., that $t_{i,j} \leq t_{i,k} + t_{k,j}$ for all $\{i, j\}, \{i, k\}, \{j, k\} \in E$. Under this assumption, the set partitioning formulation can be turned into a set covering formulation, which is easier to solve

because the dual solution π becomes nonnegative. A solution for the set partitioning formulation can always be retrieved from a set covering solution by removing nodes that appear in more than one cycle from all but one cycle. This is guaranteed to be feasible since removing a node could only decrease the total travel time and increase the minimum critical time of the cycle. By the same arguments, we can prove the following lemma that show that nodes $i \in V$ with $\pi_i = 0$ can be temporarily ignored during dynamic programming.

Lemma 3. *Suppose travel times satisfy the triangle inequality, and we are given a dual solution $\pi \in \mathbb{R}^V$, then there exists an optimal cycle C to the pricing problem $\max_{C \in \Omega} \sum_{i \in C} \pi_i$ such that $\pi_i > 0$ for all $i \in C$.*

4 Computational Results

We implemented the branch-and-price algorithm described in Sec. 3 using the Python wrapper of the branch-and-price framework SCIP [2, 15], interfacing to a Rust implementation for solving the pricing problem. It calls the dynamic programming method from Sec. 2 from each starting node in parallel using the Rayon Rust library [16]. The rest of SCIP is sequential. All experiments were run with a time limit of two hours on a cluster of identical machines equipped with Intel(R) Xeon(R) Gold 5122 processors with 3.6 GHz and 96 GB of RAM; each chip had 8 cores, and allowed for 16 threads to run in parallel. The code and the instance-wise results of the experiments are publicly available on GitHub at <https://github.com/mmghannam/lccp-bnp>.

The goal of our experiments was to analyze the performance of our branch-and-price algorithm, in particular to answer the following questions:

1. How does the new method compare to the previous approach from [11] in terms of speed and ability to solve instances to optimality?
2. How tight is the linear relaxation of the set partitioning formulation in general and in comparison to the compact formulation from [11]?
3. What is the performance impact of the individual acceleration techniques?

Moreover, we were curious whether it matters if the travel times respect the triangle inequality, and how the performance of the algorithms is affected if we modify travel times in the above instances to satisfy the triangle inequality.

We use the standard benchmark set from [11], which consists of 84 instances that have between 14 and 100 nodes. All algorithms evaluated were initialized with a solution provided by the MCV heuristic from [11]. We compare results for three major algorithms:

- BNC-SEC: branch and cut with subtour elimination constraints from [11] based on Gurobi 10.0.3 [8]
- BNP-BASIC: plain branch and price without any improvement technique
- BNP-FULL: branch and price with all improvement techniques

Additionally, we ran the following variations of BNP-FULL to quantify the impact of four components:

- BNP-NOBIDIR: branch and price with monodirectional labeling
- BNP-NOPAR: branch and price without parallelization
- BNP-NOSYMBR: branch and price without symmetry breaking
- BNP-NOEARLY: branch and price without early branching

Note that the labeling algorithm in BNP-BASIC already performs dominance checks, since otherwise not more than the smallest instances could be solved.

In Tab. 1 we report the shifted geometric mean of the solving times for all instances solved by at least one algorithm. We use a shift of 1 second; time outs are included with the time limit. Column “Ratio” states the relative solving time w.r.t. the fastest solver BNP-FULL. In Fig. 2 we plot the number of solved instances over time for each algorithm.

Overall Results. We can first observe that BNP-FULL drastically outperforms BNC-SEC, on average being more than an order of magnitude (14.6 times) faster. BNP-FULL is able to solve 13 more instances to optimality. The largest instance solved features 76 nodes compared to 52 nodes previously. Notably, all 39 instances solved by BNC-SEC are also solved by BNP-FULL.

LP Relaxations. While BNC-SEC could solve the root relaxation (including sub-tour elimination) on all 84 instances, the column generation loop of BNP-FULL converged to an optimal root LP solution for only 56 instances, running into the memory or time limit in the remaining cases. Nonetheless, whenever computed successfully, the root dual bound of the set partitioning formulation proves to be significantly tighter than the root dual bound of BNC-SEC. For the instances where both algorithms could solve the linear relaxation at the root node, the average gap with respect to the best known primal solution is 1.1% for BNP-FULL vs. 23.6% for BNC-SEC. In addition, on instances for which the optimal objective value z^* is known the root dual bound of the set partitioning formulation (rounded up due to integrality) is at least $z^* - 1$. This finding suggests that the set-partitioning model may have a round-up property similar to that investigated in [20].

Primal Bounds. In [11] it is reported that the MCV heuristic is effective in producing good, and often optimal, solutions. Specifically, it finds the optimal solution for 21 out of the 39 instances that are solved by both BNC-SEC and BNP-FULL. For the 13 instances newly solved by BNP-FULL, the heuristic finds the optimal solution for only 2 of them, displaying the importance of the exact branch-and-price run also on the primal side. Interestingly, for instances where both algorithms time out, the primal bounds imposed by the MCV heuristic are hard to improve on. Only BNP-FULL manages to improve the primal bound for 2 of these instances, by a value of 1.

Impact of the Improvement Techniques. Bidirectional labeling is the most effective amongst the improvement techniques. In comparison, monodirectional labeling generates substantially more labels and results in a 4.4 times slowdown

Solver	Time [s]	Ratio	Solved
BNC-SEC	142.8	14.6	39
BNP-BASIC	189.1	19.3	30
BNP-FULL	9.8	1.0	52
BNP-NOBIDIR	43.1	4.4	40
BNP-NOPAR	12.5	1.3	50
BNP-NOSYMBR	19.6	2.0	45
BNP-NOEARLY	16.0	1.6	45

Tab. 1: Aggregated results.

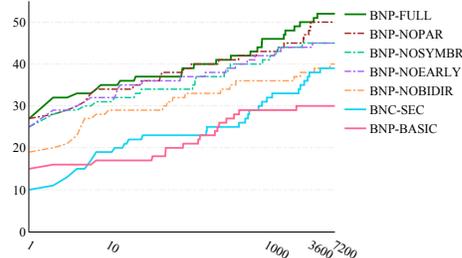


Fig. 2: Instances solved over time.

overall. The remaining improvement techniques also contribute significantly to the performance, although to a lesser degree. While disabling symmetry breaking and early branching each lead to 7 fewer instances solved, parallelization only helps to solve two more instances to optimality. Last but not least, the importance of the improvement techniques is underlined by the fact that BNP-BASIC, the algorithm without any improvement techniques, solves significantly fewer instances and is slower than BNC-SEC.

Triangle Inequality. Finally, we modified the 84 instances from [11] in order to create a new set of instances that respect the triangle inequality by replacing the travel time on each by the distance of the shortest paths between its nodes. Curiously, this change makes the instances harder to solve for both approaches. On the new benchmark set imposing the triangle inequality, the mean solving time is increased to 15.0 seconds for BNP-FULL and 187.1 seconds for BNC-SEC. BNP-FULL solves 4 instances less and BNC-SEC solves 1 instance more than on the original test set. These slowdowns may be explained by the shorter edge lengths leading to a larger number of length-feasible cycles.

Note that the above results for BNP-FULL do not yet exploit the triangle inequality. The improvements described in Sec. 3, i.e., to use a set covering formulation and to remove nodes with zero duals during labeling, enable it to solve 7 more instances (55 in total) and reduce the average solving time to 7.5 seconds. This is less than on the original test set, and around 25.2 times faster than BNC-SEC.

Conclusion. To summarize, our branch-and-price algorithm for the length-constrained cycle partition problem proves to outperform the state of the art significantly, but the results also show that this is only possible with the help of some nontrivial improvement techniques. These allow us to close 13 open instances from the standard benchmarks, and to scale to larger instances with up to 76 nodes, compared to 52 nodes for the previous branch-and-cut method. Future work should investigate whether the use of cutting planes can improve the performance of the branch-and-price algorithm even further, and whether the improvement techniques developed for LCCP are also valuable for similar problems such as kidney exchange.

References

1. Achterberg, T.: Constraint Integer Programming. Doctoral Thesis, Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften, Berlin (2007). <https://doi.org/10.14279/depositonce-1634>
2. Bestuzheva, K., Besançon, M., Chen, W.K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S.J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M.E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D., Witzig, J.: Enabling research through the SCIP Optimization Suite 8.0. *ACM Transactions on Mathematical Software* (2023). <https://doi.org/10.1145/3585516>
3. Ceselli, A., Gatto, M., Lübbecke, M.E., Nunkesser, M., Schilling, H.: Optimizing the cargo express service of Swiss Federal Railways. *Transportation Science* **42**(4), 450–465 (2008). <https://doi.org/10.1287/trsc.1080.0246>
4. Costa, L., Contardo, C., Desaulniers, G.: Exact Branch-Price-and-Cut Algorithms for Vehicle Routing. *Transportation Science* **53**(4), 946–985 (2019). <https://doi.org/10.1287/trsc.2018.0878>
5. Desaulniers, G., Desrosiers, J., Solomon, M.: Column Generation. Springer New York, NY (2005). <https://doi.org/10.1007/b135457>
6. Desrosiers, J., Lübbecke, M.: A Primer in Column Generation, pp. 1–32. Springer New York, NY (2006). https://doi.org/10.1007/0-387-25486-2_1
7. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting-stock problem. *Operations Research* **9**(6), 849–859 (1961). <https://doi.org/10.1287/opre.9.6.849>
8. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023), <https://www.gurobi.com>
9. Hoppmann, K., Mexi, G., Burdakov, O., Casselgren, C.J., Koch, T.: Minimum cycle partition with length requirements. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 273–282. Springer (2020)
10. Hoppmann-Baum, K.: Mathematical programming for stable control and safe operation of gas transport networks (2022)
11. Hoppmann-Baum, K., Burdakov, O., Mexi, G., Casselgren, C.J., Koch, T.: Length-constrained cycle partition with an application to UAV routing. *Optimization Methods and Software* **37**(6), 2080–2116 (2022). <https://doi.org/10.1080/10556788.2022.2053972>
12. Irnich, S.: Resource extension functions: properties, inversion, and generalization to segments. *OR Spectrum* **30**(1), 113–148 (2008). <https://doi.org/10.1007/s00291-007-0083-6>
13. Kantorovich, L.V., Zalgaller, V.A.: Ratsionalnyj raskroj promyshlennykh materialov [Rational cutting of industrial materials]. Lenizdat, Leningrad (1951)
14. Lam, E., Mak-Hau, V.: Branch-and-cut-and-price for the cardinality-constrained multi-cycle problem in kidney exchange. *Computers & Operations Research* **115**, 104852 (2020)
15. Maher, S., Miltenberger, M., Pedroso, J.P., Rehfeldt, D., Schwarz, R., Serrano, F.: PySCIPopt: Mathematical programming in Python with the SCIP optimization suite. In: *Mathematical Software – ICMS 2016*, pp. 301–307. Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-42432-3_37

16. Matsakis, N., Stone, J.: Rayon: A data parallelism library for Rust. <https://github.com/rayon-rs/rayon>
17. Mehrotra, A., Trick, M.A.: A column generation approach for graph coloring. *INFORMS Journal on Computing* **8**(4), 344–354 (1996). <https://doi.org/10.1287/ijoc.8.4.344>
18. Miller, C.E., Tucker, A.W., Zemlin, R.A.: Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)* **7**(4), 326–329 (1960)
19. Nunkesser, M.: Algorithm design and analysis of problems in manufacturing, logistic, and telecommunications. An algorithmic jam session. Doctoral thesis, ETH Zurich, Zürich (2006). <https://doi.org/10.3929/ethz-a-005281028>
20. Scheithauer, G., Terno, J.: The modified integer round-up property of the one-dimensional cutting stock problem. *European Journal of Operational Research* **84**(3), 562–571 (1995)
21. Tilk, C., Goel, A.: Bidirectional labeling for solving vehicle routing and truck driver scheduling problems. *European Journal of Operational Research* **283**(1), 108–124 (2020). <https://doi.org/10.1016/j.ejor.2019.10.038>
22. Tilk, C., Rothenbächer, A.K., Gschwind, T., Irnich, S.: Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research* **261**(2), 530–539 (2017). <https://doi.org/10.1016/j.ejor.2017.03.017>
23. Uchoa, E., Sadykov, R.: Kantorovich and Zalgaller (1951): the 0-th Column Generation Algorithm
24. Wolsey, L.: *Integer Programming*. John Wiley & Sons, Ltd (2020). <https://doi.org/10.1002/9781119606475>