Solving the Single Row Facility Layout Problem by K-Mebdoids Memetic Permutation Group

Lixin Tang, *Senior Member, IEEE,* Zuocheng Li, and Jin-Kao Hao*

***Abstract*— The single row facility layout problem (SRFLP) is concerned with arranging facilities along a straight line so as to minimize the sum of the products of the flow costs and distances between all facility pairs. SRFLP has rich practical applications and is however NP-hard. In this paper, we first investigate a dedicated symmetry-breaking approach based on permutation group theory for reducing the solution space of SRFLP. Relevant symmetry properties are identified through the alternating group of the original solution space or the corresponding coordinate rotation space. Then, a memetic algorithm is proposed to explore promising search regions regarding the reduced solution space. The memetic algorithm employs a problem-specific crossover operator guided by *k*-medoids clustering technique to produce meaningful offspring solutions. The algorithm additionally uses a simulated annealing procedure to intensively exploit a given search region and a distance-and-quality based population management strategy to ensure a reasonable diversity of the population. Experimental results on commonly used benchmark instances and newly introduced large-scale instances with sizes up to 2000 facilities show that the proposed algorithm competes favorably with state-of-the-art SRFLP algorithms. It attains all but one previous best known upper bounds and discovers new upper bounds for 33 instances out of the 93 popular benchmark instances.**

***Index Terms*—Single row facility layout, permutation group, memetic search, *k*-medoids clustering, simulated annealing.**

I. Introduction

THE SINGLE row facility layout problem (SRFLP) is a popular combinatorial optimization problem. It consists of arranging facilities along a straight line while minimizing the sum of the products of the flow costs and distances between all pairs of facilities.

Formally, let $\mathcal{N}=\{1,...,N\}$ be the set of given facilities, Π_N the set of all permutations of facilities in $\mathcal{N}$ (the cardinality is $N!$), $c_{ij}=c_{ji}$ the flow cost between facilities $i \in \mathcal{N}$ and $j \in \mathcal{N}$, and ℓ_i the length of facility i. SRFLP can be formulated as follows [1]–[4]:

$$\min_{\boldsymbol{\pi}\in\Pi_N} f(\boldsymbol{\pi}) = \sum\nolimits_{i,j\in\mathcal{N},i<j} c_{\pi_i,\pi_j} d_{\pi_i,\pi_j}, \tag{1}$$

where d_{π_i,π_j} is the distance that is defined as follows:

$$d_{\pi_i,\pi_j} = \frac{\ell_{\pi_i}}{2} + \sum\nolimits_{i<k<j} \ell_{\pi_k} + \frac{\ell_{\pi_j}}{2}. \tag{2}$$

SRFLP was first studied as far back as the 1960s [5]. It occurs in numerous practical applications, including generating physical layout in circuit design [6], determining machine sequence in manufacturing systems [7] and sequencing workstations in automated guided vehicle (AGV) systems [1]. SRFLP is known to be NP-hard [8] and so is a computationally challenging research problem.

Given its theoretical importance and application focuses, a number of solution methods have been proposed for solving SRFLP, which can be generally classified as exact algorithms and metaheuristic algorithms. A comprehensive review of solution methods for SRFLP till 2015 can be found in [9]. Due to the NP-hardness of SRFLP, exact algorithms often require a prohibitively expensive time for large-sized instances. The best performing exact algorithms including the branch-and-cut algorithm [10], semidefinite programming algorithms [11], [12] and the cutting plane algorithm [13] may fail to address instances with up to 42 facilities. Even the currently most effective semidefinite programming method [8] still has difficulties for instances with $N>110$. For large-sized instances that cannot be handled by exact algorithms, metaheuristic algorithms can be suitable alternatives to obtain high-quality suboptimal solutions within reasonable computation time.

In recent years, several metaheuristic algorithms have been used to solve SRFLP. In [1], [4], search approaches based on the so-called greedy randomized adaptive search procedure (GRASP), named GRASP-PR and GRASP-$_\mathrm{F}$, were proposed to solve SRFLP with size $N>300$. Palubeckis [2], [3] investigated novel speed-up solution evaluation techniques and proposed a multi-start simulated annealing (MSA) and a variable neighborhood search (VNS-LS3) to solve SRFLP considering many new instances. Samarghandi and Eshghi [14] studied a tabu search where they introduced a speed-up strategy to enhance the efficiency of the algorithm for a special case of SRFLP. Kothari and Ghosh [15] presented a tabu search with pairwise 2-opt and insertion neighborhoods to solve SRFLP. In [16], a genetic algorithm (GA) with problem-specific improvement strategies was proposed to solve the problem. Other metaheuristic algorithms, such as scatter search [17], the cross-entropy algorithm [18], and ant colony optimization [19], were also developed for the problem. These studies motivate our metaheuristic search model for SRFLP.

Despite the effectiveness and efficiency of these reviewed metaheuristic algorithms, few methods can solve large-scale

This research was partially supported by the Major Program of National Natural Science Foundation of China (71790614), the Major International Joint Research Project of the National Natural Science Foundation of China (71520107004) and the 111 Project (B16009). (*Corresponding author*: Jin-Kao Hao)

L. Tang is with Frontier Science Center for Industrial Intelligence and Systems Optimization, Northeastern University, Shenyang 110819, China (e-mail: lixintang@mail.neu.edu.cn).

Z. Li is with Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern University), Ministry of Education, Shenyang 110819, China (e-mail: zuocheng_li@163.com).

J.K. Hao is with LERIA, Université d'Angers, 2 Boulevard Lavoisier, Angers 49045, France (e-mail: jin-kao.hao@univ-angers.fr).

Manuscript received Sept 2021; revised Jan 2022; accepted March 2022.

SRFLP. Only solution approaches of [1]–[4] can find a high-quality suboptimal solution for instances with $300 \leq N \leq 1000$. This can be due to the fact that almost no metaheuristic algorithms have used the inherent symmetric properties of the solution space of SRFLP to improve the search efficiency. Besides, existing algorithms are also short of pertinent models to learn valuable knowledge while identifying building blocks in discrete search spaces. These motivate us to consider the symmetric features of the solution space of SRFLP and so devise a learning based metaheuristic algorithm able to solve large-scale problem.

In this paper, we investigate for the first time a dedicated symmetry-breaking method based on permutation group theory for reducing the solution space of SRFLP. Given the characteristics of the reduced solution space, a *k*-me doids memetic permutation group algorithm (KMPG) is designed for SRFLP. Like most memetic algorithms, the solution quality of KMPG also depends on a suitable local search procedure. However, no efficient incremental evaluation technique is known to assess the quality of candidate solutions. In this case, solution methods like tabu search and variable neighborhood search are unsuitable since the time of each iteration will be too expensive. Simulated annealing has been successful to a number of related facility layout problems [20]–[22]. The simplicity of simulated annealing makes it an interesting local search tool in our case because the search process is guided only by a simple annealing function. Thus, in KMPG we embed a simulated annealing based local search procedure. The contributions of this work are summarized as follows.

First, from a perspective of solution approach, the proposed KMPG algorithm considers the symmetric features of the solution space of SRFLP, aiming to explore more promising search regions with less effort. For this, a dedicated symmetry-breaking strategy based on the permutation group theory is presented. Indeed, for SRFLP, although previous studies offer various theoretical analysis on solution structures, we are not aware of any effective symmetry-breaking approach based on inherent mathematical properties for reducing the solution space. In particular, the KMPG algorithm combines a problem-specific crossover operator guided by the *k*-medoids clustering technique to produce promising offspring solutions, a symmetry-breaking-based simulated annealing procedure to perform local optimization, and a distance-quality-based population management strategy to strengthen the diversity of the population. These search components work together to attain a suitable balance between the global exploration and the local exploitation. We note that, this is the first approach that employs machine learning mechanism and strict mathematical method to guide the search behavior of the algorithm.

Second, from a perspective of computational performance and resulting advances, we provide comparisons of the KMPG algorithm and the most effective state-of-the-art SRFLP algorithms of [1]–[4] on traditional benchmark instances and newly introduced large-scale instances. Especially, we report new upper bounds for 33 out of the 93 traditional benchmark instances. To the best of our knowledge, this is the first solution method achieving such a performance regarding the popular benchmark instances. Moreover, we make the 20 new instances (with $1050 \leq N \leq 2000$) publicly available and present for the first time upper bounds on them, which would be valuable for future research on SRFLP. Note that existing method s have been only used to handle instances with $N \leq 1000$. This indicates that the KMPG algorithm is capable of offering a good performance for large-scale SRFLP instances. Given that SRFLP has a number of real-life applications, the proposed KMPG algorithm will be particularly useful if large-sized practical cases are considered.

Third, the idea of the proposed symmetry-breaking method is of generic nature and could be advantageously adopted by other SRFLP algorithms. Considering the problem-specific features of solved problems, the search components based on the permutation group and the *k*-medoids clustering of this work could inspire effective search approaches for other permutation problems. More generally, since any finite abstract group is isomorphic to a permutation group, the proposed symmetry-breaking approach can be used to identify meaningful symmetric features for other combinatorial optimization problems. However, the symmetric features is only one of several solution features for combinatorial optimization problems, so the case of the permutation group in this work could inspire useful methods for finding other types of solution properties. Furthermore, there may be certain kinds of symmetric structures in some optimization and machine learning methods. Thus, the philosophy of the permutation group in this work can be potentially used to discover useful symmetric information for relevant optimization and machine learning methods, for which few studies exist in the literature.

The rest of this paper is organized as follows. Section II introduces necessary preliminaries. Section III presents the KMPG algorithm. Experimental results and comparisons are reported in Section IV. In Section V, we draw concluding comments. The proofs of proposed theorems are given in the Appendix and some of experimental results are provided as supplementary files.

II. Preliminaries

In KMPG, we use a permutation of N facilities to represent a candidate solution. There is thus a “one-to-one” mapping between the search space of KMPG and the solution space of SRFLP. Below, we first perform an analysis of SRFLP in terms of the permutation group theory and then introduce the proposed solution properties and symmetry-breaking approach.

A. Permutation Group and Single Row Facility Layout

Mathematically, any solution of KMPG with N facilities is an element of a certain permutation group formed by a subset of Π_N. Meanwhile, all permutations in Π_N form a symmetric group of degree N and order $N!$. That is, the search space of KMPG is composed of the entire permutations of the related symmetric group. Especially, the population of KMPG at each generation is a permutation group of the symmetric group. As such, the permutation group theory is, in nature, suitable to analyze and understand the search space of KMPG. Thereby, we can discover useful solution properties of KMPG. Note that in this work the symmetric group is also denoted by Π_N.

Indeed, research on permutation theory started from the seminal work of Cayley in 1849 [23]. To present the solution properties of SRFLP, the following well-known definitions of the permutation group theory will be used [24].

Definition 1: Given a permutation formed by a product of transpositions, if there is an odd number of transpositions, then it is an *odd permutation* . Otherwise, it is an *even permutation*.

Note that we can calculate the number of transpositions based on cycle form. For example, a cycle form of $\boldsymbol{\pi} = [5, 7, 6, 1, 4, 3, 2]$ is shown in Fig. 1. Then, it has a product of 4 transpositions $(1, 5)(5, 4)(2, 7)(3, 6)$ and so is an even permutation. The number of transpositions may not be unique, but the sign of the permutation is either always even (i.e., sign "1") or always odd (i.e., sign "-1"). The time complexity of determining the sign is $O(N)$. In Section II-B, the signs will be used as key indicators in analyzing the symmetric solution properties of KMPG, as well as in guiding the dedicated crossover and local search operators in Section III.

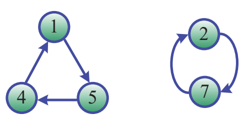


Fig. 1. Illustration of a cycle form of permutation ($\boldsymbol{\pi} = [5, 7, 6, 1, 4, 3, 2]$).

Definition 2: Given a symmetric group Π_N, all even permutations form an *alternating group* A_N that has $N!/2$ elements. It is worth noting that A_N is a subgroup of Π_N with the identity $[1, ..., N]$. Take an SRFLP instance with 3 facilities for example, and we illustrate the relationship between Π_N and A_N in Fig. 2. Apparently, the alternating group can inspire an idea of considering either sign "1" or sign"-1", reducing thus the search space of KMPG while getting rid of any information loss. In Section II-B, we will propose the symmetric solution properties of KMPG by employing the alternating group.

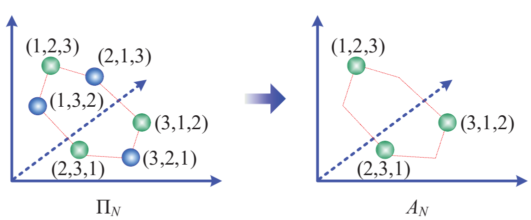


Fig. 2. Symmetric group and alternating group (for SRFLP with 3 facilities).

Definition 3: Given two solutions $\boldsymbol{\pi}^{\boldsymbol{a}}$ and $\boldsymbol{\pi}^{\boldsymbol{b}}$, the distance $dist(\boldsymbol{\pi}^{\boldsymbol{a}}, \boldsymbol{\pi}^{\boldsymbol{b}})$ is the number of transpositions for transforming one solution into another. If $dist(\boldsymbol{\pi}^{\boldsymbol{a}}, \boldsymbol{\pi}^{\boldsymbol{b}})$ is an odd or even number, then they have an *odd distance* or *even distance*.

For example, $\boldsymbol{\pi}^{\boldsymbol{a}} = [5, 7, 6, 1, 4, 3, 2]$ can be transformed to $\boldsymbol{\pi}^{\boldsymbol{b}} = [5, 4, 7, 6, 1, 3, 2]$ according to a product of transpositions $(5, 4)(4, 3)(3, 2)$, giving $dist(\boldsymbol{\pi}^{\boldsymbol{a}}, \boldsymbol{\pi}^{\boldsymbol{b}}) = 3$. Often, the distance is not unique, but it is always odd or even. Since there is no intuitive distance between two permutations, such a distance provides an alternative metric to assess the relationships among permutations. In Section II-C, the distance is used to measure the relationship between two neighborhood solutions in the proposed symmetry-breaking approach.

B. Proposed Solution Properties of KMPG

To analyze the features of the search space of KMPG, we first present the sign properties of solutions in Π_N. Let $\boldsymbol{\alpha} = inverse(\boldsymbol{\pi}) = [\pi_N, \pi_{N-1}, ..., \pi_1]$ be the inverse form of $\boldsymbol{\pi} \in \Pi_N$. We have the following Theorem 1.

Theorem 1: Suppose that $\boldsymbol{\pi}$ is an even permutation. Then $\boldsymbol{\alpha}$ is an odd permutation if and only if $N \in \{\mathcal{K} | (\mathcal{K} = 4k + 2) \vee (\mathcal{K} = 4k + 3), \forall k \in \mathbb{N}\}$.

As Theorem 1 shows, even if a solution of KMPG and its inverse form may have different permutation signs, they have the same objective value. In this case, one can attempt to find useful symmetric features of the search space of KMPG based on permutation signs. Take the even $\boldsymbol{\pi} = [5, 7, 6, 1, 4, 3, 2]$ in Fig. 1 for example, and $\boldsymbol{\alpha} = inverse(\boldsymbol{\pi}) = [2, 3, 4, 1, 6, 7, 5]$ with a product of 5 transpositions $(1, 2)(2, 3)(3, 4)(5, 6)(6, 7)$ is an odd permutation. Obviously, we have $f(\boldsymbol{\pi}) = f(\boldsymbol{\alpha})$ according to Eq. (1), indicating thus a basic symmetric feature associated with permutation signs.

Without loss of generality, in this work we consider even permutations in the alternating group A_N and have the following Theorem 2.

Theorem 2: Suppose that $\boldsymbol{\pi} \in A_N$ satisfies $N \in \{\mathcal{K} | (\mathcal{K} = 4k + 2) \vee (\mathcal{K} = 4k + 3), \forall k \in \mathbb{N}\}$. Then $\neg \exists \boldsymbol{\alpha} \in A_N : \boldsymbol{\alpha} = inverse(\boldsymbol{\pi})$ holds.

Take the symmetric group for SRFLP in Fig. 2 for example, and the permutations in $A_3 = \{[1, 2, 3], [2, 3, 1], [3, 1, 2]\}$ are not inverse to each other. Nevertheless, each permutation in $\Pi_3 \backslash A_3 = \{[3, 2, 1], [1, 3, 2], [2, 1, 3]\}$ corresponds to a unique permutation in A_3 that has the same objective value. Based on Theorems 1 and 2, we give the solution properties of KMPG in Theorem 3, regarding the condition $N \in \{\mathcal{K} | (\mathcal{K} = 4k + 2) \vee (\mathcal{K} = 4k + 3), \forall k \in \mathbb{N}\}$.

Theorem 3 (Scenario 1): Suppose that $\boldsymbol{\pi} \in A_N$ satisfies $N \in \{\mathcal{K} | (\mathcal{K} = 4k + 2) \vee (\mathcal{K} = 4k + 3), \forall k \in \mathbb{N}\}$. Then $\exists \boldsymbol{\beta} \in \Pi_N \backslash A_N : f(\boldsymbol{\pi}) = f(\boldsymbol{\beta})$ holds.

As Theorem 3 indicates, given that $N \in \{\mathcal{K} | (\mathcal{K} = 4k+2) \vee (\mathcal{K} = 4k+3), \forall k \in \mathbb{N}\}$ is satisfied, there exists a symmetric relationship between even and odd permutations in Π_N in terms of objective function landscape. As such, we can identify each pair of symmetric solutions of the objective function landscape based on the permutation signs, as illustrated in Fig. 3. Accordingly, the search space of KMPG can be reduced by considering only even permutations in the alternating group during the search process. For example, in Fig. 2, there are 6 permutations in Π_3, i.e., $[1, 2, 3]$, $[1, 3, 2]$, $[2, 1, 3]$, $[2, 3, 1]$, $[3, 1, 2]$ and $[3, 2, 1]$, and one can consider only the even permutations $[1, 2, 3]$, $[2, 3, 1]$, $[3, 1, 2]$ while discarding the other 3 permutations during the search process of KMPG.

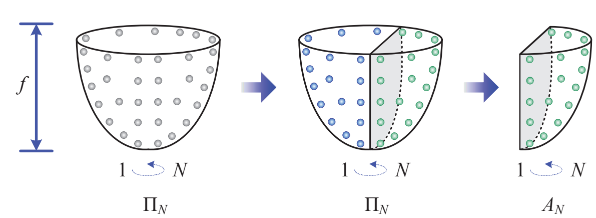


Fig. 3. Illustration of solution properties in Theorem 3 (Scenario 1).

However, Theorem 3 above is not available concerning $N \in \{\mathcal{K}|(\mathcal{K}=4k+4) \vee (\mathcal{K}=4k+5), \forall k \in \mathbb{N}\}$. It can be due to the fact that any permutation and its inverse form have the same sign for such a case (the contrapositive of Theorem 1). As such, it is necessary to perform further analysis on the solution properties of KMPG in addition to the conclusion of Theorem 3. For this, we propose to consider a mapping search space of KMPG from a perspective of the coordinate rotation of the original SRFLP. Let $\mathcal{L}$ be the set of fixed facilities, which is defined as follows:

$$\mathcal{L} = \begin{cases} \{N\} & N \in \{\mathcal{K}|\mathcal{K}=4k+4\} \\ \{N-1, N\} & N \in \{\mathcal{K}|\mathcal{K}=4k+5\} \end{cases}, \forall k \in \mathbb{N}, \quad (3)$$

where $\mathcal{L}$ also denotes the cardinality of the set. Take $\boldsymbol{\pi} = [3,1,4,2,5]$ for example, and we can write $\boldsymbol{\pi}^{\mathcal{L}=2} = [3,1,2]$ where the fixed facilities are 4 and 5.

Let $\Pi_N^{\mathcal{L}}$ be the symmetric group regarding $\mathcal{L}$ fixed facilities out of N facilities, and $A_N^{\mathcal{L}}$ the alternating group identified from $\Pi_N^{\mathcal{L}}$. For the coordinate rotation of the original search space of KMPG, we have the following Theorem 4.

Theorem 4 (Scenario 2): Suppose that $\boldsymbol{\pi}^{\mathcal{L}} \in A_N^{\mathcal{L}}$ satisfies $N \in \{\mathcal{K}|(\mathcal{K}=4k+4) \vee (\mathcal{K}=4k+5), \forall k \in \mathbb{N}\}$. Then $\exists \boldsymbol{\beta}^{\mathcal{L}} \in \Pi_N^{\mathcal{L}} \backslash A_N^{\mathcal{L}} : f(\boldsymbol{\pi}) = f(\boldsymbol{\beta})$ holds.

From Theorem 4, one observes that the symmetric solutions of the objective function landscape can be identified by means of the coordinate rotation of the original search space. Thereby, we can reduce the computational effort of KMPG by restricting the search space to the corresponding alternating group. Theorem 4 (Scenario 2) is illustrated in Fig. 4. It shows that KMPG can be afforded more opportunities to discover promising search regions within the mapping (rotated) search space without any information loss of the original search space. Take SRFLP with 4 facilities for example, and we have $A_4^1 = \{$[4,1,2,3], [4,2,3,1], [4,3,1,2], [1,4,2,3], [2,4,3,1], [3,4,1,2], [1,2,4,3], [2,3,4,1], [3,1,4,2], [1,2,3,4], [2,3,1,4], [3,1,2,4]$\}$. Each permutation in A_4^1 corresponds a unique permutation in $\Pi_4^1 \backslash A_4^1 = \{$[4,3,2,1], [4,1,3,2], [4,2,1,3], [3,4,2,1], [1,4,3,2], [2,4,1,3], [3,2,4,1], [1,3,4,2], [2,1,4,3], [3,2,1,4], [1,3,2,4], [2,1,3,4]$\}$ that has the same objective value. In this case, one can consider only the 12 permutations in A_4^1 while discarding the other 12 permutations in $\Pi_4^1 \backslash A_4^1$ during the search process of KMPG.

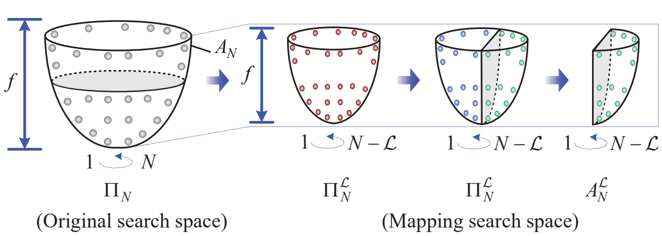


Fig. 4. Illustration of solution properties in Theorem 4 (Scenario 2).

C. Proposed Symmetry-breaking Approach of KMPG

The proposed symmetry-breaking method aims to reduce the search space of KMPG for discovering promising solutions. According to Theorems 3 and 4, we restrict the search space of KMPG to the alternating group of the original or mapping search space. For a permutation $\boldsymbol{\pi}$, the symmetry-breaking method works as follows. The sign of $\boldsymbol{\pi}$ (Scenario 1) or $\boldsymbol{\pi}^{\mathcal{L}}$ (Scenario 2) is checked. If $\boldsymbol{\pi}$ or $\boldsymbol{\pi}^{\mathcal{L}}$ is odd, we set $\boldsymbol{\pi} = inverse(\boldsymbol{\pi})$ to guarantee an even sign of the permutation. The time complexity of the above procedure is $O(N)$.

On the other hand, we need to ensure even signs for the local search operator of KMPG. For permutation problems, representative neighborhood structures include *Insert* and *Interchange* [2], [25], [26]. In KMPG, we adopt the *Insert* neighborhood that shows a good performance for SRFLP [2]. In Section IV, we will conduct discussions on the effectiveness of the two neighborhood structures. The adopted *Insert*-based neighborhood structure concerned with Scenarios 1 and 2 is shown in Fig. 5. We present the symmetry-breaking method for the local search operator of KMPG in Algorithm 1. It shows that the sign of the improved permutation is always even regarding the original or mapping search space of KMPG. As such, the local search operator can efficiently exploit given search regions while having no impact on the consistency of the search process of KMPG. In the worst case, the time complexity of the symmetry-breaking method is $O(N)$.

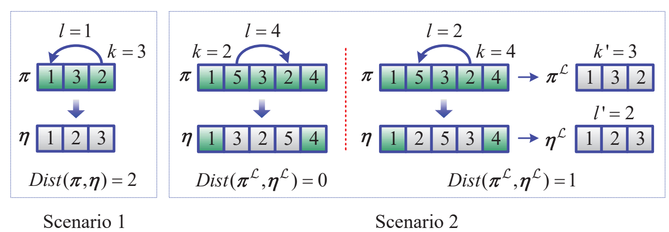


Fig. 5. Adopted *Insert*-based neighborhood structure.

Algorithm 1: Symmetry Breaking for Local Search

Input: $Scenario$, $\boldsymbol{\pi}$ ($\boldsymbol{\pi}$ or $\boldsymbol{\pi}^{\mathcal{L}}$ is even) and $\boldsymbol{\eta} = insert(\boldsymbol{\pi}, k, l)$
Output: updated $\boldsymbol{\pi}$ ($\boldsymbol{\pi}$ or $\boldsymbol{\pi}^{\mathcal{L}}$ is even)

```
switch Scenario do
    case 1 do
        dist(π, η) := |k − l|; break; /*as shown in Fig. 5*/
    case 2 do
        if π_k > N − L then
            dist(π^L, η^L) := 0; /*as shown in Fig. 5*/
        else
            Find positions k' and l' in permutations π^L and η^L;
            dist(π^L, η^L) := |k' − l'|; /*as shown in Fig. 5*/
        break;
if dist(π, η) or dist(π^L, η^L) is odd then
    π ← η; /*if even distance*/
else
    π ← inverse(η); /*if odd distance*/
return π
```

III. K-MEDOIDS MEMETIC PERMUTATION GROUP ALGORITHM FOR SRFLP

In this section, the proposed KMPG algorithm is detailed based on the preliminaries in Section II. In the following, we present the general scheme of the KMPG algorithm and then its search components.

A. General Scheme

Like any population-based evolutionary algorithm (such as [27]–[29]), memetic algorithms use a search model composed of evolutionary algorithms and local search operators to attain a suitable balance of exploration and exploitation [30], [31]. The primary highlight of KMPG consists in the dedicated symmetry-breaking method based on the permutation group for reducing the search space (Section II). Thereby, KMPG integrates complementary search components to discover high-quality solutions, including population initialization (Section III-B), k-medoids clustering based crossover (Section III-C), simulated annealing based local search (Section III-D), and distance-and-quality based population management (Section III-E). Let $genMax$ be the maximum generation of KMPG, ps the population size, $\boldsymbol{\pi}^{\mathcal{P}}(gen) = \{\boldsymbol{\pi}_1^{\mathcal{P}}(gen), ..., \boldsymbol{\pi}_{ps}^{\mathcal{P}}(gen)\}$ the population at generation gen, and $\boldsymbol{gBest}(gen)$ the global best solution found so far at generation gen. The KMPG algorithm for SRFLP is given in Algorithm 2.

Algorithm 2: Main Framework of the KMPG Algorithm

Input: SRFLP instance and parameters of KMPG
Output: $\boldsymbol{gBest}(genMax)$
1 **for** $gen := 0$ **to** $genMax$ **do**
2   **if** $gen = 0$ **then**
3   Initialize $\boldsymbol{\pi}^{\mathcal{P}}(gen)$ wherein each permutation has even sign regrading original or mapping search space; /*Section III-B*/
4   Initialize global best solution $\boldsymbol{gBest}(gen)$;
5   **else**
6   Perform k-medoids clustering based crossover operator to generate an even offspring solution regarding original or mapping search space; /*Section III-C*/
7   Perform simulated annealing based local search to obtain improved even offspring solution with respect to original or mapping search space; /*Section III-D*/
8   Update global best solution $\boldsymbol{gBest}(gen)$;
9   Perform distance-and-quality population management and update $\boldsymbol{\pi}^{\mathcal{P}}(gen)$; /*Section III-E*/
10 **return** $\boldsymbol{gBest}(genMax)$

As Algorithm 2 shows, KMPG starts its search with the initial population belonging to the relevant alternating group (Line 2). Next, it uses the k-medoids clustering based crossover to generate an offspring solution, which inherits valuable genetic information. Thereafter, KMPG undergoes the simulated annealing based local search to improve the offspring solution (Line 7) and the distance-and-quality based population management to ensure a healthy population (Line 9). The algorithm stops and returns the best solution found when $genMax$ generations are performed.

B. Population Initialization

The population initialization of KMPG is different from GA in that KMPG works with a population of local optimal solutions that are obtained by a local optimization procedure, while this is not the case for GA. Note that the initial population of GA can be generated randomly or by greedy algorithms. KMPG considers both the quality and diversity when generating the initial population. It first randomly generates ps even permutations regarding the original or mapping search space that are improved by the local search operator (Section III-D). An *Insert*-based perturbation is then introduced to enhance population diversity, where the symmetry-breaking method is used to ensure even signs. Especially, a discrete function is used to measure the similarity between two permutations [17]:

$$simi(\boldsymbol{\pi^a}, \boldsymbol{\pi^b}) = \sum\nolimits_{1 \le i \le N} |i - rp(\pi_i^a, \boldsymbol{\pi^b})|, \quad (4)$$

where $rp(\pi_i^a, \boldsymbol{\pi^b})$ represents the relative position of π_i^a in $\boldsymbol{\pi^b}$. Take $\boldsymbol{\pi^a} = [4, 1, 3, 2]$ and $\boldsymbol{\pi^b} = [1, 4, 2, 3]$ for example, and we have $simi(\boldsymbol{\pi^a}, \boldsymbol{\pi^b}) = |1-2|+|2-1|+|3-4|+|4-3| = 4$. Apparently, the smaller the $simi$ is, the larger the similarity between the two permutations is. The population initialization procedure is given in Algorithm 3. In the worst case, its time complexity is $O(ps^4 \cdot N)$.

Algorithm 3: Population Initialization Procedure

Input: $Scenario$, ps and $gen = 0$
Output: initial $\boldsymbol{\pi}^{\mathcal{P}}(gen)$
*/*generate permutations with even signs*/*
for $p := 1$ **to** ps **do**
  Randomly generate $\boldsymbol{\pi}_p^{\mathcal{P}}(gen)$;
  $\boldsymbol{\pi}_p^{\mathcal{P}}(gen) \leftarrow inverse(\boldsymbol{\pi}_p^{\mathcal{P}}(gen))$, if $\boldsymbol{\pi}_p^{\mathcal{P}}(gen)$ or $\boldsymbol{\pi}_p^{\mathcal{P}}(gen)^{\mathcal{L}}$ is odd; /*symmetry breaking based on $Scenario$*/
  Improve $\boldsymbol{\pi}_p^{\mathcal{P}}(gen)$ by local search operator; /*Section III-D*/
*/*Insert-based perturbation*/*
Set $count := 0$;
while $count < ps \cdot (ps - 1)/2$ **do**
  for $p := 1$ **to** $ps - 1$ **do**
    for $q := p + 1$ **to** ps **do**
      if $simi(\boldsymbol{\pi}_p^{\mathcal{P}}(gen), \boldsymbol{\pi}_q^{\mathcal{P}}(gen)) = 0$ **then**
        while $simi(\boldsymbol{\pi}_p^{\mathcal{P}}(gen), \boldsymbol{\pi}_q^{\mathcal{P}}(gen)) = 0$ **do**
          Randomly generate $k \neq l \in \{1, ..., N\}$;
          $\boldsymbol{\pi}_q^{\mathcal{P}}(gen) \leftarrow Insert(\boldsymbol{\pi}_q^{\mathcal{P}}(gen), k, l)$;
      else
        $count := count + 1$;
for $p := 1$ **to** ps **do**
  $\boldsymbol{\pi}_p^{\mathcal{P}}(gen) \leftarrow inverse(\boldsymbol{\pi}_p^{\mathcal{P}}(gen))$, if $\boldsymbol{\pi}_p^{\mathcal{P}}(gen)$ or $\boldsymbol{\pi}_p^{\mathcal{P}}(gen)^{\mathcal{L}}$ is odd; /*symmetry breaking based on $Scenario$*/
return $\boldsymbol{\pi}^{\mathcal{P}}(gen)$

C. Proposed K-medoids Clustering Based Crossover

It is widely recognized that crossover operator plays a crucial role within a memetic algorithm [32]. A suitable crossover operator should pass problem-specific genetic knowledge from parent solutions to offspring solutions. For this, we go deep into the problem-specific properties of KMPG and have the following Theorem 5.

Theorem 5: For any facility $n_x \in \{1, ..., N\}$ $(N > 3)$, there are $N!/4$ permutations in A_N (Scenario 1) or $A_N^{\mathcal{L}}$ (Scenario 2) such that $rp(n_x, \boldsymbol{\pi}) \le \lceil N/2 \rceil$ $(\forall \boldsymbol{\pi} \in A_N$ or $A_N^{\mathcal{L}})$.

As Theorem 5 shows, there is a special "quasi-symmetry" of the search space of KMPG (i.e., A_N or $A_N^{\mathcal{L}}$). Such "quasi-symmetry" means that any permutation corresponds to a certain permutation in the search space that has similar objective value, due to the features presented in Theorem 5. For example, we consider $n_x = 3$ and enumerate all the 12 permutations of $A_4^{\mathcal{L}}$ in Fig. 6. There are 6 permutations in $A_4^{\mathcal{L}(A)}$ such that facility 3 is in the first half of them. One can also find a pair of $\boldsymbol{\pi^a} \in A_4^{\mathcal{L}(A)}$ (e.g. [2,3,1,4] in Fig. 6) and $\boldsymbol{\pi^b} \in A_4^{\mathcal{L}(B)}$ (e.g. [4,1,2,3] in Fig. 6) satisfying $simi(\boldsymbol{\pi^a}, inverse(\boldsymbol{\pi^b})) = 2$, so that $\boldsymbol{\pi^a}$ and $\boldsymbol{\pi^b}$ have similar objective values. Given that $\boldsymbol{\pi^a}$ and $\boldsymbol{\pi^b}$ are parent solutions for crossover operator, pertinent problem-specific genetic knowledge is likely to be lost.

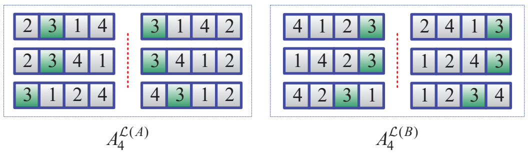


Fig. 6. Illustration of "quasi-symmetry" of the search space of KMPG.

In view of the above "quasi-symmetry" of the search space of KMPG, it is reasonable to select parent solutions for the crossover operator from only one of the two sets:

$$A_N^{(\mathcal{L})(A)} = \{\boldsymbol{\pi} | rp(n_x, \boldsymbol{\pi}) \leq \lceil N/2 \rceil, \forall \boldsymbol{\pi} \in A_N^{(\mathcal{L})}, \quad n_x \in \{1, ..., N\}\}, \tag{5}$$

and

$$A_N^{(\mathcal{L})(B)} = A_N^{(\mathcal{L})} \backslash A_N^{(\mathcal{L})(A)}. \tag{6}$$

Based on Eqs. (5) and (6), we treat the permutations found by KMPG as *two* categories in the sense that they account for distinct "quasi-symmetry" scenarios. As Tang and Meng [33] pointed out, machine learning can provide guidance for optimization algorithms. This motivates us to adopt clustering method to analyze the population of KMPG while guiding the crossover operator. Classical clustering techniques consist mainly of k-means and k-medoids methods [34]. Since the solutions of KMPG are defined in discrete domains, the cluster centers should be feasible permutations. It is, thus, suitable to use the k-medoids method. The k-medoids clustering method is extensively used for clustering problems where the value of k depends strongly on problem-specific properties. Here in our case we fix $k = 2$ accordingly.

To be specific, we first partition the population into two clusters using the k-medoids algorithm, and then select parent solutions of interest from the resulting clusters for the crossover operator. Again, the similarity in Eq. (4) is used as a "clustering distance" between permutation pairs. The adopted k-medoids clustering algorithm starts with two random centers. Next, each permutation of the population is attributed to the closest cluster, each center is set to the permutation that has the smallest sum of similarity of permutations belonging to that cluster, and the procedure continues until the centres don't change anymore. Note that the k-medoids clustering algorithm is invoked if and only if the population is updated (Section III-E). Take $N = 8$ and $ps = 10$ for example, and we illustrate the resulting clusters of the k-medoids clustering algorithm in Fig. 7. In Section IV, we will perform relevant discussions on the KMPG variants considering different versions of k-medoids clustering based crossovers to verity the reasonability of the adopted k-medoids clustering algorithm.

Let pr be the rate of reserving partial facilities, and $\boldsymbol{\pi}^{\mathcal{C}_1}$ and $\boldsymbol{\pi}^{\mathcal{C}_2}$ the resulting clusters of the k-medoids algorithm. The k-medoids clustering based crossover is given in Algorithm 4. In the worst case, its time complexity is $O(ps \cdot N)$.

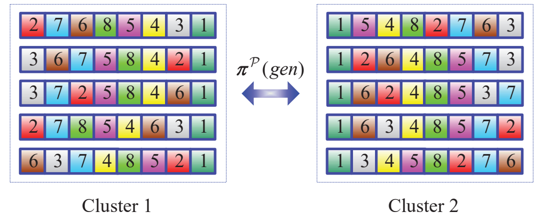


Fig. 7. Illustration of resulting clusters of k-medoids clustering algorithm.

Algorithm 4: K-medoids Clustering Based Crossover

Input: $Scenario$, $\boldsymbol{\pi}^{\mathcal{C}_1}$, $\boldsymbol{\pi}^{\mathcal{C}_2}$ and pr
Output: offspring solution $\boldsymbol{\pi}$
*/*preparations for crossover*/*
1 Set $nr := \lfloor N \cdot (1 - pr) \rfloor$; /*number of reserved facilities*/
2 Randomly select a cluster $k \in \{1, 2\}$ such that $|\boldsymbol{\pi}^{\mathcal{C}_k}| > 1$;
3 Calculate number of distinct facilities for each position, denoted by $\boldsymbol{nd} = \{nd_1, ..., nd_N\}$, according to solutions in cluster k;
4 Identify reserved positions, denoted by $\boldsymbol{nm} = \{nm_1, ..., nm_{nr}\}$, that correspond to the smallest nr numbers in $\boldsymbol{nd}$;
*/*generate offspring solution*/*
5 Randomly select two parent solutions p and q ($p \neq q$) as well as a reference solution r from cluster k;
6 Initialize $\boldsymbol{\pi} \leftarrow \emptyset$ and $cp := 1$ and $cq := 1$ for solutions p and q; /*cp and cq are available positions of p and q*/
7 **for** $i := 1$ **to** nr **do**
8 $\quad$ $\pi_{nm_i} \leftarrow \pi_{r,nm_i}^{\mathcal{C}_k}$; /*build partial offspring solution*/
9 **for** $(i := 1$ **to** $N) \wedge (\pi_i = \emptyset)$ **do**
10 $\quad$ **while** cp (cq) is not available **do**
11 $\quad\quad$ cp (cq) $:= cp$ (cq) $+ 1$; /*current available positions*/
12 $\quad$ **if** $random[0, 1] \leq 0.5$ **then** $\pi_i \leftarrow \pi_{p,cp}^{\mathcal{C}_k}$; **else** $\pi_i \leftarrow \pi_{q,cq}^{\mathcal{C}_k}$;
*/*symmetry-breaking procedure*/*
13 $\boldsymbol{\pi} \leftarrow inverse(\boldsymbol{\pi})$, if $\boldsymbol{\pi}$ or $\boldsymbol{\pi}^{\mathcal{L}}$ is odd; /*based on $Scenario$*/
14 **return** $\boldsymbol{\pi}$

In Algorithm 4, preparations for the crossover operator are made first (Lines 1 to 4). Especially, we identify reserved positions based on the overall features of the selected cluster (Line 4). Then, the partial offspring solution is built using the reserved facilities of the reference solution to inherit good properties (Lines 7 to 8). Next, we complete the offspring solution with uniformly selected available facilities from the parent solutions to ensure diversity (Lines 9 to 12). Complementing the reserved and uniformly selected facilities, the dedicated crossover operator has the advantage of producing meaningful offspring solutions regarding both quality and diversity.

D. Simulated Annealing Based Local Search Considering Symmetry Breaking

To intensively examine neighbor solutions, the general simulated annealing procedure and the objective gain calculation method of [2] are used. However, we perform the local search in the philosophy of the dedicated symmetry breaking (see Algorithm 1), which is thus different from [2].

The simulated annealing procedure has two nested loops, namely *outer* loop and *inner* loop. Let n_{out} be the number of the outer loop, n_{in} the number of the inner loop, t_{max} the maximum temperature, ct the cooling factor of temperature, t_{min} the minimum temperature, n_{sam} the sample size for calculating t_{max}, and pl the coefficient for determining n_{in}. The simulated annealing procedure is presented in Algorithm 5. As mentioned in [2], the time complexity of calculating the objective gain of the *Insert*-based neighborhood structure is $O(N)$, so the time complexity of Algorithm 5 is $O(n_{out} \cdot N^2)$.

Algorithm 5: Simulated Annealing Procedure

Input: $Scenario$, ct, n_{sam}, t_{min}, pl and offspring solution $\boldsymbol{\pi}$
Output: improved $\boldsymbol{\pi}$
1 Set $t_{max} := \max_{1 \le r \le n_{sam}} |f(insert(\boldsymbol{\pi}, k_r, l_r)) - f(\boldsymbol{\pi})|$ where k_r and l_r $(k_r \neq l_r)$ are randomly selected from $\{1, ..., N\}$;
2 Set $n_{out} := \lfloor (\log t_{min} - \log t_{max}) / \log ct \rfloor$ and $n_{in} := pl \cdot N$;
3 Initialize temperature $T := t_{max}$;
4 Set $\boldsymbol{\beta} := \boldsymbol{\pi}$ and $f(\boldsymbol{\beta}) := f(\boldsymbol{\pi})$; /*record offspring solution*/
5 **for** $p := 1$ **to** n_{out} **do**
6   **for** $q := 1$ **to** n_{in} **do**
7   Initialize acceptance flag $accept := 0$;
8   Randomly select k and l $(k \neq l)$ from $\{1, ..., N\}$;
9   Calculate objective gain $\delta := f(insert(\boldsymbol{\beta}, k, l)) - f(\boldsymbol{\beta})$;
10   **if** $\delta \le 0$ **then**
11   Set $accept := 2$;
12   **else**
13   **if** $random[0, 1] \le \exp(-\delta / T)$ **then**
14   Set $accept := 1$;
15   **if** $accept = 0$ **then**
16   **continue**;
17   **else**
18             $\boldsymbol{\eta} \leftarrow insert(\boldsymbol{\beta}, k, l)$; /*neighborhood moving*/
19   Update $\boldsymbol{\beta}$ based on $\boldsymbol{\eta}$ by using symmetry-breaking procedure in *Algorithm 1*; /*based on $Scenario$*/
20   Set $T := ct \cdot T$; /*temperature cooling*/
21 **if** $f(\boldsymbol{\beta}) < f(\boldsymbol{\pi})$ **then**
22   Set $\boldsymbol{\pi} := \boldsymbol{\beta}$ and $f(\boldsymbol{\pi}) := f(\boldsymbol{\beta})$;
23 **return** $\boldsymbol{\pi}$

In Algorithm 5, to guide the simulated annealing procedure to the reduced search space, the symmetry-breaking method in Algorithm 1 is used to modify the accepted neighboring solutions (Line 19). Thanks to the control of the permutation signs of neighbor solutions, KMPG is afforded more possibilities, yet requires less computational efforts to find promising solutions in a given search region.

E. Distance-and-quality Based Population Management

To maintain a health population and avoid a premature convergence, we use a population updating mechanism similar to [31]. The adopted mechanism simultaneously considers offspring solution's quality and its distance to the permutations in the population. Since the entire search behavior of KMPG is restricted to the reduced search space, any offspring solution $\boldsymbol{\pi}$ satisfies $\boldsymbol{\pi} \neq inverse(\boldsymbol{\pi}_p^{\mathcal{P}}(gen))$ $(\forall p \in \{1, ..., ps\})$. For this reason, we calculate the similarity between $\boldsymbol{\pi}$ and each $\boldsymbol{\pi}_p^{\mathcal{P}}(gen)$ according to Eq. (4). If one of the ps similarities is 0, $\boldsymbol{\pi}$ is dropped and the population is not updated. Otherwise, if $\boldsymbol{\pi}$ is better than the worst permutation in the population, the worst one is replaced by $\boldsymbol{\pi}$; else, we drop $\boldsymbol{\pi}$. The time complexity of updating the population is $O(N)$.

IV. Experimental Results

A. Benchmark Instances and Experimental Setup

To evaluate the performance of the proposed KMPG algorithm, we conduct extensive experiments on four sets of well-known benchmark instances and one set of newly introduced large-scale instances. In recent years, the four sets of the traditional benchmark instances have been commonly used to test the performance of SRFLP algorithms. These five sets are summarized as follows[1].

- **Amaral set (3 instances)**: This set contains 3 medium scale instances with $N = 110$.
- **Anjos set (40 instances)**: This set includes 20 small-scale instances with $N = 60$ to 80, 10 medium scale instances with $N = 200$ and 300, and 10 large-scale instances with $N = 400$ and 500.
- **Sko set (40 instances)**: This set consists of 20 small-scale instances with $N = 64$ to 100, 10 medium scale instances with $N = 200$ and 300, and 10 large-scale instances with $N = 400$ and 500.
- **Palubeckis set (50 instances)**: This set is composed of 20 medium scale instances with $N = 110$ to 300, and 30 large-scale instances with $N = 310$ to 1000.
- **RanLarge set (20 instances)**: This set contains 20 new large-scale instances introduced in this paper with $N =$ 1050 to 2000. Based on the generation methods of [2] and [3], the length of each facility and the flow cost of each facility pair are randomly generated from $\{1, ..., 10\}$ and $\{0, ..., 10\}$, respectively.

Note that the 40 small-scale instances with $N \le 100$ in the Anjos and Sko sets are ignored because the best known upper bounds can be easily attained by KMPG with short running times. Hence, there are a total of 113 instances including 93 traditional benchmark instances and 20 new instances.

For each instance, KMPG and compared algorithms (if source codes are available) are independently run 30 times with different random seeds. The results are reported as *BEST*, *AVG* and *SD*, which are the best objective value, average objective value and standard deviation of solved solutions. Meanwhile, we use *NoB* to represent the total number of instances for which an algorithm can obtain the best result of a quality indicator. To analyze the statistical significance of the comparisons between KMPG and the compared algorithms, the *p*-values from the nonparametric Friedman test with a confidence interval (CI) of 95% are provided. Besides, we mark the *BEST* of an instance with an asterisk (*) to indicate a strictly best objective value among the compared algorithms, which also corresponds to a newly discovered upper bound. KMPG is coded in C++ and complied using g++ (with '-O3' flag), and all experiments are carried out on a computing platform with an AMD Opteron-6134 2.3 GHz CPU and 2 GB memory under Linux.

B. Parameter Tuning

The parameters of KMPG include the population size ps, the rate of reserving partial facilities pr, the coefficient for

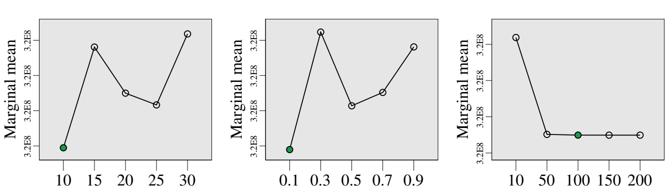
[1]Instances are available at http://dx.doi.org/10.13140/RG.2.2.27898.41926

determining the inner loop number pl, the temperature cooling factor ct, the minimum temperature t_{min}, and the sample size n_{sam}. As mentioned in [2], the settings of ct, t_{min} and n_{sam} are well known, which have relatively less impact on the performance of the simulated annealing procedure. In our preliminary experiments, we observed that KMPG can achieve good search performances by adopting the following values: $ct = 0.95$, $t_{min} = 0.0001$, and $n_{sam} = 5000$. Therefore, the parameter tuning of KMPG is devoted to finding a suitable combination of ps, pr and pl. Toward this goal, we carry out the design of experiment (DOE) based on the instance "p450" in the Palubeckis set in which the orthogonal array L_5^3 with 25 parameter combinations is used. For each combination, we independently run KMPG 30 times with the CPU time 3600s and adopt *AVG* as response value. The orthogonal array and response values are given in Table I, in which each combination of parameter values corresponds to a certain *AVG* value. Thereby, we can determine the parameters of KMPG and check the robustness of KMPG to the parameters.

TABLE I
ORTHOGONAL ARRAY AND RESPONSE VALUES

No.	Combination			*AVG*	No.	Combination			*AVG*
	ps	pr	pl			ps	pr	pl	
1	30	0.5	150	324488489.5	14	30	0.9	200	324488494.7
2	30	0.7	50	324488506.7	15	15	0.9	10	324491451.1
3	25	0.7	150	324488495.3	16	20	0.7	10	324491140.6
4	20	0.9	150	324488492.1	17	20	0.3	200	324488492.7
5	10	0.3	150	324488493.1	18	15	0.5	200	324488491.5
6	25	0.1	200	324488491.9	19	10	0.7	200	324488491.6
7	25	0.5	10	324491040.1	20	10	0.1	10	324490741.2
8	20	0.5	100	324488495.8	21	30	0.1	100	324488492.3
9	15	0.7	100	324488495.6	22	20	0.1	50	324488503.6
10	10	0.5	50	324488518.1	23	15	0.3	50	324488518.3
11	15	0.1	150	324488495.1	24	10	0.9	100	324488493.1
12	25	0.3	100	324488492.1	25	30	0.3	10	324491561.8
13	25	0.9	50	324488521.9	—	—	—	—	—

Based on the results of Table I, the estimated level trends are shown in Fig. 8. Therefore, we set the three parameters of KMPG as follows: $ps = 10$, $pr = 0.1$, and $pl = 100$. We also conduct additional experiments for tuning these three parameters considering other instances, while the results are the same as mentioned above. Moreover, we perform the analysis of variance (ANOVA) with the 95% CI to examine the sensitivity of the three parameters, as shown in Table II.



(a) ps (b) pr (c) pl

Fig. 8. Estimated Level trends of ps, pr and pl (marginal means of *AVG*).

As Table II shows, the p-values for ps and pr are larger than 0.05 and, in turn, reveal the robustness of KMPG to these two parameters. In addition, the p-value for pl is smaller than 0.05, which can thus confirm the reasonability of the integration of the global search model and the local search procedure.

TABLE II
ANOVA RESULTS OF PARAMETERS

Parameter	Mean square	F-value	p-value	Rank
ps	21271.835	0.989	0.450	3
pr	22437.413	1.043	0.425	2
pl	7229561.398	336.052	0.000	1

C. Comparisons of KMPG and State-of-the-art Algorithms

In this section, we report comparative results with the most recent state-of-the-art SRFLP algorithms with respect to the five sets of benchmark instances[2]. According to the results of experiments in one of the latest works on SRFLP [1], the following four algorithms can be regard as the best performing ones: GRASP-$_\mathrm{F}$ (2019) [1], GRASP-PR (2016) [4], VNS-LS3 (2015) [3] and MSA (2017) [2]. To our knowledge, the best known upper bounds (BKS) in the literature are achieved by these four algorithms together. We thus use them as the reference algorithms for the comparisons.

1) Computational Results for Amaral and Anjos Sets: For the Amaral and Anjos sets, we use GRASP-$_\mathrm{F}$ and GRASP-PR as reference algorithms. Since the source codes of GRASP-$_\mathrm{F}$ and GRASP-PR are not available to us, we adopt the results of these two algorithms reported in the related references. Considering the differences in computer configurations and programming environments, the running times of KMPG and the compared algorithms will not be further discussed. For each run, the stopping condition of KMPG is a cutoff CPU time of 3600s. In terms of *BEST*, *AVG* and *SD*, the comparative results on the 23 instances in the Amaral and Anjos sets are summarized in Fig. 9. The statistical significance of the comparisons is given in Table III.

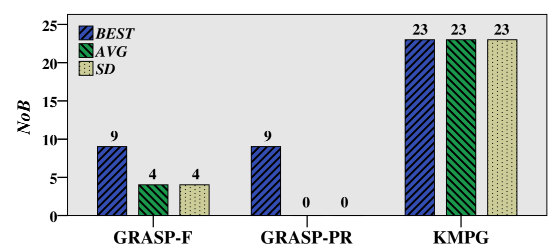


Fig. 9. Summary of comparisons of GRASP-$_\mathrm{F}$, GRASP-PR and KMPG on the 23 instances in the Amaral and Anjos sets with $N = 110$ to 500. Note that only *BEST* values were provided for GRASP-PR in [4].

TABLE III
STATISTICAL SIGNIFICANCE OF COMPARISONS BETWEEN GRASP-$_\mathrm{F}$, GRASP-PR AND KMPG (AMARAL AND ANJOS SETS)

Indicator	p-value for algorithm pair			
	KMPG vs. GRASP-$_\mathrm{F}$	Sig.	KMPG vs. GRASP-PR	Sig.
BEST	0.000	Yes	0.000	Yes
AVG	0.000	Yes	—	—
SD	0.000	Yes	—	—

From Fig. 9, we can see that KMPG outperforms the compared algorithms associated with *BEST*, *AVG* and *SD*. In

[2]Details of comparative results of KMPG and state-of-the-art algorithms are available at http://dx.doi.org/10.13140/RG.2.2.29156.71041

Table III, the p-values are all less than 0.05, which shows the significance of difference between KMPG and the two reference algorithms.

In addition to the above summary, we report the detailed results of KMPG in Table IV. For each instance, the average CPU time in seconds (i.e., T(s)), which is related to the best objective value found for the first time during the search process of KMPG, is listed for information. The results show that KMPG has a good search performance for large-scale instances with $N \geq 400$. On the other hand, the values of *SD* are 0 for 12 out of 23 instances, which indicates the robustness of KMPG. To our knowledge, for the Amaral and Anjos sets, KMPG is the first method achieving such a stable performance. Especially, KMPG is able to discover new upper bounds for 11 out 23 instances, while reaching the BKS for the other 12 instances. These newly obtained upper bounds show the convergence of KMPG for finding high-quality solutions of SRFLP.

TABLE IV
RESULTS OF KMPG ON THE AMARAL AND ANJOS SETS

Instance	BKS	KMPG (this work)			
		BEST	*AVG*	*SD*	T(s)
Amaral_110_1	144296664.5	144296664.5	144296664.5	0.0	40.6
Amaral_110_2	86050037	86050037	86050037.0	0.0	44.6
Amaral_110_3	2234743.5	2234743.5	2234743.5	0.0	4.2
Anjos_200_1	305461818	305461818	305461818.0	0.0	1464.3
Anjos_200_2	178806828.5	178806828.5	178806828.5	0.0	1012.1
Anjos_200_3	61891275	61891275	61891275.0	0.0	1056.0
Anjos_200_4	127735691	127735691	127735691.0	0.0	1786.7
Anjos_200_5	89057121.5	89057121.5	89057121.5	0.0	371.3
Anjos_300_1	1549423272	***1549422266**	1549422301.1	28.1	2010.9
Anjos_300_2	955538302.5	955538302.5	955538302.5	0.0	1595.7
Anjos_300_3	308257630.5	308257630.5	308257630.5	0.0	1284.8
Anjos_300_4	602873168.5	602873168.5	602873168.5	0.0	1246.0
Anjos_300_5	466151295	***466150775**	466150901.5	84.6	2055.7
Anjos_400_1	4999816802	***4999801252.5**	4999804517.2	3219.6	1849.7
Anjos_400_2	2910265496	***2910265439**	2910265560.3	57.2	1688.9
Anjos_400_3	920861309	***920860291**	920860560.2	117.2	1887.3
Anjos_400_4	1805638949	1805638949	1805638949.0	0.0	1779.2
Anjos_400_5	1401835315	***1401831869.5**	1401832115.8	231.5	2010.2
Anjos_500_1	12290696250	***12290667266**	12290684656.0	11093.8	1774.3
Anjos_500_2	7491700551	***7491697298.5**	7491721869.3	13710.8	1850.2
Anjos_500_3	2478348156	***2478331774**	2478337666.0	3804.8	1684.8
Anjos_500_4	4281107260	***4281106062.5**	4281107387.9	790.1	1679.3
Anjos_500_5	3675369229	***3675293000.5**	3675296069.6	4218.4	2168.8

* KMPG discovers new upper bounds for 11 out of the 23 instances.

2) Computational Results for Sko Set: In terms of *BEST*, *AVG* and *SD*, we summarize the comparisons of GRASP-$_{\text{F}}$, GRASP-PR and KMPG on the 20 instances in the Sko set in Fig. 10. It should be noted that we also adopt the results of the two reference algorithms from the related references and the same stopping condition as mentioned in subsection IV-C1. In addition, the corresponding p-values for the two algorithm pairs are listed in Table V. Moreover, the results of KMPG are provided in Table VI.

TABLE V
STATISTICAL SIGNIFICANCE OF COMPARISONS BETWEEN GRASP-$_{\text{F}}$, GRASP-PR AND KMPG (SKO SET)

Indicator	p-value for algorithm pair			
	KMPG vs. GRASP-$_{\text{F}}$	Sig.	KMPG vs. GRASP-PR	Sig.
BEST	0.000	Yes	0.000	Yes
AVG	0.000	Yes	—	—
SD	0.000	Yes	—	—

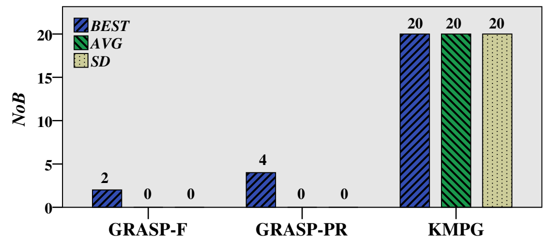


Fig. 10. Summary of comparisons of GRASP-$_{\text{F}}$, GRASP-PR and KMPG on the 20 instances in the Sko set with N = 200 to 500.

TABLE VI
RESULTS OF KMPG ON THE SKO SET

Instance	BKS	KMPG (this work)			
		BEST	*AVG*	*SD*	T(s)
Sko_200_1	3231044	***3230706**	3230708.0	1.6	2056.3
Sko_200_2	7758927	7758927	7758927.0	0.0	1473.0
Sko_200_3	12739043	12739043	12739043.0	0.0	1632.2
Sko_200_4	20260531	20260531	20260531.0	0.0	1624.9
Sko_200_5	26871976.5	26871976.5	26871976.5	0.0	2054.9
Sko_300_1	11251960	***11249787**	11249811.9	11.6	1739.3
Sko_300_2	28991854	***28991767**	28991778.1	5.0	1841.5
Sko_300_3	48791025.5	***48790881.5**	48790882.9	2.8	1898.4
Sko_300_4	71185259.5	***71185096.5**	71185109.4	14.0	1678.0
Sko_300_5	86789543.5	***86789519.5**	86789746.0	229.3	1695.0
Sko_400_1	26708999	***26696039**	26696113.8	30.0	1806.7
Sko_400_2	67950673	***67950486**	67950507.8	13.0	1709.1
Sko_400_3	115881934	***115881368**	115881475.1	67.6	1764.3
Sko_400_4	162933020	***162932778**	162932804.1	14.5	1696.8
Sko_400_5	227633018.5	***227632499.5**	227632525.5	12.7	1285.8
Sko_500_1	52873391	***52853174**	52853510.6	168.1	1723.7
Sko_500_2	127364274	***127362138**	127362192.5	33.8	2004.8
Sko_500_3	230919458.5	***230877719.5**	230877781.1	33.7	1490.4
Sko_500_4	340274982	***340273141**	340278964.6	6217.1	1955.6
Sko_500_5	445699477	***445698570**	445698708.0	82.3	2146.7

* KMPG discovers new upper bounds for 16 out of the 20 instances.

It can be seen from Fig. 10 and Table V that KMPG achieves a more competitive performance than the reference algorithms. As Table VI shows, KMPG attains all the BKS and finds new upper bounds for 16 out 20 instances. For the instances Sko_200_2 to 5 that match the best known upper bounds, the *SD* values are all 0, confirming the robustness of KMPG. For the Sko set, KMPG is also the first method that achieves such performance in terms of both convergence and stability.

From the above results, one concludes that the population-based KMPG is more suitable than single trajectory methods. This is because, in KMPG, the population-based search model can capture valuable information while requiring less effort. Due to GRASP is one of the best performing search models for combinatorial optimization problems, it is worthwhile to devise effective methods based on our symmetry-breaking method and GRASP for SRFLP in future research.

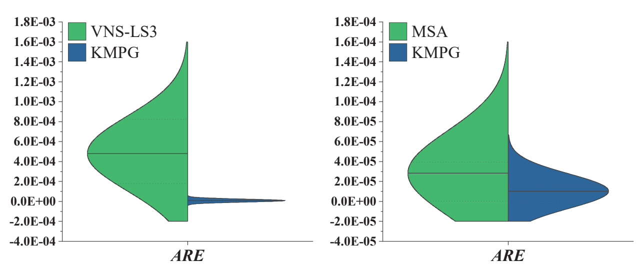
3) Computational Results for Palubeckis Set: We use VNS-LS3 and MSA as the reference algorithms, which hold the best known results for the Palubeckis set. The source codes of the reference algorithms are available to us, making it possible to perform a fair comparison. For each instance, we execute the three algorithms with a cutoff time of 3600s. The results are reported in Table VII.

From Table VII, we observe that KMPG outperforms the reference algorithms for all the instances regarding *BEST*, *AVG* and *SD*. The p-values are all less than 0.05, which indicates the significance of the difference between KMPG and the

TABLE VII
COMPARISONS OF VNS-LS3, MSA AND KMPG ON THE PALUBECKIS SET

Instance	BKS	VNS-LS3			MSA			KMPG (this work)			
		BEST	*AVG*	*SD*	*BEST*	*AVG*	*SD*	*BEST*	*AVG*	*SD*	*T*(s)
p110	4435868	**4435868**	**4435868.0**	**0.0**	**4435868**	**4435868.0**	**0.0**	**4435868**	**4435868.0**	**0.0**	18.0
p120	6282721	**6282721**	**6282721.0**	**0.0**	**6282721**	**6282721.0**	**0.0**	**6282721**	**6282721.0**	**0.0**	39.3
p130	7880929.5	**7880929.5**	**7880929.5**	**0.0**	**7880929.5**	**7880929.5**	**0.0**	**7880929.5**	**7880929.5**	**0.0**	40.3
p140	9257162	**9257162**	**9257162.0**	**0.0**	**9257162**	**9257162.0**	**0.0**	**9257162**	**9257162.0**	**0.0**	181.4
p150	10624389.5	**10624389.5**	10624411.0	115.6	**10624389.5**	**10624389.5**	**0.0**	**10624389.5**	**10624389.5**	**0.0**	458.4
p160	14873277	**14873277**	**14873277.0**	**0.0**	**14873277**	**14873277.0**	**0.0**	**14873277**	**14873277.0**	**0.0**	355.6
p170	16630187	**16630187**	**16630187.0**	**0.0**	**16630187**	**16630187.0**	**0.0**	**16630187**	**16630187.0**	**0.0**	170.7
p180	18746031.5	**18746031.5**	**18746031.5**	**0.0**	**18746031.5**	**18746031.5**	**0.0**	**18746031.5**	**18746031.5**	**0.0**	329.3
p190	24453272	**24453272**	24453379.0	400.4	**24453272**	**24453272.0**	**0.0**	**24453272**	**24453272.0**	**0.0**	227.9
p200	27482649.5	**27482649.5**	**27482649.5**	**0.0**	**27482649.5**	**27482649.5**	**0.0**	**27482649.5**	**27482649.5**	**0.0**	1212.8
p210	29512000.5	**29512000.5**	29512323.6	823.7	**29512000.5**	**29512000.5**	**0.0**	**29512000.5**	**29512000.5**	**0.0**	383.8
p220	37351850.5	**37351850.5**	37355394.4	4082.4	**37351850.5**	**37351850.5**	**0.0**	**37351850.5**	**37351850.5**	**0.0**	1026.2
p230	46744886	**46744886**	**46744886.0**	**0.0**	**46744886**	**46744886.0**	**0.0**	**46744886**	**46744886.0**	**0.0**	143.2
p240	46717781	**46717781**	46718520.1	3446.8	**46717781**	**46717781.0**	**0.0**	**46717781**	**46717781.0**	**0.0**	758.9
p250	54526293.5	**54526293.5**	54532207.4	5695.7	**54526293.5**	54526295.1	3.2	**54526293.5**	**54526293.5**	**0.0**	1497.5
p260	63300360.5	**63300360.5**	63306142.0	8940.0	**63300360.5**	63300440.3	277.9	**63300360.5**	**63300360.5**	**0.0**	1675.8
p270	68960438.5	**68960438.5**	**68960438.5**	**0.0**	**68960438.5**	**68960438.5**	**0.0**	**68960438.5**	**68960438.5**	**0.0**	935.5
p280	73845821	**73845821**	73861709.0	13956.7	**73845821**	**73845821.0**	**0.0**	**73845821**	**73845821.0**	**0.0**	1041.8
p290	86255267.5	**86255267.5**	86264186.5	10237.6	**86255267.5**	86255423.9	126.6	**86255267.5**	**86255287.2**	**12.9**	1596.4
p300	95937735	**95937735**	95949587.9	17670.6	**95937735**	95937735.7	2.5	**95937735**	**95937735.0**	**0.0**	1903.5
p310	105754955	**105754955**	105763088.2	12716.6	**105754955**	105754969.3	60.1	**105754955**	**105754955.0**	**0.0**	1719.0
p320	119522881.5	**119522881.5**	119544321.6	18806.3	**119522881.5**	119523002.4	207.6	**119522881.5**	**119522889.0**	**9.3**	1650.3
p330	124891823.5	**124891823.5**	124906998.0	34457.0	**124891823.5**	124891823.6	0.5	**124891823.5**	**124891823.5**	**0.0**	1449.5
p340	129796777.5	**129796777.5**	129810720.0	17371.1	**129796777.5**	129796786.1	24.8	**129796777.5**	**129796777.8**	**0.7**	1797.6
p350	149594388	**149594388**	149595230.2	1471.4	**149594388**	149594388.3	1.5	**149594388**	**149594388.0**	**0.0**	1193.3
p360	141122187	**141122187**	141138594.3	19708.2	**141122187**	141122284.6	256.0	**141122187**	**141122190.6**	**3.8**	1798.5
p370	174663159.5	**174663159.5**	174692270.7	27993.2	**174663159.5**	**174663159.5**	0.0	**174663159.5**	**174663159.5**	**0.0**	1125.3
p380	189452403.5	**189452403.5**	189502413.3	31974.3	**189452403.5**	189461691.8	6617.3	**189452403.5**	**189455471.4**	**5526.8**	1661.9
p390	208688557	**208688557**	208693245.3	9043.1	**208688557**	208688559.6	3.1	**208688557**	**208688557.0**	**0.0**	1490.7
p400	213768810.5	**213768810.5**	213848612.5	67994.7	**213768810.5**	213768899.0	316.1	**213768810.5**	**213768810.5**	**0.0**	1772.4
p410	243494269	**243494269**	243570586.1	74788.7	**243494269**	243494572.3	306.4	**243494269**	**243494279.0**	**14.8**	1674.5
p420	270756527.5	**270756527.5**	270811024.1	42410.3	**270756527.5**	270756803.4	1387.2	**270756527.5**	**270756539.8**	**9.8**	1518.4
p430	286334521.5	286351751.5	286408285.1	49235.7	286335277.5	286344928.3	5844.4	***286335267.5**	**286339826.8**	**847.1**	2177.1
p440	301067264.5	**301067264.5**	301165922.1	74547.3	**301067264.5**	301067292.7	28.9	**301067264.5**	**301067268.7**	**9.1**	1722.5
p450	324488485	**324488485**	324573532.8	60206.5	**324488485**	324489383.6	1774.1	**324488485**	**324488493.2**	**9.1**	1819.8
p460	314884659	**314884659**	315010645.1	63683.2	**314884659**	314885000.5	1071.4	**314884659**	**314884665.7**	**5.5**	1876.8
p470	379529990	**379529990**	379655676.2	86359.1	379530013	379535885.7	4198.6	**379529990**	**379533829.4**	**1566.3**	1712.1
p480	366821075	***366821074**	366978488.1	102207.1	366821075	366821571.3	1479.2	***366821074**	**366821095.6**	**14.4**	2017.8
p490	413901954.5	**413901954.5**	414076791.0	145356.9	**413901954.5**	413902076.0	108.6	**413901954.5**	**413901999.4**	**21.1**	1818.0
p500	465570835.5	465594639.5	465788516.5	124793.2	**465570835.5**	465580952.1	7862.5	**465570835.5**	**465572650.0**	**1953.2**	1476.7
p550	587090450.5	587108114.5	587518463.1	156550.5	**587090450.5**	587097195.0	8468.8	**587090450.5**	**587090798.2**	**1137.9**	1806.5
p600	801567664.5	801690450.5	802124564.8	304503.0	801567667.5	801594835.8	12810.8	***801567648.5**	**801581607.9**	**10990.1**	1990.5
p650	927512834	927952277	928277814.3	207330.6	**927512856**	927549195.7	41294.0	**927512856**	**927514366.0**	**2968.9**	1779.2
p700	1158462340	1159000394	1159546083.0	236033.4	1158462512	1158608173.9	69306.8	***1158462254**	**1158530076.7**	**58893.9**	1994.9
p750	1438408860.5	1438608485.5	1439876911.8	513149.4	1438408866.5	1438451140.7	51290.7	***1438408836.5**	**1438423171.8**	**8659.0**	1819.6
p800	1861593391	1862675842	1863417335.3	453352.1	**1861593391**	1861731247.2	76969.0	**1861593391**	**1861663477.8**	**32127.9**	2093.0
p850	2126675923	2127852799	2128931188.6	549425.5	**2126675923**	2126998373.6	138431.2	**2126675923**	**2126737164.2**	**50596.1**	2439.5
p900	2600124305	2601629984	2602727296.1	636069.3	2600124395	2600327533.4	115827.6	***2600003561**	**2600220480.5**	**55532.9**	2124.5
p950	2993153592.5	2994239057.5	2996450969.5	889743.9	**2993153592.5**	2993293074.1	188240.7	**2993153592.5**	**2993182477.4**	**29802.4**	2207.9
p1000	3426647267	3428816846	3430872286.5	749515.8	3426647371	3427074861.0	363800.6	***3426646969**	**3426854979.9**	**140676.7**	2051.8
NoB		38	10	10	42	17	16	**50**	**50**	**50**	
p-value		0.001	0.000	0.000	0.005	0.000	0.000				

* KMPG discovers new upper bounds for 6 out of the 50 instances.



(a) (b)

Fig. 11. Violin plots for the solutions of VNS-LS3, MSA and KMPG on the 30 instances in the Palubeckis set with $N = 310$ to 1000. These plots are based on an average relative error $ARE{=}(AVG{-}BKS)/BKS$.

reference algorithms. Especially, KMPG attains all but one best known upper bounds and discovers 6 new upper bounds for the 50 instances. To further show the statistical distributions of the results, we draw the violin plots for algorithm pairs "KMPG vs. VNS-LS3" and "KMPG vs. MSA" in Fig. 11. It is clear then that KMPG has better statistical distributions than the reference algorithms.

In view of the results in subsections IV-C1 to IV-C3, the integration of the population-based model and our symmetry-breaking method can be considered as an effective strategy for solving SRFLP. These results also indicate that, although the four reference algorithms are all based on single trajectory models, it is still highly desirable to investigate other kinds of population-based search models with our symmetry-breaking method for SRFLP. However, this is obviously beyond the scope of this work.

4) Computational Results for RanLarge Set: We here adopt VNS-LS3 and MSA as the reference algorithms, because they are among the best performing SRFLP algorithms for large-scale instances currently available in the literature. The comparison results are reported in Table VIII. It is worth noting that we use the same experimental settings as the ones mentioned in subsection IV-C3, except that KMPG and the reference algorithms are run with a CPU time of 6000s. It can be observed from Table VIII that KMPG performs significantly better than the reference algorithms with less than 0.05 p-values. These results verify the competitive performance of KMPG for solving large-scale SRFLP instances. Since we introduce for the first time these large-scale SRFLP instances with $N = 1050$ to 2000, the reported best known upper bounds can be useful for future research.

Based on the results of Table VIII, we draw the violin plots for algorithm pairs "KMPG vs. VNS-LS3" and "KMPG vs. MSA" in Fig. 12. Meanwhile, we give the box plots for instances L1300, L1500, L1700 and L1900 in Fig. 13. It is clear that KMPG achieves a more competitive distribution than the two reference algorithms. The competitiveness of KMPG may benefit from the interactions of the dedicated search components based on both the mathematical properties and machine learning mechanism. Toward that end, there would be a large potential to solve large-scale SRFLP instances with KMPG. Given that SRFLP has strong application focuses, KMPG will be particularly helpful to decision makers in different real-life fields.

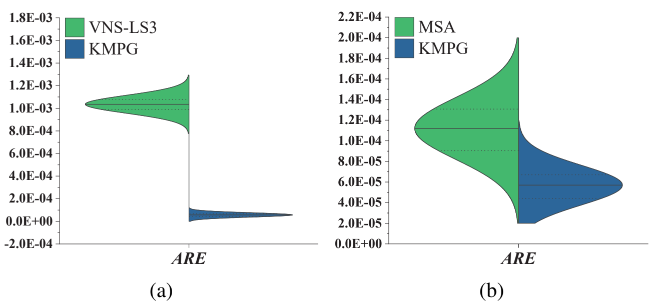


Fig. 12. Violin plots for the solutions of VNS-LS3, MSA and KMPG on the 20 instances in the RanLarge set with $N = 1050$ to 2000.

D. Assessments of KMPG's Search Components

1) Effectiveness of Main Search Components: The main search components of KMPG are the symmetry-breaking mechanism, the k-medoids clustering based crossover, and the simulated annealing based local search. To assess their impacts, we consider the following variants. 1) KMPG_V1 is a KMPG variant where the symmetry-breaking mechanism is disabled; 2) KMPG_V2 is a KMPG variant where the parent solutions for the crossover are randomly selected from the current population rather than from one resulting cluster of the k-medoids clustering method; and 3) KMPG_V3 is a KMPG variant where the *Insert*-based neighborhood is replaced by the *Interchange*-based neighborhood. Note that in KMPG_V3 the objective gain method of [2] is used. For a fair comparison, we run KMPG and the variants with the same CPU time of 3600s. For the 30 instances in the Palubeckis set with $N = 310$ to 1000, the results are summarized in Fig. 14[3]. Meanwhile, the p-values for three algorithm pairs are given in Table IX.

[3]Details of comparative results of KMPG and the three variants are available at http://dx.doi.org/10.13140/RG.2.2.20243.81448

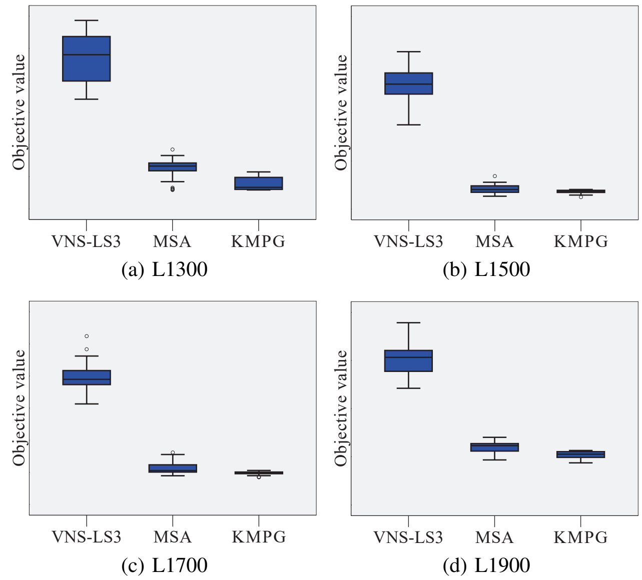


Fig. 13. Box plots for the solutions of VNS-LS3, MSA and KMPG on the instances L1300, L1500, L1700 and L1900 in the RanLarge set.

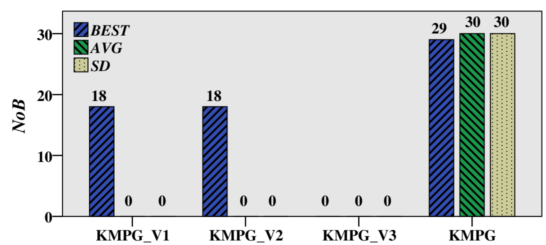


Fig. 14. Summary of comparisons of the three variants (V1-V3) and KMPG.

From Fig. 14, one can observe that KMPG outperforms KMPG_V1, indicating that the proposed symmetry-breaking strategy strengthens the search ability. We see that KMPG is afforded more opportunities to find promising search regions within the reduced search space. In this sense, it can be further used to improve other SRFLP algorithms. Meanwhile, KMPG outperforms KMPG_V2 which verifies the effectiveness of the k-medoids clustering based crossover. This also indicates that machine learning is suitable for capturing useful problem-specific solution features to guide effective search components (such as crossover operators), which could be extensively advocated in other metaheuristic algorithms. Moreover, KMPG outperforms KMPG_V3 for all the instances regarding *BEST*, *AVG* and *SD*. Therefore, adopting the *Insert*-based neighborhood leads to better local exploitation than the *Interchange* neighborhood. It would be worthwhile to consider investigating other neighborhood structures with our symmetry-breaking method for SRFLP algorithms. Furthermore, we can see from Table IX that the p-values are all less than 0.05. These results reveal that the main search components are significant for enhancing the performance of KMPG. We also draw the related violin plots in Fig. 15. Apparently, KMPG can obtain more consistent results than the three variants.

2) Contributions of Main Search Components: To further assess the contributions of the main search components for

TABLE VIII
COMPARISONS OF VNS-LS3, MSA AND KMPG ON THE RANLARGE SET

Instance	VNS-LS3			MSA			KMPG (this work)			
	BEST	*AVG*	*SD*	*BEST*	*AVG*	*SD*	*BEST*	*AVG*	*SD*	T(s)
L1050	3652253111.5	3653599680.5	794299.4	***3649886413.5**	3650216755.7	177492.2	3649886570.5	**3650019879.0**	**58804.1**	3252.7
L1100	4267978583	4269654379.8	989581.3	4264719909	4265092238.0	326248.7	***4264699102**	**4264937600.2**	**121001.7**	3465.1
L1150	4783709999.5	4785589142.2	1044003.7	4780194693.5	4780806644.3	305159.7	***4780188161.5**	**4780690360.7**	**139226.5**	3515.8
L1200	5436307666.5	5438482630.4	995662.5	***5432873749.5**	5433486809.9	404550.2	5432876337.5	**5433346735.8**	**197955.4**	3539.1
L1250	6036980599.5	6039825925.2	1396372.1	6033618049.5	6034080821.1	266008.9	***6033543601.5**	**6033871375.4**	**132482.2**	3468.9
L1300	6904829425.5	6907410707.6	1572324.3	6899176958.5	6900498196.9	685456.7	***6899176513.5**	**6899566161.8**	**405966.9**	3822.7
L1350	7675172460	7677922118.7	1577113.1	7670259077	7670914983.6	349804.0	***7669901243**	**7670530490.3**	**282670.3**	3659.1
L1400	8545759893	8548988557.8	1671990.6	***8539721102**	8540279120.3	309799.6	8539828377	**8540164010.6**	**129739.0**	3275.6
L1450	9665703631.5	9670338419.3	2014737.9	***9660010880.5**	9661174962.0	726253.5	9660242222.5	**9660731379.0**	**249577.5**	3708.8
L1500	11033752227.5	11037714178.5	1969400.8	11026341357.5	11027124527.5	469127.9	***11026245477.5**	**11026822635.4**	**182347.5**	3425.6
L1550	11912003713.5	11917653668.7	2205007.9	11904973241.5	11906000461.8	833114.2	***11904924505.5**	**11905351094.2**	**210377.0**	3734.8
L1600	13481309883.5	13485844536.6	2459184.7	***13471833322.5**	13472960008.5	652443.4	13471891763.5	**13472271299.8**	**190160.4**	3722.1
L1650	14497146697	14501954272.8	2663528.2	14486392919	14488145456.0	1003295.6	***14486152642**	**14487014957.8**	**449324.7**	4159.2
L1700	15795655978.5	15799940029.5	2186238.1	15784470479.5	15785694873.1	864671.4	***15784233191.5**	**15784890053.2**	**275940.7**	4588.5
L1750	16813954935	16820330045.9	3145316.3	16804575189	16806214588.0	945783.7	***16804197725**	**16805170276.0**	**401763.6**	4872.9
L1800	18396571638	18401930006.0	2601497.2	18384379806	18386199663.3	895394.7	***18384140209**	**18385034553.9**	**359149.8**	5399.0
L1850	20152085395	20158070655.6	2505175.2	20141394830	20142991949.8	977860.4	***20140843753**	**20141773315.9**	**453570.8**	5402.5
L1900	21874225895	21880275442.8	3369975.4	21859348121	21861997946.5	1117974.6	***21858740700**	**21860417333.8**	**662456.6**	5367.7
L1950	23145487034.5	23150221230.4	2860263.4	23127328719.5	23130194933.8	1504215.0	***23126765575.5**	**23127911906.3**	**539894.6**	5316.6
L2000	25541775753.5	25552878153.1	4591220.2	25529989111.5	25531913514.6	1304330.7	***25529463596.5**	**25530358759.6**	**397758.0**	5400.6
NoB	0	0	0	5	0	0	**15**	**20**	**20**	
p-value	0.000	0.000	0.000	0.025	0.000	0.000				

* The strictly best values of *BEST* among the three algorithms are also reported as BKS for the 20 instances.

TABLE IX
STATISTICAL SIGNIFICANCE OF COMPARISONS BETWEEN KMPG_V1, KMPG_V2, KMPG_V3 AND KMPG

Indicator	p-value for algorithm pair					
	KMPG vs. V1	Sig.	KMPG vs. V2	Sig.	KMPG vs. V3	Sig.
BEST	0.002	Yes	0.001	Yes	0.000	Yes
AVG	0.000	Yes	0.000	Yes	0.000	Yes
SD	0.000	Yes	0.000	Yes	0.000	Yes

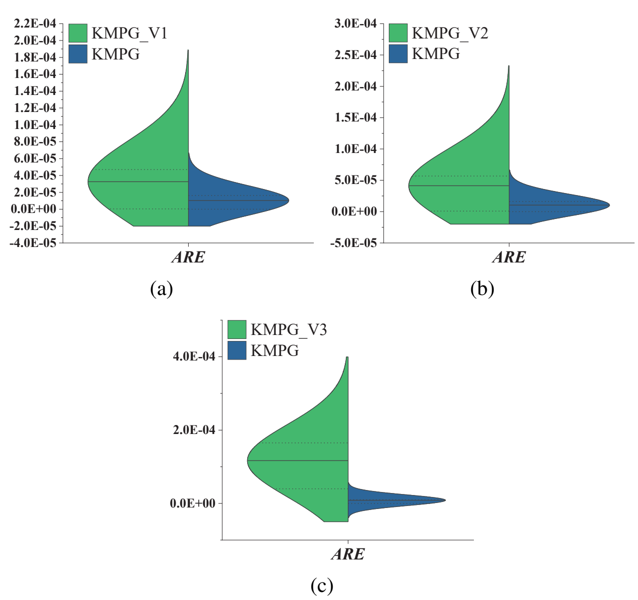


Fig. 15. Violin plots for the solutions of KMPG (V1-V3) and KMPG on the 30 instances in the Palubeckis set with $N = 310$ to 1000.

the performance of KMPG, we perform an analysis on the relationships between the three variants and KMPG. For this, the objective values obtained by a given variant and KMPG are treated as two variables. Thereby, the degree of correlation between them can be captured. Obviously, a larger degree of correlation implies a larger contribution of the corresponding component for the performance of KMPG. We give the related scatter plots and fit lines in Fig. 16.

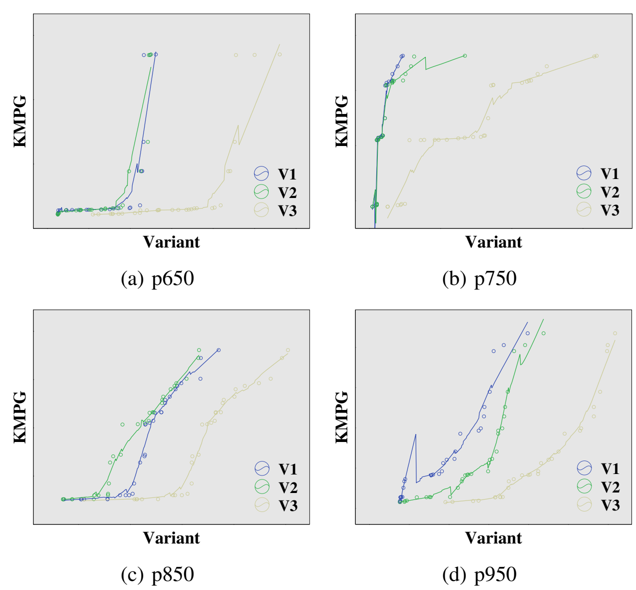


Fig. 16. Scatter plots and fit lines for KMPG (V1-V3) and KMPG on the instances p650, p750, p850 and p950 in the Palubeckis set.

Fig. 16 shows that there exist dependencies between the three variants and KMPG in terms of the considered instances, confirming that the performance of KMPG is closely related to its main search components. In addition, the gradients of the fit lines for KMPG_V1 and KMPG_V2 are relatively larger than those of KMPG_V3, which means that the proposed symmetry-breaking approach and k-medoids clustering based crossover contribute more significant than the simulated annealing based local search. Considering the symmetry features of SRFLP, KMPG can be afforded more opportunities to find promising solutions with less computational effort. Therefore, one can try to extend relevant works on inherent symmetric features and machine learning guided search components for solving other combinatorial operation problems.

E. Discussions

In this section, we provide discussions on KMPG concerning the following issues: 1) the performance of KMPG with other versions of k-medoids clustering guided crossover operators; 2) the performance of KMPG with a variable neighborhood descent (VND) based local search; and 3) the performance of KMPG with a long CPU time. Thereby, we can obtain useful insights into the impacts of the k-medoids clustering based crossover operator and the simulated annealing based local search procedure, and assess the potential of KMPG for further resulting advances[4].

1) Performance of KMPG Considering Other Versions of K-medoids Clustering Based Crossovers: We introduce three variants of KMPG as follows. 1) KMPG_C1 is a KMPG variant where we select two parent solutions from one cluster such that they have the smallest similarity among all solution pairs of the cluster; 2) KMPG_C2 is a KMPG variant where we generate the offspring solution using all solutions of one cluster; and 3) KMPG_C3 is a KMPG variant where we generate two offspring solutions associated with two resulting clusters at each generation of KMPG. To make a fair comparison, we run KMPG and the three variants with the same CPU time of 3600s. Results on the 30 instances in the Palubeckis set with $N = 310$ to 1000 are summarized in Fig. 17. In addition, the p-values for the three algorithm pairs are given in Table X. Moreover, we draw the trends of T(s) in Fig. 18.

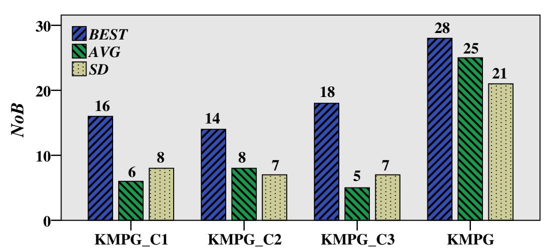


Fig. 17. Summary of comparisons of KMPG (C1-C3) and KMPG.

TABLE X
STATISTICAL SIGNIFICANCE OF COMPARISONS BETWEEN KMPG_C1, KMPG_C2, KMPG_C3 AND KMPG

Indicator	*p*-value for algorithm pair					
	KMPG vs. C1	Sig.	KMPG vs. C2	Sig.	KMPG vs. C3	Sig.
BEST	0.001	Yes	0.000	Yes	0.008	Yes
AVG	0.000	Yes	0.000	Yes	0.000	Yes
SD	0.003	Yes	0.000	Yes	0.006	Yes

As Fig. 17 shows, for the values of *NoB*, KMPG reports the values of 28, 25, and 21 out of the 30 instances with respect to *BEST*, *AVG*, and *SD* that are much better than those of the three variants. Meanwhile, we find from Table X and Fig. 18 that the p-values are all less than 0.05 and the four algorithms have similar trends of T(s). These results prove the benefit of the k-medoids clustering technique for improving KMPG, which can be due to the fact that the k-medoids clustering method is able to help the crossover operator to pass pertinent genetic knowledge from parent solutions to offspring solutions.

[4]Details of discussions on KMPG with respect to the three issues are available at http://dx.doi.org/10.13140/RG.2.2.22760.39683

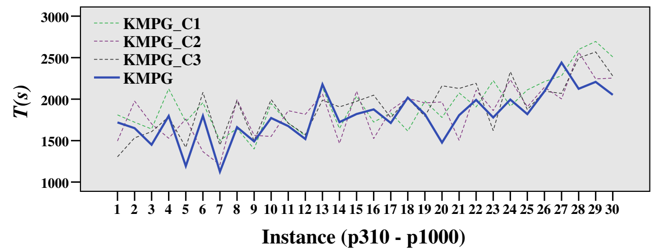


Fig. 18. Trends of T(s) for KMPG (C1-C3) and KMPG.

As such, the underlying idea could inspire effective search methods for SRFLP or other permutation problems.

2) Performance of KMPG Considering VND Based Local Search: We introduce a variant of KMPG using VND based local search, namely, KMPG_VND. KMPG_VND is the same as KMPG except that the simulated annealing procedure is replaced by the VND method [35]. KMPG_VND adopts two neighborhoods, i.e., the *Insert*-based neighborhood and the *Interchange*-based neighborhood. Note that in KMPG_VND the objective gain methods of [2] are used as well. In [3], the VND-based local search is used in VNS-LS3 which helps the algorithm to obtain high-quality solutions of SRFLP. Thus, the VND method can be seen as one of the most effective local search methods for SRFLP. Accordingly, the comparisons of KMPG_VND and KMPG are useful to show the effectiveness and reasonability of the simulated annealing procedure. We run KMPG and KMPG_VND with the same CPU time of 3600s. For the 30 instances in the Palubeckis set with $N =$ 310 to 1000, we give the trends of three introduced indicators in Fig. 19.

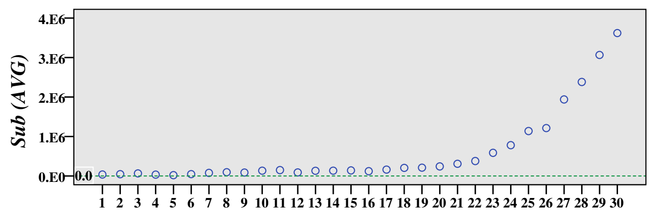
From Fig. 19, one observes that KMPG performs better than KMPG_VND associated with the trends of *Sub*(*AVG*) and *SD*, indicating that the simulated annealing procedure of KMPG is beneficial to enhance the search performance. On the other hand, KMPG has smaller values of T(s) for almost all these instances, which shows the efficiency of adopting the simulated annealing procedure.

3) Performance of KMPG With a Long CPU Time: We examine whether KMPG can still find better solutions if it is run with a long CPU time. For this, we run KMPG with the CPU time of 7200s (denoted as KMPG_L). For the 30 instances in the Palubeckis set with $N = 310$ to 1000, the trends of the three indicators are given in Fig. 20.

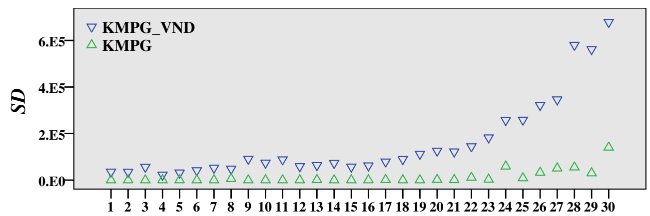
As shown in Fig. 20, the values of *Sub*(*AVG*) are no less than 0 for relatively large-scale instances among the 30 instances, whereas KMPG_L has smaller values of *SD* for most instances. In addition, KMPG_L corresponds to larger T(s) for each instance, confirming the reasonability of the above solution improvements. Therefore, KMPG is capable of finding even better results if it is given more CPU time.

V. CONCLUSION

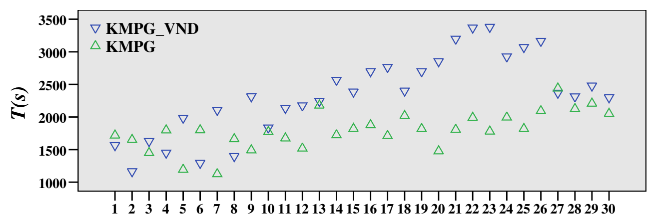
We proposed a k-medoids memetic permutation group algorithm (KMPG) to solve the well-known single row facility layout problem (SRFLP). To the best of our knowledge, this is the first memetic algorithm based on the permutation group theory for SRFLP. KMPG combined a permutation



(a) **Instance (p310 - p1000)**

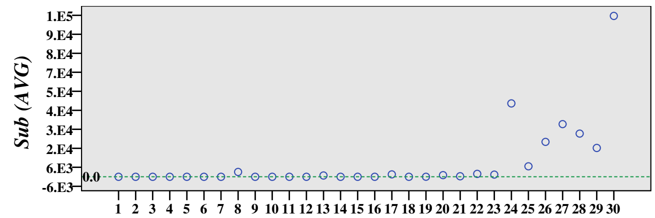


(b) **Instance (p310 - p1000)**

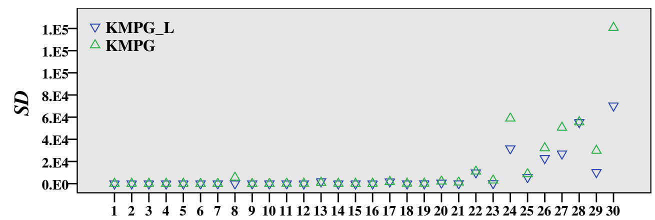


(c) **Instance (p310 - p1000)**

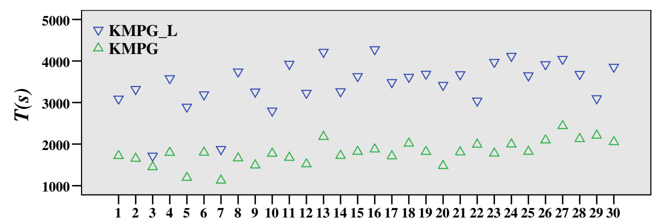
Fig. 19. Trends of quality indicators for KMPG_VND and KMPG on the 30 instances in the Palubeckis set with $N = 310$ to 1000. Note that in (a) we use $Sub(AVG) = AVG(\text{KMPG\_VND}) - AVG(\text{KMPG})$ for each instance.



(a) **Instance (p310 - p1000)**



(b) **Instance (p310 - p1000)**



(c) **Instance (p310 - p1000)**

Fig. 20. Trends of quality indicators for KMPG_L and KMPG on the 30 instances in the Palubeckis set with $N = 310$ to 1000. Note that in (a) we use $Sub(AVG) = AVG(\text{KMPG}) - AVG(\text{KMPG\_L})$ for each instance.

group based symmetry-breaking method to reduce the solution space of SRFLP, a k-medoids clustering guided crossover to produce meaningful offspring solutions that are improved by a simulated annealing based optimization procedure, and a distance-and-quality population management strategy to ensure a suitable diversity of the population. Complementing each other, these search components worked together to achieve a desirable balance of exploration and exploitation of the search process of KMPG.

Experimental results on 113 benchmark instances, including 93 traditional instances and 20 newly introduced large-scale instances, showed that KMPG performed better than the state-of-the-art methods. In particular, KMPG attained all but one previous best known upper bounds and found new upper bounds for 33 out of the 93 traditional instances. We presented for the first time upper bounds on the 20 new instances, which would be useful for performance assessment of new SRFLP algorithms.

The idea of integrating search components based on the permutation group and the k-medoids clustering is of generic scientific applicability. It would be interesting to investigate its application to more permutation problems, such as permutation flow-shop scheduling problem (PFSP), quadratic assignment problem (QAP), and traveling salesman problem (TSP). Moreover, the permutation group theory accounts for many basic solution features in discrete search domains, the findings of this work could be extended to other solution methods for combinatorial optimization problems.

ACKNOWLEDGMENT

We thank Dr. Palubeckis for sharing the problem instances and making the codes of MSA and VNS-LS3 publicly available, which helped us to sufficiently conduct our experiments. We are grateful to the reviewers for their comments and suggestions.

REFERENCES

[1] G. L. Cravo and A. R. Amaral, “A grasp algorithm for solving large-scale single row facility layout problems,” *Computers & Operations Research*, vol. 106, pp. 49–61, 2019.

[2] G. Palubeckis, “Single row facility layout using multi-start simulated annealing,” *Computers & Industrial Engineering*, vol. 103, pp. 1–16, 2017.

[3] ——, “Fast local search for single row facility layout,” *European Journal of Operational Research*, vol. 246, no. 3, pp. 800–814, 2015.

[4] M. Rubio-Sánchez, M. Gallego, F. Gortázar, and A. Duarte, “Grasp with path relinking for the single row facility layout problem,” *Knowledge-Based Systems*, vol. 106, pp. 1–13, 2016.

[5] D. M. Simmons, “One-dimensional space allocation: an ordering algorithm,” *Operations Research*, vol. 17, no. 5, pp. 812–826, 1969.

[6] T. S. Moh, S. L. Hakimi, and W. M. Moh, “Almost linear time optimization for single-row floorplanning,” in *Proceedings of the 37th Midwest Symposium on Circuits and Systems*, 1994.

[7] M. S. Kumar, M. N. Islam, N. Lenin, D. Vignesh Kumar, and D. Ravindran, “A simple heuristic for linear sequencing of machines in layout design,” *International Journal of Production Research*, vol. 49, no. 22, pp. 6749–6768, 2011.

[8] P. Hungerländer and F. Rendl, “A computational study and survey of methods for the single-row facility layout problem,” *Computational Optimization and Applications*, vol. 55, no. 1, pp. 1–20, 2013.

[9] B. Keller and U. Buscher, “Single row layout models,” *European Journal of Operational Research*, vol. 245, no. 3, pp. 629–644, 2015.

[10] A. R. Amaral and A. N. Letchford, “A polyhedral approach to the single row facility layout problem,” *Mathematical Programming*, vol. 141, no. 1, pp. 453–477, 2013.

[11] M. F. Anjos and A. Vannelli, "Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes," *INFORMS Journal on Computing*, vol. 20, no. 4, pp. 611–617, 2008.

[12] M. F. Anjos and G. Yen, "Provably near-optimal solutions for very large single-row facility layout problems," *Optimization Methods & Software*, vol. 24, no. 4-5, pp. 805–817, 2009.

[13] A. R. Amaral, "A new lower bound for the single row facility layout problem," *Discrete Applied Mathematics*, vol. 157, no. 1, pp. 183–190, 2009.

[14] H. Samarghandi and K. Eshghi, "An efficient tabu algorithm for the single row facility layout problem," *European Journal of Operational Research*, vol. 205, no. 1, pp. 98–105, 2010.

[15] R. Kothari and D. Ghosh, "Tabu search for the single row facility layout problem using exhaustive 2-opt and insertion neighborhoods," *European Journal of Operational Research*, vol. 224, no. 1, pp. 93–100, 2013.

[16] D. Datta, A. R. Amaral, and J. R. Figueira, "Single row facility layout problem using a permutation-based genetic algorithm," *European Journal of Operational Research*, vol. 213, no. 2, pp. 388–394, 2011.

[17] R. Kothari and D. Ghosh, "A scatter search algorithm for the single row facility layout problem," *Journal of Heuristics*, vol. 20, no. 2, pp. 125–142, 2014.

[18] X. Ning and P. Li, "A cross-entropy approach to the single row facility layout problem," *International Journal of Production Research*, vol. 56, no. 11, pp. 3781–3794, 2018.

[19] J. Guan and G. Lin, "Hybridizing variable neighborhood search with ant colony optimization for solving the single row facility layout problem," *European Journal of Operational Research*, vol. 248, no. 3, pp. 899–909, 2016.

[20] C. Chen and L. K. Tiong, "Using queuing theory and simulated annealing to design the facility layout in an agv-based modular manufacturing system," *International Journal of Production Research*, vol. 57, no. 17, pp. 5538–5555, 2019.

[21] A. Tayal and S. P. Singh, "Integrating big data analytic and hybrid firefly-chaotic simulated annealing approach for facility layout problem," *Annals of Operations Research*, vol. 270, no. 1, pp. 489–514, 2018.

[22] M. Z. Allahyari and A. Azab, "Mathematical modeling and multi-start search simulated annealing for unequal-area facility layout problem," *Expert Systems with Applications*, vol. 91, pp. 46–62, 2018.

[23] A. Cayley, "Note on the theory of permutations," *Philosophical Magazine*, vol. 34, no. 232, pp. 527–529, 1849.

[24] P. J. Cameron, *Permutation groups*. Cambridge University Press, 1999.

[25] B. Qian, L. Wang, R. Hu, W.-L. Wang, D.-X. Huang, and X. Wang, "A hybrid differential evolution method for permutation flow-shop scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 38, no. 7-8, pp. 757–777, 2008.

[26] Z.-Q. Zhang, B. Qian, R. Hu, H. P. Jin, and L. Wang, "A matrix-cube-based estimation of distribution algorithm for the distributed assembly permutation flow-shop scheduling problem," *Swarm and Evolutionary Computation*, early access, Oct. 1, 2021.

[27] L. Tang, Y. Zhao, and J. Liu, "An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 209–225, 2013.

[28] L. Tang, Y. Dong, and J. Liu, "Differential evolution with an individual-dependent mechanism," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 4, pp. 560–574, 2014.

[29] L. Tang and X. Wang, "A hybrid multiobjective evolutionary algorithm for multiobjective optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 20–45, 2012.

[30] X. Chen, Y.-S. Ong, M. H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5, pp. 591–607, 2011.

[31] Y. Zhou, J. K. Hao, Z. Fu, Z. Wang, and X. Lai, "Variable population memetic search: A case study on the critical node problem," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 1, pp. 187–200, 2021.

[32] J.-K. Hao, "Memetic algorithms in discrete optimization," in *Handbook of Memetic Algorithms*. Springer, 2012, pp. 73–94.

[33] L. Tang and Y. Meng, "Data analytics and optimization for smart industry," *Frontiers of Engineering Management*, vol. 8, no. 2, pp. 157–171, 2021.

[34] P. Arora, Deepali, and S. Varshney, "Analysis of *k*-means and *k*-medoids algorithm for big data," *Procedia Computer Science*, vol. 78, pp. 507–512, 2016.

[35] X. Lai and J.-K. Hao, "Iterated variable neighborhood search for the capacitated clustering problem," *Engineering Applications of Artificial Intelligence*, vol. 56, pp. 102–120, 2016.

APPENDIX

PROOFS OF PROPOSED THEOREMS 1 TO 5

In the following, we detail the proofs of Theorems 1 to 5 proposed in Sections II and III.

Theorem 1: Suppose that $\boldsymbol{\pi}$ is an even permutation. Then $\boldsymbol{\alpha}$ is an odd permutation if and only if $N \in \{\mathcal{K}|(\mathcal{K} = 4k + 2) \vee (\mathcal{K} = 4k + 3), \forall k \in \mathbb{N}\}$.

Proof: Since $\boldsymbol{\pi}$ is an even permutation, one can represent it by employing a product of transpositions with even number, that is, $\boldsymbol{\pi} = (\mathcal{T}_{11}, \mathcal{T}_{12}), ..., (\mathcal{T}_{2l,1}, \mathcal{T}_{2l,2})$ $(l \in \mathbb{N})$ according to *Definition 1*. Then, we can transform $\boldsymbol{\pi}$ into $\boldsymbol{\alpha}$, that is, $\boldsymbol{\alpha} = (\mathcal{T}_{11}, \mathcal{T}_{12}), ..., (\mathcal{T}_{2l,1}, \mathcal{T}_{2l,2})(1, N)(2, N-1), ..., (\lfloor N/2 \rfloor, \lceil N/2 \rceil + 1)$. If $\boldsymbol{\alpha}$ is an odd permutation, then $2l + \lfloor N/2 \rfloor$ is an odd number. We thus have $N \in \{\mathcal{K}|(\mathcal{K} = 4k + 2) \vee (\mathcal{K} = 4k + 3), \forall k \in \mathbb{N}\}$. □

Theorem 2: Suppose that $\boldsymbol{\pi} \in A_N$ satisfies $N \in \{\mathcal{K}|(\mathcal{K} = 4k + 2) \vee (\mathcal{K} = 4k + 3), \forall k \in \mathbb{N}\}$. Then $\neg\exists \boldsymbol{\alpha} \in A_N : \boldsymbol{\alpha} = inverse(\boldsymbol{\pi})$ holds.

Proof: We first assume that there exists $\boldsymbol{\alpha} \in A_N$ such that $\boldsymbol{\alpha} = inverse(\boldsymbol{\pi})$. Then, according to *Theorem 1*, one can obtain $\boldsymbol{\alpha}$ from $\boldsymbol{\pi}$ by using transpositions $(1, N), (2, N-1), ..., (\lfloor N/2 \rfloor, \lceil N/2 \rceil + 1)$. Obviously, $\lfloor N/2 \rfloor$ is always odd and $\boldsymbol{\alpha}$ is thus an odd permutation, which contradicts the initial assumption that there exists such a permutation $\boldsymbol{\alpha}$. □

Theorem 3 (Scenario 1): Suppose that $\boldsymbol{\pi} \in A_N$ satisfies $N \in \{\mathcal{K}|(\mathcal{K} = 4k + 2) \vee (\mathcal{K} = 4k + 3), \forall k \in \mathbb{N}\}$. Then $\exists \boldsymbol{\beta} \in \Pi_N \backslash A_N : f(\boldsymbol{\pi}) = f(\boldsymbol{\beta})$ holds.

Proof: From *Theorem 1*, the inverse form of each even permutation in Π_N is an odd permutation. And, as *Theorem 2* shows, any pairs of solutions in A_N do not display an inverse relationship between them. In this case, each even permutation in A_N has a corresponding odd permutation in $\Pi_N \backslash A_N$ with the same objective function. Therefore, we have $\exists \boldsymbol{\beta} \in \Pi_N \backslash A_N : f(\boldsymbol{\pi}) = f(\boldsymbol{\beta})$, $\forall \boldsymbol{\pi} \in A_N$, with respect to $N \in \{\mathcal{K}|(\mathcal{K} = 4k + 2) \vee (\mathcal{K} = 4k + 3), \forall k \in \mathbb{N}\}$. □

Theorem 4 (Scenario 2): Suppose that $\boldsymbol{\pi}^{\mathcal{L}} \in A_N^{\mathcal{L}}$ satisfies $N \in \{\mathcal{K}|(\mathcal{K} = 4k + 4) \vee (\mathcal{K} = 4k + 5), \forall k \in \mathbb{N}\}$. Then $\exists \boldsymbol{\beta}^{\mathcal{L}} \in \Pi_N^{\mathcal{L}} \backslash A_N^{\mathcal{L}} : f(\boldsymbol{\pi}) = f(\boldsymbol{\beta})$ holds.

Proof: First, it is apparent that there are $N!/4$ permutations in both sets $C_N \subset A_N$ and $D_N \subset \Pi_N \backslash A_N$ regarding $A_N^{\mathcal{L}}$. And, each permutation in C_N (or D_N) has an inverse form in $A_N \backslash C_N$ (or $\Pi_N \backslash \{A_N \cup D_N\}$). From *Theorem 2*, we have $\boldsymbol{\eta}^{\mathcal{L}} \neq inverse(\boldsymbol{\xi}^{\mathcal{L}})$ and hence $\boldsymbol{\eta} \neq inverse(\boldsymbol{\xi})$ $(\forall \boldsymbol{\eta}, \boldsymbol{\xi} \in C_N)$ and, likewise, $\boldsymbol{\eta}' \neq inverse(\boldsymbol{\xi}')$ $(\forall \boldsymbol{\eta}', \boldsymbol{\xi}' \in D_N)$. Besides, we have $\boldsymbol{\gamma} \neq inverse(\boldsymbol{\rho})$ ($\forall \boldsymbol{\gamma} \in C_N$ and $\forall \boldsymbol{\rho} \in D_N$) based on *Theorem 1*. We can therefore conclude the theorem. □

Theorem 5: For any facility $n_x \in \{1, ..., N\}$ $(N > 3)$, there are $N!/4$ permutations in A_N (Scenario 1) or $A_N^{\mathcal{L}}$ (Scenario 2) such that $rp(n_x, \boldsymbol{\pi}) \leq \lceil N/2 \rceil$ ($\forall \boldsymbol{\pi} \in A_N$ or $A_N^{\mathcal{L}}$).

Proof: For *Scenario 1*, we first assume that there exist $N!/4 + n_0$ $(n_0 > 0)$ permutations in A_N such that each $\boldsymbol{\pi}$ of them satisfies $rp(n_x, \boldsymbol{\pi}) \leq \lceil N/2 \rceil$. Then, we transform each $\boldsymbol{\pi}$ as follows: $interchange(\boldsymbol{\pi}, rp(n_x, \boldsymbol{\pi}), \lceil N/2 \rceil + 1) \rightarrow interchange(\boldsymbol{\pi}, \lceil N/2 \rceil + 1, \lceil N/2 \rceil + 2)$. We can obtain $N!/4 + n_0$ different even permutations and each $\boldsymbol{\pi}'$ of them satisfies $rp(nx, \boldsymbol{\pi}') > \lceil N/2 \rceil$, so there will be $2 \cdot (N!/4 + n_0) > N!/2$ permutations in A_N. This does not satisfy *Theorem 3*, and, thus, contradicts the initial assumption. If we assume a permutation number $N!/4 - n_0$, the same conclusion can be drawn. Likewise, the conclusion for *Scenario 2* can be proved. □