Efficient and Robust Neural Combinatorial Optimization via Wasserstein-based Coresets

Authors: Xu Wang, Fuyou Miao, Wenjie Liu, Yan Xiong

Affiliations: School of Computer Science and Technology, University of Science and Technology of China

Conference: ICLR 2025

2025.03.18

Outline

- Research Background & Motivation
- Challenges for NCO
- Overview of the Proposed Method
  - RWD & Coreset Construction
  - Accelerate Coreset Construction with Merge-and-Reduce
  - Efficient Framework for NCO methods
- Experimental Results
- Conclusion & Future Work

Research Background & Motivation

- Combinatorial Optimization (CO) is crucial in fields like transportation[1], logistics[2], and manufacturing[3].

[1] MacZ. Transportation - 575184[EB/OL]. [2025-02-21]. Available: https://sc.macz.com/pic/575184.html.
[2] Thailand Translation. *Logistics Image*[EB/OL]. (2017-10) [2025-02-21]. Available: https://thailandtranslation.net/wp-content/uploads/2017/10/logistic-1.jpg.
[3] Yingluo. Advances in Combinatorial Optimization Applications[EB/OL]. Plasway, 2025-02-21[2025-02-21]. Available at: https://yingluo.plasway.com/news/120277.html.

Research Background & Motivation

- Traditional exact and heuristic solvers have limitations in scalability and adaptability.
- Neural Combinatorial Optimization (NCO) methods offer a data-driven alternative but demand high computational resources.
- An important issue for NCO is the reduced robustness when the training and test data distributions differ.

Challenges for NCO

- **High storage and computational costs** due to large-scale training datasets.
- NCO models often struggle with robustness when facing **distribution shifts** between training and testing.
- The need for an efficient training strategy that maintains performance while using limited resources.

Overview of the Proposed Method

- RWD and Coreset Construction
- Accelerate Coreset Construction with Merge-and-Reduce
- Efficient Framework for NCO methods

RWD and Coreset Construction

- RWD: Definition and its invariance to rigid transformations.

Definition 1 (Wasserstein distance under rigid transformations, RWD)

Let $\mu = \sum_{i=1}^{n} a_i \delta_{x_i}, \nu = \sum_{j=1}^{n} b_j \delta_{y_j} \in \mathcal{P}(\mathbb{R}^d)$, where $\mathbf{a}, \mathbf{b} \in \mathbb{R}_+^n$ are their weight vectors and $\{x_i\}_{\{i\in[n]\}}$, $\{y_j\}_{\{j\in[n]\}} \subset \mathbb{R}^d$ are their locations. Then, the value of RWD between μ and ν is

$$\mathcal{W}(\mu,\nu) := \left(\min_{\mathbf{P}\in\mathbf{\Pi}(\mathbf{a},\mathbf{b}), e\in E(d)} \sum_{i=1}^{n}\sum_{j=1}^{n} P_{ij} \| x_i - e(y_j) \|^2 \right)^{1/2},$$

Where $\mathbf{\Pi}(\mathbf{a},\mathbf{b}) := \{\mathbf{P} \in \mathbb{R}_+^{n\times n} \mid \mathbf{P1} = \mathbf{a}, \mathbf{P}^T\mathbf{1} = \mathbf{b}\}$ is the coupling set, $E(d)$ is the Euclidean group on $\mathbb{R}^d$, and $e: \mathbb{R}^d \to \mathbb{R}^d$ is the rigid transformation.

Intuitions: The solutions to CO problems such as TSP remain invariant under rigid transformations such as translation, rotation, and reflection.

Alternative metrics: Wasserstein distance, Gromov-Wasserstein distance, ...

RWD and Coreset Construction

- What is coreset?
  - A small-size weighted proxy of the original dataset.
  - Preserves the value of an objective function evaluated on the full dataset.
  - Saves computational and storage resources.

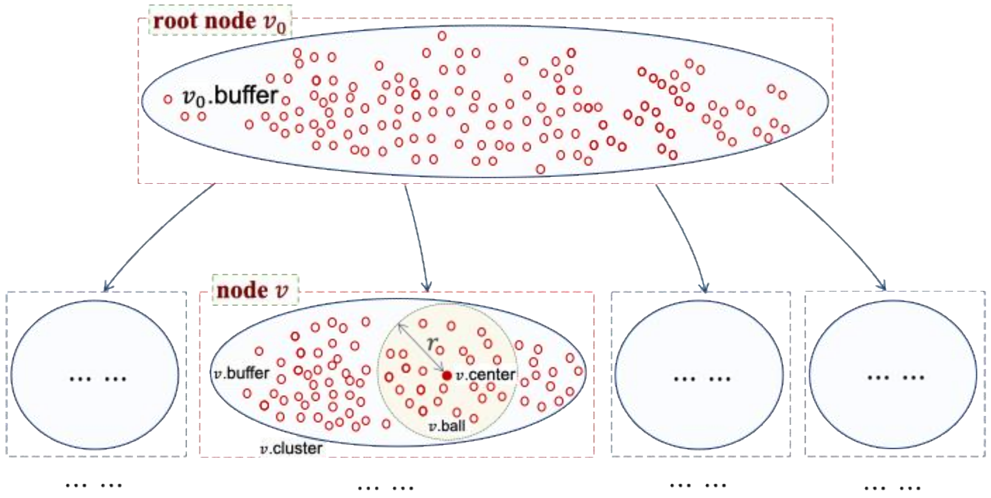
Definition 2 (Coreset)

Let $0 < \epsilon < 1$ and ℓ be a loss function. Let $\mathcal{Q} \subset \mathcal{P}(\mathbb{R}^d)$ be a set of measures with weight function $W_{\mathcal{Q}}: \mathcal{P}(\mathbb{R}^d) \to \mathbb{R}_+$. Let $\sum_{\mu \in \mathcal{Q}} W_{\mathcal{Q}}(\mu) = 1$. Then, a weighted set $\mathcal{S}$ with weight function $W_{\mathcal{S}}$ is an ϵ-coreset of $\mathcal{Q}$ if

$$\ell(\mathcal{S}, \theta) \in (1 \pm \epsilon) \cdot \ell(\mathcal{Q}, \theta) \quad \text{for all} \quad \theta \in \Theta.$$

RWD and Coreset Construction

- Based RWD, design a coreset method.



Algorithm 1 Algorithm for constructing coresets

Input: a set $\mathcal{Q} := \{\mu_i\}_{i\in[N]} \subset \mathcal{P}(\mathbb{R}^d)$ of measures, doubling dimension ddim of $\mathcal{Q}$, radius r

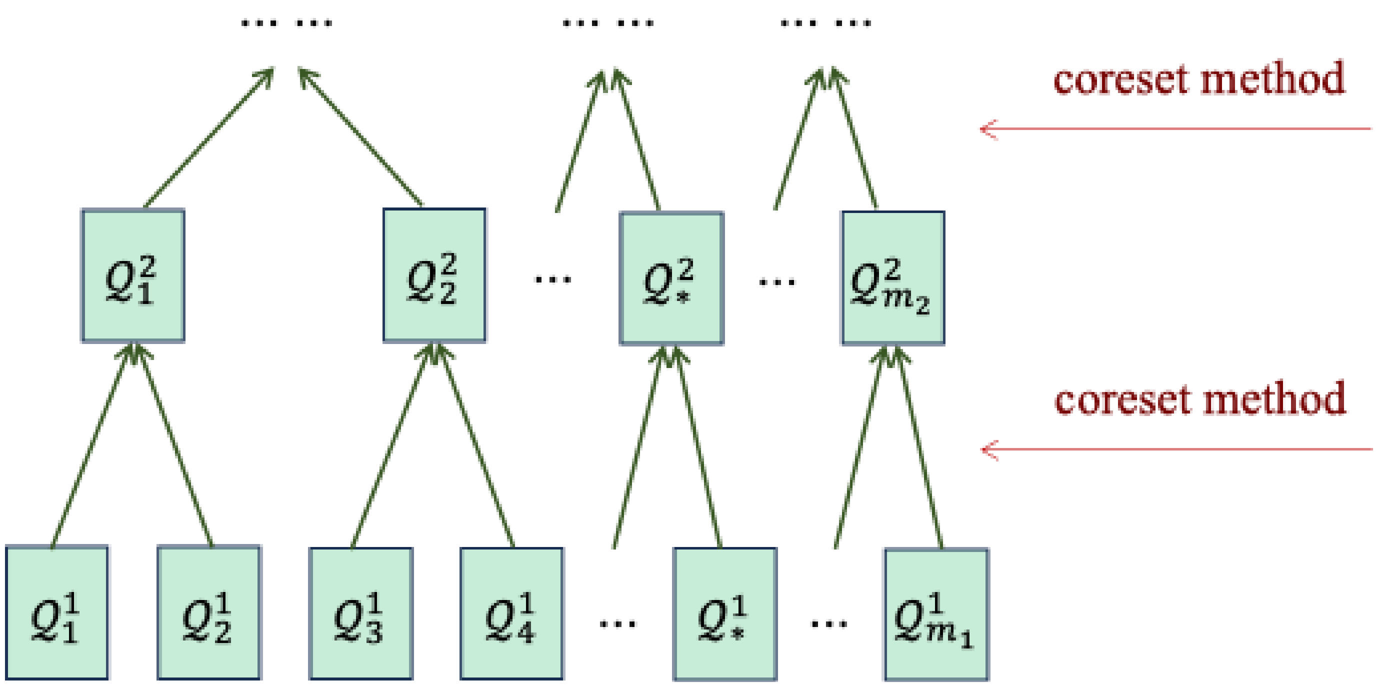
Initialize an empty tree $\mathcal{T}$, and set its root node as v_0;
Set $v_0.\mathsf{buffer} = \mathcal{Q}$;
▷ The root node v_0 only has an attribute buffer, and it is not associated with any node.
Construct the nodes of $\mathcal{T}$ recursively as follows: ▷ v is the current node.
if $v.\mathsf{buffer}$ is $\emptyset$ **then**
  The current node v is a leaf node, and we stop adding children to it;
else
  Set $k = \min\{|v.\mathsf{buffer}|, 2^{2\cdot \mathsf{ddim}}\}$ and add k children node $\{v'_j\}_{j\in[k]}$ to the current node v;
  Run Gonzalez's algorithm k rounds on $v.\mathsf{buffer}$. For each children node v'_j, we set its attributions $\mathsf{cluster}$, center, ball, buffer according to Equation (3) and Equation (4);
end if
Set $\mathcal{S} = \{v.\mathsf{center} \mid v \text{ is a node of } \mathcal{T}\}$ and set the weight as $w_{\mathcal{S}}(\mu) = |v.\mathsf{ball}|$;

Output: $\mathcal{T}, \mathcal{S}$

- Theoretical guarantee: Under a Lipschitz continuous loss function, the coreset approximates the full dataset's objective within a $(1 \pm \epsilon)$ factor.
- The resulting tree structure is retained for later use in the inference phase.

Accelerate Coreset Construction with Merge-and-Reduce

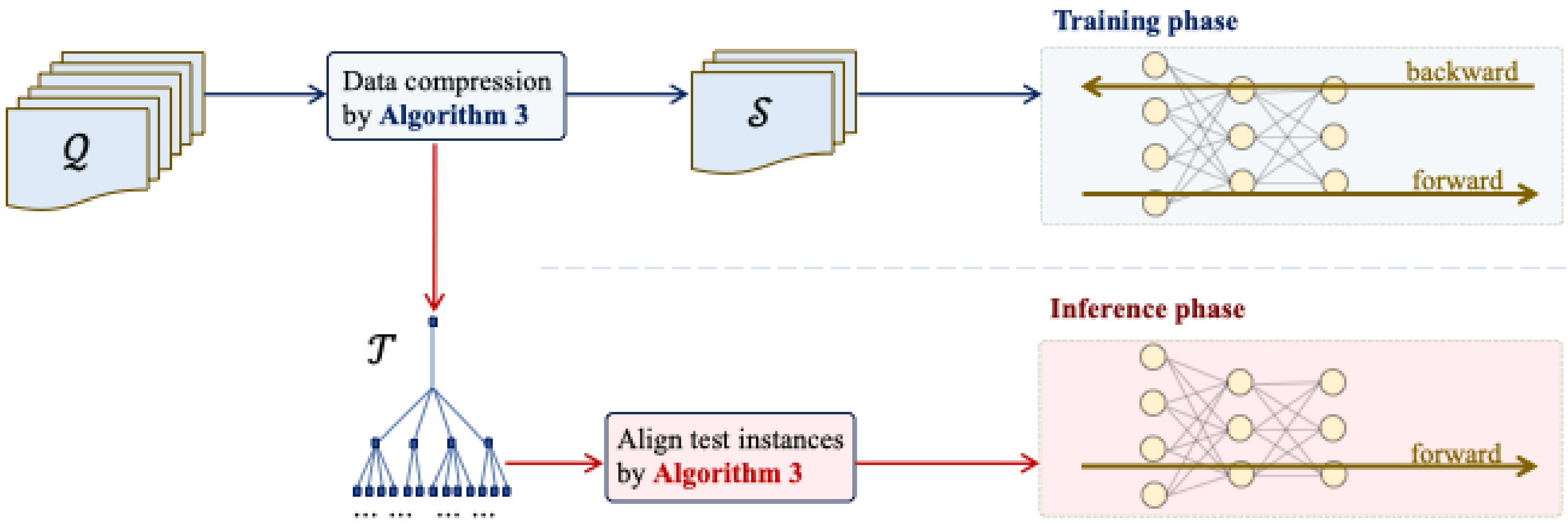
- Introduce the merge-and-reduce framework: partition the data into blocks and process them in parallel.



- This approach reduces the time complexity of coreset construction and supports streaming data scenarios.

Efficient Framework for NCO methods

- Training Phase: Replace the full dataset with the small-size coreset $\mathcal{S}$, saving computational and storage resources.



- Inference Phase: Align test instances with the coreset using the pre-constructed tree $\mathcal{T}$ to quickly find the closest representative.

Experimental Results (TSP)

- Datasets: TSP100 (2D & 3D instances), TSPLIB, etc.
  - Metrics: Tour length and runtime.

- Comparisons: The coreset method outperforms uniform sampling, particularly under distribution shifts.

- Results:
  - Comparable performance with reduced sample sizes.
  - Robustness to distribution shifts.
  - Better generalization to larger problem sizes (e.g., TSP200, TSP500, TSP1000).

Experimental Results (TSP-2D)

Table 1: Comparison of uniform sampling and our coreset method using TSP100-2D-$\mathcal{N}(0,1)$ as the training dataset on test data TSP100-2D from different distributions.

Sample size	Method	Test distribution	Greedy Length (↓)	Greedy Time (↓)	Greedy+2-opt Length (↓)	Greedy+2-opt Time (↓)
128000	Org	$\mathcal{N}(0,1)$	20.39	386	18.61	384
		$\mathcal{N}(0,4)$	76.41	374	67.39	388
		$\mathcal{U}(0,10)$	89.29	372	79.82	385
4003	US	$\mathcal{N}(0,1)$	22.34	378	18.92	387
		$\mathcal{N}(0,4)$	101.95	379	69.28	395
		$\mathcal{U}(0,10)$	119.78	380	82.59	395
	CS	$\mathcal{N}(0,1)$	22.21	372	**18.87**	379
		$\mathcal{N}(0,4)$	**80.63**	372	67.92	379
		$\mathcal{U}(0,10)$	**94.73**	373	80.64	377
	CS-aligned	$\mathcal{N}(0,1)$	**22.18**	359	18.88	363
		$\mathcal{N}(0,4)$	80.66	362	**67.91**	358
		$\mathcal{U}(0,10)$	94.94	361	**80.53**	360
8245	US	$\mathcal{N}(0,1)$	22.12	377	18.87	388
		$\mathcal{N}(0,4)$	83.17	377	68.13	378
		$\mathcal{U}(0,10)$	97.31	377	80.80	387
	CS	$\mathcal{N}(0,1)$	**21.79**	366	**18.84**	383
		$\mathcal{N}(0,4)$	78.72	372	**67.79**	378
		$\mathcal{U}(0,10)$	**92.99**	374	**80.35**	377
	CS-aligned	$\mathcal{N}(0,1)$	21.80	360	18.86	359
		$\mathcal{N}(0,4)$	**78.50**	361	67.82	358
		$\mathcal{U}(0,10)$	93.04	355	80.42	361
12951	US	$\mathcal{N}(0,1)$	21.99	390	18.87	377
		$\mathcal{N}(0,4)$	80.78	384	67.94	379
		$\mathcal{U}(0,10)$	95.01	369	80.60	379
	CS	$\mathcal{N}(0,1)$	21.57	372	18.81	382
		$\mathcal{N}(0,4)$	77.80	369	67.58	379
		$\mathcal{U}(0,10)$	**92.01**	378	80.23	375
	CS-aligned	$\mathcal{N}(0,1)$	**21.50**	361	**18.79**	358
		$\mathcal{N}(0,4)$	**77.67**	362	**67.57**	357
		$\mathcal{U}(0,10)$	**92.01**	358	**80.21**	359

Table 2: Comparison of uniform sampling and our coreset method using TSP100-2D-$\mathcal{N}(0,1)$ as the training dataset on test data of varying sizes. We fix the sample size as 12951.

TSP size	Method	Test distribution	Greedy Length (↓)	Greedy Time (↓)	Greedy+2-opt Length (↓)	Greedy+2-opt Time (↓)
TSP200	US	$\mathcal{N}(0,1)$	33.69	109	27.14	112
		$\mathcal{N}(0,4)$	125.99	108	96.70	112
		$\mathcal{U}(0,10)$	145.41	109	113.39	112
	CS	$\mathcal{N}(0,1)$	**30.75**	107	26.69	110
		$\mathcal{N}(0,4)$	**110.48**	109	94.84	111
		$\mathcal{U}(0,10)$	129.77	107	**111.47**	109
	CS-aligned	$\mathcal{N}(0,1)$	30.77	77	**26.68**	79
		$\mathcal{N}(0,4)$	110.99	78	**94.59**	79
		$\mathcal{U}(0,10)$	**129.28**	76	111.49	78
TSP500	US	$\mathcal{N}(0,1)$	59.81	1012	43.41	1020
		$\mathcal{N}(0,4)$	237.72	1012	154.28	1022
		$\mathcal{U}(0,10)$	263.66	1015	180.75	1022
	CS	$\mathcal{N}(0,1)$	**49.11**	1012	**42.25**	1016
		$\mathcal{N}(0,4)$	**178.56**	1010	**149.50**	1016
		$\mathcal{U}(0,10)$	**208.36**	1011	**174.93**	1016
	CS-aligned	$\mathcal{N}(0,1)$	49.38	680	42.26	683
		$\mathcal{N}(0,4)$	178.88	679	149.63	682
		$\mathcal{U}(0,10)$	208.77	678	175.03	682
TSP1000	US	$\mathcal{N}(0,1)$	94.71	2823	61.72	2848
		$\mathcal{N}(0,4)$	382.77	4224	219.16	2847
		$\mathcal{U}(0,10)$	426.61	4215	255.95	4254
	CS	$\mathcal{N}(0,1)$	**69.76**	2823	59.59	2833
		$\mathcal{N}(0,4)$	**252.92**	4224	**210.71**	2832
		$\mathcal{U}(0,10)$	**299.80**	4215	**246.57**	4234
	CS-aligned	$\mathcal{N}(0,1)$	69.96	2825	**59.57**	2832
		$\mathcal{N}(0,4)$	253.63	2821	210.81	2830
		$\mathcal{U}(0,10)$	300.03	2812	246.61	2827

Experimental Results (TSP-2D)

Table 3: Comparison of uniform sampling and our coreset method using TSP100-2D-$\mathcal{N}(0,1)$ as the training dataset on test data TSPLIB(Reinelt, 1991).

Sample size	Method	Test distribution	Greedy Length (↓)	Greedy Time (↓)	Greedy+2-opt Length (↓)	Greedy+2-opt Time (↓)
128000	Org	$\mathcal{N}(0,1)$	129.35	108	112.23	106
4003	US	$\mathcal{N}(0,1)$	190.79	109	115.87	108
	CS	$\mathcal{N}(0,1)$	153.08	107	**113.56**	108
	CS-aligned	$\mathcal{N}(0,1)$	**152.70**	103	113.71	105
8245	US	$\mathcal{N}(0,1)$	166.40	106	114.47	108
	CS	$\mathcal{N}(0,1)$	140.49	107	113.04	106
	CS-aligned	$\mathcal{N}(0,1)$	**140.18**	104	**112.91**	104
12951	US	$\mathcal{N}(0,1)$	162.19	107	114.31	107
	CS	$\mathcal{N}(0,1)$	133.63	106	**112.45**	105
	CS-aligned	$\mathcal{N}(0,1)$	**133.14**	103	112.52	110

Experimental Results (TSP-3D)

Table 10: Comparison of uniform sampling and our coreset method with training dataset TSP100-3D-$\mathcal{N}(0,1)$ on test data of varying sizes. We fix the sample size as 12058.

TSP size	Method	Test distribution	Greedy Length (↓)	Greedy Time (↓)	Alignment time (↓)
TSP-200	Org	$\mathcal{N}(0,1)$	30.02	77	-
		$\mathcal{N}(0,2)$	61.03	76	-
		$\mathcal{N}(0,4)$	109.78	77	-
		$\mathcal{N}(0,8)$	126.15	76	-
		$\mathcal{U}(0,10)$	128.35	77	-
TSP-500	Org	$\mathcal{N}(0,1)$	48.66	682	-
		$\mathcal{N}(0,2)$	101.71	682	-
		$\mathcal{N}(0,4)$	184.94	682	-
		$\mathcal{N}(0,8)$	210.36	680	-
		$\mathcal{U}(0,10)$	212.62	683	-
TSP-1000	Org	$\mathcal{N}(0,1)$	69.08	2828	-
		$\mathcal{N}(0,2)$	144.10	2826	-
		$\mathcal{N}(0,4)$	264.58	2824	-
		$\mathcal{N}(0,8)$	303.02	2821	-
		$\mathcal{U}(0,10)$	313.23	2818	-
TSP-200	US	$\mathcal{N}(0,1)$	**32.30**	76	-
		$\mathcal{N}(0,2)$	67.81	77	-
		$\mathcal{N}(0,4)$	130.76	77	-
		$\mathcal{N}(0,8)$	152.74	77	-
		$\mathcal{U}(0,10)$	153.90	77	-
	CS	$\mathcal{N}(0,1)$	32.72	76	-
		$\mathcal{N}(0,2)$	**66.08**	76	-
		$\mathcal{N}(0,4)$	120.64	78	-
		$\mathcal{N}(0,8)$	139.91	77	-
		$\mathcal{U}(0,10)$	142.84	77	-
	CS-aligned	$\mathcal{N}(0,1)$	33.25	76	2
		$\mathcal{N}(0,2)$	69.79	75	5
		$\mathcal{N}(0,4)$	**118.44**	77	7
		$\mathcal{N}(0,8)$	**135.90**	77	10
		$\mathcal{U}(0,10)$	**138.27**	76	12
TSP-500	US	$\mathcal{N}(0,1)$	58.45	682	-
		$\mathcal{N}(0,2)$	127.53	681	-
		$\mathcal{N}(0,4)$	264.93	682	-
		$\mathcal{N}(0,8)$	316.73	681	-
		$\mathcal{U}(0,10)$	318.39	680	-
	CS	$\mathcal{N}(0,1)$	59.69	682	-
		$\mathcal{N}(0,2)$	122.24	681	-
		$\mathcal{N}(0,4)$	231.96	682	-
		$\mathcal{N}(0,8)$	266.08	677	-
		$\mathcal{U}(0,10)$	268.75	680	-
	CS-aligned	$\mathcal{N}(0,1)$	**56.47**	682	16
		$\mathcal{N}(0,2)$	**121.33**	681	19
		$\mathcal{N}(0,4)$	**205.45**	682	23
		$\mathcal{N}(0,8)$	**238.14**	681	26
		$\mathcal{U}(0,10)$	**245.01**	681	30
TSP-1000	US	$\mathcal{N}(0,1)$	**86.01**	2828	-
		$\mathcal{N}(0,2)$	191.39	2828	-
		$\mathcal{N}(0,4)$	397.40	2825	-
		$\mathcal{N}(0,8)$	470.55	2820	-
		$\mathcal{U}(0,10)$	550.11	2819	-
	CS	$\mathcal{N}(0,1)$	86.57	2826	-
		$\mathcal{N}(0,2)$	**181.84**	2827	-
		$\mathcal{N}(0,4)$	342.64	2822	-
		$\mathcal{N}(0,8)$	395.66	2820	-
		$\mathcal{U}(0,10)$	441.82	2818	-
	CS-aligned	$\mathcal{N}(0,1)$	86.87	2828	236
		$\mathcal{N}(0,2)$	186.88	2823	433
		$\mathcal{N}(0,4)$	**320.68**	2825	624
		$\mathcal{N}(0,8)$	**372.34**	2820	861
		$\mathcal{U}(0,10)$	**383.92**	2819	980

Continued on next page

– continued from previous page

Experimental Results (TSP-3D)

Table 9: Comparison of uniform sampling and our coreset method using TSP100-3D-$\mathcal{N}(0,1)$ as the training dataset on test data TSP100-3D from different distributions.

Sample size	Method	Test distribution	Greedy Length (↓)	Greedy Time (↓)	Alignment time (↓)
128000	Org	$\mathcal{N}(0,1)$	20.80	364	-
		$\mathcal{N}(0,2)$	42.25	366	-
		$\mathcal{N}(0,4)$	76.57	362	-
		$\mathcal{N}(0,8)$	88.45	360	-
		$\mathcal{U}(0,10)$	90.55	358	-
4103	US	$\mathcal{N}(0,1)$	24.92	480	-
		$\mathcal{N}(0,2)$	49.96	482	-
		$\mathcal{N}(0,4)$	96.60	483	-
		$\mathcal{N}(0,8)$	116.65	482	-
		$\mathcal{U}(0,10)$	119.78	481	-
	CS	$\mathcal{N}(0,1)$	24.89	364	-
		$\mathcal{N}(0,2)$	50.46	384	-
		$\mathcal{N}(0,4)$	106.35	360	-
		$\mathcal{N}(0,8)$	109.12	360	-
		$\mathcal{U}(0,10)$	111.63	353	-
	CS-aligned	$\mathcal{N}(0,1)$	**23.36**	479	2
		$\mathcal{N}(0,2)$	**47.00**	480	4
		$\mathcal{N}(0,4)$	**91.94**	479	7
		$\mathcal{N}(0,8)$	**106.50**	482	9
		$\mathcal{U}(0,10)$	**108.81**	483	11
7960	US	$\mathcal{N}(0,1)$	23.62	477	-
		$\mathcal{N}(0,2)$	48.32	477	-
		$\mathcal{N}(0,4)$	92.92	484	-
		$\mathcal{N}(0,8)$	111.98	479	-
		$\mathcal{U}(0,10)$	115.62	481	-
	CS	$\mathcal{N}(0,1)$	23.41	362	-
		$\mathcal{N}(0,2)$	47.10	361	-
		$\mathcal{N}(0,4)$	86.20	362	-
		$\mathcal{N}(0,8)$	99.50	365	-
		$\mathcal{U}(0,10)$	101.25	359	-
	CS-aligned	$\mathcal{N}(0,1)$	**22.83**	476	12
		$\mathcal{N}(0,2)$	**46.06**	474	14
		$\mathcal{N}(0,4)$	**85.71**	483	16
		$\mathcal{N}(0,8)$	**97.61**	480	17
		$\mathcal{U}(0,10)$	**99.10**	481	19

Please refer to the original article for more experimental and technical details.

Conclusion and Future Work

- Contributions:
  - A coreset construction method and its acceleration
  - Efficient Framework for NCO methods
- Future Work:
  - Extend the approach to other optimization problems with graph structures.
  - Explore enhanced alignment and acceleration strategies for high-dimensional data.