From Sorting Algorithms to Scalable Kernels: Bayesian Optimization in High-Dimensional Permutation Spaces

Anonymous authors
Paper under double-blind review

Abstract

Bayesian Optimization (BO) is a powerful tool for black-box optimization, but its application to high-dimensional permutation spaces is severely limited by the challenge of defining scalable representations. The current state-of-the-art BO approach for permutation spaces relies on an exhaustive $\Omega(n^2)$ pairwise comparison, inducing a dense representation that is impractical for large-scale permutations. To break this barrier, we introduce a novel framework for generating efficient permutation representations via kernel functions derived from sorting algorithms. Within this framework, the Mallows kernel can be viewed as a special instance derived from enumeration sort. Further, we introduce the **Merge Kernel** , which leverages the divide-and-conquer structure of merge sort to produce a compact, $\Theta(n \log n)$ to achieve the lowest possible complexity with no information loss and effectively capture permutation structure. Our central thesis is that the Merge Kernel performs competitively with the Mallows kernel in low-dimensional settings, but significantly outperforms it in both optimization performance and computational efficiency as the dimension n grows. Extensive evaluations on various permutation optimization benchmarks confirm our hypothesis, demonstrating that the Merge Kernel provides a scalable and more effective solution for Bayesian optimization in high-dimensional permutation spaces, thereby unlocking the potential for tackling previously intractable problems such as large-scale feature ordering and combinatorial neural architecture search.

1 Introduction

As one of the most widely adopted approaches to black-box optimization, Bayesian optimization (BO) Shahriari et al. (2015) has found broad application in machine-learning hyper-parameter tuning Wu et al. (2019), financial portfolio optimization Gonzalvez et al. (2019), chemical and material discovery Luo et al. (2025), and catalyst formulation design Xie et al. (2023). BO employs probabilistic surrogate models—most commonly Gaussian processes (GPs)—to approximate the unknown objective and uses an acquisition function to balance exploration and exploitation, thereby approaching the global optimum with the fewest possible evaluations.

Most research on BO to date has concentrated on continuous Greenhill et al. (2020) and categorical design spaces Garrido-Merchán & Hernández-Lobato (2020); Nguyen et al. (2020); Bartoli et al. (2025), whereas applications to permutation spaces remain comparatively underexplored. This gap is striking given the ubiquity of permutation optimization in both theory and practice: canonical examples include the traveling-salesperson problem (TSP), the sequencing of operations in automated experimental pipelines Guidi et al. (2020), and the sequential order-of-addition experiments Lin & Rios (2025). In addition, permutation optimization frequently arises in diverse AI applications, including scheduling tasks in robotic planning Alatartsev et al. (2015), optimizing experimental sequences Blau et al. (2022), and other sequential decision-making problems Sun & Giles (2001); Wen et al. (2023). Therefore, extending Bayesian optimization to permutation spaces holds significant theoretical and practical value for artificial intelligence as well as broader academic and industrial applications.

Successfully deploying BO in permutation spaces hinges on equipping the GP surrogate with a kernel that faithfully quantifies the similarity between two permutations. Existing approaches have evolved along two principal lines: (1) General-purpose discrete BO frameworks—most notably COMBO Oh et al. (2019), which combines a graph-Laplacian kernel with local search to accommodate heterogeneous discrete variables and thus enjoys broad applicability. However, it relies on manually crafted adjacency graphs or hash encodings, which struggle to capture the comparison sequences and cyclic shifts unique to permutations; consequently, its efficiency degrades on tasks that involve frequent high-order swaps or mixed local/global reorderings. (2) Permutation-specific kernels—chief among them the Mallows kernel in BOPS Deshwal et al. (2022). This kernel is built on the Kendall–τ distance, representing a permutation via the inversion counts generated by enumeration sort, yielding a feature dimension of $O(n^2)$. The dimension therefore scales quadratically with n, and because $2^{n^2} \gg n!$, the vast majority of features fail to correspond to any valid permutation, resulting in both statistical and computational redundancy.

To address the limitations of existing approaches, we propose a sorting-algorithm–driven kernel-design framework for permutations and instantiate it with the Merge kernel, which reduces the feature dimension to $O(n \log n)$—the information-theoretic lower bound for encoding a permutation. The central insight is that any comparison-based sorting algorithm is defined by a fixed sequence of element comparisons; recording the binary outcome of each comparison yields a feature vector for the permutation. Choosing an algorithm with a deterministic comparison tree—such as merge sort or bitonic sort—thus produces a representation that is both fixed in length and highly compact.

Contributions. Our work makes three principal contributions:

- **General framework.** We propose a unified design framework that constructs permutation-space kernels by treating any comparison-based sorting algorithm as a feature generator. Within this view, the classic Mallows kernel is recovered as the special case obtained when the framework is instantiated with enumeration sort.
- **Merge kernel.** Applying the framework to merge sort yields Merge Kernel, whose $O(n \log n)$ construction matches the information-theoretic lower bound on comparison complexity.
- **Comprehensive evaluation.** We assess the effectiveness of our kernels on diverse synthetic and real-world benchmarks. Results on low-dimensional benchmarks show competitive performance against the state-of-the-art Mallows kernel, while it significantly outperforms the Mallows kernel on high-dimensional benchmarks.

Our results demonstrate that the Merge kernel provides a practical and efficient tool for permutation optimization, significantly enhancing BO's applicability to diverse AI scenarios.

2 Background and Related Works

2.1 Permutation Optimization

Here we describe the problem formulation of permutation optimization with a fixed length $n \in \mathbb{N}$. Let $[n] = \{1, 2, ..., n\}$, a permutation is a function $\pi : [n] \longrightarrow [n]$ such that π is bijective. The set of all permutations of $[n]$ is the symmetric group

$$\mathcal{S}_n \ = \ \bigl\{\pi \mid \pi : [n] \longrightarrow [n] \text{ is bijective}\bigr\}.$$

We are given a costly-to-evaluate, possibly noisy black-box function $f : \ \mathcal{S}_n \longrightarrow \mathbb{R}$, which assigns a real-valued quality (e.g., cost, loss, reward) to every permutation π. Hence, the optimization problem can be formulated as

$$\pi^* \ = \ \arg\min_{\pi \in \mathcal{F}} f(\pi)$$

where $\mathcal{F} \subseteq \mathcal{S}_n$ is the feasible set. In this study we only consider the unconstrained case, therefore we have $\mathcal{F} \ = \ \mathcal{S}_n$; in practice $\mathcal{F}$ may exclude permutations violating domain rules.

2.2 Bayesian Optimization

Bayesian optimization (BO) Shahriari et al. (2015) is an optimization algorithm for black-box objective functions that no closed-form expression or gradient information is available and whose evaluation is often an expensive physical or computational experiment. The algorithm first fits the observed data with a surrogate model, most commonly a Gaussian process (GP) Williams & Rasmussen (2006), and then employs an acquisition function to select the next query point, balancing exploration of uncertain regions against exploitation of promising areas identified by the surrogate.

The kernel $K(\mathbf{x}, \mathbf{x}')$ is the central design lever in a Gaussian-process surrogate: it defines the similarity metric between inputs, thereby specifying the prior smoothness assumptions and, through GP inference, the posterior mean and uncertainty. Extending BO to any new search domain is therefore tantamount to endowing that domain with an appropriate kernel function. Whereas Euclidean spaces typically rely on Gaussian (RBF) kernels, discrete structures—and permutations in particular—require bespoke constructions that faithfully encode ordering relationships. For example, Oh et al. (2022) uses the Position kernel Zaefferer et al. (2014) and L-ensemble with Acquisition Weights to extend BO on permutation space to a batched BO scheme.

While significant progress has been made in scaling BO to high-dimensional continuous domains (e.g., Eriksson et al. (2019); Wang et al. (2016)), extending BO to large-scale structured discrete domains like permutations presents a distinct set of challenges centered on kernel design, which is the primary focus of this work.

2.3 Mallows Kernel for Permutation Space

BOPS-H Deshwal et al. (2022) is the current state-of-the-art BO algorithm for permutation optimization, which proposes to employ Mallows kernel Jiao & Vert (2015) on the symmetric group $\mathcal{S}_n$ in a similar manner to the RBF kernel on the Euclidean space. The Mallows kernel $K_{mal}(\pi, \pi')$ for the permutation pair (π, π') is defined as the exponential negative of the number of discordant pairs $n_d(\pi, \pi')$ between π and π':

$$K_{Mal}(\pi, \pi') = \exp(-l d(\pi, \pi')) \tag{1}$$

where $l \geq 0$ is the length-scale parameter of the Mallows kernel, and $d(\pi, \pi')$ is the Kendall-τ distance Kendall (1938) which counts the number of pairs of elements ordered oppositely by π and π':

$$\begin{aligned} d(\pi, \pi') = \sum_{i<j} \big[& 1_{\pi(i)>\pi(j)} 1_{\pi'(i)<\pi'(j)} \\ & + 1_{\pi(i)<\pi(j)} 1_{\pi'(i)>\pi'(j)} \big] \end{aligned} \tag{2}$$

Intuitively, the Kendall-τ distance counts the differences of all pair-wise comparisons between π and π'. For example, let $\pi = (1, 2, 3, 4)$ and $\pi' = (2, 1, 4, 3)$. Two pairs are discordant among the six unordered pairs: $(1, 2), (3, 4)$, hence $d(\pi, \pi') = 2$ and $K_{Mal}(\pi, \pi') = \exp(-2l)$.

3 Merge Kernel: Generating Kernels from sorting algorithms

In the previous section, we have shown that the core of the Mallows kernel is the pairwise comparison of all elements. Equivalently, it maps a permutation π to a feature vector

$$\Phi_{Mal}(\pi) \in \{0, 1\}^{\binom{n}{2}}$$

where each coordinate corresponds to the comparison of a pair of elements: 0 if they are in ascending order, and 1 otherwise. We then have

$$\begin{aligned} K_{Mal}(\pi, \pi') &= \exp\Big(-\frac{\|\Phi_{Mal}(\pi) - \Phi_{Mal}(\pi')\|^2}{2\ell^2} \Big) \\ &= K_{\mathrm{RBF}}\big(\Phi_{Mal}(\pi), \Phi_{Mal}(\pi')\big). \end{aligned} \tag{3}$$

under the reparameterization $l = \frac{1}{2\ell^2}$ for the length-scale parameter ℓ in the Gaussian RBF kernel. Since the RBF kernel K_{RBF} is strictly positive definite on $\mathbb{R}^d$ and thus satisfies Mercer's condition Mercer (1909), and because positive definiteness is preserved under composition with any deterministic mapping Φ, it follows that $K(\pi, \pi')$ constructed from Φ also satisfies Mercer's condition and is therefore a valid kernel function.

Algorithm 1 MERGE FEATURE MAPPING $\Phi_{Mer}(\pi)$

```
Input: Permutation $\pi$ of length $n$
Output: Feature vector $\Phi_{Mer}(\pi)$
  if length($\pi$) == 1 then
    return []
  end if
  if length($\pi$) == 2 then
    return [1] if $\pi[0] > \pi[1]$ else [0]
  end if
  Let $mid = \lfloor \frac{n}{2} \rfloor$
  Let $V_{Left}, V_{Right} = \Phi_{Mer}(\pi[: mid]), \Phi_{Mer}(\pi[mid :])$
  Let $\hat{\pi}_l, \hat{\pi}_r$ = sorted($\pi[: mid]$), sorted($\pi[mid :]$)
  Let $V_{Merge} = []$, $i = j = 0$
  while $i <$ length($\hat{\pi}_l$) and $j <$ length($\hat{\pi}_r$) do
    if $\hat{\pi}_l[i] > \hat{\pi}_r[j]$ then
      $V_{Merge}$.append(1)
      $j+ = 1$
    end if
    if $\hat{\pi}_l[i] < \hat{\pi}_r[j]$ then
      $V_{Merge}$.append(0)
      $i+ = 1$
    end if
  end while
  return $V_{Left} + V_{Right} + V_{Merge}$
```

Thus, other pairwise comparison methods can also be used to construct analogous feature vectors, which—when combined with an RBF kernel—yield valid kernel functions. Naturally, we can extend the idea of pairwise comparison to sorting algorithms: the essence of a sorting algorithm is to compare elements in a sequence and swap them when necessary. Consequently, each sorting algorithm embodies a unique pairwise comparison strategy, suggesting that we can build permutation-space kernels based on sorting procedures. As a sorting algorithm traverses all elements, it records whether each comparison leads to a swap, thereby fully reconstructing the original permutation; hence, the resulting feature vector retains all information without any loss. Viewed in this light, the Mallows kernel's exhaustive enumeration of every element pair can be interpreted as an enumeration-sort–inspired featurization, where enumeration sort ranks each item by comparing it with every other element and then places it directly in its final position.

Here is a more detailed explanation. Consider that each entry in the Mallows feature (or Kendall-τ distance) represents whether the order of elements at two positions i, j in a pair of permutations π_a and π_b is the same. Considering only one permutation, we compare all element pairs within it: if the element at the earlier position is greater than the element at the later position, we mark it as 1; otherwise, we mark it as 0. The value of each entry here is equivalent to the swap information examined for that pair during an enumerate sort referring to all pairwise comparisons, we can thus obtain the feature vectors for these two permutations. By comparing the elements of these two feature vectors position by position, if the two elements are the same, it means π_a and π_b share the same order for the element pair represented at this position (i.e., Φ_{mal} is 0 at this position in the distance calculation); otherwise, it is 1. Now, we can replace the above enumerate sort with a sorting algorithm to generate another feature vector for permutation kernel. That is, we use sorting algorithm to obtain the information on whether a swap occurred during every comparison in the comparison map.

However, not every sorting algorithm can induce a valid feature mapping suitable for kernel construction. This is because the mapping from permutations to feature space must yield feature vectors of fixed length; otherwise, feature vectors of differing lengths would not be compatible with the RBF kernel. Hence, only sorting algorithms that have a fixed comparison path and a constant number of comparisons across all inputs can generate valid feature mappings, e.g., a fixed sorting network with a predetermined comparator sequence Batcher (1968). This strict constraint allows us to exclude the vast majority of $O(n \log n)$ complexity sorting algorithms that are stochastic or adaptive, such as quicksort, heapsort, and standard merge sort, since their comparisons are data-dependent, not fixed. Nevertheless, merge sort could be an exception: although typically merge sort ceases comparisons once all elements from one subsequence have been merged, redundant comparisons can be artificially introduced during the merge procedure, resulting in a fixed number of comparisons $L + R - 1$, where L and R are the length of the two subsequences. This ensures that both the comparison path and the number of comparisons remain identical across different permutations, thus establishing a fixed comparison path.

Stabilizing the comparison map of other $O(n \log n)$ sorting algorithms is quite challenging, and other $O(n^2)$ sorting algorithms are equivalent to Mallows kernel. At the moment we can confirm that Bitonic sort satisfy the above constraint, however it is of $O(n \log^2 n)$ complexity. Consequently, we select merge sort as the feature mapping $\Phi_{Mer}(\pi)$ for constructing permutation-space kernel function K_{Mer}, following Equation 3:

$$\begin{aligned} K_{Mer}(\pi, \pi') &= \exp\Big(-\frac{\|\Phi_{Mer}(\pi) - \Phi_{Mer}(\pi')\|^2}{2\ell^2}\Big) \\ &= K_{\mathrm{RBF}}\big(\Phi_{Mer}(\pi),\, \Phi_{Mer}(\pi')\big). \end{aligned} \tag{4}$$

The element-pair comparison mapping $\Phi_{Mer}(\pi)$ is shown in Algorithm 1. Here we present an example for permutation [1,4,3,2] to show the **Merging** process.

Example: Feature Mapping for $\pi = (1, 4, 3, 2)$

1. **Initial Split:** π splits into $L_0 = (1, 4)$ and $R_0 = (3, 2)$.
2. **Recurse on $L_0 = (1, 4)$:** Merging sorted lists '[1]' and '[4]' yields feature vector $V_L = [0]$.
3. **Recurse on $R_0 = (3, 2)$:** Merging sorted lists '[2]' and '[3]' yields feature vector $V_R = [1]$.
4. **Final Merge:** Merge sorted lists '(1,4)' and '(2,3)'. The fixed comparison path generates the merge vector $V_{Merge} = [0(1 < 2), 1(4 > 2), 1(4 > 3), 1(\text{left padding})]$.
5. **Concatenate:** The final feature vector is $\Phi_{Mer}(\pi) = V_L \oplus V_R \oplus V_{Merge} = [0] \oplus [1] \oplus [0, 1, 1, 1] = [0, 1, 0, 1, 1, 1]$.

We have established that merge sort, with a specially designed fixed-comparison procedure, is uniquely capable of constructing valid kernel functions among common sorting algorithms with a complexity of $\Omega(n \log n)$. We now demonstrate that the feature vector derived from merge sort achieves the theoretical lower bound on vector length for lossless permutation encoding. First, note that the lower bound on time complexity for any comparison-based sorting algorithm is $\Omega(n \log n)$; as this complexity directly corresponds to the number of element comparisons during sorting, it similarly sets a lower bound on the length of the feature vector. On the other hand, consider the permutation space consisting of all $n!$ possible permutations of length n. From an information-theoretic viewpoint, encoding all $n!$ permutations without loss using a binary feature vector composed solely of $0, 1$ requires a minimum vector length of $\log_2(n!)$. Applying Stirling's approximation Donald et al. (1999), we have: $\log_2(n!) = n \log_2 n - nlog_2 e + O(\log n) = \Omega(n \log n)$. Consequently, the feature vector length of the merge-sort-based kernel (Merge Kernel) reaches this theoretical lower bound for lossless permutation encoding.

It is important to clarify that the feature vector lower bound discussed here refers to the information-theoretic lower bound on the number of pairwise comparisons required to reconstruct the relative order of two permutations, i.e., the lower bound implied by lossless information compression. This should be distinguished from the algorithmic lower bound for computing a distance regarding to the original permutations. For example, the Mallows kernel relies on the Kendall–τ distance, and although the latter can be computed in $O(n \log n)$ time using algorithms such as modified merge sort, this does not reduce the information requirement to $O(n \log n)$. Such algorithms still implicitly depend on the relative order of all $O(n^2)$ pairs of elements, but accelerate computation by batch processing rather than by reducing the underlying information model. In contrast, our method does not require explicit access to all $O(n^2)$ pairs. Instead, it achieves a complete reconstruction of the relative order using only $O(n \log n)$ pairwise comparisons.

It is worth noting that, when constructing the Merge kernel via merge sort, we have not required the feature mapping Φ to possess any group invariance property, such as right-invariance. Traditionally, a permutation-distance measure should be invariant under right multiplication, meaning that applying an identical right-multiplication operation to two permutations should not alter the distance between them. However, only sorting algorithms with complexity $O(n^2)$ can yield fully right-invariant kernel functions with no information loss (otherwise, a simple Spearman's footrule Diaconis & Graham (1977) with $O(n)$ complexity can hold right-invariance as well), since such invariance necessitates exhaustive pairwise comparisons among all $\frac{n(n-1)}{2}$ pairs of elements—an impossibility for more efficient sorting algorithms like merge sort. Consequently, although the Merge kernel achieves better computational efficiency through a more compact encoding, it sacrifices a certain degree of performance due to the loss of right-invariance.

We can view the relationship between the Merge and Mallows kernels as a feature selection process. Given that the $\Phi_{\text{mer}}(\pi)$ vector corresponds to a structured subset of the complete $\Phi_{\text{mal}}(\pi)$ feature space, incrementally adding the missing comparison positions to Φ_{mer} is equivalent to a gradual transformation towards the Φ_{mal} vector. This transformation represents a principled trade-off: the process of "buying back" the property of right-invariance through the incorporation of more features is achieved at the explicit cost of sacrificing computational efficiency. However, because this requires the development of appropriate analytical tools to quantify the exact marginal gain in invariance per added comparison, this remains a fascinating, yet highly complex, direction for future research that is beyond the scope of this paper.

4 Experiments

4.1 Benchmarks and Experiment Settings

Our empirical evaluation focuses on the state-of-the-art BOPS-H algorithm Deshwal et al. (2022) as the primary control baseline. This choice is twofold: first, BOPS-H was shown to substantially outperform other permutation-specific methods like COMBO Oh et al. (2019). Second, our core objective is a principled comparison between our Merge Kernel and the Mallows Kernel—both natively designed for permutations. This comparison serves as a direct evaluation of representation power without confounding factors from domain adaptation. Furthermore, we adapt TuRBO Eriksson et al. (2019) as a high-dimensional BO algorithm of general purpose to evaluate the overall competitiveness of our framework. Since TuRBO is designed for continuous space, we apply a continuous relaxation to the permutation space: we define the search space as a $d-$dimensional unit hypercube $[0, 1]^d$, where the discrete permutation is induced by the argsort of the continuous vector elements.

To disentangle whether the performance of the Merge kernel stems merely from its compact dimensionality or from the specific structured information it captures, we introduce a randomized baseline. Specifically, we randomly subsample a fixed number of pairwise comparisons from the full Mallows feature vector, ensuring the total dimensionality exactly matches that of the Merge kernel. We then apply the same RBF kernel to these features. If the success of the Merge kernel were driven solely by "compression" rather than the "informative structure" of the features, this baseline should achieve comparable performance. In addition, discussion of using Spearman's footrule as featurization method is also added to Appendix B.1, due to limited space in the main manuscript.

4.1.1 Low-dimensional Benchmarks

The study by Deshwal et al. (2022) exclusively considers problems with dimensions of 30 or less. Accordingly, we adopt their experimental settings to form our suite of low-dimensional benchmarks. The BOPS-H algorithm follows and modifies the local-search strategy used in COMBO, examining only the set of neighbouring permutations of the current incumbent to restrict the combinatorial search space, we therefore adopt the same procedure in our experiments. GPyTorch Gardner et al. (2018) and BoTorch Balandat et al. (2020) libraries are used to implement both algorithms. Expected Improvement acquisition function is used for all the experiments, and 10 restarts are used for local search based EI optimization for BOPS-H and MergeBO. Each benchmark is evaluated with 20 independent trials, each consisting of 50 iterations. The random seed for each trial is set to its trial index.

We evaluate our method on the same two synthetic benchmarks and the same two real-world applications in Deshwal et al. (2022). Detailed information for all benchmarks are listed below:

(1)Quadratic Assignment (QAP)$_{n=15}$**.** This is a classic facility-location problem: assign n facilities to n locations so that flow costs and distances align optimally. We use the 15-city instances from **QAPLIB** Burkard et al. (1997), each with cost matrix A and distance matrix B, and minimise $\mathrm{Tr}(APBP^{\mathsf{T}})$ over permutation matrices P.

(2)Travelling Salesman (TSP)$_{n=15}$**.** The TSP seeks the shortest Hamiltonian cycle through a set of cities and is a standard benchmark for route-planning. Our instances are 15 city PCB drill tours from **TSPLIB** Reinelt (1995); the score is the total travel time to visit all holes exactly once and return.

(3)Floor Planning (FP)$_{n=30}$**.** Floor planning is an NP-hard VLSI layout task that packs rectangular modules on a chip while minimising area and manufacturing cost. We evaluate two 15-block variants (FP-1 and FP-2); each permutation defines a block placement whose cost we minimise.

(4)Cell Placement (CP)$_{n=30}$**.** Cell placement arranges logic cells on a row to reduce wire-length and hence circuit delay. We consider 30 equal-height cells with a fixed net-list; the objective is the total Manhattan wire-length induced by a permutation of cell positions.

Because the publicly available implementation of the Mallows kernel (https://github.com/aryandeshwal/BOPS) does not provide the interface required to run the Rodinia's heterogeneous many-core benchmark Che et al. (2009), we did not perform experiments on this benchmark. We note a numerical discrepancy between our replicated results and those reported in the original paper, which we attribute to subtle implementation details not specified in the publication, such as problem instance choices. However, we emphasize that within our experimental framework, both the Merge Kernel and the Mallows Kernel were evaluated under identical conditions, ensuring a fair and controlled comparison of their relative performance.

4.1.2 High-dimensional Benchmarks: Traveling Thief Problems

We introduce traveling thief problems (TTP) Bonyadi et al. (2013) as high-dimensional benchmarks, which is a defined as a combination of TSP and knapsack problem: a thief must determine a tour through n cities with distance matrix $D = \{d_{ij}\}$ while simultaneously selecting items of varying weights w_k and values p_k to maximize profit without exceeding a knapsack capacity W. This structure defines a complex, hybrid search space, combining an n-dimensional permutation space for the city tour with a $\{0,1\}^m$ discrete space for item selection. Despite its typical application in evaluating white-box or heuristic algorithms Polyakovskiy et al. (2014); Gupta et al. (2015), we adapt the TTP as a true black-box benchmark, providing no structural information to the optimizer.

Following the implementation in Polyakovskiy et al. (2014), we create three distinct instances based on a $n = 280$-city problem (a 280-dimensional permutation space). These instances feature a large number of items with varying properties: (1) $m = 279$ items with uncorrelated weights; (2) $m = 837$ items with bounded strong correlation in the weights; and (3) $m = 837$ items with uncorrelated weights. These benchmarks provide a strenuous test for our proposed MergeBO and the baseline BOPS-H. Notice that the TTP is a hybrid space problem, we modified both MergeBO and BOPS-H by multiplying an RBF kernel on the $\{0,1\}^m$ discrete space:

$$K_{TTP}((\pi,\sigma),(\pi',\sigma')) = K(\pi,\pi')K_{RBF}(\sigma,\sigma') \tag{5}$$

Where σ is the item picking strategy. The neighbouring permutation search method for BOPS-H is computationally infeasible on such a vast space. Therefore, we adopt a continuous relaxation approach, treating the entire set of optimization variables (π,σ) as a continuous vector for gradient-based optimization, and the result is subsequently projected back to the nearest permutation and binary vectors.

Crucially, this relaxation approach relies on the existence of a feasible projection from the continuous feature space back to the permutation space. While our Merge kernel features preserve the structural logic of the sorting algorithm to allow for valid reconstruction, the randomized baseline lacks this structural consistency (e.g., potentially inducing cyclic or conflicting comparisons). Consequently, a valid projection for the randomized baseline is ill-defined, rendering it inapplicable to this high-dimensional optimization setting.

Table 1: Feature length comparison of Merge and Mallows kernel over problems of different scales.

Problem	Dimension	Merge feature length	Mallows feature length
TSP	15	45	105
QAP	15	45	105
FP	30	119	435
CR	30	119	435
TTP	280	2009	39060

All experiments use the EI acquisition function. Each benchmark is evaluated with 50 independent trials, each consisting of 55 iterations (5 iterations of random initialization and 50 iterations of optimization). The random seed for each trial is set to its trial index.

4.1.3 Evaluation Metrics

In this study, we employ two evaluation metrics: the final simple regret and the area under the best-so-far regret curve (AUC). In optimization, regret is defined as the difference between the best objective value observed to date and the global optimum:

$$r_t = f_t^{\text{best}} - f^*$$

Following this concept, the final simple regret is the regret value obtained in the last iteration and reflects the algorithm's ultimate optimization capability when computational cost is disregarded. By contrast, the regret AUC, or cumulative regret—the sum (area under the curve) of the simple regret across all iterations—quantifies the convergence speed of the entire optimization process:

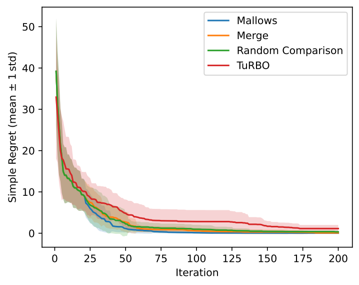
$$\text{AUC}_T = \Sigma r_t$$

Generally speaking, the final simple regret and AUC represent different, valuable aspects of an algorithm's performance: the former represents the the quality of the solution it ultimately finds while the latter represents the "journey", or convergence speed during optimization. For sample-efficient methods like BO, these two evaluation metrics are standard practices as they allow for a comprehensive comparison.

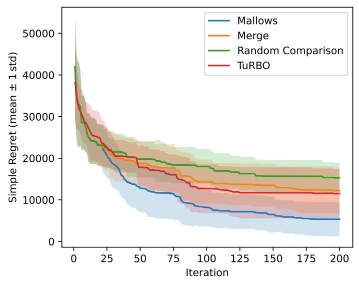
Table 2: Performance comparison between MergeBO, BOPS-H (Mallows kernel), BOPS-H with random comparisons and TuRBO. Underlined results indicate the best numerical results in terms of mean value $\pm$ standard deviation of all trials, and bold font indicates statistically significant superiority of MergeBO against BOPS-H as determined by a binomial sign test ($p < 0.05$, corresponding to more than 15 wins of 20 trials).

Problem	**Simple final Regret**				**Regret Wins**		
	Merge	Mallows	Random	TuRBO	Merge Wins	Ties	Mallows Wins
$\text{TSP}_{n=15}$	0.077 ± 0.125	$\underline{0.013 \pm 0.039}$	0.329 ± 0.332	1.213 ± 0.879	1	12	7
$\text{QAP}_{n=15}$	$14.9 \pm 5.5 \times 10^3$	$\underline{8.1 \pm 4.1 \times 10^3}$	$18.1 \pm 3.6 \times 10^3$	$14.2 \pm 5.9 \times 10^3$	1	3	**16**
$\text{FP}_{n=30}$	24.0 ± 9.7	30.1 ± 12.8	35.7 ± 11.2	$\underline{20.5 \pm 8.4}$	10	4	6
$\text{CR}_{n=30}$	$\underline{6.1 \pm 2.2}$	6.1 ± 3.0	52.15 ± 19.0	33.85 ± 15.7	9	3	8
$\text{TTP1}_{n=280}$	$\underline{23.0 \pm 11.3 \times 10^3}$	$88.9 \pm 7.5 \times 10^3$		$54.8 \pm 12.6 \times 10^3$	**50**	0	0
$\text{TTP2}_{n=280}$	$\underline{14.9 \pm 7.2 \times 10^4}$	$56.5 \pm 6.1 \times 10^4$		$36.8 \pm 9.4 \times 10^4$	**50**	0	0
$\text{TTP3}_{n=280}$	$\underline{8.0 \pm 3.2 \times 10^4}$	$28.1 \pm 2.8 \times 10^4$		$19.1 \pm 3.6 \times 10^4$	**50**	0	0
Problem	**Best so far AUC**				**AUC Wins**		
	Merge	Mallows	Random	TuRBO	Merge Wins	Ties	Mallows Wins
$\text{TSP}_{n=15}$	527.6 ± 162.8	$\underline{428.2 \pm 121.9}$	559.7 ± 224.9	877.2 ± 352.4	5	0	**15**
$\text{QAP}_{n=15}$	$38.3 \pm 8.4 \times 10^5$	$\underline{27.5 \pm 7.4 \times 10^5}$	$42.5 \pm 6.1 \times 10^5$	$46.7 \pm 9.2 \times 10^5$	1	2	**17**
$\text{FP}_{n=30}$	8097.5 ± 2163.7	8665.7 ± 2638.8	9481.0 ± 2086.2	$\underline{5932.1 \pm 1636.9}$	10	1	9
$\text{CR}_{n=30}$	5495.6 ± 687.7	$\underline{5350.5 \pm 910.1}$	13970.8 ± 2408.8	10340.5 ± 2477.3	8	0	12
$\text{TTP1}_{n=280}$	$\underline{20.5 \pm 4.4 \times 10^5}$	$48.5 \pm 3.1 \times 10^5$		$40.0 \pm 4.9 \times 10^5$	**50**	0	0
$\text{TTP2}_{n=280}$	$\underline{12.3 \pm 3.2 \times 10^6}$	$30.5 \pm 2.4 \times 10^6$		$25.9 \pm 3.8 \times 10^6$	**50**	0	0
$\text{TTP3}_{n=280}$	$\underline{6.7 \pm 1.4 \times 10^6}$	$15.2 \pm 1.3 \times 10^6$		$13.2 \pm 1.5 \times 10^6$	**50**	0	0

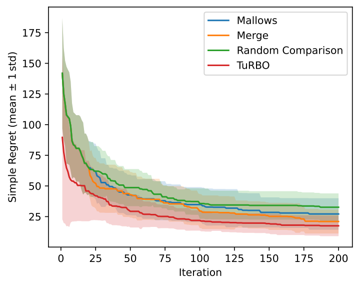
It is important to note that this study does not include a comparison of wall-clock computation times. This is a deliberate choice grounded in the fundamental premise of BO, where the cost of



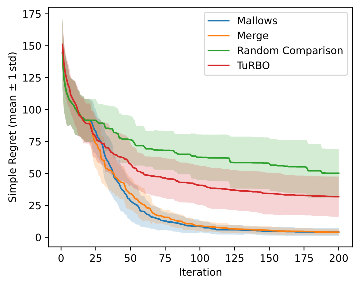
(a) TSP (n=15)



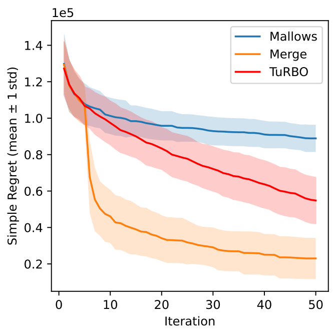
(b) QAP (n=15)



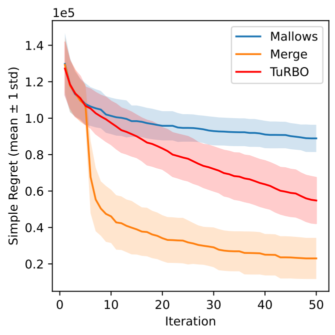
(c) Floor planning (n=30)



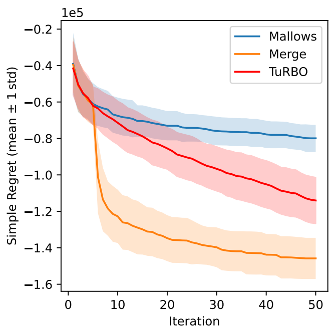
(d) Cell placement (n=30)



(e) TTP (1)



(f) TTP (2)



(g) TTP (3)

Figure 1: Low- and high-dimensional results comparing Mallows kernel (BOPS-H) and Merge kernel (MergeBO) on the current regret value (difference between best-so-far and optimal value) vs. number of iteration. Solid lines show the average regrets, while the shaded areas denote one standard deviation.

function evaluations (e.g., physical experiments or complex simulations) is assumed to far outweigh the computational cost of the algorithm itself. Consequently, our analysis prioritizes metrics related to sample efficiency, which is the primary bottleneck in such real-world scenarios. As a more stable and implementation-agnostic proxy for computational complexity, we instead report the feature vector dimensions generated by each kernel in Table 1, which directly reflects the compactness of the learned representations. Furthermore, as our experiments were conducted on a shared high-performance computing (HPC) cluster, reported wall-clock times would be subject to scheduler-induced variability, making them an unreliable metric for rigorous algorithmic comparison.

Instead, we can report a rough time estimation based on local, small-scale experiments here: the Merge kernel is approximately 10% slower than the Mallows kernel in low-dimensional problems. This is because the Mallows kernel's calculation relies on two clean for-loops, whereas the Merge kernel requires recursive calls to merge sort, which involves significant constant overhead from function calls, Python list slicing, and deepcopy operations. However, in high-dimensional problems, we believe the kernel value computation bottleneck arising from the $O(n^2)$ feature space will lead to significant performance degradation.

4.2 Results and Discussion

Our experimental results, as presented in Table 2 and Figure 1, systematically reveal a strong correlation between the performance advantage of the Merge kernel and the problem's dimensionality. For low-dimensional permutation problems (TSP and QAP), the Mallows kernel, which is specifically designed for such tasks, exhibited a slight performance advantage. However, on problems of intermediate dimensionality (FP and CR), the performance of the two kernels was comparable, with no statistically significant difference observed. This trend shifted decisively on the higher-dimensional TTP problems, where the Merge kernel demonstrated definitive superiority. The statistical data in Table 2 indicates that the Merge kernel significantly outperformed the Mallows kernel across all TTP instances on both the final regret and AUC metrics. Furthermore, the convergence curves in Figure 1 confirm its substantially faster convergence speed. This validates the superior performance and scalability of our proposed method on complex, high-dimensional optimization problems.

The consistent underperformance of the randomized baseline confirms that unstructured feature selection fails to capture necessary permutation similarities. This validates that our method's efficiency stems from preserving principled structural information, rather than mere dimensionality reduction. Regarding TuRBO, while it demonstrates scalability by outperforming BOPS-H in high-dimensional tasks, it remains significantly inferior to our method. This substantial gap underscores the necessity of our specialized permutation optimization framework over generic continuous relaxation strategies and general high-dimensional optimization approaches.

These results corroborate our core hypothesis: the Merge kernel possesses an inherent advantage in high-dimensional permutation optimization problems, owing to its more compact structural design. The disparity between its $O(n \log n)$ feature complexity and the Mallows kernel's $O(n^2)$ complexity widens dramatically as the dimensionality n increases. This is explicitly quantified by the feature length comparison in Table 1: for low, intermediate, and high-dimensional problems, the feature dimensionality of the Mallows kernel is approximately 2, 3.5, and 19.5 times that of the Merge kernel, respectively. These results indicate that the experimental performance is governed by a trade-off between two key factors: (1) the ability to capture global information via the distance metric, and (2) the search efficiency driven by the compactness of the feature space. Due to its right-invariance property, the Mallows kernel possesses a stronger distance metric capability than the Merge kernel. However, as dimensionality increases, the vast disparity in feature vector length leads to a more pronounced space-compression effect. The resulting gains in search efficiency begin to outweigh the performance benefits afforded by the superior distance metric. Consequently, as the problem dimensionality continues to grow, the performance of the Merge kernel ultimately surpasses that of the Mallows kernel by a significant margin. This naturally suggests a dimension-dependent heuristic for practitioners: leveraging the Mallows kernel's robust, right-invariant distance metric for low-dimensional tasks, while switching to the Merge kernel to capitalize on its superior scalability in high-dimensional regimes.

5 Conclusions

In this work, we proposed a novel kernel construction framework for permutation spaces by leveraging sorting algorithms as structured comparison schemes. Within this framework, we introduced the Merge kernel—an efficient, compact, and theoretically grounded alternative to the quadratic Mallows kernel. We showed that the Merge kernel achieves the information-theoretic lower bound on feature complexity while preserving meaningful structural information. This contribution bridges sorting theory and kernel design, revealing a fundamental trade-off between a model's structural invariance and the compactness of its feature space.

Empirical experiments confirmed this trade-off: while the state-of-the-art BOPS-H algorithm held a marginal advantage in low-dimensional problems, their performances were comparable on intermediate-dimensional tasks. In high-dimensional settings, however, MergeBO's compact representation enabled far superior search efficiency, allowing it to significantly outperform BOPS-H and TuRBO. This work opens exciting possibilities for scaling Bayesian optimization to larger permutation spaces. Future directions include exploring other sorting algorithms with constant comparison counts or constructing stabilized sorting algorithm of $O(n \log n)$ complexity for kernel design, and applying this framework to challenging real-world domains like combinatorial neural architecture search and computational biology.

REFERENCES

Sergey Alatartsev, Sebastian Stellmacher, and Frank Ortmeier. Robotic task sequencing problem: A survey. *Journal of intelligent & robotic systems*, 80(2):279–298, 2015.

Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33:21524–21538, 2020.

Nathalie Bartoli, Thierry Lefebvre, Rémi Lafage, Paul Saves, Youssef Diouane, Joseph Morlier, Jasper Bussemaker, Giuseppa Donelli, Joao Marcos Gomes de Mello, Massimo Mandorino, et al. Multi-objective bayesian optimization with mixed-categorical design variables for expensive-to-evaluate aeronautical applications. *arXiv preprint arXiv:2504.09930*, 2025.

Kenneth E Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pp. 307–314, 1968.

Tom Blau, Edwin V Bonilla, Iadine Chades, and Amir Dezfouli. Optimizing sequential experimental design with deep reinforcement learning. In *International conference on machine learning*, pp. 2107–2128. PMLR, 2022.

Mohammad Reza Bonyadi, Zbigniew Michalewicz, and Luigi Barone. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *2013 IEEE congress on evolutionary computation*, pp. 1037–1044. IEEE, 2013.

Rainer E Burkard, Stefan E Karisch, and Franz Rendl. Qaplib–a quadratic assignment problem library. *Journal of Global optimization*, 10:391–403, 1997.

Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE international symposium on workload characterization (IISWC)*, pp. 44–54. Ieee, 2009.

Aryan Deshwal, Syrine Belakaria, Janardhan Rao Doppa, and Dae Hyun Kim. Bayesian optimization over permutation spaces. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 6515–6523, 2022.

Persi Diaconis and Ronald L Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 39(2):262–268, 1977.

E Knuth Donald et al. The art of computer programming. *Sorting and searching*, 3(426-458):4, 1999.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in neural information processing systems*, 31, 2018.

Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes. *Neurocomputing*, 380:20–35, 2020.

Joan Gonzalvez, Edmond Lezmi, Thierry Roncalli, and Jiali Xu. Financial applications of gaussian processes and bayesian optimization. *arXiv preprint arXiv:1903.04841*, 2019.

Stewart Greenhill, Santu Rana, Sunil Gupta, Pratibha Vellanki, and Svetha Venkatesh. Bayesian optimization for adaptive experimental design: A review. *IEEE access*, 8:13937–13948, 2020.

Mara Guidi, Peter H Seeberger, and Kerry Gilmore. How to approach flow chemistry. *Chemical Society Reviews*, 49(24):8910–8932, 2020.

Abhishek Gupta, Yew-Soon Ong, and Liang Feng. Multifactorial evolution: Toward evolutionary multitasking. *IEEE Transactions on Evolutionary Computation*, 20(3):343–357, 2015.

Yunlong Jiao and Jean-Philippe Vert. The kendall and mallows kernels for permutations. In *International Conference on Machine Learning*, pp. 1935–1944. PMLR, 2015.

Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.

Dennis KJ Lin and Nicholas Rios. Order-of-addition experiments: A review and some recommendations. *Wiley Interdisciplinary Reviews: Computational Statistics*, 17(2):e70024, 2025.

Man Luo, Zikai Xie, Huirong Li, Baicheng Zhang, Jiaqi Cao, Yan Huang, Hang Qu, Qing Zhu, Linjiang Chen, Jun Jiang, et al. Physics-informed, dual-objective optimization of high-entropy-alloy nanozymes by a robotic ai chemist. *Matter*, 8(4), 2025.

James Mercer. Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209(441-458):415–446, 1909.

Dang Nguyen, Sunil Gupta, Santu Rana, Alistair Shilton, and Svetha Venkatesh. Bayesian optimization for categorical and category-specific continuous inputs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5256–5263, 2020.

Changyong Oh, Jakub Tomczak, Efstratios Gavves, and Max Welling. Combinatorial bayesian optimization using the graph cartesian product. *Advances in Neural Information Processing Systems*, 32, 2019.

Changyong Oh, Roberto Bondesan, Efstratios Gavves, and Max Welling. Batch bayesian optimization on permutations using the acquisition weighted kernel. *Advances in Neural Information Processing Systems*, 35:6843–6858, 2022.

Sergey Polyakovskiy, Mohammad Reza Bonyadi, Markus Wagner, Zbigniew Michalewicz, and Frank Neumann. A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, pp. 477–484, 2014.

Gerhard Reinelt. Tsplib95. *Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg*, 338:1–16, 1995.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2015.

Ron Sun and C Lee Giles. Sequence learning: From recognition and prediction to sequential decision making. *IEEE Intelligent Systems*, 16(4):67–70, 2001.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando De Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

Muning Wen, Runji Lin, Hanjing Wang, Yaodong Yang, Ying Wen, Luo Mai, Jun Wang, Haifeng Zhang, and Weinan Zhang. Large sequence models for sequential decision-making: a survey. *Frontiers of Computer Science*, 17(6):176349, 2023.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.

Zikai Xie, Xenophon Evangelopoulos, Joseph CR Thacker, and Andrew I Cooper. Domain knowledge injection in bayesian search for new materials. In *ECAI 2023*, pp. 2768–2775. IOS Press, 2023.

Martin Zaefferer, Jörg Stork, and Thomas Bartz-Beielstein. Distance measures for permutations in combinatorial efficient global optimization. In *International Conference on Parallel Problem Solving from Nature*, pp. 373–383. Springer, 2014.

A The Use of Large Language Models

We declare that the large language models (LLMs) ChatGPT and Gemini are only used to aid and polish writing. No further applications of LLMs are used in this research, including but not limited to retrieval, research ideation and experiment designing.

B Extra Experiment Results

B.1 Discussion on Spearman's Footrule

To evaluate the trade-off between permutation information, dimension reduction and right-invariance, we employ Spearman's footruleDiaconis & Graham (1977) distance as another benchmark baseline. Similar to Euclidean measures on raw coordinates, Spearman's footrule operates directly on the permutation group S_n. Formally, for two permutations (rankings) σ and π of n elements, the distance is defined as the sum of the absolute differences between the ranks of each element:

$$d_{\text{footrule}}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{n} |r_i - r'_i| \tag{6}$$

Table 3: Performance comparison between MergeBO, BOPS-H (Mallows kernel), BOPS-H with random comparisons, TuRBO and Spearman's footrule. Underlined results indicate the best numerical results in terms of mean value ± standard deviation of all trials.

Problem	**Simple final Regret**				
	Merge	Mallows	Random	TuRBO	Spearman
$\text{TSP}_{n=15}$	0.077 ± 0.125	$\underline{0.013 \pm 0.039}$	0.329 ± 0.332	1.213 ± 0.879	0.026 ± 0.052
$\text{QAP}_{n=15}$	$14.9 \pm 5.5 \times 10^3$	$\underline{8.1 \pm 4.1 \times 10^3}$	$18.1 \pm 3.6 \times 10^3$	$14.2 \pm 5.9 \times 10^3$	$10.2 \pm 4.7 \times 10^3$
$\text{FP}_{n=30}$	24.0 ± 9.7	30.1 ± 12.8	35.7 ± 11.2	$\underline{20.5 \pm 8.4}$	34.6 ± 9.0
$\text{CR}_{n=30}$	6.1 ± 2.2	6.1 ± 3.0	52.15 ± 19.0	33.85 ± 15.7	$\underline{1.5 \pm 2.4}$
$\text{TTP1}_{n=280}$	$\underline{23.0 \pm 11.3 \times 10^3}$	$88.9 \pm 7.5 \times 10^3$		$54.8 \pm 12.6 \times 10^3$	$88.7 \pm 10.1 \times 10^3$
$\text{TTP2}_{n=280}$	$\underline{14.9 \pm 7.2 \times 10^4}$	$56.5 \pm 6.1 \times 10^4$		$36.8 \pm 9.4 \times 10^4$	$56.2 \pm 6.2 \times 10^4$
$\text{TTP3}_{n=280}$	$\underline{8.0 \pm 3.2 \times 10^4}$	$28.1 \pm 2.8 \times 10^4$		$19.1 \pm 3.6 \times 10^4$	$28.4 \pm 2.5 \times 10^4$

Problem	**Best so far AUC**				
	Merge	Mallows	Random	TuRBO	Spearman
$\text{TSP}_{n=15}$	527.6 ± 162.8	428.2 ± 121.9	559.7 ± 224.9	877.2 ± 352.4	$\underline{397.1 \pm 98.0}$
$\text{QAP}_{n=15}$	$38.3 \pm 8.4 \times 10^5$	$\underline{27.5 \pm 7.4 \times 10^5}$	$42.5 \pm 6.1 \times 10^5$	$46.7 \pm 9.2 \times 10^5$	$31.8 \pm 8.5 \times 10^5$
$\text{FP}_{n=30}$	8097.5 ± 2163.7	8665.7 ± 2638.8	9481.0 ± 2086.2	$\underline{5932.1 \pm 1636.9}$	9187.7 ± 2024.1
$\text{CR}_{n=30}$	5495.6 ± 687.7	5350.5 ± 910.1	13970.8 ± 2408.8	10340.5 ± 2477.3	$\underline{4673.4 \pm 1020.15}$
$\text{TTP1}_{n=280}$	$\underline{20.5 \pm 4.4 \times 10^5}$	$48.5 \pm 3.1 \times 10^5$		$40.0 \pm 4.9 \times 10^5$	$48.2 \pm 4.3 \times 10^5$
$\text{TTP2}_{n=280}$	$\underline{12.3 \pm 3.2 \times 10^6}$	$30.5 \pm 2.4 \times 10^6$		$25.9 \pm 3.8 \times 10^6$	$30.3 \pm 2.4 \times 10^6$
$\text{TTP3}_{n=280}$	$\underline{6.7 \pm 1.4 \times 10^6}$	$15.2 \pm 1.3 \times 10^6$		$13.2 \pm 1.5 \times 10^6$	$15.3 \pm 9.4 \times 10^6$

Obviously, the corresponding featurization method $\Phi_{footrule}(\pi)$ is an identical mapping that directly uses the permutation π as its feature vector. The computational complexity of Spearman's footrule is $O(n)$, and it is fully right-invariant since it simply calculates the L_1 distance between two vectors.

However, while this mapping retains the raw rank values, it treats permutations merely as vectors in a Euclidean space, thereby ignoring the underlying algebraic structure of the symmetric group. Unlike the Mallows kernel or our proposed Merge kernel, which embed specific probabilistic or hierarchical priors, this naive representation doesn't project the permutation to a compact manifold and fails to capture the compact dependencies within the feasible space. Nevertheless, its L_1 nature allows it to effectively approximate local structural discrepancies through simple summation.

The experiment results of Spearman's footrule are added in Table 3 above. In low-dimensional settings, Spearman's footrule performs slightly worse than the Mallows kernel but marginally better than the Merge kernel. This can be attributed to the high relevance of right-invariance in lower dimensions, where accurately measuring the similarity between local regions—which often share similar performance characteristics—is critical. Notably, the method exhibits rapid convergence on TSP and CR tasks, stemming from the inherent local additivity of these problems. However, Spearman's footrule offers lower discriminative resolution for permutations compared to the Mallows kernel. This coarser granularity tends to bias the search towards exploitation rather than exploration; while this enables quick convergence to local optima, it may limit the model's ability to escape them for a global solution.

Conversely, in high-dimensional problems, Spearman's footrule performs comparably to the Mallows kernel but significantly lags behind the Merge kernel. This stark contrast highlights the Merge kernel's superior capability to compress high-dimensional search spaces and accelerate optimization through hierarchical decomposition.