Solving the Capacitated Vehicle Routing Problem with Graph Transformers

Evgeny Polyachenko[1][a], Daniel Antunes Pedrozo[1][b], Jorge Augusto Meira[1][c], and Radu State[1][d]
[1] *2Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg*
{evgeny.polyachenko, jorge.meria}@uni.lu

Keywords: Vehicle Routing Problem, Graph Attention Network, Reinforcement Learning

Abstract: This paper presents a novel approach to solving the Capacitated Vehicle Routing Problem (CVRP) using Graph Transformers. We leverage the power of deep reinforcement learning to train an agent that can dynamically construct high-quality routes for a fleet of vehicles. Our model is evaluated on a set of well-known CVRP benchmarks, and we show that it outperforms existing methods in terms of both solution quality and computational time.

1 INTRODUCTION

2 Model settings

The Capacitated Vehicle Routing Problem (CVRP) studied in our benchmark consists of finding minimum-cost routes for a fleet of homogeneous vehicles to service a set of customers with known demands. Each vehicle has a fixed capacity, starts and ends its route at a central depot, and each customer must be visited exactly once by exactly one vehicle.

2.1 Instance Generation

CVRP instances are generated with the following specifications:

- **Number of customers**: $n \in \{7, 8, 9, 10, \dots, 20\}$ (varies by experiment)
- **Depot location**: Node 0 with coordinates sampled uniformly
- **Customer locations**: Nodes $1, 2, \dots, n$ with coordinates sampled uniformly
- **Coordinate generation**:
  1. Sample integer coordinates $(x_i, y_i) \sim \text{Uniform}\{0, 1, \dots, 100\}$ for all nodes $i \in \{0, 1, \dots, n\}$
  2. Normalize to unit square: $(x'_i, y'_i) = (x_i/100, y_i/100)$
- **Customer demands**: $q_i \sim \text{Uniform}\{1, 2, \dots, 10\}$ for $i \in \{1, \dots, n\}$
- **Depot demand**: $q_0 = 0$
- **Vehicle capacity**: $Q = 30$ (fixed for all instances)
- **Distance metric**: Euclidean distance in the normalized unit square
- **Random seed**: seed $= 1000 * n + i$ (where i is instance id-1)

2.2 Mathematical Formulation

Let $\mathcal{V} = \{0, 1, \dots, n\}$ denote the set of all nodes, where node 0 represents the depot and $\mathcal{N} = \{1, 2, \dots, n\}$ represents the set of customers. Let $\mathcal{K}$ denote the set of available vehicles (unbounded in our formulation).

2.2.1 Parameters

- $(x_i, y_i) \in [0, 1]^2$: Normalized coordinates of node $i \in \mathcal{V}$
- $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$: Euclidean distance between nodes i and j
- $q_i \in \{1, 2, \dots, 10\}$: Demand of customer $i \in \mathcal{N}$ (with $q_0 = 0$)
- $Q = 30$: Vehicle capacity

[a] https://orcid.org/0000-0002-4596-1222
[b] https://orcid.org/0009-0005-0718-3908
[c] https://orcid.org/0000-0002-4086-5784
[d] https://orcid.org/0000-0002-4751-9577

2.2.2 Decision Variables

- $x_{ijk} \in \{0,1\}$: Binary variable equal to 1 if vehicle k travels directly from node i to node j
- $u_i \in [0,Q]$: Cumulative demand served when visiting node i (for subtour elimination)

2.2.3 Objective Function

Minimize the total travel distance:

$$\min \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{V}} \sum_{j\in\mathcal{V}} d_{ij} \cdot x_{ijk} \tag{1}$$

2.2.4 Constraints

1. Each customer is visited exactly once:

$$\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{V}, i\neq j} x_{ijk} = 1 \quad \forall j \in \mathcal{N} \tag{2}$$

2. Flow conservation (route continuity):

$$\sum_{i\in\mathcal{V}, i\neq h} x_{ihk} = \sum_{j\in\mathcal{V}, j\neq h} x_{hjk} \quad \forall h \in \mathcal{V}, \forall k \in \mathcal{K} \tag{3}$$

3. Each vehicle starts from the depot:

$$\sum_{j\in\mathcal{N}} x_{0jk} \leq 1 \quad \forall k \in \mathcal{K} \tag{4}$$

4. Each vehicle returns to the depot:

$$\sum_{i\in\mathcal{N}} x_{i0k} \leq 1 \quad \forall k \in \mathcal{K} \tag{5}$$

5. Vehicle capacity constraint:

$$\sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{V}, j\neq i} q_i \cdot x_{ijk} \leq Q \quad \forall k \in \mathcal{K} \tag{6}$$

6. Subtour elimination (Miller-Tucker-Zemlin):

$$u_i - u_j + Q \cdot x_{ijk} \leq Q - q_j \quad \forall i,j \in \mathcal{N}, i \neq j, \forall k \in \mathcal{K} \tag{7}$$

$$q_i \leq u_i \leq Q \quad \forall i \in \mathcal{N} \tag{8}$$

2.3 Solution Representation

A CVRP solution is represented as a set of vehicle routes, where each route is a sequence of customer nodes. The solution format used in the implementation is:

- **Individual routes**: Each vehicle route is stored as a list $[c_1, c_2, \ldots, c_m]$ containing only customer nodes
- **Complete solution**: A list of vehicle routes $\mathcal{R} = \{r_1, r_2, \ldots, r_{|\mathcal{K}|}\}$
- **Flattened representation**: For output, routes are combined with explicit depot nodes: $[0, c_{11}, \ldots, c_{1m_1}, 0, c_{21}, \ldots, c_{2m_2}, 0, \ldots]$

2.4 Solution Validation Procedure

The following algorithm validates a CVRP solution and computes its objective value:

```
Require: Solution $\mathcal{R} = \{r_1, r_2, \ldots, r_K\}$, instance
    data $(d_{ij}, q_i, Q)$
Ensure: Feasibility status and total cost
 1: visited ← $\emptyset$ {Track visited customers}
 2: total_cost ← 0
 3: feasible ← true
 4: for each route $r_k \in \mathcal{R}$ do
 5:     route_demand ← 0
 6:     route_cost ← $d_{0,r_k[1]}$ {Depot to first customer}
 7:     for $i = 1$ to $|r_k|$ do
 8:         if $r_k[i] \in$ visited then
 9:             feasible ← false {Customer visited
                twice}
10:             return (false, $\infty$)
11:         end if
12:         visited ← visited $\cup \{r_k[i]\}$
13:         route_demand ← route_demand + $q_{r_k[i]}$
14:         if $i < |r_k|$ then
15:             route_cost ← route_cost + $d_{r_k[i], r_k[i+1]}$
16:         else
17:             route_cost ← route_cost + $d_{r_k[i],0}$ {Last
                customer to depot}
18:         end if
19:     end for
20:     if route_demand > $Q$ then
21:         feasible ← false {Capacity violated}
22:         return (false, $\infty$)
23:     end if
24:     total_cost ← total_cost + route_cost
25: end for
26: if |visited| $\neq n$ then
27:     feasible ← false {Not all customers visited}
28:     return (false, $\infty$)
29: end if
30: return (true, total_cost)
```

Algorithm 1: CVRP Solution Validation

2.5 Distance Calculation

The exact Euclidean distance calculation used in the implementation:

2.6 Instance Generation Algorithm

The complete instance generation procedure as implemented:

```
Require: Coordinates $\{(x_i, y_i)\}_{i=0}^{n}$
Ensure: Distance matrix $D \in \mathbb{R}^{(n+1)\times(n+1)}$
 1: for $i = 0$ to $n$ do
 2:     for $j = 0$ to $n$ do
 3:         $D[i][j] \leftarrow \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
 4:     end for
 5: end for
 6: return $D$
```
Algorithm 2: Distance Matrix Computation

```
Require: Number of customers $n$, instance ID $id$,
    attempt number $a$
Ensure: CVRP instance
    $(coords, demands, distances, capacity)$
 1: seed $\leftarrow 1000 * n + i$
 2: Set random seed to seed
 3: // Generate coordinates
 4: for $i = 0$ to $n$ do
 5:     $x_i \leftarrow \text{RandomInt}(0, 100)$
 6:     $y_i \leftarrow \text{RandomInt}(0, 100)$
 7:     coords$[i] \leftarrow (x_i/100, y_i/100)$
 8: end for
 9: // Generate demands
10: demands$[0] \leftarrow 0$ {Depot has no demand}
11: for $i = 1$ to $n$ do
12:     demands$[i] \leftarrow \text{RandomInt}(1, 10)$
13: end for
14: // Compute distance matrix
15: distances $\leftarrow$ ComputeDistanceMatrix(coords)
16: // Set capacity
17: capacity $\leftarrow 30$
18: return (coords, demands, distances, capacity)
```
Algorithm 3: CVRP Instance Generation

2.7 Key Implementation Notes

1. **Coordinate Precision**: Coordinates are first sampled as integers from $\{0, 1, \ldots, 100\}$ then normalized to $[0, 1]$ by dividing by 100, ensuring consistent precision across instances.
2. **Distance Computation**: Distances are computed using double-precision floating-point arithmetic to minimize rounding errors.
3. **Vehicle Fleet**: The number of vehicles is unbounded (sufficient vehicles are always available), making this effectively a fleet-size-and-mix VRP with homogeneous unlimited vehicles.
4. **Solution Quality**: Solutions are evaluated solely by total distance traveled; the number of vehicles used is not part of the objective function.
5. **Reproducibility**: The deterministic seeding scheme ensures that instances can be exactly reproduced given the problem size, instance ID, and attempt number.

2.8 Key differences vs. Daniel's thesis that can worsen the optimal solution.

- **Forced use of every vehicle (Daniel's thesis) vs. optional use (this paper).** Daniel's thesis imposes per-vehicle equalities at the depot (each vehicle must depart and return exactly once), which *forces all $|M|$ vehicles to be used.* If $|M|$ exceeds what is actually needed to minimize distance, this restriction can strictly *increase* the optimal travel cost by splitting demand across unnecessary routes. In this paper, the depot constraints are inequalities (≤ 1), allowing unused vehicles and thus never forcing extra tours.
- **Fixed fleet size (Daniel's thesis) vs. unbounded fleet here.** Combined with the previous point, a fixed fleet that *must* all be used can degrade the optimum when the fleet is larger than necessary for the instance, again increasing the total traveled distance. (By contrast, the unbounded-fleet model in this paper cannot worsen the distance-only objective, because it never forces the use of extra vehicles.)
- **MTZ subtour elimination here vs. load-flow formulation in Daniel's thesis (practical optimality under time limits).** While both are exact if solved to proven optimality, the MTZ (big-M) formulation used here typically has a *weaker LP relaxation* than the single-commodity load-flow model in Daniel's thesis. Under realistic solver time limits or heuristic search, this weaker relaxation can lead to *worse incumbent (higher-cost) solutions* being returned before optimality is proven.

Why the MTZ (big-M) model here can return worse solutions under time limits (novice explanation).

- **How MILP solvers search.** Solvers (Branch-and-Bound) repeatedly solve a relaxed version of the model where binary decisions are allowed to be fractional (between 0 and 1). This is called the *LP relaxation*. Two things matter in practice:
  1. a *lower bound* from the relaxation (an optimistic cost that no true integer solution can beat), and
  2. the current best *incumbent* (a feasible integer solution).

Tighter relaxations give better (less optimistic) lower bounds and usually shrink the search effort, helping the solver find better incumbents within a fixed time.

- **What "weaker" means here.** In the MTZ model used in this paper, subtours are blocked with big-M inequalities involving the u_i variables (e.g., $u_i - u_j + Qx_{ijk} \leq Q - q_j$). When x_{ijk} is fractional in the relaxation (say 0.3 or 0.5), it is often easy to choose u values so that these inequalities still hold. Intuitively, the relaxation can "pretend" that a fraction of a vehicle partially visits many customers at once. This produces LP solutions that look artificially cheap and very different from any real set of routes, i.e., the lower bound is *too optimistic*. The solver then must branch a lot to repair these fractions, which consumes time and can delay finding high-quality incumbents.
- **Why Daniel's thesis (load-flow) is stronger.** In Daniel's thesis, every used arc must carry a consistent amount of *load* ℓ_{ijm}:
  - If an arc (i, j) is used (even fractionally in the relaxation), the load on it must be at least the demand of the next customer ($q_j x_{ijm} \leq \ell_{ijm}$) and at most the remaining capacity.
  - Load is *conserved*: total load flowing into a customer minus the load flowing out must equal that customer's demand.

  These flow and capacity couplings make it much harder for the relaxation to "cheat" with fractional vehicles visiting many customers simultaneously. As a result, the LP relaxation is *closer to the true integer problem* (a "stronger" relaxation), giving tighter lower bounds and guiding the search more effectively.
- **Practical outcome under time limits.** If you stop the solver early (as is common for larger CVRP instances), a weaker relaxation (MTZ) tends to:
  - explore larger search trees (more time spent fixing fractional artifacts),
  - provide looser bounds (less guidance),
  - and, consequently, return *worse incumbents* (higher cost routes) on average within the same time budget.

  A stronger relaxation (load-flow from Daniel's thesis) typically does the opposite: smaller trees, tighter bounds, and better incumbents sooner.
- **Important caveat.** If the solver runs to *proven optimality*, both formulations reach the same optimum. The difference matters most when time or memory limits prevent full optimality proofs.

2.9 GAT

2.10 GT

3 Optimal and suboptimal benchmarks

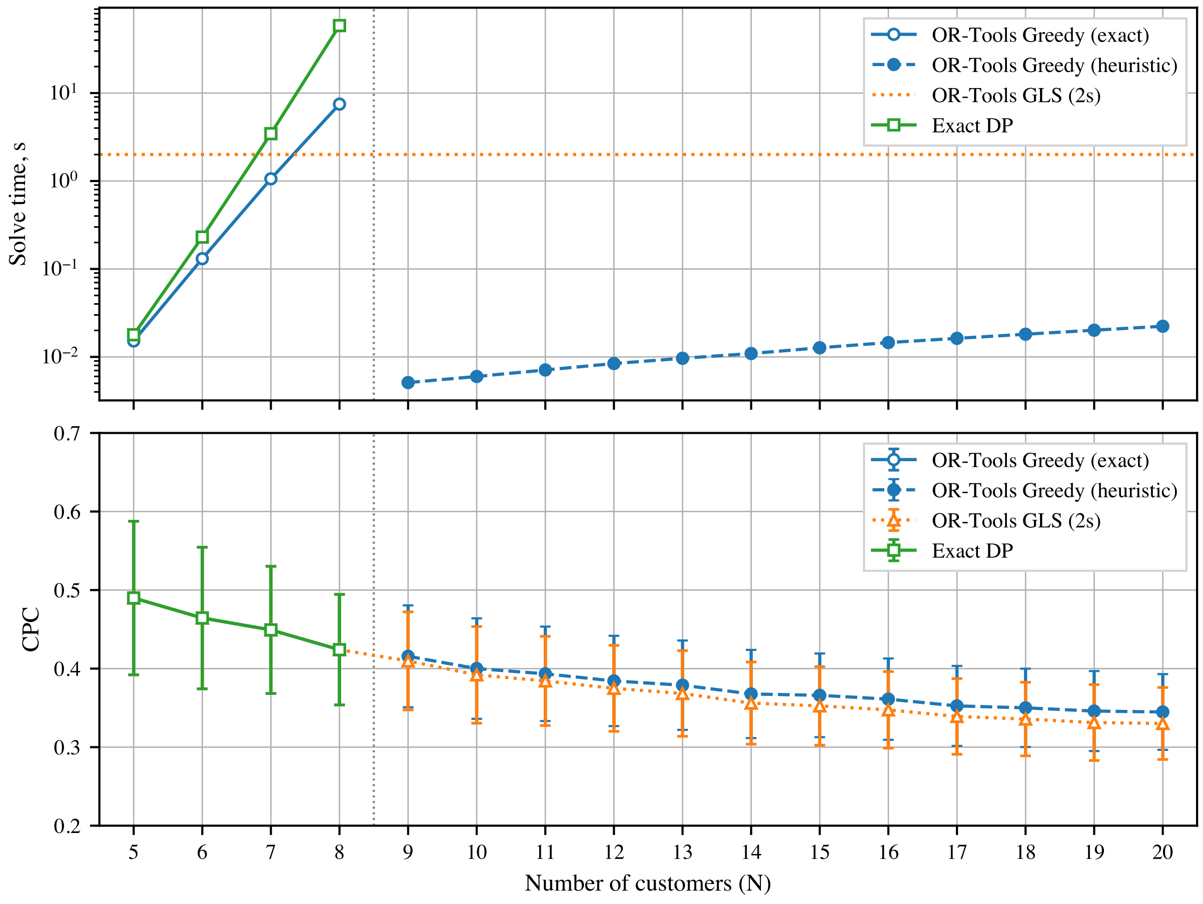

Figure 1: Performance comparison of CVRP solvers on CPU benchmarks. **Top panel:** Solve time (log scale) versus problem size for different solvers. Exact DP provides optimal solutions but is limited to small instances (N≤8). OR-Tools Greedy operates in exact mode for N≤8 (solid line) and switches to heuristic mode for larger instances (dashed line). OR-Tools GLS with 2-second time limit maintains consistent performance across all problem sizes (dotted line). **Bottom panel:** Cost per customer (CPC) showing solution quality with error bars indicating standard deviation across 1000 instances per problem size. The vertical line at N=8.5 marks the transition point between exact and heuristic approaches for OR-Tools Greedy.

Table 1: Cost per Customer (CPC) for various CVRP sizes

Method	**N**	**Cap.**	**Time**	**Mean**	**GM**	**Median**	**Error**	**95% Range**	**KS CPC**	**KS log(CPC)**
GPU-DP-Exact (sub)	8	30	–	0.4265	0.4206	0.4237	9.07e-05	[0.2966, 0.5696]	< 0.01	< 0.01
CPU-DP-Exact (opt)	8	30	58.54s	0.4241	0.4182	0.4213	3.01e-03	[0.2957, 0.5653]	0.46	0.38
OR-Tools Greedy (opt)	8	30	7.48s	0.4241	0.4182	0.4213	3.01e-03	[0.2957, 0.5653]	0.46	0.38
OR-Tools GLS (sub)	8	30	2.00s	0.4241	0.4182	0.4213	3.01e-03	[0.2957, 0.5653]	0.46	0.38
GPU-DP-Exact (sub)	10	30	–	0.3947	0.3900	0.3913	7.35e-05	[0.2854, 0.5272]	< 0.01	< 0.01
OR-Tools GLS (sub)	10	30	2.00s	0.3920	0.3873	0.3884	2.39e-03	[0.2858, 0.5259]	0.13	0.70
OR-Tools Greedy (sub)	10	30	5.99ms	0.4000	0.3950	0.3958	2.55e-03	[0.2879, 0.5362]	0.17	0.97
Paper I (SCIP opt)	10	30	6.7ms	0.4360	-	-	-	-	-	-
OR-Tools GLS (sub)	20	30	2.00s	0.3301	0.3270	0.3284	1.93e-03	[0.2469, 0.4261]	0.20	0.87
OR-Tools Greedy (sub)	20	30	0.022s	0.3448	0.3414	0.3439	1.88e-03	[0.2547, 0.4461]	0.68	0.07
Paper I (GR sub)	10	30	42ms	0.3550	-	-	-	-	-	-

- Statistics computed over $10^6/10^3$ instances for GPU-DP-Exact/Others per configuration
- CPU DP-Exact solver uses brute-force enumeration of all customer partitions and route permutations ($O(n! \times 2^n)$), ensuring truly optimal solutions but restricting feasibility to $N \leq 8$.
- GPU "DP-Exact" solver combines optimal Held-Karp dynamic programming for partitioning ($O(n^2 2^n)$) with nearest neighbor heuristic for route ordering, achieving near-optimal solutions with dramatically improved scalability through GPU parallelization.
- Time: Average time required to solve 1 instance
- Mean: Arithmetic mean; GM: Geometric mean
- Error: Maximum of SE(Mean), SE(GM), SE(Median) where:
  SE(Mean) = $s/\sqrt{n}$; SE(GM) = GM $\times \log(\text{GSD})/\sqrt{n}$; SE(Median) via KDE
- 95% Range: [2.5th percentile, 97.5th percentile] of CPC values
- KS: Kolmogorov-Smirnov test p-values for normality, CPC and log(CPC)
- (opt): optimal solver; (sub): sub-optimal solver
- Paper I: Pedrozo D.A. Integrating Machine Learning and Optimisation to ...

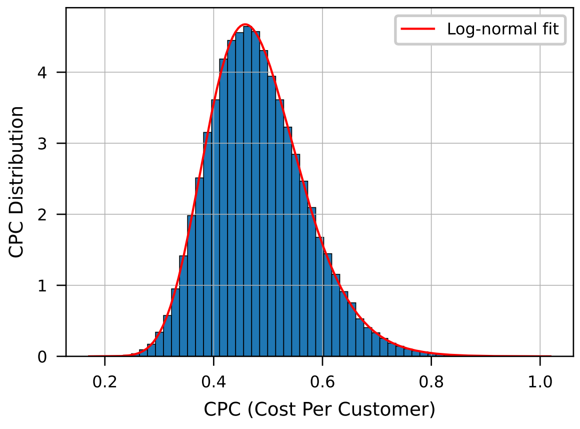

Figure 2: Distribution of CPC for N=10 CVRP instances solved using GPU dynamic programming. The histogram shows 100,000 random instances with the fitted log-normal distribution overlaid ($\mu = -0.748$, $\sigma = 0.183$). The log-normal fit demonstrates that CPC follows a log-normal distribution (Kolmogorov-Smirnov test, $p = 0.189$).

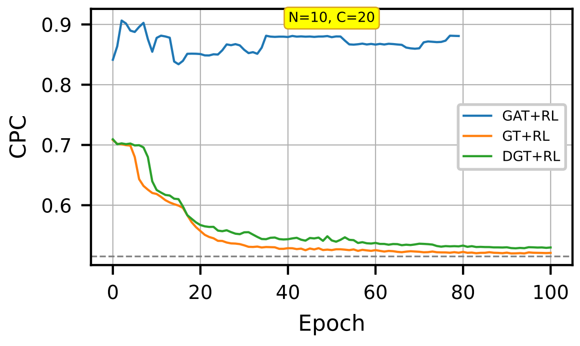

Figure 3: Training batch 75

Table 2: Timing Comparison: Exact DP vs OR-Tools Greedy (Optimal Mode)

N	Exact DP	OR-Tools	1K Exact	1K OR-Tools	10K Exact	10K OR-Tools
5	18ms	15ms	0.3m	0.2m	3.0m	2.5m
6	230ms	131ms	3.8m	2.2m	38.3m	21.8m
7	3.5s	1.1s	57.5m	17.7m	09.35h	02.56h
8	58.5s	7.5s	16.15h	02.04h	06.18d	20.46h
9	19.3m	48.2s	13.08d	13.22h	133.17d	05.13d
10	7.3h	4.6m	303.12d	03.05d	3035.09d	32.04d

Single instance timing per method (N=5-8: actual data; N=9-10: quadratic extrapolation)
1K/10K columns show total time for 1,000/10,000 instances
Time format: ms=milliseconds, s=seconds, m=minutes, h=hours (HH.MM), d=days (DD.HH)
OR-Tools run in optimal mode with same solution quality as Exact DP

Table 3: Optimal and sub-optimal (OR-Tools GLS) CPC values (10k instances per configuration)

Method	**N**	**Cap.**	**Timeout**	**GM**	**GSD**	**95% Range**	**95% CI**	**KS**
Exact DP (1000k).	8	30	–	0.4206	1.1833	[0.3024, 0.5850]	[0.4204, 0.4207]	<0.01
Exact DP (1000k)*	8	30	–	0.4237	nan	[0.2966, 0.5696]	[0.0003, 0.0006]	<0.01
Exact (100k)	10	20	–	0.4735	1.2012	[0.3306, 0.6782]	[0.4730, 0.4741]	0.19
Exact (10k)	10	20	–	0.4737	1.1996	[0.3316, 0.6767]	[0.4720, 0.4754]	0.93
Heuristic	10	20	2s	0.4744	1.2025	[0.3305, 0.6809]	[0.4727, 0.4761]	0.91
Heuristic	20	30	2s	0.3257	1.1473	[0.2488, 0.4264]	[0.3248, 0.3266]	0.15
Heuristic	20	30	10s	0.3234	1.1459	[0.2477, 0.4224]	[0.3226, 0.3243]	0.08
Heuristic	50	40	10s	0.2366	1.1233	[0.1884, 0.2972]	[0.2270, 0.2467]	0.89
Heuristic	50	40	20s	0.2391	1.1270	[0.1892, 0.3023]	[0.2323, 0.2462]	0.05
Heuristic	50	40	30s	0.2253	1.1294	[0.1775, 0.2860]	[0.2247, 0.2258]	< 0.01
Heuristic	50	40	60s	0.2237	1.1293	[0.1763, 0.2839]	[0.2232, 0.2243]	< 0.01
Heuristic	100	50	10s	0.1729	1.1061	[0.1419, 0.2108]	[0.1676, 0.1784]	< 0.01
Heuristic	100	50	20s	0.1673	1.1353	[0.1305, 0.2146]	[0.1612, 0.1737]	< 0.01
Heuristic	100	50	30s	0.1552	1.0338	[0.1454, 0.1657]	[0.1538, 0.1567]	< 0.01
Heuristic	100	50	60s	0.1754	1.1258	[0.1391, 0.2213]	[0.1750, 0.1758]	< 0.01
Heuristic.	100	50	120s	0.1737	1.1259	[0.1377, 0.2192]	[0.1733, 0.1741]	< 0.01
OR-Tools GLS	100	50	240s	0.1721	1.1265	[0.1362, 0.2173]	[0.1717, 0.1725]	<0.01

Exact:
Heuristic: OR-Tools GLS
GM: Geometric Mean, GSD: Geometric Standard Deviation
95% Range: GM $\times$ [GSD$^{-1.96}$, GSD$^{+1.96}$]
KS: Kolmogorov-Smirnov, D'Agost.: D'Agostino, JB: Jarque-Bera (p-values for log(CPC) normality)
AD: Anderson-Darling test statistic (critical value at 5% = 0.787; * indicates rejection)
Timeout column indicates per-instance time limit used by OR-Tools GLS
Test run with 20 instances per configuration