Column Generation by Subset Enumeration and Pruning for the Multi-Pickup and Delivery Problem with Time Windows

Xavier Tepoorten, Connor Kikkert, Lucca Herbert
Student ID: 47434301, 47480700, 48013897

31st October 2025

1 Abstract

This report presents a Column Generation approach enhanced with dynamic pruning for the multi-pickup and delivery problem with time windows (MPDPTW). The framework decomposes the problem into a master and a subproblem: the subproblem generates feasible single-vehicle routes by solving a pricing problem that either returns the best route for a subset of requests or declares infeasibility, while the master problem selects the optimal combination of routes to cover all requests. The dynamic pruning mechanism incrementally explores combinations of requests, discarding any subset that cannot form a feasible route and skipping all of its supersets, thereby reducing redundant evaluations. A capacity correction step ensures load feasibility across routes.

Parallelisation and pruning jointly reduce the computational burden of Column Generation without compromising accuracy. Experiments on benchmark instances inspired by online food delivery show that the proposed model solves 116 out of 120 instances to optimality and produces high-quality feasible solutions for both the uncapacitated and capacitated variants of the problem. The capacitated variant is handled by correcting route loads after the master problem identifies the optimal combination of uncapacitated routes. Although the dataset is synthetically generated, the formulation provides insight into how Column Generation with pruning can be applied to large-scale delivery and routing problems.

2 Introduction

The industry problem that was identified in this report was that online food delivery platforms face the challenge of assigning drivers with limited carrying capacity to collect items from multiple restaurants and deliver them as a single order to the customer within promised time windows. In short, we want to route a limited fleet of vehicles from a depot to a pick up location and deliver paired requests, within time windows, without breaking vehicle capacity, and minimise total cost/ travel time.

With the rise of apps and businesses like Uber, Deliveroo, Menulog and so forth, the idea of multi-side marketplaces in which customer demand for things like travel, fast-food etc is matched to a supply of drivers or delivery vectors, have taken hold as a very prominent industry concept. Minimising expenditure and costs within this dynamic environment, whilst ensuring customer and stakeholder satisfaction has now become a billion dollar problem.

3 Background & Paper Summary

3.1 Problem setup:

In their paper "Exact algorithms for the multi-pickup and delivery problem with time windows," Aziez et al. (2020) propose a a MIP (mixed integer program) to solve the MPDPTW, and they experimented with three initial formulations to setup the problem.

First, a Two-index solution was utilised, which was previously implemented in a paper by Naccache et al. (2018) [7], followed by a Three-index approach and finally an asymmetric representatives formulation (ARF), inspired by Jans and Desrosiers (2013) [4].

3.2 3 Index Approach

The Three-index formulation includes both arcs and vehicles, which includes a binary term x_{ij}^k indexed both by the arc (i, j) and vehicle k, then another

binary term y_{rk}, indexed by request r and vehicle k. This formulation makes vehicle assignment explicit but greatly increases the number of variables and model size.

3.3 2 Index Approach

The Two-index formulation is similar to a traditional vehicle routing model, but instead of assigning arcs to specific vehicles it only uses a binary term x_{ij} to show if any vehicle uses the arc (i, j), with extra constraints added to enforce capacity, pairing, and time windows.

3.4 Asymmetric Representative Formulation (ARF)

Aziez et al. (2020) propose the Asymmetric Representative Formulation (ARF) to address the high symmetry inherent in traditional two- and Three-index formulations of the MPDPTW. In such formulations, identical vehicles can often be interchanged without affecting solution quality, resulting in a large number of equivalent solutions and an expanded search space.

The ARF introduces asymmetry by constructing K clusters, each anchored by a representative request k. Only requests with higher indices can be assigned to cluster k, effectively preventing equivalent vehicle permutations. In this formulation, a binary variable, x_{ij}^k indicates whether arc (i, j) is used within cluster k, and a second binary variable, y_r^k denotes whether request r is assigned to cluster k. Each cluster therefore represents a potential route, and a feasible solution is obtained by selecting a subset of clusters that together cover all requests.

This asymmetric representation significantly reduces redundant solutions and improves solver efficiency, particularly in instances where symmetry dominates the search space.

3.5 Other literature on the MPDPTW

There have been other literature published following Aziez et al. (2020) that further expand upon possible formulations and constraints that can improve upon computation times for the problem instances used in their paper.

"A capacitated multi pickup online food delivery problem with time windows: a branch-and-cut algorithm" by Kohar and Jakhar (2021) [5] improves upon the results presented in Aziez et al. (2020) by augmenting their Two-index, while also including capacity restraints to make the formulation more realistic, resulting in a stronger branch-and-cut algorithm in comparison to Aziez et al. (2020). The Two-index formulation in Kohar and Jakhar (2021) adds two additional variables, R_{mn}: equals to 1, if requests $m \in R$ and $n \in R$ are served by same vehicle, 0 otherwise. and V_{rk}: equals to 1, if a vehicle $k \in K$ serves the request $r \in R$, 0 otherwise. The addition of these variables and the capacity value Q allows for more constraints to be added in the formulation, and strengthens the set of infeasible pairs of requests W, by adding valid inequalities that ensure that requests being considered are not being served on the same route or by the same vehicle. This change in formulation led to a model that had a larger variable space than the Two-index formulation in Aziez et al. (2020), but still much smaller than the ARF and Three-index formulations, while in contrast, the number of constraints is greater than the previous Two-index formulation. The decrease in computation time due to the amount of variables seemed to outweigh the additional computation time required for the new constraints introduced into the formulation. Paired with the addition of capacity constraints, this lead to an improvement in the results as compared to Aziez et al. (2020). [5]

"Strong cutting planes for the capacitated multi-pickup and delivery problem with time windows" by Kohar, Jakhar, and Agarwal (2023) [6] extends the capacitated multi-pickup and delivery problem with time windows (MPDPTW) through a tighter Two-index mixed-integer linear formulation integrated within a branch-and-cut framework focused on the design and efficient separation of cutting planes. Building on the earlier work by Kohar and Jakhar (2021) [5], which strengthened the Two-index formulation of Aziez et al. (2020) [2] by introducing capacity-related variables and constraints, the 2023 study shifts the emphasis from model enhancement to the generation of strong cuts.

Whereas the 2021 formulation primarily improved model realism through explicit capacity representation, the 2023 approach focuses on strengthening the relaxation through a richer family of cutting planes and improved separation procedures. This shift results in tighter LP bounds, smaller root gaps, and faster convergence across benchmark instances from Aziez et al. (2020), highlighting the value of

systematic cut design in enhancing branch-and-cut performance for the capacitated MPDPTW.

"Efficient Heuristic to Solve the Multi-pickup and Delivery Problem with Time Windows" by Bouhlla et al. (2024) [3] also improves upon the results presented in Aziez et al. (2020) [2], and this was done by utilising the same Three-index formulation, however they implemented an advanced construction operator in their algorithm. This algorithm first requires:

$$R : \text{the } n \text{ requests to serve}$$

$$S_{\text{best}} : \text{best solution seen}$$

$$S_i : \text{current working solution}$$

$$d_{\max} : \text{max diversification degree}$$

$$URequests : \text{temporarily unrouted requests}$$

$$ms_i : \text{number of non-empty routes in } S_i$$

$$\text{iter}_{\max} = \alpha \cdot n : \text{patience/plateau limit}$$

The algorithm then begins by removing $d \sim \text{uniform}\{1, \ldots, d_{\max}\}$ requests from S_i, then collects $URequests$ from the "partially ruined" solution and then recreates S_i by reinserting those requests using a heuristic that tries to keep costs/feasibility good (e.g., cheapest-feasible insertion with penalties for TW violations, load, ride-time, etc.).. This rebuilt solution is then searched for a local optimum.
This algorithm is an improvement upon the branch-and-cut framework from Aziez et al. (2020) as this algorithm is heuristic in nature, it does not search "exactly" for the most optimal solution, it instead finds good feasible solutions quickly and then searches for improvements upon them. When implemented, this reduces the computation time, as the algorithm stops after αn iterations without improvement — meaning runtime grows roughly linearly with n, however the branch-and-cut utilised by Aziez et al. (2020), can be exponential in problem-size in the worst case scenario.
.

3.6 The Data

The benchmark instances are the same as those provided by the original authors. Each instance is characterised by three features: (i) *time-window type*, (ii) *request length*, and (iii) *total number of nodes*. The time-window type controls scheduling flexibility: N-type instances have narrow local time windows at each pickup and delivery; L-type instances have wider local windows (more flexibility but still per-node limits); and W-type instances remove local windows entirely and instead impose a single global time horizon, yielding many more feasible timing combinations. The request length indicates how many pickup–delivery nodes a request contains (four or eight nodes per request). The total number of nodes is 25, 35, 50, or 100, covering both small verification cases and larger instances for scalability assessment. For each *(time-window type, request length, node count)* combination, five distinct instance variants are provided, yielding a total of $3 \times 2 \times 4 \times 5 = 120$ instances.

4 Methodology & Implementation:

4.1 Column Generation

To reduce the computational time of the optimisation process, we implemented a Column Generation (CG) algorithm. This approach was motivated by preliminary testing of the earlier formulations, where we observed that the LP solver could readily (and in many cases, almost instantly) handle cases in which multiple requests were assigned to a single route. These observations suggested that generating columns this way could be an effective strategy.

4.1.1 Sub-problem:

The sub-problem for the Column Generation procedure acts as a route builder. Given a subset of requests, it aims to find the best feasible route for a single vehicle that serves all pickups and deliveries in order, within their respective time windows (and capacity limits, when applicable). Instances without time window constraints are solved without enforcing these bounds (as discussed in 4.6.1).

Let R denote the set of all requests, V the set of all nodes (pickup, delivery, depot, and sink), A the set of all directed arcs between nodes, and $N = V \setminus \{\text{depot, sink}\}$ the set of non-depot nodes. The sub-problem formulation is defined as follows:

Sets:

$A_\rho \subseteq A$	Set of feasible arcs for a given subset
$V_\rho \subseteq V$	Set of nodes for a given subset
$N_\rho = V_\rho \setminus \{\text{depot, sink}\}$	

Data:

$\rho \subseteq R$	The subset of requests to construct a route for
$i = 0$	starting depot node (depot)
$i = p + n + 1$	final depot node (sink)
$\{a_i, b_i\}$	time window to begin service at node $i \in N$
s_i	service time at node $i \in V$
t_{ij}	travel time/cost for arc $(i, j) \in A$
d_r	delivery node for request $r \in R$
$r(i)$	request associated with node $i \in N$
$M_{ij} =$	for $(i, j) \in A$
$\max\{0,\ b_i + s_i + t_{ij} - a_j\}$	

Variables:

$x_{ij} \in \{0, 1\}$	1 if the route uses arc $(i, j) \in A_\rho$; 0 otherwise
$S_i \in \mathbb{R}^+$	Start time at node $i \in V_\rho$

Objective Function:

$$\min \sum_{(i,j)\in A_\rho} x_{ij}\,(t_{ij} + s_i) \tag{1}$$

Time-Window Constraints:

$$a_i \le S_i \le b_i, \quad \forall\, i \in V_\rho \tag{2}$$

$$S_i + s_i + t_{ij} - M_{ij}\,(1 - x_{ij}) \le S_j, \quad \forall\,(i, j) \in A_\rho \tag{3}$$

Degree/Flow Constraints:

$$\sum_{i\in A_\rho^-(j)} x_{ij} = 1, \quad \forall\, j \in N_\rho \tag{4}$$

$$\sum_{j\in A_\rho^+(i)} x_{ij} = 1, \quad \forall\, i \in N_\rho \tag{5}$$

$$\sum_{i\in A_\rho^-(\omega)} x_{i\omega} = 1 \tag{6}$$

$$\sum_{j\in A_\rho^+(0)} x_{0j} = 1 \tag{7}$$

4.1.2 Master Problem:

The master problem in Column Generation acts as a route selector: it chooses a combination of routes from the set of feasible routes returned by the sub-problem in order to cover every request exactly once. The objective is to minimise the total travel time of the selected routes.
The sub-problem serves as a feasibility check that accounts for service times. Given a feasible subset ρ covering the node set N_ρ with sub-problem objective value t^*, the corresponding column in the master problem has cost:

$$\text{cost}_\rho = t^* - \textstyle\sum_{i\in N_\rho} s_i$$

The master problem can be formulating with the following:

Sets:

K	The set of vehicles
$\mathcal{R} \subseteq \mathcal{P}(R) \setminus \{\emptyset\}$	Set of all feasible routes.
$\mathcal{R}_i = \{\, \rho \in \mathcal{R} \mid i \in \rho \,\}$	Set of routes that include request $i \in R$.

Data:

cost_ρ	Travel time (or cost) of route $\rho \in \mathcal{R}$

Variables:

$z_\rho \in \{0, 1\}$	1 if route $\rho \in \mathcal{R}$ is selected; 0 otherwise

Objective Function:

$$\min \sum_{\rho\in\mathcal{R}} z_\rho\, \text{cost}_\rho \tag{8}$$

Constraints:

$$\sum_{\rho\in\mathcal{R}_i} z_\rho = 1, \quad \forall\, i \in R \tag{9}$$

$$\sum_{\rho\in\mathcal{R}} z_\rho \le |K| \tag{10}$$

4.2 Preprocessing:

4.2.1 Infeasible Pairs

Aziez et al. (2020) [2] includes pre-processing to spare the solver from handling feasibility checks, thereby accelerating the optimisation process. This is done by defining the potential groups of requests beforehand, so a set W of infeasible request pairs can be generated, to be eliminated whenever they appear in a cluster. While they used a heuristic based approach in the form of ALNS (Adaptive Large Neighbourhood Search), our Column Generation subproblem model already can easily determine feasibility between two requests. By cutting out arcs between infeasible pairs of requests as part of the preprocessing (outlined in 4.2.3), this can help speed up the sub-problem model.

4.2.2 Time-Window Tightening

While the data provides an initial earliest and latest time window for a node (a, b) (Naccache et al., 2018), these bounds can be improved by applying the tightening procedure described in Section 5.1.2 of [2], which follows the algorithm of Ascheuer, Fischetti, and Grötschel [1].

Step 1: $a_k \leftarrow \max\{a_k, \min_{i \in A^-(k)}(a_i + s_i + t_{ik})\},\ \forall k \in N \mid A^-(k) \neq \emptyset.$

Step 2: $a_k \leftarrow \max\{a_k, \min\{b_k, \min_{j \in A^+(k)}(a_j - s_k - t_{kj})\}\},\ \forall k \in N \mid A^+(k) \neq \emptyset.$

Step 3: $b_k \leftarrow \min\{b_k, \max\{a_k, \max_{i \in A^-(k)}(b_i + s_i + t_{ik})\}\},\ \forall k \in N \mid A^-(k) \neq \emptyset.$

Step 4: $b_k \leftarrow \min\{b_k, \max_{j \in A^+(k)}(b_j - s_k - t_{kj})\},\ \forall k \in N \mid A^+(k) \neq \emptyset.$

The authors [2] further tighten the time windows using an algorithm based on fully enumerating all possible routes. While this may be feasible in a single preprocessing run, our Column Generation model must solve pricing subproblem thousands of times, making such an approach computationally infeasible. Instead, after running the 4-steps above, we propose a quick and easy time-window tightening that applies directly to all delivery nodes:

Step 1: $k^* \longleftarrow \arg\max_{k \in P_r}\{a_k\}$

Step 2: $a_{d_r} \longleftarrow \max\{a_{d_r},\ a_{k^*} + s_{k^*} + t_{k^* d_r}\}$

This tightening ensures that the delivery time window begins no earlier than the latest earliest start time among its associated pickups, plus the service time at that pickup and the travel time from the pickup to the delivery node.

4.2.3 Arc pruning

During pre-processing, infeasible arcs can be easily pruned. For this implementation, the following rules (Aziez et al., 2020 [2]) were applied to all instances:

1. Arcs $(0, d_r)$, $(p, 0)$, and (d_r, p) are infeasible for each request $r \in R$ with pickup node $p \in P_r$ and drop node d_r.
2. Arc $(i, j) \in A$ is infeasible if $a_i + s_i + t_{ij} > b_j$.
3. Arc $(i, j) \in A$ is infeasible if $(r(i), r(j)) \in W$.

4.3 Lazy Constraints:

Aziez et al. (2020) also utilise valid inequalities, as the basic formulation of the problem can allow for infeasible solutions in the linear programming relaxation. Such valid inequalities ensure that these infeasible solutions are cut from all possible solutions. These inequalities are added in our sub-problem model as lazy constraints.

4.3.1 Subtour Elimination Constraints (LSEC):

LCES are constraints where possible solutions including sub-tours, which are cycles in a route that do not include the depot, are removed. These LCES are split into two constraints, the successor and predecessor inequalities. The successor inequality forbids a subset S of nodes from forming an isolated sub-tour by enforcing that any deliveries in S must remain connected to their pickups, while the predecessor inequality enforces that any pick ups in S must remain connected to their deliveries, which both ensure at least one arc must leave S.

$$x(S) + \sum_{i \in \bar{S} \cap \sigma(S)} \sum_{j \in S} x_{ij} + \sum_{i \in \bar{S} \setminus \sigma(S)} \sum_{j \in S \cap \sigma(S)} x_{ij} \leq |S| - 1 \quad (11)$$

$\forall S \subseteq P,\ r \in R$

$$x(S) + \sum_{i \in S} \sum_{j \in \bar{S} \cap \pi(S)} x_{ij} + \sum_{i \in S \cap \pi(S)} \sum_{j \in \bar{S} \setminus \pi(S)} x_{ij} \leq |S| - 1 \quad (12)$$

$\forall S \subseteq P,\ r \in R$

4.3.2 Infeasible Path Constraints:

Each formulation implements infeasible path constraints, which aim to forbid paths that are considered infeasible as they breach conditions specific to the MPDPTW. The types of paths that are considered

infeasible are those that breach the precedence constraints, where a vehicle must leave the depot, must visit request delivery nodes after visiting the request pickup nodes, and must end at the depot. Additionally, paths with nodes that are pairs of infeasible requests are forbidden, along with groups of infeasible requests that share common arcs (fork constraints).

$$\sum_{j \in S} x_{0j} + \sum_{i \in S \cup \{d_r\}} \sum_{j \notin S \cup \{d_r\}} x_{ij} \leq |S| \quad (13)$$

$\forall S \subseteq P,\ r \in R$

Constraint (13) ensures that for any subset S of requests, the total number of arcs entering S from the starting depot node (0) plus the arcs leaving S together with the associated delivery node d_r cannot exceed $|S|$. This further prevents sub-tours from occurring by ensuring the model cannot choose a closed cycle of nodes which are disconnected from the depot, while also including the delivery node d_r to respect the pickup–delivery precedence.

$$\sum_{j \in S} x_{d_r j} + \sum_{i \in S} \sum_{j \notin S} x_{ij} + \sum_{i \in S} x_{ip} \leq |S| \quad (14)$$

$\forall S \subseteq P,\ r \in R$

Constraint (14) ensures that for any subset S of requests, the sum of the total number of arcs entering S from the the delivery node of the request d_r, plus the arcs that start in S and leave S, and plus the arcs from S to the pickup request p is less than $|S|$. This constraint also prevents sub-tours from occurring as this inequality ensures the number of arcs that connect with either of the pickup and delivery pair and the arcs in S are capped by $|S|$. Additionally, including both p and d_r in the constraints means that the pick-up delivery order cannot be satisfied in a way such that a sub-tour would be feasible.

$$\sum_{i \in S \cup \{p\}} \sum_{j \in S \cup \{p\}} x_{ij} \ + \sum_{i \in S} x_{i,\, p+n+1} \ \leq \ |S| \quad (15)$$

$\forall\ p \in P_r,\ r \in R$

Constraint (15) ensures that for a subset of requests S and a pickup p, then the number of requests within S plus the arcs leaving S to the node $p+n+1$ cannot exceed $|S|$. This constraint prevents a partial route (containing a pickup without its corresponding delivery) from forming inside the set S, maintaining the precedence inherent to the MPDPTW.

4.4 Pruning in Subset Enumeration

Column Generation builds columns by generating feasible vehicle routes for subsets of requests. Enumerating all subsets is infeasible, so we prune families of subsets early whenever feasibility can be ruled out.

Pruning rules.

1. **Infeasibility propagation (Lemma 1).** If a subset S has no feasible route, then any superset $S \cup \{r\}$ is also infeasible. We record S as an *infeasible mask* and skip all supersets (e.g., if $\{1, 3\}$ is infeasible, then every $\{1, 3, \cdot\}$ is discarded).

2. **Horizon lower bound (Lemma 2).** If adding request r necessarily pushes the service-time lower bound for $S \cup \{r\}$ beyond the horizon H, the extension is discarded without solving a subproblem.

Search frontier. We maintain a frontier of feasible subsets. Starting from singletons, candidates of size $k{+}1$ are formed by adding one request to subsets in the current frontier. Each candidate is screened by Lemma 1 and Lemma 2; only survivors are solved. Feasible solutions enter the next frontier; infeasible ones yield new masks.

Lemma 1 (Monotonicity of infeasibility)**.** *Let R be a set of pickup–delivery requests and let $S \subseteq R$. A route is feasible if it starts and ends at the depot, serves all requests with pickups before deliveries, and respects node time windows (with waiting allowed). If no feasible route exists that serves all of S, then for any request $r \in R \setminus S$, no feasible route exists that serves all of $S \cup \{r\}$.*

Proof. Assume for contradiction that there exists a feasible route ρ' serving $S \cup \{r\}$. Remove the pickup and delivery nodes of r from ρ' and connect the gap directly, obtaining a sequence ρ.

Structure: ρ still starts and ends at the depot, and all requests in S maintain the pickup-before-delivery order.

Timing: Since travel times are Euclidean and satisfy the triangle inequality, the direct arc in ρ cannot increase travel time. Thus arrivals in ρ occur no later than in ρ'.

Time windows: Because waiting is permitted, earlier arrivals do not violate any time window constraints. Hence ρ remains feasible for all requests in S.

Therefore ρ is a feasible route for S, contradicting the assumption that no feasible route exists for S. Thus, if S is infeasible, any superset $S \cup \{r\}$ is also infeasible. *Note:* this argument can be adapted to include a vehicle capacity constraint but the result still stands. □

Lemma 2 (Service-time lower bound for horizon pruning)**.** *Let H be the global horizon cutoff. For a subset S of requests, let $T^*(S)$ denote the optimal (depot-to-depot) finish time of a feasible route serving all of S (with waiting allowed). Let r be a new request whose service nodes are $V(r)$ with service times $\{d_i\}_{i\in V(r)}$. If*

$$T^*(S) \;+\; \sum_{i\in V(r)} d_i \;>\; H, \tag{16}$$

then no feasible route exists that serves all of $S \cup \{r\}$, and the superset can be pruned.

Proof. Any route serving $S \cup \{r\}$ must (i) complete all services in S and (ii) perform every service in $V(r)$, each requiring time d_i. Travel and waiting times are nonnegative. Hence the finish time of any such route is at least $T^*(S) + \sum_{i\in V(r)} d_i$. If this quantity exceeds H, the horizon is violated even under zero travel and zero waiting, so $S \cup \{r\}$ is infeasible. □

4.5 Adapting for Capacity Constraint

While we initially focused on the uncapacitated variant of this problem, modifications can be made to exactly solve it with capacity constraints.

4.5.1 Subproblem model

To allow the subproblem model to compute the feasible time of a given subset with capacity included a flow based variable can be added:

Data:

$q_i \in \mathbb{R}$ Load change at node $i \in N_\rho$

$Q \in \mathbb{R}^+$ Maximum capacity of a vehicle

Variables:

$f_{ij} \in \mathbb{R}^+$ Continuous load on arc $(i,j) \in A_\rho$

Capacity Constraints:

$$\sum_{(i,j)\in A_\rho} f_{ij} \le Q \cdot x_{ij} \tag{17}$$

$$\sum_{j\in A^+(v)} f_{vj} - \sum_{i\in A^-(v)} f_{iv} = q_v \quad \forall v \in N_\rho \tag{18}$$

4.5.2 Dynamic Column Correction

The master problem generates many candidate subsets of requests, but in the final solution only a small fraction are actually selected. Adding full capacity constraints to every candidate subset is therefore wasteful and can dominate the computational budget. To address this, we use a dynamic correction step that selectively enforces capacity feasibility only on those routes that appear in the current solution.

Procedure. After solving the master problem without embedding all capacity constraints, we examine the set of routes that are selected. Then these routes are checked for possible capacity violations using a simple arc traversal loop. If the capacity is violated, we re-solve the underlying subproblem model with the capacity constraint explicitly enforced. If the route proves feasible, its column is updated with the increased cost; if infeasibility is detected, the route is discarded from the master problem along with all its supersets (per Lemma 1). This may change the master problem solution, so the optimisation is re-run with the updated column set. The process continues until no routes with capacity violations are detected in the master problem.

Convergence. Since each iteration either removes at least one infeasible column or strictly increases the cost of a route, the sequence of corrections is finite, which guarantees convergence to a stable solution.

4.6 Code Optimisations

While column pruning makes most instances computationally feasible, we applied additional code optimisations to further improve solution efficiency.

4.6.1 Gurobi Parameters

The goal of the subproblem is to determine the optimal end time of a feasible route. A route is only feasible if its end time does not exceed the sink's latest finish time, b_{sink}. Since the subproblem's objective is to compute this very end time, we can exploit this property by using Gurobi's `Cutoff` parameter.

The `Cutoff` parameter allows the solver to terminate early as soon as it finds a solution with an objective value greater than the specified cutoff. By setting `Cutoff` $= b_{\text{sink}}$, any route whose optimal end time would exceed the sink's latest finish time is automatically discarded without further search. This significantly reduces computation time because Gurobi avoids exploring infeasible or dominated solutions. Moreover, the cutoff parameter enables us to handle instances with wide time windows without explicitly embedding individual time–window constraints, since the cutoff

indirectly enforces a single global horizon on all routes.

Beyond the cutoff parameter, we also found that during testing on uncapacitated wide–time–window instances the subproblem benefited from setting Gurobi's `Heuristics` parameter to 0. While we did not formally benchmark this, we observed runtime improvements of roughly 5–10%. This most likely occurred because the subproblem model was being solved without time–window or capacity constraints; solver preprocessing through heuristics was unlikely to be beneficial, since route contiguity and pickup–delivery precedence constraints were being enforced lazily.

4.6.2 Bitmask Frontier Generation

During the development of the pruning algorithm, we opted to use a bitmask representation for routes. Let the requests be indexed $R = \{0, 1, \ldots, |R| - 1\}$. For a subset of requests $\rho \subseteq R$, the bitmask is defined as

$$\text{mask}(\rho) \;=\; \sum_{r \in \rho} 2^r.$$

Given the set of infeasible masks M, a subset ρ is infeasible if

$$\exists m \in M \quad \text{s.t.} \quad \text{mask}(\rho)\ \&\ m \;=\; m.$$

Encoding subsets (routes) with a bitmask is expected to improve runtimes, particularly as the number of subsets grows, since bit-level operations in Python are substantially faster than manipulating built-in data structures such as sets or lists. However, we did not rigorously test these improvements.

4.6.3 Multi-threading

As discussed previously, a major reason why running many LPs as sub-problems is computationally feasible is that the single-route, multi-request case can be solved extremely quickly. However, when solving instances with thousands of sub-problems, a significant portion of the total runtime is spent repeatedly setting up and loading the subproblem LP model. A straightforward improvement is therefore to multi-thread the subproblem evaluations, since most sub-problems execute just as quickly on a single thread as they do when all available threads are used.

In the implementation, parallel execution was achieved using Python's `concurrent.futures` library, specifically the `ProcessPoolExecutor` class, which allows for efficient distribution of independent subproblem evaluations across multiple processes. Due to the nature of parallel processing, noticeable performance gains were only observed once the problem instances became sufficiently large. For smaller instances, runtime occasionally increased slightly, as the overhead of initialising and managing multiple processes dominated the time required to solve the sub-problems sequentially. However, these smaller instances typically solve in under a few seconds regardless. For this reason, we dynamically enable multi-processing, activating it only when subsets reach a sufficient size. The impact of multi-threading on runtime is discussed further in the Results section.

5 Data & Results

5.1 Model Implementation and Experimental Setup

To obtain a reliable representation of our Column Generation model, we re-implemented the authors' Two-index, Three-index, and ARF formulations. This was motivated by the expectation that improvements in LP solver technology since the original publication could lead to better performance, even without major structural changes. The following provides a brief overview of the implemented models used to generate the reported results.

Preprocessing: For consistency, all models underwent the same pre-processing steps described in Section 4.2 prior to execution.

Three-index Model: Our Three-index formulation closely followed the authors' base implementation. However, neither lazy constraints nor valid inequalities were applied.

Two-index Model: The Two-index model was implemented similarly to the authors' version. Lazy constraints were included, however only the LSEC and valid path inequalities were enforced.

ARF Model: The ARF model utilised the authors' optimal cluster assignments as an additional preprocessing step (included in the total time recorded) and did not employ valid inequalities. Due to the advancements in LP solver performance since the publishing of the original paper, our ARF implementation was able to successfully solve a greater number of instances than reported in the original study.

Solver Configuration: To ensure consistency across all experiments, the Gurobi time limit parameter (`TimeLimit`) was set to 1800 seconds (30 minutes). For the Column Generation model, no Gurobi time limit parameter was used; instead, the entire process was externally terminated after 30 minutes due to the iterative nature of the Column Generation procedure. While the authors of the original study used a one-hour cutoff, we adopted a shorter limit due to practical time constraints. All experiments were executed on a laptop equipped with an Intel(R) Core(TM) Ultra 5 125H (1.20 GHz) processor and 16 GB of RAM, running a 64-bit operating system. This cpu had 14 available physical cores that where used for the multi-threading implementation.

The Gurobi `Work` metric has been included in the recorded results. This measurement aims to provide an indication of the computational complexity of the model being solved while remaining relatively independent of hardware-specific conditions. Since the Column Generation approach produces columns by solving multiple ILPs, the total `Work` reported corresponds to the sum of the work recorded for each subproblem model and the master problem.

5.2 Table 1

Table 1 shows the average work units, Gap Final %, and time to computation across instances types, for each formulation. For time to computation, the specific instances that were unable to be solved, or reached the time limit if 1800 seconds, were excluded to calculate more accurate averages for each instance type.

Similar to the results of Aziez et al. (2020), both the Two-index and ARF formulations showed slight improvements to the total number of instances solved in comparison to the Three-index formulation, with ARF formulation achieving a total number of instances solved of 62. Compared to all these formulations, Column Generation showed massive amounts of improvement, solving a total of 114 instances. Additionally, for the instances solved by all three models, the average computation for Column Generation was significantly decreased, such as for the l_4_25 instances, where for the Three-index, two index and ARF formulations, the average time for computation was 481.33 seconds, 41.14 seconds, and 8.918 seconds respectively, where as Column Generation had an average computation time of 1.442 seconds.

Table 1: Comparison of Formulations

	3-index				2-index				ARF				Col Gen			
Instance Type	Solved	Gap Final (%)	Time (s)	Work Units	Solved	Gap Final (%)	Time (s)	Work Units	Solved	Gap Final (%)	Time (s)	Work Units	Solved	Gap Final (%)	Time (s)	Work Units
l_4_25	1	17.28	481.33	222.80	2	10.67	41.14	25.23	5	0.00	8.92	5.27	5	0.00	1.44	0.54
l_4_35	0	35.39	—	—	0	24.57	—	—	4	1.22	195.62	86.08	5	0.00	3.44	1.26
l_4_50	0	47.39	—	—	0	38.63	—	—	0	18.82	—	—	5	0.00	17.29	7.86
l_4_100	0	59.74	—	—	0	52.81	—	—	0	52.50	—	—	4	0.00	610.15	330.08
l_8_25	4	2.36	573.35	161.03	5	0.00	7.89	7.71	5	0.00	1.36	0.19	5	0.00	0.56	0.24
l_8_35	1	10.22	66.00	22.97	4	2.95	460.25	296.31	5	0.00	3.36	0.58	5	0.00	0.86	0.39
l_8_50	0	28.12	—	—	1	16.06	1800.00	1365.17	4	2.44	26.86	7.96	5	0.00	2.94	1.51
l_8_100	0	42.66	—	—	0	35.78	—	—	0	17.53	—	—	5	0.00	19.09	10.04
n_4_25	5	0.00	300.68	121.83	5	0.00	7.56	7.06	5	0.00	1.94	0.10	5	0.00	0.18	0.02
n_4_35	1	8.17	74.64	58.29	5	0.00	217.92	137.10	5	0.00	7.12	0.25	5	0.00	0.34	0.05
n_4_50	0	23.86	—	—	2	12.91	654.97	315.72	5	0.00	16.41	4.24	5	0.00	1.09	0.20
n_4_100	0	44.49	—	—	0	36.79	—	—	1	11.40	1205.22	1404.61	5	0.00	23.49	6.59
n_8_25	5	0.00	11.65	8.29	5	0.00	0.72	0.54	5	0.00	0.39	0.04	5	0.00	0.15	0.03
n_8_35	5	0.00	230.63	85.48	5	0.00	1.03	0.76	5	0.00	0.78	0.05	5	0.00	0.22	0.04
n_8_50	2	4.38	4.10	2.80	5	1.12	370.20	279.84	5	0.00	5.48	0.20	5	0.00	0.45	0.09
n_8_100	0	17.50	—	—	1	5.79	49.18	36.62	5	0.00	27.26	3.41	5	0.00	1.77	0.37
w_4_25	0	28.35	—	—	0	22.14	—	—	0	19.65	—	—	5	0.00	2.42	0.89
w_4_35	0	37.26	—	—	0	32.79	—	—	0	32.56	—	—	5	0.00	28.27	12.49
w_4_50	0	46.02	—	—	0	42.55	—	—	0	43.92	—	—	5	0.00	166.72	77.07
w_4_100	0	57.72	—	—	0	53.91	—	—	0	56.90	—	—	0	—	—	—
w_8_25	0	21.85	—	—	0	15.55	—	—	2	9.32	251.23	81.09	5	0.00	0.53	0.19
w_8_35	0	35.54	—	—	0	30.73	—	—	1	16.17	12.32	7.13	5	0.00	0.93	0.35
w_8_50	0	47.49	—	—	0	43.39	—	—	0	38.07	—	—	5	0.00	6.52	2.91
w_8_100	0	57.67	—	—	0	55.23	—	—	0	47.70	—	—	5	0.00	128.02	60.02
Total Solved	24				40				62				114			

Table 2: Column Generation vs Multi Thread Column Generation Computation Time

Instance Type	Col. Gen. Time (s)	Multi-Thread Time (s)	Time Change (%)	Avg. No. Col. Generated
l_4_100	610.15	261.05	-57.22	9735.5
l_4_50	20.14	10.3	-48.86	629.0
n_4_100	23.49	21.68	-7.68	2062.6
w_4_35	28.27	11.33	-59.91	1611.8
w_4_50	166.72	60.61	-63.65	7853.0
w_8_100	128.02	96.86	-24.34	3953.2
w_8_50	8.97	6.44	-28.26	445.0

5.3 Table 2

Table 2 describes the benefits of Multi-Threading in terms of the average % time change of computation in comparison to Column Generation, for each instance type. The comparison was limited to instances where over 400 columns are generated, as this is the threshold number of subsets with which the process of Multi-Threading begins, as previously stated. Additionally, the data in the table was limited to the instances which both Column Generation and Multi Thread Column Generation could solve, to ensure a fair comparison in terms of the time to computation.

As can be observed in the table, for all instance types, Multi-Threading decreased the computation time for all instance types, and in some cases, by a significant amount, such as by -63.35% for the w_4_50 instances and by -59.91% for the w_4_35 instances. This improvement was mostly observed in the w instances, which is most likely due to their lack of time windows. A further benefit of Multi-Threading is that it was able to solve one more l_4_100 instance type and one more w_4_100 instance type compared to the sequential Column Generation.

Table 3: Multi Thread Column Generation for Previously Unsolvable Instances

Instance Type	Mutli-Thread Time (s)	No. Columns Generated
l_4_100_2	843.25	21802
w_4_100_3	1554.70	66618

5.4 Table 3

Table 3 describes the two instances, l_4_100_2 and w_4_100_3, that standard Column Generation was unable to solve, likely due to the high number of columns generated (21802 and 66618 respectively), whereas they were successfully solved using Multi-Thread Column Generation.

While these instances require a substantial amount of computation time, both Aziez et al. (2020) and standard Column Generation failed to solve these instances, so Multi-Thread Column Generation reaching a solution is notable.

Table 4: Column Generation Improvement on Author's and Our Best Upper Bound

Instance Type	Improvement on Author's Best Upper Bound	Improvement on Our Best Upper Bound
l_4_25	0.00	0.00
l_4_35	0.00	0.00
l_4_50	0.00	0.00
l_4_100	-0.01	-0.01
l_8_25	0.00	0.00
l_8_35	0.00	0.00
l_8_50	-0.15	-0.15
l_8_100	-0.11	-0.11
n_4_25	0.00	0.00
n_4_35	0.00	0.00
n_4_50	0.00	0.00
n_4_100	-0.02	-0.02
n_8_25	0.00	0.00
n_8_35	0.00	0.00
n_8_50	0.00	0.00
n_8_100	0.00	0.00
w_4_25	0.00	0.00
w_4_35	-2.53	-1.63
w_4_50	-0.11	-0.11
w_4_100	0.00	0.00
w_8_25	0.00	0.00
w_8_35	-1.14	-0.58
w_8_50	-0.13	-0.13
w_8_100	-0.27	-0.27

5.5 Table 4

Table 4 shows the average improvement of the best upper bound that the Column Generation across the Three-index, Two-index and ARF formulations, averaged across every instance type.

As can be observed in the table, for 9 out of the 24 instance types, Column Generation had a slight improvement on the best upper bound found by both ours and the authors Three-index, Two-index and ARF formulations, meaning in some cases, Column Generation was able to achieve a more optimal solution. As the w_4_35 instance types are complex enough such that the previous formulations struggle, but not complex enough that a large amount of columns (≥ 200) are generated, they had the most noticeable upper bound improvement of -1.63%.

Table 5: Capacity Table

Instance Type	Solved	Avg. Iterations	Avg Extra Work	Avg. Extra Time (s)
l_4_100	3	2.33	2.36	2.36
l_8_50	2	1.00	0.06	0.11
l_8_100	5	1.60	0.17	0.30
n_8_35	1	1.00	0.00	0.02
n_8_50	1	1.00	0.01	0.03
n_8_100	1	1.00	0.01	0.03
w_4_35	1	2.00	0.94	1.09
w_4_50	1	1.00	0.74	0.75
w_8_25	1	1.00	0.03	0.06
w_8_35	5	1.40	0.37	0.41
w_8_50	5	4.00	1.29	1.53
w_8_100	5	36.60	429.85	469.02

5.6 Table 5

Table 5 illustrates the impact of incorporating the dynamic column correction process used to implement vehicle capacity constraints within the multi-threaded Column Generation framework.

The table is conditioned on only those instances where the objective value was affected by the vehicle capacity constraint (i.e., instances that required at least one re-run), with results averaged across each instance type. As shown, the largest impact occurred for the w_8_100 instance type, which required an additional 469.02 seconds on average and 429.85 additional work units compared to runs without the capacity constraint. In contrast, the n_8 instances were minimally affected, requiring only 0.0 to 0.1 extra work units and an average of 0.02 to 0.03 additional seconds to solve.

6 Discussion & Critical Analysis

6.1 Interpretation and Comparison

6.1.1 Column Generation Improvements on Past Results

Based on the results presented, it is clear that Column Generation proved superior to the Three-index, Two-index and ARF models, not only in the computation time required for instances solvable by all models (see Table 1) but also by having more accurate solutions (see Table 4).

Another improvement made by Column Generation is that it was able to solve an additional 52 instances that were previously unsolvable by the ARF formulation, which was the best performing formulation presented by Aziez et al. (2020). Majority of the additional instances that were able to be solved by Column Generation, but not by previous models, were the instance types without time windows (w_instances). These instances were made solvable since they ran without any time propagation constraint and instance harnessed Gurobis `Cutoff` parameter to encode this as discussed in 4.6.1.

6.1.2 Performance with the Capacity Constraint

In most cases, the uncapacitated and capacitated objectives were identical. As shown in Table 5, only 31 out of 116 instances required re-optimisation with the capacity constraint enabled. Furthermore, in the majority of these cases, the column correction algorithm converged within just a few iterations, adding only a few seconds to the total runtime. This suggests that the vehicle capacity limit acted as a relatively weak constraint within the model.

It was observed, however, that the w_8_100 instances were considerably more challenging to solve once capacity constraints were introduced. This was likely because these instances allowed requests of up to length 8, meaning combinations of such requests had a higher likelihood of violating the maximum capacity before any load was removed at a delivery node. In these cases, the wide time windows meant that this part of the constraint did not dominate the capacity constraint, as it did in many other instance sets.

6.1.3 Improvements from Multi-Threading

Further improvements in both the number of instances solved and the computation time for solvable instances were achieved through the use of multi-threading. This approach distributes the solving of sub-problems across multiple CPU cores, allowing several to be processed simultaneously. Noticeable performance gains were observed once the number of generated columns reached approximately 400 or more, at which point multi-threading produced a significant reduction in total computation time (see Table 2). The observed decrease in runtime is attributed to the hardware's ability to process up to 14 subproblem solves in parallel.

With multi-threading enabled, two additional instances (l_4_100_2 and w_4_100_3) were successfully solved (see Table 3). These instances were previously unsolvable due to the large number of columns generated for each, which made it infeasible for the Column Generation process to converge within a reasonable time. However, by running multiple sub-problems in parallel, multi-threading effectively distributed the computational workload, allowing these instances to reach a feasible solution.

6.1.4 Unsolvable Instances

Even with the addition of multi-threading to the Column Generation process, four out of five of the w_4_100 instances remained unsolved. These instances are among the most challenging due to their large size and structural complexity. Each of the four unsolved instances contained 34–35 requests, and, in the absence of time windows, the pruning mechanisms were unable to effectively reduce the number of columns that needed to be generated. As a result, the number of columns grew rapidly, and the process failed to complete within the allotted time limit.

6.1.5 Comparison with Previous Studies

Existing exact formulations for the MPDPTW, including the Two-index, Three-index, ARF, and ATIF models, share a common computational bottleneck arising from the use of continuous time-propagation variables. These variables, defined for each node, substantially increase model size and solution time as the number of requests grows. In contrast, the proposed Column Generation framework limits or removes these variables entirely. In instances without explicit time windows, the sub-problem formulation eliminates them altogether, contributing to a significant reduction in computational effort.

In terms of performance, the formulations reported by Kohar et al. (2023) successfully solved 47, 49, 60, and 66 benchmark instances for the Two-index, Three-index, ARF, and ATIF models, respectively. Their accompanying heuristic approach achieved 67 solved instances. When disaggregated by instance family, the strongest previously reported results were 33 of 40 N-instances, 20 of 40 L-instances, and 14 of 40 W-instances solved to optimality. The proposed Column Generation approach solved all N- and L-instances and 36 of 40 W-instances, representing a substantial improvement across all instance types.

6.2 Methodological Reflection

6.2.1 Sensitivity and Stability of Results

As the size of the model increased, we noticed that the results became less consistent when the model was repeatedly ran. The generation of additional columns required solving a greater number of linear programs, which in turn increased the likelihood of the solver employing heuristics or cuts that adversely affected runtime performance. Furthermore, external factors such as CPU temperature and background system processes appeared to influence the precision and stability of the results.

Due to the project's deadlines, no systematic analysis of result variability could be conducted. Consequently, the extent to which these factors impacted the results remains uncertain.

6.2.2 Alternative Modeling Methods

Based on our results, it is clear that the Column Generation approach is vastly superior to the best known approaches in literature. This success could indicate that further improvements could be gained with a dynamic Column Generation method like branch-price.

6.2.3 Extensions and Open Questions

As discussed, the capacity component of the data instances was relatively weak. This led to the decision not to include it initially in the Column Generation model. If the maximum vehicle capacity were reduced, the constraint would likely become more active, potentially allowing for stricter pruning during Column Generation. Moreover, incorporating this constraint into the original uncapacitated models (Two-index,

Three-index, and ARF formulations) could also be beneficial if it results in tighter relaxations.

While it is understandable that no larger benchmark instances currently exist, as no known approach has come close to solving all of the existing ones, the success of our Column Generation method suggests that further investigation into its performance on larger instances would be valuable. Although the W-type instances appeared to approach the computational limits of this approach, the N and L instance sets still showed considerable room for scaling to larger problem sizes.

7 Conclusion

In this paper, we reproduced and reviewed the branch-and-cut algorithm presented by Aziez et al. (2020), which they used through three formulations: Two-index, Three-index and ARF. As per the findings of Aziez et al. (2020), ARF performed the best of these three formulations, both in terms of number of instances solved and computation time.

We also presented a Column Generation algorithm, which had two parts, a sub-roblem and a master problem. The sub problem acted as a route builder, that when given a subset of requests, finds feasible routes one vehicle can complete that serves all the pickup and deliveries for these requests, while respecting precedence and time constraints. Then, the master problem selects a combination of the feasible routes built by the sub-problem, minimising total travel time while ensuring that each request is covered exactly once. This algorithm presented significant improvements compared to the results found by Aziez et al. (2020) and other recent literature regarding the MPDPTW, with Column Generation solving more instances, finding better bounds on some previously solvable instances, and doing so in less time.

Further improvements were found with the addition of multi-threading, where two instances that were previously unable to be solved, even with Column Generation, could now be solved. Multi-threading found the most use for instances where over 400 columns were generated, and offered improvements to the computation speed of our algorithm for such instances.

Potential extensions to our work could include even larger benchmark instances for the N and L instances, or including tighter capacity constraints which would require stricter pruning during Column Generation.

References

[1] Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Mathematical Programming*, 90(3):475–506, 2001. ISSN 1436-4646. doi: 10.1007/PL00011432. URL https://doi.org/10.1007/PL00011432.

[2] Imadeddine Aziez, Jean-François Côté, and Leandro C. Coelho. Exact algorithms for the multi-pickup and delivery problem with time windows. *European Journal of Operational Research*, 284(3):906–919, 2020. ISSN 0377-2217. doi: 10.1016/j.ejor.2020.01.040. URL https://www.sciencedirect.com/science/article/pii/S0377221720300771.

[3] Salma Bouhlla, Rym Guibadj, and Aziz Moukrim. Efficient heuristic to solve the multi-pickup and delivery problem with time windows. *Lecture Notes in Networks and Systems*, 1311:175–187, 2025. doi: 10.1007/978-3-031-90606-0_15. URL https://doi.org/10.1007/978-3-031-90606-0_15.

[4] Raf Jans and Jacques Desrosiers. Efficient symmetry breaking formulations for the job grouping problem. *Computers & Operations Research*, 40(4):1132–1142, 2013. ISSN 0305-0548. doi: 10.1016/j.cor.2012.11.017. URL https://doi.org/10.1016/j.cor.2012.11.017.

[5] Amit Kohar and Suresh Kumar Jakhar. A capacitated multi pickup online food delivery problem with time windows: a branch-and-cut algorithm. *Annals of Operations Research*, 2021. doi: 10.1007/s10479-021-04145-6. URL https://doi.org/10.1007/s10479-021-04145-6.

[6] Amit Kohar, Suresh Kumar Jakhar, and Yogesh K. Agarwal. Strong cutting planes for the capacitated multi-pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 176:102806, 2023. ISSN 0191-2615. doi: https://doi.org/10.1016/j.trb.2023.102806. URL https://www.sciencedirect.com/science/article/pii/S0191261523001315.

[7] Salma Naccache, Jean-François Côté, and Leandro C. Coelho. The multi-pickup and delivery problem with time windows. *European Journal of Operational Research*, 269(2):353–362, 2018. ISSN 0377-2217. doi: 10.1016/j.ejor.2018.01.035. URL https://doi.org/10.1016/j.ejor.2018.01.035.

7.1 The Data

Table 6: Instance types.

Instance Size	Without TW		Normal TW		Large TW	
	Short	Long	Short	Long	Short	Long
25	25_w_4	25_w_8	25_n_4	25_n_8	25_l_4	25_l_8
35	35_w_4	35_w_8	35_n_4	35_n_8	35_l_4	35_l_8
50	50_w_4	50_w_8	50_n_4	50_n_8	50_l_4	50_l_8
100	100_w_4	100_w_8	100_n_4	100_n_8	100_l_4	100_l_8

7.2 General Formulation of Problem:

7.2.1 Sets:

$P = \{1, \cdots, p\}$ set of pickup nodes

$D = \{p+1, \cdots, p+n\}$ set of delivery nodes

$V = P \cup D \cup \{0, p+n+1\}$ set of vertices

$N = P \cup D$ set of all customer nodes (pickups and deliveries)

$R = \{r_1, \cdots, r_n\}$ set of requests to be routed

$C_j = \{R | j \leq i\}$ cluster of requests for an requests with index i and cluster index j

$P_r \subseteq P$ set of pickup nodes for request $r \in R$

$K = \{1, \cdots, m\}$ set of m uncapacitated vehicles

I set of arcs that lead to infeasible solutions

$A = V \times V - I$ set of arcs with feasible solutions

$A^+(i) = \{j \in V | (i,j) \in A\}$ set of nodes j such that there is an arc from i to j

$A^-(i) = \{j \in V | (j,i) \in A\}$ set of nodes j such that there is an arc from j to i

$\pi(S) = \{i \in \Pr \mid d_r \in S\}$ set of predecessors of S

$\sigma(S) = \{d_r \in D \mid i \in \Pr \cap S\}$ set of successors of S

7.2.2 Parameters:

$i = 0$ starting depot node

$i = p + n + 1$ final depot node

$[a_i, b_i]$ time window to begin service at node $i \in N$

s_i service time at node $i \in N$

t_{ij}/c_{ij} travel time/cost for arc $(i,j) \in A$

d_r delivery node for request $r \in R$

$r(i)$ request associated with node $i \in N$

M a big number, can be set to return time to depot (b_{p+n+1})

7.2.3 σ (successor) inequality:

$$x(S) + \sum_{i \in \bar{S} \cap \sigma(S)} \sum_{j \in S} x_{ij} + \sum_{i \in \bar{S} \setminus \sigma(S)} \sum_{j \in S \cap \sigma(S)} x_{ij} \leq |S| - 1 \tag{19}$$

7.2.4 π (predecessor) inequality:

$$x(S) + \sum_{i \in S} \sum_{j \in \bar{S} \cap \pi(S)} x_{ij} + \sum_{i \in S \cap \pi(S)} \sum_{j \in \bar{S} \setminus \pi(S)} x_{ij} \leq |S| - 1 \tag{20}$$

7.2.5 Infeasible drop/pickup paths inequalities:

$$x_{(0,S)} + x_{(S \cup \{d_r\})} \leq |S| \tag{21}$$
$$x_{(S \cup \{p\})} + x_{(S, p+n+1)} \leq |S| \tag{22}$$
$$x_{(d_r,S)} + x_{(S)} + x_{(S,p)} \leq |S| \tag{23}$$

7.3 Three-index Formulation:

7.3.1 Three-index Decision Variables:

$x_{ij}^k \in \{0,1\}$	1 if arc $(i,j) \in A$ is traversed by vehicle $k \in K$; 0 otherwise
$y_{rk} \in \{0,1\}$	1 if request $r \in R$ is satisfied by vehicle $k \in K$; 0 otherwise
$S_i \in \mathbb{R}^+$	Start time of service at node $i \in V$

7.3.2 Three-index Objective Function:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} \cdot x_{ij}^k \tag{24}$$

7.3.3 Three-index Constraints:

$$\sum_{j \in A^+(i)} x_{ij}^k = y_{r(i)}^k \qquad k \in K,\ i \in N \tag{25}$$
$$\sum_{j \in A^-(i)} x_{ji}^k = y_{r(i)}^k \qquad k \in K,\ i \in N \tag{26}$$
$$\sum_{j \in A^+(0)} x_{0j}^k \leq 1 \qquad k \in K \tag{27}$$
$$\sum_{k \in K} y_{rk} = 1 \qquad r \in R \tag{28}$$
$$S_j \ \geq \ S_i + (s_i + t_{ij} + M) \sum_{k \in K} x_{ij}^k - M \qquad (i,j) \in A \tag{29}$$
$$a_i \leq S_i \leq b_i \qquad i \in V \tag{30}$$
$$S_{d_r} \ \geq \ S_i + s_i + t_{id_r} \qquad i \in P_r,\ r \in R \tag{31}$$
$$x_{ij}^k,\ y_{rk} \in \{0,1\} \qquad (i,j) \in A,\ r \in R,\ k \in K \tag{32}$$
$$S_i \geq 0 \qquad i \in V \tag{33}$$

7.4 Two-index Formulation:

7.4.1 Two-index Decision Variables:

$x_{ij} \in \{0,1\}$ 1 if arc $(i,j) \in A$ is traversed by a vehicle ; 0 otherwise

$S_i \geq 0$ Start time of service at node $i \in V$

7.4.2 Two-index Objective Function:

$$\min \sum_{(i,j)\in A} c_{ij} \cdot x_{ij} \tag{34}$$

7.4.3 Two-index Constraints:

$$\sum_{i\in V} x_{ij} = 1 \qquad j \in N \tag{35}$$

$$\sum_{j\in V} x_{ij} = 1 \qquad i \in N \tag{36}$$

$$\sum_{j\in N} x_{0j} \leq |K| \tag{37}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1 \qquad S \subseteq N \tag{38}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 2 \qquad S \in \mathcal{S} \tag{39}$$

$$S_j \ \geq \ S_i + s_i + t_{ij} - M(1 - x_{ij}) \qquad (i,j) \in A \tag{40}$$

$$a_i \leq S_i \leq b_i \qquad i \in V \tag{41}$$

$$S_{d_r} \ \geq \ S_i + s_i + t_{id_r} \qquad i \in P_r,\ r \in R \tag{42}$$

$$x_{ij} \in \{0,1\} \qquad (i,j) \in A \tag{43}$$

$$S_i \geq 0 \qquad i \in V \tag{44}$$

7.5 ARF Formulation:

7.5.1 ARF Decision Variables:

$x_{ij}^k \in \{0,1\}$ 1 if arc $(i,j) \in A$ is present in cluster $k \in C$; 0 otherwise

$y_{rk} \in \{0,1\}$ 1 if request $r \in R$ is present in cluster $k \in C$; 0 otherwise

$S_i \geq 0$ Start time of service at node $i \in V$

7.5.2 ARF Objective Function:

$$\min \sum_{k \in C} \sum_{(i,j) \in A} c_{ij} \cdot x_{ij}^k \tag{45}$$

7.5.3 ARF Constraints:

$$\sum_{j \in A^+(i)} x_{ij}^k = y_{r(i)}^k \qquad k \in R,\ i \in N \tag{46}$$

$$\sum_{j \in A^-(i)} x_{ji}^k = y_{r(i)}^k \qquad k \in R,\ i \in N \tag{47}$$

$$\sum_{j \in A^+(0)} x_{0j}^k \leq 1 \qquad k \in R \tag{48}$$

$$\sum_{k \in R} y_{kk} \leq |K| \tag{49}$$

$$\sum_{k \in R} y_{rk} = 1 \qquad r \in R \tag{50}$$

$$\sum_{\substack{r \in R \\ r > k}} y_{rk} \leq y_{kk}(|R| - k) \qquad k \in R \tag{51}$$

$$y_{rk} + y_{r'k} \leq 1 \qquad k, r, r' \in R,\ (r, r') \in W \tag{52}$$

$$S_j \ \geq \ S_i + (s_i + t_{ij} + M) \sum_{k \in K} x_{ij}^k - M \qquad (i,j) \in A \tag{53}$$

$$a_i \leq S_i \leq b_i \qquad i \in V \tag{54}$$

$$S_{d_r} \ \geq \ S_i + s_i + t_{id_r} \qquad i \in P_r,\ r \in R \tag{55}$$

$$x_{ij}^k,\ y_{rk} \in \{0,1\} \qquad (i,j) \in A,\ r,k \in R \tag{56}$$

$$S_i \geq 0 \qquad i \in V \tag{57}$$

7.5.4 Cluster Assignment Model

7.5.5 Decision Variables:

$y_{rk} \in \{0,1\}$ 1 if request $r \in R$ is assigned to cluster $k \in R$; 0 otherwise

$y_{kk} \in \{0,1\}$ 1 if cluster k is active (its representative is present); 0 otherwise

$z_{rr'}^k \in \{0,1\}$ 1 if $r < r'$, $r, r' \in R$ are both assigned to cluster k; 0 otherwise

7.5.6 Objective Function:

$$\min \sum_{k \in R} \sum_{\substack{r,r' \in R \\ r < r'}} e_{rr'}\, z_{rr'}^k \ + \sum_{k \in R} \sum_{r \in R} e_r\, y_{rk} \tag{58}$$

7.5.7 Constraints:

$$\sum_{k \in R} y_{rk} = 1 \qquad r \in R \tag{59}$$

$$y_{rk} \leq y_{kk} \qquad r \in R,\ k \in R \tag{60}$$

$$y_{rk} = 0 \qquad r \leq k,\ r \neq k,\ k \in R \tag{61}$$

$$y_{rk} + y_{r'k} \leq 1 \qquad (r, r') \in W,\ k \in R \tag{62}$$

$$z^k_{rr'} \leq y_{rk}, \quad z^k_{rr'} \leq y_{r'k}, \quad z^k_{rr'} \geq y_{rk} + y_{r'k} - 1 \qquad r < r',\ r, r', k \in R \tag{63}$$

$$y_{rk},\ y_{kk},\ z^k_{rr'} \in \{0, 1\} \qquad r, r', k \in R \tag{64}$$