中国科学技术大学
University of Science and Technology of China

From Sorting Algorithm to Scalable Kernels on Permutation Space

Authors: Zikai Xie, Linjiang Chen

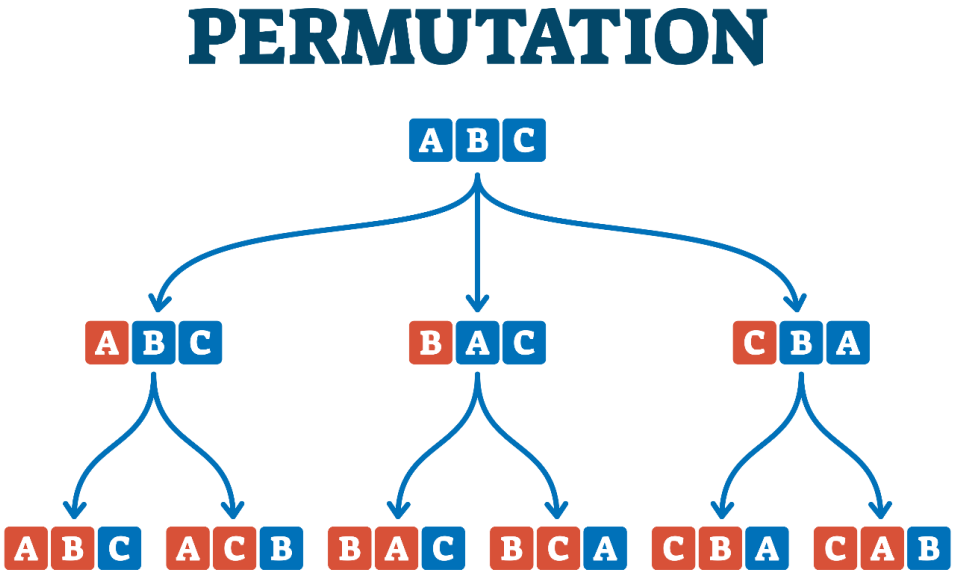
State Key Laboratory of Precision and Intelligent Chemistry, USTC

BO on Permutation Space

Permutation:
A sequence of operations where each operation is performed exactly once.

Examples:
The order of adding each seasoning when cooking, the order of visiting attractions during a trip, etc.

Characteristics:
A fixed set of operations,
each operation must be selected
exactly once, and the order varies.



BO on Permutation Space

Bayesian Optimization:
A sequential optimization algorithm that uses surrogate models (typically a Gaussian process) to fit experiment data.

Gaussian Process:
A distribution over functions, where any finite collection of function values follows a multivariate Gaussian distribution.
Defined by a prior mean function (usually constant 0 after normalization) and a kernel function to measure similarities between inputs.

BO on Permutation Space:
To use BO on permutation space, we need to design a kernel function over permutations space to make GP work on this space.

BO on Permutation Space

SOTA: Mallows Kernel

$$K_{Mal}(\pi, \pi') = \exp(-ld(\pi, \pi'))$$

Where d(π, π′) is the Kendall-tau distance between two permutations：

$$d(\pi, \pi') = \sum_{i<j} \left[1_{\pi(i)>\pi(j)} 1_{\pi'(i)<\pi'(j)} + 1_{\pi(i)<\pi(j)} 1_{\pi'(i)>\pi'(j)}\right]$$

In other word, Kendall-tau measures, for all two-element pairs in two permutations, whether their relative ordering is consistent. For example：

π = (1, 2, 3, 4), π′ = (2, 1, 4, 3).

2 of the 6 two-element pairs are inconsistent： (1, 2), (3, 4).

Therefore d(π, π′) = 2， K_mal(π, π′) = exp(−2l).

BO on Permutation Space

$$d(\pi, \pi') = \sum_{i<j} \left[1_{\pi(i)>\pi(j)} 1_{\pi'(i)<\pi'(j)} + 1_{\pi(i)<\pi(j)} 1_{\pi'(i)>\pi'(j)} \right]$$

Kendall-tau measures, for all two-element pairs in two permutations, whether their relative ordering is consistent.

We can observe that this measurement is related to sorting algorithms:

Is relative ordering consistent? ⇔ Do they need to swap during sorting?

Therefore, Kendall-tau distance can be equivalently seen as checking the swap status during enumerate sort:

π = (1, 2, 3, 4): (0, 0, 0, 0, 0, 0) ([1,2],[1,3],[1,4],[2,3],[2,4],[3,4])

π′= (2, 1, 4, 3): (1, 0, 0, 0, 0, 1) ([2,1],[2,4],[2,3],[1,4],[1,3],[4,3])

BO on Permutation Space

Algorithm:

For a given sorting algorithm, when transforming a permutation into the canonical order $[1, 2, 3, \ldots, n]$, the algorithm performs a sequence of pairwise comparisons. Each comparison determines whether to swap the two elements based on their relative order.

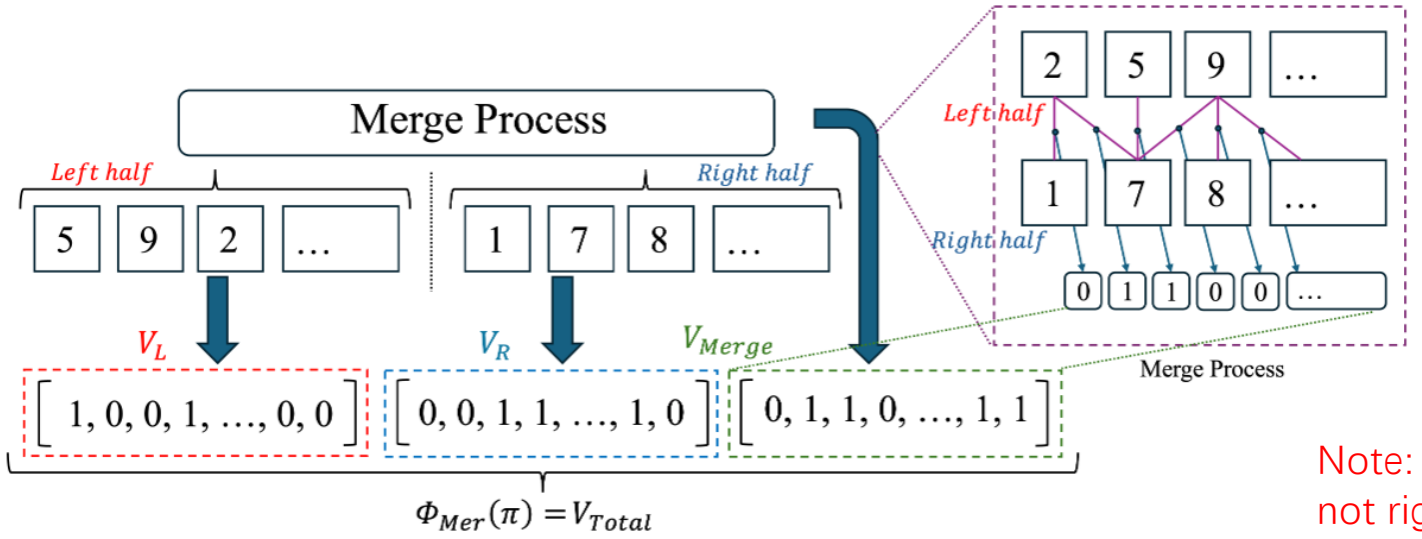
By recording the outcomes of these comparisons (i.e., whether a swap occurs), we can construct a feature vector for the permutation (e.g., [0,1,1,0, ⋯, 1]).

Applying an RBF kernel to this feature vector yields a kernel function over the permutation space (PSD).
A necessary condition for a sorting algorithm to be applicable in this framework is that the number and positions of compared element pairs are fixed.

The state-of-the-art Mallows kernel can be viewed as a special case of this framework using *Enumerate Sort*.

Our proposed method corresponds to applying this framework with *Merge Sort*, achieving the lower bound in sorting complexity while also attaining the minimal information encoding in the feature representation.

BO on Permutation Space



Note: our kernel is not right-invariant

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 $\Phi_{Mer}(\pi) = V_L \oplus V_R \oplus V_{Merge} = [0] \oplus [1] \oplus [0, 1, 1, 1] = [0, 1, 0, 1, 1, 1]$.

Experiments

Experiments：

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

Low-dim:

- **Quadratic Assignment (QAP, n=15)**
- **Travelling Salesman (TSP, n=15)**
- **Floor Planning (FP, n=30)**
- **Cell Placement (CP, n=30)**

High-dim: **Travelling Thief Problems (TTP, n=280)**

Experiment Results

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

Conclusion

Our Contributions:

- We propose a general framework for constructing kernel functions over permutation spaces based on sorting algorithms.
- Not all sorting algorithms can be used to construct valid kernels; they must satisfy the property of having a fixed comparison map.
- Under this framework, a previously state-of-the-art method, Kendall-tau distance, can be viewed as a special case: *Enumerate Sort*.
- Based on this framework, we identify a kernel that achieves the lower bound of information complexity by leveraging *Merge Sort*.
- Experiments show that higher feature dimensionality leads to better performance, revealing a trade-off between right invariance and dimensionality reduction.