An adaptive memetic algorithm for the bidirectional loop layout problem

Wen Sun [a,b], Jin-Kao Hao [c], Wenlong Li [a], Qinghua Wu [d,*]

[a] *School of Cyber Science and Engineering, Southeast University, 2 Road Southeast University, 211189 Nanjing, China*

[b] *Frontiers Science Center for Mobile Information Communication and Security, Southeast University, 2 Road Southeast University, 211189 Nanjing, China*

[c] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[d] *School of Management, Huazhong University of Science and Technology, 1037 Road Luoyu, 430074 Wuhan, China*

Knowledge-Based Systems, https://doi.org/10.1016/j.knosys.2022.110002, 2022

Abstract

Given a set of facilities, a set of positions arranged in a loop configuration and a flow cost matrix, the bidirectional loop layout problem (BLLP) is to find a facility-to-position assignment with the minimum sum of the products of flow costs and facility distances. The flow cost between two facilities is the unit cost of transporting items between two facilities, and the distance between two positions is shorter in the clockwise or counterclockwise direction in the loop. The BLLP is a relevant model for manufacturing systems; however, as the problem is known to be NP-hard, solving the problem is computationally challenging. This paper presents a memetic algorithm for BLLP that features an adaptive crossover selection and a dedicated 3-phase local search. Experiments of the proposed algorithm are conducted on 65 benchmarks, demonstrating a competitive performance compared with the current best algorithms. Additional experiments are conducted to study the impacts of the essential components of the algorithm.

Keywords: Memetic algorithm; heuristics; quadratic optimization; facility layout.

* Corresponding author.
Email address: qinghuawu1005@gmail.com (Qinghua Wu).

1 Introduction

Given a flow c_{ij} from facility i to facility j for all $i, j \in \{1, \ldots, n\}$ and a distance d_{qp} between positions q and p for all $q, p \in \{1, \ldots, n\}$, the popular quadratic assignment problem (QAP) is to determine a minimal cost assignment of n facilities to n positions. Let Π denote the set of permutation functions $\pi : \{1, \ldots n\} \rightarrow \{1, \ldots, n\}$. Then, NP-hard QAP [1] can be formulated as

$$min_{\pi \in \Pi} f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} d_{\pi_i \pi_j} \tag{1}$$

where c and d are the flow and distance matrices, respectively, and π_i represents the position chosen for facility i in permutation $\pi \in \Pi$. Then, the objective is to find a permutation π^* in Π that minimizes the sum of the products of the flow and distance matrices, i.e., $f(\pi^*) \leq f(\pi), \forall \pi \in \Pi$.

One typical application of the QAP is in flexible manufacturing systems, where an automated guided vehicle transports materials in a loop layout path, and one wants to assign the facilities to suitable positions to minimize the transportation cost. The transportation cost refers to the sum of the flow times the distance of each pair of facilities. Other applications include backboard wiring in electronics, design of typewriter keyboards, campus planning, analysis of chemical reactions for organic compounds, and balancing turbine runners [2].

The bidirectional loop layout problem (BLLP) can be considered as a variant of the QAP with the following considerations [3]: all materials enter into and leave from the manufacturing system at the Load/Unload (LUL) station (the first position) and the first facility is placed at the first position; a closed material handling loop goes through each facility exactly once; an automated guided vehicle transports materials along the loop in one or the other direction, depending on which route is shorter. The flow and distance matrices are symmetric, implying that the flows and distances from i to j and from j to i are the same. The BLLP can be expressed as:

$$min_{\pi \in \Pi} f(\pi) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} d_{\pi_i \pi_j}, \tag{2}$$

subject to

$$\pi_1 = 1, \tag{3}$$

$$d_{\pi_i\pi_j} = \begin{cases} min\{\sum_{k=\pi_i}^{\pi_j-1} d_{kk+1}, \sum_{k=\pi_j}^{n-1} d_{kk+1} + d_{1n} + \sum_{k=1}^{\pi_i-1} d_{kk+1}\}, \ \pi_i < \pi_j; \\ min\{\sum_{k=\pi_j}^{\pi_i-1} d_{kk+1}, \sum_{k=\pi_i}^{n-1} d_{kk+1} + d_{1n} + \sum_{k=1}^{\pi_j-1} d_{kk+1}\}, \ \pi_i > \pi_j. \end{cases} \tag{4}$$

where constraint (3) ensures that the first facility is fixed in the first place, and constraint (4) indicates that $d_{\pi_i\pi_j}$ is the length of the shortest route between two positions (π_i and π_j) along the loop. In fact, the BLLP problem is a variant of the constrained facility layout problem [4,5].

Figure 1 shows an example of the BLLP with 9 positions and 9 facilities. Figure 1(a) presents the flow matrix between facilities, and Figure 1(b) presents the distance matrix whose values are either piling up clockwise or counterclockwise. For example, suppose $\pi_i = 1$, $\pi_j = 5$, the sum of $d_{12} + d_{23} + d_{34} + d_{45}$, i.e., 20, is less than that of $d_{19} + d_{98} + d_{87} + d_{76} + d_{65}$, i.e., 25. Therefore, the distance d_{15} between π_i and π_j equals 20. For any position p, let i $(i = 0, 1, 2 \ldots n-1)$ be the number of positions between p and q in the clockwise (or counterclockwise) direction. As positions are placed in a loop, the distance d_{pq} from position p to q first increases with the increase of i and then decreases with the increase of i after reaching the maximum value. For example, suppose $p = 1$, $q \leq 5$, and d_{pq} is equal to $\{6, 10, 13, 20\}$, which increases with the increase of i. On the contrary, when $q \geq 6$, d_{pq} is equal to $\{20, 14, 12, 5\}$, which decreases with the increase of i. Figure 1(c) and 1(d) show a random permutation with the objective value of 2094 and an optimal permutation with the objective value of 1797, respectively.

The BLLP was first formulated by Bozer and Rim [3], where they also proved the NP-hardness of the problem. Consequently, it is computationally challenging to solve this problem.

In practice, BLLP arises naturally in circumstances such as the design of flexible manufacturing systems (FMS) [6], tool indexing [7,8,9] and broadcast scheduling [10].

In the last few decades, the QAP has been studied intensively, while BLLP has received much less attention. Bozer and Rim [3] presented an exact solution procedure for the BLLP based on the branch-and-bound (B&B) method, which outperformed the best algorithm available for general QAPs as the problem size increases. Given the NP-hard nature of the BLLP, the computational time required by exact algorithms increases exponentially, limiting their practical use. To tackle large and difficult BLLP instances, Palubeckis [11] proposed five heuristic approaches: one simulated annealing algorithm (SA) and four 2-phase algorithms that adopt SA in the first phase and variable neighborhood search (VNS) in the second phase. Specifically, the four 2-phase algorithms are SA-VNS-0-0 (both SA and VNS use the swap oper-

$c =$

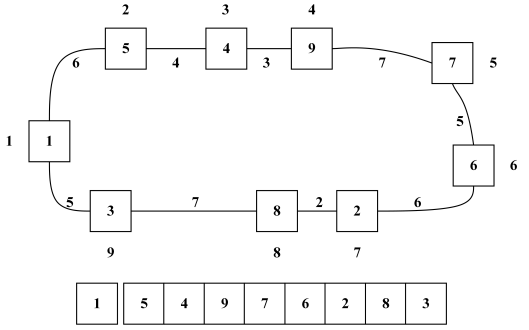
0	0	7	1	5	0	9	8	8
0	0	3	9	1	0	8	3	2
7	3	0	2	9	0	6	5	7
1	9	2	0	9	4	8	10	9
5	1	9	9	0	3	3	2	6
0	0	0	4	3	0	4	4	2
9	8	6	8	3	4	0	2	3
8	3	5	10	2	4	2	0	7
8	2	7	9	6	2	3	7	0

(a) A flow matrix

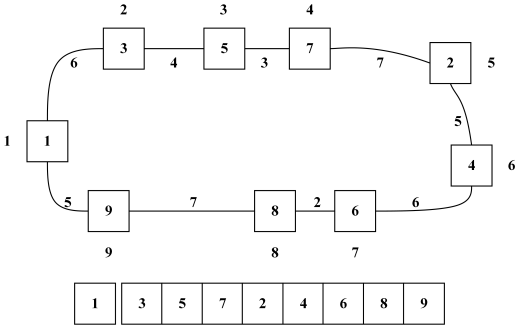
$d =$

0	6	10	13	**20**	**20**	14	12	5
6	0	4	7	14	19	**20**	18	11
10	4	0	3	10	15	**21**	**22**	15
13	7	3	0	7	12	18	**20**	18
20	14	10	7	0	5	11	13	**20**
20	19	15	12	5	0	6	8	15
14	**20**	**21**	18	11	6	0	2	9
12	18	**22**	**20**	13	8	2	0	7
5	11	15	18	**20**	15	9	7	0

(b) A distance matrix



(c) A random permutation with f=2094



(d) An optimal permutation with f=1797

Fig. 1. An example of the BLLP with its flow matrix, distance matrix, a random permutation, and an optimal permutation

ator only), SA-VNS-0-1 (SA uses the swap operator while VNS adopts the insertion operator), SA-VNS-1-0 (SA adopts the insertion operator while VNS adopts the swap operator), and SA-VNS-1-1 (both SA and VNS adopt the insertion operator). These algorithms produced state-of-the-art results on different benchmark instances. Thus, they are used as our main reference algorithms for the performance assessment in this study.

We observe that unlike the popular QAP for which many algorithms have been proposed, there are few practical algorithms for effectively solving the BLLP. The B&B exact algorithm in [3] suffers from the problem of inevitable exponential time complexity. It was only tested on very small instances ($n \leq 10$), and cannot be used in practice on large instances. The SA heuristics were shown to be much more successful in solving many instances up to $n = 300$. Still, these methods lack stability, especially on large-scale instances (e.g., $n > 160$), suggesting that there is still room for further improving their results. Thus, it is of interest to enhance the toolbox for effectively solving BLLP. Finally, we observe that the current heuristic algorithms for the BLLP are based on the single trajectory SA approach, ignoring the powerful population-based memetic framework. Indeed, it is known that hybrid memetic algorithms are among the best performing methods for several related facility layout problems, such as the QAP [12], single row facility layout problem [13], and distributed permutation flowshop scheduling problem [14]. In this study, we investigate for the first time

the potential of the memetic search framework for solving the BLLP.

We present the first adaptive memetic algorithm (AMA) for the BLLP that features an adaptive strategy for dynamically choosing the most suitable crossover among three crossover operators and a 3-phase local search that can explore distant local optima. The algorithm also integrates an elite population initialization procedure and quality-based pool updating procedure. We assess the proposed algorithm on a total of 65 benchmark instances, including 55 existing instances with up to 300 facilities ($n \leq 300$) and 10 new large instances with $310 \leq n \leq 400$. We report 14 new best results and 51 equivalent best results, which can be used to assess new BLLP algorithms. We will also make the code of the AMA algorithm publicly available, which can benefit future research on the BLLP and help solve related practical problems.

The rest of the paper is structured as follows. Section 2 illustrates the proposed algorithm. Section 3 provides computational results and comparisons. Section 4 analyzes the impacts of key components of the proposed algorithm. Conclusions and future work are discussed in the last section.

2 Memetic algorithm with adaptive crossover selection

Memetic algorithm (MA) is a general metaheuristic that combines population-based evolutionary search and single trajectory local search [15]. It aims to exploit the advantages offered by these two complementary search frameworks. In particular, MA uses a crossover to generate offspring solutions and applies a local search to perform an intensified search around each offspring solution. As such, MA favors the creation of a suitable balance between exploration and exploitation, which is necessary to locate high-quality solutions in the search space. As a general method, MA has been successfully adopted to solve various difficult optimization problems such as the quadratic assignment problem [16,17], the traveling salesman problem [18,19], scheduling problems [14,20], and many routing problems [21,22]. We introduce the first memetic algorithm applied to the BLLP.

2.1 Adaptive memetic algorithm

The proposed adaptive memetic algorithm (AMA) algorithm relies on the general MA method and has two distinct features. First, it adopts an adaptive crossover selection for offspring generation. Second, it employs a 3-phase local search for offspring improvement, which includes a dedicated

perturbation for local search diversification.

Algorithm 1 Adaptive memetic algorithm for the BLLP

Input: flow matrix c and distance matrix d, facility quantity n.
Output: the best permutation π^{global_best} found

1: $P \leftarrow Population\_initialization(c, d, n)$ /* Section 2.2 */
2: $\pi^{global\_best} \leftarrow argmin\{f(\pi^i)|\pi^i \in P\}$
3: **for** $m = 1, 2, 3$ **do**
4: $\gamma_m \leftarrow 1/3$ /* Initialize the m-th crossover probability γ_m, Section 2.3.2 */
5: **end for**
6: $\gamma \leftarrow \{\gamma_1, \gamma_2, \gamma_3\}$ /* Initialize crossover probability array γ, Section 2.3.2 */
7: **while** Stopping condition is not met **do**
8: Randomly select 2 permutations π^a, π^b in P as two parents
9: $(\pi^0, t) \leftarrow Adaptive\_Crossover(\pi^a, \pi^b, \gamma)$ /* Generate an offspring permutation π^0 by an adaptively selected crossover of type t according to crossover probability, Section 2.3 */
10: $\pi^{best} \leftarrow$ 3-$phase\_Local\_Search(\pi^0)$ /* Apply local search to improve the offspring permutation, Section 2.4 */
11: $(P, \pi^{global\_best}, \gamma) \leftarrow Update(P, \pi^{best}, \pi^{global\_best}, \gamma, t)$ /* Update the population, the global best permutation π^{global_best} and the crossover probability array, Section 2.5 */
12: **end while**
13: **return** π^{global_best}

The general architecture of the AMA algorithm is described in Algorithm 1, which contains an initialization procedure (line 1), adaptive crossover procedure (line 9), a local search procedure (line 10), and a procedure for information updating (line 11).

2.2 Population Initialization

The solutions of the initial population are generated in two steps: random facility-to-position assignment, followed by local improvement.

To generate an initial permutation randomly, each unassigned facility is assigned to a random unassigned position, except for the first facility that is always assigned to position 1. From this random assignment, we improve its quality by applying a local improvement procedure using a simple descent search. Let $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ be the current permutation; the descent search iteratively removes a facility i ($2 \leq i \leq n$) and inserts it after facility j ($2 \leq j \leq n$) to obtain a neighbor permutation better than the current permutation (see Section 2.4). This process is repeated until no improvement is possible.

If the improved solution is not a clone of any other permutation in the population, it will be inserted into the population. Otherwise, this solution is discarded. The population initialization procedure is iterated until the population is filled with p (population size) distinct permutations.

In addition to this two-step initialization, we also tested random initialization. Our experiments showed that for small and medium instances, the two initialization methods led to quite similar best results of the AMA algorithm. However, for large instance, the two-step initialization allows the algorithm to obtain more final best results and makes the algorithm more stable with better average results.

2.3 Crossovers and their adaptive application

One key to the success of population-based memetic approaches lies in crossover operators, which should be able to transfer meaningful components from parents to offspring while ensuring offspring diversity [23]. Different crossover operators inherit different parental genes, resulting in differences in offspring performance and population diversity. In addition, different instances may need the transmission of particular features, or even one instance needs the transmission of specific features at different search stages. As a result, a single crossover operator is not suitable for all instances and different search stages. Thus, it is an interesting strategy to employ an ensemble of crossover operators that are applied adaptively according to specific conditions, as shown in [18,24,25].

2.3.1 Crossovers

The proposed AMA algorithm applies adaptively, according to a probabilistic learning technique, three types of permutation crossover: one-point order crossover (OPOX) [26] (line 3, Algorithm 2), linear order crossover (LOX) [27] (line 6, Algorithm 2) and order-based crossover (OBX) [28] (line 9, Algorithm 2).

One-point order crossover's offspring inherits the positions of the left-half facilities of parent π^a and the relative order of the remaining facilities of parent π^b. Linear order crossover's offspring inherits the positions of the middle facilities of parent π^a and the relative order of the remaining facilities of parent π^b; Order-based crossover's offspring inherits the positions of $n/2$ random facilities of parent π^a and the relative order of the remaining facilities of parent π^b. These three crossover operators can inherit elite components of different parts of parent π^a and the relative order of the remaining facilities of parent π^b; thus, they can complement each other to fit

different situations. The details of these operators are as follows.

OPOX: The one-point order crossover is widely applied to problems related to the traveling salesman problem (TSP) (e.g., [18,29]). OPOX randomly selects one crossing point between 2 and n and passes the sub-sequence from one parent on one side of the crossing point to the offspring. The remaining parts are copied to the offspring in the same sequence as they appear in the other parent. Figure 2 shows an example where the selected crossing point is between positions 5 and 6. The subsequence to the left of π^a is copied first, and then the remaining assignments of π^b are copied in the same order to generate the offspring π^0.

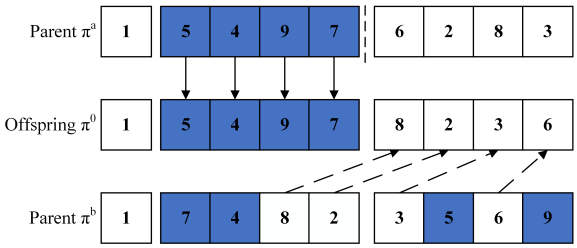


Fig. 2. An example of the one-point order crossover

LOX: The linear order crossover is widely used to solve problems related to the TSP [30]. Our experimental results (Section 4.1) show that LOX is suitable for BLLP. In this operator, two crossing points are randomly selected. The offspring π_0 is obtained by first transferring the sub-sequence between the two cutting points from one parent to the offspring, and then copying the remaining parts from the other parent in the same order. Figure 3 shows an example of generating an offspring solution with this crossover. The first crossing point is between positions 3 and 4, and the second crossing point is between positions 7 and 8. The subsequence of π^a between the crossing points is copied to π^0, and then the remaining assignments of π^b are copied in the same order to generate the offspring π^0.

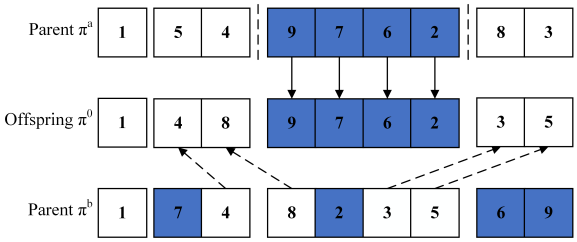


Fig. 3. An example of the linear order crossover

OBX: The order-based crossover are quite successful on scheduling problems [28], where the objective is to obtain the best order (sequence) in which jobs are processed. The basic feature of this crossover is that it preserves the relative order of assignments. OBX operates as follows. First, it randomly selects k ($k = n/2$ in this paper) facility-to-position assignments and passes

them from parent π^a to offspring. Then, it copies the remaining assignments to the offspring according to their orders in π^b. Figure 4 gives an example of the order-based crossover operator.

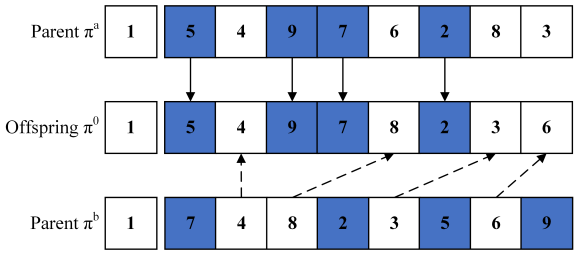


Fig. 4. An example of the order-based crossover

As shown in Section 4.1, the adaptive application of these three crossovers constitutes an important component of the proposed algorithm.

2.3.2 *Learning-based adaptive application of crossovers*

The crossover probability array γ records the possibility γ_t of the t-th type of crossovers being selected, and $\sum_{t=1}^{3} \gamma_t = 1$. Initially, the selection probability of each type of crossover is equal; i.e., $\gamma_t = 1/3$ (lines 3-5, Algorithm 1). First, a random number *rand* between 0 and 1 is generated. Then, if $rand < \gamma_1$, the one-point order crossover (the 1st type of crossover) is selected; if $\gamma_1 \leq rand < \gamma_1 + \gamma_2$, the linear order crossover (the 2nd type of crossover) is selected; otherwise, the order-based crossover (the 3rd type of crossover) is selected.

Algorithm 2 Learning-based adaptive crossover application

Input: parent permutations π^a and π^b, the crossover probability array γ.
Output: offspring permutation π^0, type of crossover t

1: $rand \leftarrow$ *random number between* 0 *and* 1
2: **if** $rand < \gamma_1$ **then**
3: $\quad \pi^0 \leftarrow OnePointOrderCrossover(\pi^a, \pi^b)$
4: $\quad t \leftarrow 1$ /* t is the type of the applied crossover */
5: **else if** $rand < \gamma_1 + \gamma_2$ **then**
6: $\quad \pi^0 \leftarrow LinearOrderCrossover(\pi^a, \pi^b)$
7: $\quad t \leftarrow 2$
8: **else**
9: $\quad \pi^0 \leftarrow OrderBasedCrossover(\pi^a, \pi^b)$
10: $\quad t \leftarrow 3$
11: **end if**
12: **return**

If an improved offspring is successfully inserted into the population during the updating procedure presented in Section 2.5, the applied crossover of type t is rewarded by increasing its selection probability γ_t, while the

other crossovers are penalized by decreasing their probability, according to the learning strategy described in Section 2.5. From the use of adaptively learned probabilities, it is expected that the most suitable crossover is selected at each generation of the algorithm to create promising offspring solutions.

2.4 3-phase local search

It is known that local search can be trapped in local optimal solutions. To cope with this difficulty, multi-phase local search framework is one possible remedy. In this work, we adopt a 3-phase local search procedure whose key idea is illustrated in Figure 5, where the X-axis indicates feasible solutions π, and the Y-axis indicates the corresponding objective values $f(\pi)$. In Figure 5, B, C, D, E, F, H and I are local optima of different quality. Starting from an initial solution, say A, the search calls *Descent_Search* (phase 1) to reach the first local optimum B and then uses *Simulated_Annealing* (phase 2) to discover nearby local optima C and D. At this point, phase 3 starts. Since D is the current local optimal solution, the *Perturbation* procedure is executed to jump from D to a distant solution J, which is subsequently optimized by *Descent_Search* (J $\rightarrow$ I). Suppose that I is better than D; the next *Perturbation* will be executed from I (I $\rightarrow$ G), which is subsequently optimized by *Descent_Search* (G $\rightarrow$ H), to obtain a high-quality solution H. We now explain each of these three local search phases.

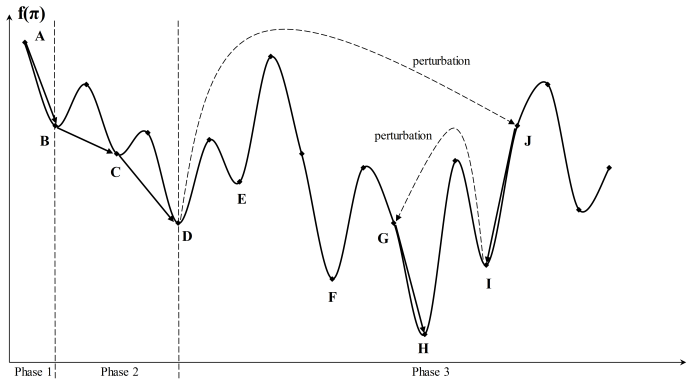


Fig. 5. An example of the 3-phase local search

1) Phase 1 (Descent search, line 1, Algorithm 3): The first local search phase is based on fast descent search. Starting from the given input permutation, this phase iteratively improves the solution until no further improvement is possible. At each iteration, facility i ($2 \leq i \leq n$) is removed and inserted after facility j ($2 \leq j \leq n$) such that the new (neighbor) solution is better than the current permutation.

2) Phase 2 (Simulated annealing, line 3, Algorithm 3): From the local opti-

mal solution π obtained in the first phase, we invoke a simulated annealing (SA) algorithm to further improve the solution. Specifically, each iteration of SA adopts the same insertion operator as in the first phase but accepts a new (neighbor) solution π' under two different conditions. If π' is no worse than π, the former is accepted to replace the current solution. Otherwise, π' will be accepted with probability $P(\delta_f, T)$ where $\delta_f = f(\pi') - f(\pi)$, and T is the temperature parameter that starts with a high value and progressively decreases according to $T = \alpha * T$ (α is a parameter called the cooling factor) when the maximum number of iterations at the current temperature exceeds parameter Q (called the search depth of SA). This second phase terminates when the temperature T is below 0.01.

3) Phase 3 (Iterated descent search, lines 6-15, Algorithm 3): Finally, when SA falls into a deep local optimal trap, the third phase is invoked, which alternates between the descent search of phase 1 and the perturbation presented in Section 2.6. This phase stops if the best-found solution cannot be improved for L consecutive search rounds (L is discussed in Section 3.2.1). Owing to the dedicated perturbation, this iterated descent search phase allows to explore more distant local optimal solutions.

Algorithm 3 Local search for solving BLLP

Input: flow matrix c and distance matrix d, facilities n, and input permutation π^0.
Output: the best improved permutation π^{best}

/* Phase 1: descent-based local search without diversification */
1: $\pi^0 \leftarrow Descent\_Search(c, d, n, \pi^0)$ /* Reach the first local optimum quickly */
2: $\pi^{best} \leftarrow \pi^0$ /* The best permutation found so far during 3-phase local search */
/* Phase 2: SA-based local search with limited degree of diversification */
3: $\pi^0 \leftarrow Simulated\_Annealing(c, d, n, \pi^0)$/* Further improve the quality of the permutation */
4: $\pi^{best} \leftarrow argmin\{f(\pi^{best}), f(\pi^0)\}$ /* Record the best permutation π^{best} found */
/* Phase 3: iterated descent search with high degree of diversification */
5: $\omega \leftarrow 0$ /* Record the consecutive non-improvement descent searches */
6: **while** Stopping condition is not met **do**
7: $\pi^0 \leftarrow Perturbation(\omega, \pi^{best})$ /* Escape from the local optimum, Section 2.6 */
8: $\pi^0 \leftarrow Descent\_Search(c, d, n, \pi^0)$
9: **if** $f(\pi^0) < f(\pi^{best})$ **then**
10: $\pi^0 \leftarrow \pi^{best}$
11: $\omega \leftarrow 0$
12: **else**
13: $\omega \leftarrow \omega + 1$
14: **end if**
15: **end while**
16: **return** π^{best}

These three phases are based on the *insertion* operator [11], which is de-

scribed formally here. Let π be a given permutation, and $\pi' \leftarrow \pi \oplus insert(\pi_i, \pi_j)$ be the new (neighbor) permutation obtained by deleting the facility i from the position π_i and inserting i to the position π_j. Thus, the neighborhood $N_I(\pi)$, i.e., the set of neighbor solutions induced by applying the insertion operator to π is given by

$$N_I(\pi) = \{\pi \oplus insert(\pi_i, \pi_j) : \pi_i, \pi_j \in \{2, \ldots, n\}, \pi_i \neq \pi_j\} \tag{5}$$

whose size is $O((n-1)^2)$.

To ensure a fast evaluation of neighbor permutations, an incremental evaluation technique [11] is adopted for the BLLP. The main idea is to maintain a special data structure to record the move gains of the neighbor solutions. Particularly, each insertion move is decomposed into several swap moves. Let $swap(\pi_i, \pi_j)$ be the swap operation which exchanges the positions of two facilities in permutation π, $\pi^S \leftarrow \pi \oplus swap(\pi_i, \pi_j)$ be the neighbor solution obtained by applying $swap(\pi_i, \pi_j)$ to π. Then the move gain $\Delta(\pi_i, \pi_j)$ of the insertion move $insert(\pi_i, \pi_j)$ can be calculated by,

$$\Delta(\pi_i, \pi_j) = \begin{cases} \sum_{\pi_k=\pi_i+1}^{\pi_j} \Delta_S(\pi_k - 1, \pi_k), \ \pi_i < \pi_j; \\ \sum_{\pi_k=\pi_i-1}^{\pi_j} \Delta_S(\pi_k + 1, \pi_k), \ \pi_i > \pi_j. \end{cases} \tag{6}$$

where $\Delta_S(\pi_k - 1, \pi_k)$ and $\Delta_S(\pi_k + 1, \pi_k)$ are the move gains of the swap moves when facility i is swapped from position $\pi_k - 1$ and $\pi_k + 1$ to its neighboring position π_k.

$$\begin{cases} \Delta_S(\pi_k - 1, \pi_k) = -d_{\pi_k - 1\pi_k} \sum_{r \in A} (c_{ir} - c_{kr}) + d_{\pi_k - 1\pi_k} \sum_{r \in B} (c_{ir} - c_{kr}) \\ + \sum_{r \in C} (d_{\pi_k - 1\pi_r} - d_{\pi_k \pi_r})(c_{ir} - c_{kr}) \\ \Delta_S(\pi_k + 1, \pi_k) = d_{\pi_k + 1\pi_k} \sum_{r \in A} (c_{ir} - c_{kr}) - d_{\pi_k + 1\pi_k} \sum_{r \in B} (c_{ir} - c_{kr}) \\ - \sum_{r \in C} (d_{\pi_k + 1\pi_r} - d_{\pi_k \pi_r})(c_{ir} - c_{kr}) \end{cases} \tag{7}$$

where $A = K_{\pi_i} \backslash \{k\}$, $B = K \backslash K_{\pi_k}$ and $C = K_{\pi_k} \backslash K_{\pi_i}$. $K = \{1, \ldots, n\} \backslash \{i, k\}$ and K_{π_i} is a facility set in which the clockwise distance from the position of a facility to π_i is shorter than its counterclockwise distance.

The 3-phase local search procedure is applied to each offspring. We discuss the complexity of this local search procedure in Section 2.7.

2.5 Update

After the 3-phase local search, the population, the best permutation ever found, and the probability array of crossovers are updated.

Algorithm 4 Update

Input: Population P, new permutation π^{best}, best permutation π^{global_best}, the crossover probability array γ, type of applied crossover t.

Output: Updated population P, best permutation π^{global_best}, the crossover probability array γ.

1: $\pi^{worst} \leftarrow argmax\{f(\pi^1), f(\pi^2), \ldots, f(\pi^p)\}$ /* Identify the worst permutation π^{worst} in P */
   /* Update the Population */
2: **if** $f(\pi^{best}) \leq f(\pi^{worst})$ **then**
3: $\quad P \leftarrow P \cup \{\pi^{best}\} \backslash \{\pi^{worst}\}$
4: $\quad q_t = q_t + 1$;
5: **end if**
   /* Update the best permutation π^{global_best} */
6: $\pi^{global\_best} \leftarrow argmin\{f(\pi^{best}), f(\pi^{global\_best})\}$
   /* Update the probability of each applied crossover */
7: Update γ by calculate each γ_m according to equation (8)
8: **return**

1) Update the population and the best permutation ever found (lines 2-6, Algorithm 4). The quality-based population updating strategy is adopted to update the population, which has been successfully applied to the QAP [17] and the traveling repairman problem with profits [25]. Since the local search already guarantees the diversity of descendants through perturbations and SA, pool replacement only needs to consider the quality of descendants. Specifically, if the objective value (quality) of the input permutation π^{best} is no worse than the worst permutation π^{worst} of the population, then π^{best} replaces π^{worst}. Otherwise, π^{best} is ignored and the population remains unchanged.

2) Update the crossover probability array (line 7, Algorithm 4). The crossover probability array $\gamma = \{\gamma_1, \gamma_2, \gamma_3\}$ is updated according to a probability learning method (Equation (8)). Our probabilistic updating scheme is inspired by learning automata [31], which helps to determine the optimal strategy in the space of candidate strategies [32]. Similar probability updating schemes were used to solve the traveling repairman problem with profits [30], the orienteering problem [33], feature selection in classification [34], and the traveling salesman problem with hotel selection [18].

$$\gamma_m = \frac{1 + q_m}{3 + q_1 + q_2 + q_3} (m \in \{1, 2, 3\}) \tag{8}$$

where m is the crossover type and q_m is the number of times that an improved offspring is successfully inserted into the population by adopting the crossover of type m. If q_m increases in the current generation, both numerator and denominator of the equation (8) will increase by the same amount. Their total γ_m will rise, while the selection probability of other crossovers will decrease correspondingly.

2.6 Perturbation for iterated descent search

As explained in Section 2.4, the third local search phase iterates descent search and perturbation to explore additional local optimal solutions starting from the deep local optimum reached by the SA procedure. To make the perturbation as effective as possible, two types of perturbation (random and symmetric-based) (line 2 and line 4, Algorithm 5) are employed and applied according to the number of consecutive iterations during which π^{best} is not improved during the 3rd phase of the local search.

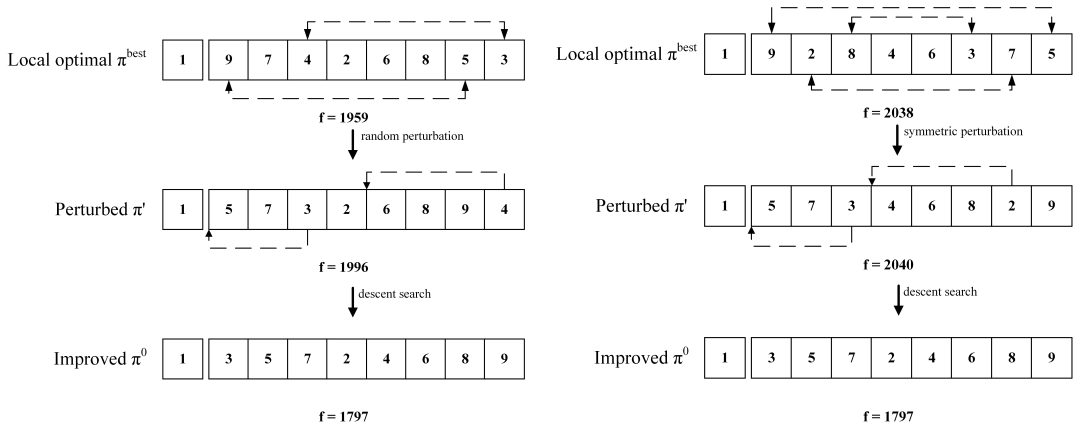
Algorithm 5 The perturbation phase for the 3rd-phase local search

Input: the number of consecutive iterations ω during which π^{best} is not improved.
Output: the permutation π' after perturbation
1: **if** $\omega < \beta \times L$ **then**
2: $\quad \pi' \leftarrow random\_perturbation(\pi^{best})$
3: **else**
4: $\quad \pi' \leftarrow symmetric\_perturbation(\pi^{best})$
5: **end if**
6: **return** π'

Recalled that the 3rd phase of the local search iterates the perturbation and the descent search. If the iteration is slightly stagnating, i.e., no better permutation can be found during $\beta \times L$ consecutive descent searches (β is a parameter, L is the allowed maximum number of descent searches), the first type of perturbation is adopted to slightly modify the current permutation π^{best}. If the search is trapped in a deep local optimum (i.e., no improvement is reached after $\beta \times L$ consecutive descent searches), the perturbation strategy applies the second perturbation to enable a stronger modification.

To perform a perturbation, two types of moves, both are based on the swap operator (see Section 2.4) are adopted according to ω. The first type is called random tabu perturbation that randomly selects performs η (a random number in $[0.1 \times n, \eta_{max}]$) random $swap(i, j)$ moves, where η_{max} is discussed in Section 3.2.1. Each time a perturbation move is performed, the swapped facilities i and j are recorded in the tabu list and will not be considered during the current perturbation procedure. The second type is the symmetric perturbation that swaps $0.45 * n$ facilities i from their positions π_i with their

symmetric positions (defined as position $n + 1 - \pi_i$).



(a) An example of random perturbation (b) An example of symmetric perturbation

Fig. 6. Two types of perturbation with descent search

Figure 6(a) gives a permutation $\pi = \{1, 9, 7, 4, 2, 6, 8, 5, 3\}$ with the objective value of 1959. A new permutation $\pi' = \{1, 5, 7, 3, 2, 6, 8, 9, 4\}$ is obtained after randomly swapping two pairs of facilities $\{9, 5\}$ and $\{4, 3\}$ of π. The objective value of the random perturbed permutation π' is 1996, which is improved by the descent search, leading to an improved permutation with the objective value of 1797. Figure 6(b) illustrates the symmetric permutation applied to permutation $\pi = \{1, 9, 2, 8, 4, 6, 3, 7, 5\}$ with the objective value of 2038. This permutation relocates three pairs of facilities, $\{9, 5\}$, $\{2, 7\}$ and $\{8, 3\}$ to their symmetrical positions, leading to the perturbed permutation $\pi' = \{1, 5, 7, 3, 4, 6, 8, 2, 9\}$ with the objective value of 2040. This objective value is reduced to 1797 by the descent search, which is much better than that of initial solution.

Finally, during the perturbation procedure, the best solution ever found is updated each time an even better solution is found. The permutation resulting from the perturbation procedure is then used as the new starting permutation of the next round of the descent search procedure.

2.7 Computational complexity and discussion

We first consider the population initialization procedure, which can be divided into two steps. The first step of randomly assigning facilities can be achieved in $O(n)$. The complexity of one iteration of descent search is $O(n^2)$. Then, the second step of applying the descent search is bounded by $O(K_1 \times n^2)$, where K_1 is the number of iterations of the descent search. Thus, the time complexity of the population initialization procedure is $O(p \times K_1 \times n^2)$, where p is the population size.

We now consider the four procedures in the main loop of the AMA algorithm: parent selection, crossover operator, the 3-phase local search, and updating procedure. The parent selection procedure is realized in $O(1)$. The crossover operator requires time $O(n)$. The complexity of the 3-phase local search procedure is $O(n \times Q \times \tau + K_1 \times K_2 \times n^2)$, where Q is the search depth of SA, τ is the number of temperature levels, and K_2 is the number of runs of descent search. Because Q is linearly related to n according to Section 3.2.1, the complexity expression can be simplified to $O((\tau + K_1 \times K_2) \times n^2)$. The updating procedure can be achieved in $O(n)$. Then, the complexity of one iteration of the main loop of the AMA algorithm is $O((\tau + K_1 \times K_2) \times n^2)$. This complexity is quite similar to the complexity of one iteration of the main loop of the SA-VNS-1-1 algorithm of [11], which is $O((\tau + K_1 \times K_3) \times n^2)$, where K_3 is the number of runs of VNS.

3 Experimental results and comparisons

We evaluate the proposed AMA algorithm by studying 65 benchmark instances and comparing it with the best-performing algorithms.

3.1 Benchmark instances

We use 65 benchmark instances, which are classified into four categories [1]. The first three categories (55 instances with up to 300 facilities) are from the literature and in particular are thoroughly tested in [11]. The last category introduced in this study includes 10 very large instances with 310–400 facilities.

(1) The first category contains 20 instances with sko* in their names. These instances came from the unified testing platform QAPlib proposed by J. Skorin-Kapov [35] [2]. The flow is the same as those in QAP and ranges from 0 to 10. The distance between adjacent positions are random integer numbers from the intervals [1, 101]. These sko* instances were tailored for the single row facility layout problem [36], and largely tested in the literature on the facility layout problem [37,38,39,40,41].
(2) The second category contains 20 randomly generated large scale instances with 110 to 300 facilities [11]. These instances have p* in their names. The flow between two facilities and the distances between ad-

[1] https://github.com/sunseu2022/Instances-of-BLLP
[2] https://www.opt.math.tugraz.at/qaplib/inst.html#Sk

jacent positions were random integer numbers in the intervals [0, 10] and [1, 10], respectively.

(3) The third category contains 15 instances with 100, 150, and 200 facilities [11]. They are generated in the same way as the second category. Each size includes 5 instances. Names of these instances begin with p* and end with -*.

(4) The fourth category contains 10 instances with 310 to 400 facilities (one instance per size), generated in the same way as the second and the third categories. These instances also have p* in their names.

3.2 Experiment settings

The proposed AMA algorithm is programmed in C++ and compiled by GNU g++ 4.1.2 with the -O3 flag. Experiments are conducted on a computer with an Intel (R) Core (TM) 2 Duo CPU T7700 2.40GHz processor running Ubuntu CentOS Linux release 7.9.2009 (Core). We will make the code of the algorithm available upon the publication of the paper at the link of footnote 1.

3.2.1 Parameters

The main parameters are related to the population initialization, simulated annealing algorithm and the iterated descent search (see Sections 2.2, 2.4 and 2.6). The AMA algorithm requires 6 parameters: the population size p, the number of iterations Q at which the temperature is kept constant, the cooling factor α, the search depth L of iterated descent search, The threshold of applying random perturbation β, and the maximum length of random perturbation η_{max}. To tune these parameters, we used the "irace" package [42], which implements the Iterated F-Race method to determine automatically the most suitable parameter settings from a set of possible parameter configurations. The irace is run on 7 randomly selected instances, and the tuning budget is set to be 200 executions under the normal cutoff time. Table 1 presents the range of parameter values and final values recommended by the "irace". All experiments adopted the final values.

3.2.2 Reference algorithms

To evaluate the performance of the AMA algorithm, the state-of-the-art heuristic algorithm SA and its variants (SA-VNS-0-0, SA-VNS-0-1, SA-VNS-1-0, and SA-VNS-1-1) from [11] are used as the main reference algorithms, where SA-VNS is a 2-phase algorithm that combines the SA algorithm with a variable neighborhood search (VNS). These reference algorithms are run

Table 1
Settings of important parameters

Parameters	Section	Description	Candidate values	Final value
p	2.2	Population size	$\{7, 10, 12, 15\}$	10
Q	2.4	The number of iterations at each temperature in SA	$\{50n, 100n, 150n, 200n\}$	$100n$
α	2.4	Cooling factor of SA	$\{0.8, 0.85, 0.9, 0.95\}$	0.95
L	2.4	Search depth of iterative descent search	$\{50, 75, 100, 125\}$	75
β	2.6	The threshold of applying random perturbation	$\{0.33, 0.5, 0.67, 0.75\}$	0.67
η_{max}	2.6	The maximum length of random perturbation	$\{0.25n, 0.3n, 0.35n, 0.4n\}$	$0.25n$

on a computer with an Intel Core i5-6200U CPU (2.30GHz) [11]. We extracted the results of these reference algorithms from [11] for comparisons.

Following the common practice of reporting comparative results, performance assessment focuses on the best permutation found by an algorithm with the minimum objective value. For the conventional 55 instances, we cite the best-known objective value (BKV), which was reported in [11] under a relaxed cutoff time (see Section 3.2.3). It is worth noting that, according to the results reported in [11], no single existing reference algorithm could obtain all BKVs for these conventional 55 instances. Even the best-performing algorithm SA-VNS-1-1 misses 16 BKVs. As shown in Section 3.3, under the same relaxed cutoff time as in [11], our AMA algorithm reaches all BKVs and improves 4 BKVs. Under the normal cutoff time (see Section 3.2.3), AMA is still able to improve 4 BKVs and match other 46 BKVs while missing only 5 BKVs.

3.2.3 Stopping condition

The algorithms of [11] (coded in C++) were tested with different cutoff time limits (given below) and each algorithm was run to solve each instance 10 times independently. The experiments in [11] were performed on a computer with an Intel Core i5-6200U CPU running at 2.30GHz, which is comparable to our Intel computer running at 2.40GHz. To ensure a meaningful comparison, we adopted the same experimental protocol to report our results for the proposed AMA algorithm (coded in C++) and used the same cutoff time as in [11] to solve each instance 10 times.

Normal cutoff time: The five reference algorithms (SA, SA-VNS-0-0, SA-VNS-0-1, SA-VNS-1-0 and SA-VNS-1-1) [11] were run to solve the 55 benchmark instances of the first three categories under the following cutoff times: 30 seconds for $n \leq 80$, 60 seconds for $80 < n \leq 100$, 300 seconds for $100 < n \leq 150$, 600 seconds for $150 < n \leq 200$, 1200 seconds for $200 < n \leq 250$, 1800 seconds for $250 < n \leq 300$. Similarly, AMA was run 10 times to solve each instance. Besides, for the 10 newly generated very large instances, we use the following cutoff times: 3600 seconds for $300 < n \leq 350$ and 5400

seconds for $350 < n \leq 400$.

Relaxed cutoff time: SA-VNS-1-1 (the best reference algorithm) was also run under much relaxed cutoff time in [11], that is, 30 times each of the above-mentioned normal cutoff times. Under these relaxed stopping conditions, SA-VNS-1-1 reported the current best objective values for the benchmark instances of categories 1 to 3 (the BKV values as shown in Table 2-4).

3.3 Comparison with state-of-the-art algorithms

The results of AMA on the three categories of instances are presented and compared with those of the reference algorithms.

Table 2 reports the results of our AMA algorithm on the 20 instances of the first category, as well as the results of the reference algorithms. The first column of Table 2 indicates the name of each instance and the second column indicates its BKV reported in the literature. Recalled that the BKV values were obtained by the representative algorithm SA-VNS-1-1 under relaxed time conditions. The next 15 columns report the best results (f_{best}), average results (f_{avg}), and average running time (t_{avg}) of SA, SA-VNS-0-0, SA-VNS-0-1, SA-VNS-1-0, and SA-VNS-1-1 over 10 independent runs. The last 3 columns show the results of AMA for each instance over 10 runs: the best results, average results, and average computation time (t_{avg}) in seconds of the successful runs to obtain the best results. The best results among the compared values are indicated in bold. Entries with "-" mean that the corresponding results are not available in the literature. Additionally, the row "Avg." indicates the average values of the columns. In [11], no timing information is indicated for each specific instance, but the average time for each group of instances (i.e., instances with the same n) is provided, which is shown in row Avg. in Table 2.

Table 2 indicates that SA-VNS-0-1, SA-VNS-1-1, and AMA can easily find the best-known results for these 20 instances, while SA, SA-VNS-0-0, and SA-VNS-1-0 miss at least 1 best-known result. Moreover, in terms of f_{avg}, AMA has better or equal f_{avg} values in all instances compared with the reference algorithms, which shows that the AMA algorithm is highly stable compared to the reference algorithms. Finally, among the compared algorithms, AMA needs the least time to reach the BKVs.

Table 3 reports the comparative results of the reference algorithms and the AMA algorithm on 20 instances of the second category with the same information as in Table 2. It can be seen from Table 3 that AMA improves the BKV values of 4 instances (p180, p280, p290, and p300, indicated by the '*' symbol) and reaches the BKV values of other 12 instances. Besides,

AMA dominates the reference algorithms SA, SA-VNS-0-0, SA-VNS-0-1, SA-VNS-1-0, and SA-VNS-1-1, with 15, 17, 14, 13, and 11 better results, respectively. As to the computation time, we note that for the 4 instances for which all algorithms can obtain the BKVs, AMA is the fastest to reach the best–known results. For the remaining instances except two, AMA spends

Table 2
Comparison of AMA with the reference algorithms [11] on the 20 instances of the first category

Instance	BKV	SA			SA-VNS-0-0			SA-VNS-0-1			SA-VNS-1-0			SA-VNS-1-1			AMA		
		f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
sko_64_1	74,067	**74,067**	74,067.1	-	**74,067**	74,069.3	-	**74,067**	**74,067.0**	-	**74,067**	74,067.3	-	**74,067**	**74,067.0**	-	**74,067**	**74,067.0**	1.9
sko_64_2	573,458	**573,458**	573,528.7	-	**573,458**	573,531.1	-	**573,458**	573,460.9	-	**573,458**	573,495.0	-	**573,458**	573,498.0	-	**573,458**	**573,458.0**	9.2
sko_64_3	363,994	**363,994**	364,005.1	-	**363,994**	364,005.1	-	**363,994**	364,004.0	-	**363,994**	**363,994.0**	-	**363,994**	363,994.2	-	**363,994**	**363,994.0**	2.1
sko_64_4	243,966	**243,966**	243,974.0	-	**243,966**	243,984.8	-	**243,966**	**243,966.0**	-	**243,966**	243,970.8	-	**243,966**	**243,966.0**	-	**243,966**	**243,966.0**	2.8
sko_64_5	430,063	**430,063**	430,177.6	-	**430,063**	430,171.2	-	**430,063**	**430,086.0**	-	430,178	430,207.0	-	**430,063**	430,101.4	-	**430,063**	**430,086.0**	14.4
sko_72_1	107,431	**107,431**	107,434.2	-	**107,431**	107,483.1	-	**107,431**	**107,431.0**	-	**107,431**	**107,431.0**	-	**107,431**	**107,431.0**	-	**107,431**	**107,431.0**	6.3
sko_72_2	609,044	**609,044**	609,129.3	-	**609,044**	609,125.4	-	**609,044**	609,073.7	-	**609,044**	609,121.3	-	**609,044**	609,070.2	-	**609,044**	**609,044.0**	9.8
sko_72_3	1,009,747	**1,009,747**	1,009,764.3	-	**1,009,747**	1,009,768.0	-	**1,009,747**	**1,009,747.0**	-	**1,009,747**	1,009,847.5	-	**1,009,747**	1,009,847.5	-	**1,009,747**	**1,009,747.0**	3.8
sko_72_4	853,106	**853,106**	853,191.7	-	**853,106**	853,350.0	-	**853,106**	853,245.7	-	**853,106**	853,183.2	-	**853,106**	853,160.4	-	**853,106**	**853,112.9**	7.7
sko_72_5	351,489	**351,489**	351,570.7	-	**351,489**	351,586.1	-	**351,489**	351,494.4	-	**351,489**	351,582.9	-	**351,489**	351,552.2	-	**351,489**	**351,489.0**	5.8
sko_84_1	155,730	**155,730**	**155,730.0**	-	**155,730**	155,730.6	-	**155,730**	**155,730.0**	-	**155,730**	**155,730.0**	-	**155,730**	**155,730.0**	-	**155,730**	**155,730.0**	1.5
sko_84_2	447,633	**447,633**	447,635.4	-	**447,633**	447,640.2	-	**447,633**	**447,633.0**	-	**447,633**	447,633.8	-	**447,633**	**447,633.0**	-	**447,633**	**447,633.0**	5.1
sko_84_3	848,904	**848,904**	848,917.3	-	**848,904**	848,908.5	-	**848,904**	**848,904.0**	-	**848,904**	848,908.4	-	**848,904**	848,908.4	-	**848,904**	**848,904.0**	11.8
sko_84_4	1,768,175	**1,768,175**	1,768,188.6	-	**1,768,175**	1,768,186.1	-	**1,768,175**	**1,768,175.0**	-	**1,768,175**	1,768,178.5	-	**1,768,175**	1,768,177.5	-	**1,768,175**	**1,768,175.0**	10.9
sko_84_5	1,175,705	**1,175,705**	1,175,715.5	-	**1,175,705**	1,175,705.1	-	**1,175,705**	**1,175,705.0**	-	**1,175,705**	**1,175,705.0**	-	**1,175,705**	**1,175,705.0**	-	**1,175,705**	**1,175,705.0**	1.5
sko_100_1	288,678	**288,678**	288,693.9	-	288,698	288,742.3	-	**288,678**	288,698.4	-	**288,678**	288,709.3	-	**288,678**	288,699.4	-	**288,678**	**288,678.0**	20.1
sko_100_2	1,806,738	1,806,879	1,807,173.7	-	1,806,959	1,807,521.2	-	**1,806,738**	1,807,210.6	-	**1,806,738**	1,807,311.2	-	**1,806,738**	1,807,130.4	-	**1,806,738**	**1,806,991.9**	18.9
sko_100_3	14,871,217	**14,871,217**	14,872,537.5	-	14,871,260	14,873,074.1	-	**14,871,217**	14,872,263.2	-	**14,871,217**	14,872,719.9	-	**14,871,217**	14,872,558.4	-	**14,871,217**	**14,871,596.4**	24.1
sko_100_4	2,980,012	2,980,038	2,980,282.1	-	2,980,090	2,980,666.9	-	**2,980,012**	2,980,423.5	-	**2,980,012**	2,980,434.1	-	**2,980,012**	2,980,262.8	-	**2,980,012**	**2,980,051.6**	22.8
sko_100_5	879,038	**879,038**	879,296.3	-	**879,038**	879,324.7	-	**879,038**	879,233.5	-	**879,038**	879,324.4	-	**879,038**	879,297.4	-	**879,038**	**879,177.7**	22.7
Avg.	1,491,910	1,491,918	1,492,050.7	17.8	1,491,928	1,492,128.7	19.7	**1,491,910**	1,492,027.6	14.6	1,491,916	1,492,077.7	17.4	**1,491,910**	1,492,039.5	15.7	**1,491,910**	**1,491,951.9**	10.2

Table 3
Comparison of AMA with the reference algorithms [11] on the 20 instances of the second category

Instance	BKV	SA			SA-VNS-0-0			SA-VNS-0-1			SA-VNS-1-0			SA-VNS-1-1			AMA		
		f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
p110	4,121,976	**4,121,976**	4,121,977.8	77.3	**4,121,976**	4,122,028.8	197.9	**4,121,976**	**4,121,976.0**	60.7	**4,121,976**	**4,121,976.0**	53.3	**4,121,976**	**4,121,976.0**	46.2	**4,121,976**	**4,121,976.0**	9.8
p120	5,712,477	**5,712,477**	5,712,479.4	110.4	**5,712,477**	5,712,829.7	161.0	**5,712,477**	5,712,503.1	101.0	**5,712,477**	5,712,477.8	86.5	**5,712,477**	**5,712,477.0**	64.1	**5,712,477**	**5,712,477.0**	36.3
p130	7,163,273	**7,163,273**	7,163,581.9	130.8	**7,163,273**	7,163,645.6	104.0	**7,163,273**	7,163,424.5	128.7	**7,163,273**	7,163,545.8	141.4	**7,163,273**	7,163,542.9	151.0	**7,163,273**	**7,163,340.3**	155.2
p140	8,531,830	8,531,878	8,532,073.6	132.0	8,531,885	8,532,407.3	176.4	**8,531,830**	8,532,217.5	193.1	**8,531,830**	8,532,131.5	144.3	**8,531,830**	8,532,102.4	144.0	**8,531,830**	**8,531,884.9**	148.7
p150	10,223,765	**10,223,765**	10,223,909.3	108.4	10,224,061	10,224,957.7	154.2	**10,223,765**	10,223,978.8	130.7	**10,223,765**	10,223,848.7	154.3	**10,223,765**	**10,223,814.0**	102.5	10,223,814	10,223,815.8	105.4
p160	13,831,552	**13,831,552**	13,831,705.0	327.3	13,832,078	13,832,671.9	338.2	**13,831,552**	13,832,315.0	359.5	**13,831,552**	13,831,763.9	378.0	**13,831,552**	13,831,658.8	345.3	**13,831,552**	**13,831,603.5**	320.4
p170	15,765,986	15,766,043	15,766,846.2	265.9	15,766,299	15,767,515.4	211.1	15,766,014	15,766,619.8	290.0	15,766,049	15,767,071.2	343.2	15,766,049	15,766,894.3	306.2	**15,765,986**	**15,766,148.5**	343.5
p180	17,879,424	17,879,893	17,880,274.8	274.6	17,879,718	17,880,254.0	315.7	17,879,454	17,879,953.1	334.5	17,879,655	17,880,405.0	220.3	17,879,681	17,880,299.5	249.5	**17,879,398***	**17,879,702.1**	474.5
p190	22,570,679	22,570,709	22,570,856.3	219.9	22,570,801	22,571,928.0	276.7	22,570,724	22,571,183.2	306.9	22,570,709	22,570,785.6	249.7	**22,570,679**	22,570,712.0	206.8	**22,570,679**	**22,570,680.1**	326.2
p200	25,948,640	25,948,749	25,949,075.1	288.8	25,949,406	25,951,403.3	364.2	25,948,857	25,950,692.7	312.4	25,948,749	25,949,144.3	293.5	**25,948,647**	25,948,933.5	298.9	**25,948,647**	**25,948,777.5**	466.2
p210	28,275,043	28,275,129	28,275,264.3	607.1	28,275,289	28,275,975.7	497.7	28,275,071	28,275,242.3	484.2	28,275,070	28,275,396.4	611.9	28,275,062	**28,275,219.9**	581.9	**28,275,043**	28,275,238.2	988.2
p220	34,618,625	34,618,795	34,619,312.0	468.4	34,619,072	34,621,239.5	695.7	34,618,750	34,620,209.5	644.5	34,618,697	34,619,191.6	502.5	34,618,653	34,619,108.0	630.9	**34,618,625**	**34,618,754.8**	834.0
p230	42,559,657	42,560,001	42,560,285.2	465.7	42,559,966	42,560,963.9	804.3	42,559,959	42,560,313.8	905.8	42,559,773	42,560,176.6	839.6	42,559,770	42,560,017.8	824.7	**42,559,657**	**42,559,756.3**	792.0
p240	43,876,137	43,876,455	43,877,660.6	767.6	43,877,232	43,878,940.3	611.6	43,876,518	43,878,050.4	582.6	43,876,414	43,878,105.5	592.6	43,876,644	43,877,989.9	488.1	**43,876,137**	**43,876,729.1**	740.8
p250	50,981,493	50,981,829	50,982,127.1	511.3	50,982,667	50,985,643.3	612.7	50,981,760	50,982,845.4	583.0	**50,981,493**	50,981,974.5	693.0	50,981,543	50,981,821.8	645.3	50,981,496	**50,981,655.8**	919.8
p260	58,694,312	58,694,376	58,694,655.3	959.9	58,696,442	58,699,841.0	760.5	58,695,090	58,698,410.1	1088.0	58,694,322	58,694,882.0	1321.7	**58,694,312**	58,694,793.5	1172.5	**58,694,312**	**58,694,567.0**	1036.9
p270	64,033,556	64,033,642	64,034,117.1	411.2	64,035,059	64,037,662.2	1028.8	64,034,235	64,035,914.4	1059.6	64,033,630	64,034,165.1	578.2	64,033,592	**64,033,893.5**	559.0	**64,033,558**	64,033,985.5	1182.0
p280	69,343,736	69,344,215	69,344,961.9	932.4	69,344,451	69,347,417.5	808.4	69,344,455	69,346,128.8	998.2	69,344,150	69,345,159.3	730.7	69,343,780	69,344,754.5	759.6	**69,343,717***	**69,343,872.2**	1198.3
p290	80,334,743	80,335,259	80,336,907.9	1087.7	80,338,235	80,340,459.0	965.3	80,336,224	80,339,177.9	890.0	80,335,793	80,337,038.7	1049.5	80,335,278	80,336,667.1	1172.6	**80,334,587***	**80,335,870.3**	1288.8
p300	89,325,779	89,326,209	89,328,369.7	976.3	89,328,004	89,334,459.3	925.2	89,327,323	89,332,625.3	1059.6	89,326,100	89,329,428.8	1032.8	89,325,801	89,328,908.9	966.5	**89,325,729***	**89,328,162.0**	1253.1
Avg.	34,689,634	34,689,811	34,690,322.0	456.2	34,690,420	34,692,112.2	500.5	34,689,965	34,691,189.1	525.6	34,689,774	34,690,433.4	500.9	34,689,718	34,690,279.3	485.8	**34,689,625**	**34,689,949.8**	631.0

slightly more time finding better solutions, including one record-breaking result, for instance p180.

According to [11], SA-VNS-1-1 is most representative among the reference algorithms. Therefore, Table 4 focuses on a comparison between AMA and SA-VNS-1-1 on the 15 instances of the third category. The results show that AMA can reach 14 BKV values (against 12 for SA-VNS-1-1), and a slightly worse result than BKV in only one case (p200-5) (against 3 for SA-VNS-1-1). AMA performs better than SA-VNS-1-1 on 3 instances in terms of f_{best} and reports 15 better f_{avg} results.

Table 4
Comparison of AMA with the reference algorithm SA-VNS-1-1 [11] on the 15 instances of the third category

Instance	BKV	SA-VNS-1-1			AMA		
		f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
p100-1	3,003,926	**3,003,926**	3,003,945.8	-	**3,003,926**	**3,003,926.9**	15.2
p100-2	3,148,061	**3,148,061**	3,148,147.8	-	**3,148,061**	**3,148,061.2**	27.5
p100-3	2,995,405	**2,995,405**	2,995,424.5	-	**2,995,405**	**2,995,407.6**	24.3
p100-4	2,709,226	**2,709,226**	2,709,227.8	-	**2,709,226**	**2,709,226.2**	26.9
p100-5	3,081,967	**3,081,967**	3,082,212.1	-	**3,081,967**	**3,082,064.7**	25.6
p150-1	11,209,802	**11,209,802**	11,210,555.5	-	**11,209,802**	**11,210,503.8**	198.6
p150-2	9,592,147	**9,592,147**	9,592,258.1	-	**9,592,147**	**9,592,173.8**	182.6
p150-3	10,363,199	**10,363,199**	10,363,401.5	-	**10,363,199**	**10,363,269.9**	216.7
p150-4	10,306,319	**10,306,319**	10,306,364.9	-	**10,306,319**	**10,306,319.0**	147.9
p150-5	10,345,363	**10,345,363**	10,345,380.5	-	**10,345,363**	**10,345,363.0**	71.9
p200-1	26,003,404	26,003,406	26,006,396.1	-	**26,003,404**	**26,004,820.2**	392.3
p200-2	25,812,802	**25,812,802**	25,812,921.4	-	**25,812,802**	**25,812,802.8**	357.0
p200-3	25,047,165	25,047,231	25,047,987.8	-	**25,047,165**	**25,047,267.**1	340.4
p200-4	24,982,077	24,982,092	24,984,932.1	-	**24,982,077**	**24,983,098.1**	406.6
p200-5	26,576,831	**26,576,846**	26,577,150.3	-	**26,576,846**	**26,577,011.2**	392.8
Avg.	13,011,846	13,011,853	13,012,420.4	-	**13,011,847**	**13,012,087.7**	188.4

Table 5 summarizes the comparison results of AMA with SA-VNS-1-1 on the 10 (very large) instances of the fourth category. Since these instances have not been previously tested by any BLLP algorithm, we use the most representative algorithm, SA-VNS-1-1, as our reference. We run both SA-VNS-1-1 and AMA under the above-mentioned normal and relaxed cutoff time, respectively. Column 1 of Table 5 indicates the name of each instance. Columns 2-4 and 7-9 report respectively the best results (f_{best}), average results (f_{avg}), and average computation time (t_{avg}) of SA-VNS-1-1 and AMA over 10 runs for each instance under the cutoff time. Columns 5-6 and 10-11 report the best results f_{best} and the computation time ($t(s)$) of SA-VNS-1-1 and AMA over 1 run for each instance under the relaxed cutoff time. We observe that under the normal cutoff time, AMA obtains better results than SA-VNS-1-1 for all the instances except one at the expense of more computation time (3076 seconds against 2467 seconds for SA-VNS-1-1). Under the relaxed cutoff time, AMA dominates SA-VNS-1-1 by finding better results for all 10 instances with significantly less computation time (53577 seconds against 63321 seconds for SA-VNS-1-1).

Table 5
Comparison of AMA with the reference algorithm SA-VNS-1-1 [11] on the 10 instances of the fourth category

Instance	SA-VNS-1-1					AMA				
	Normal cutoff time			Relaxed cutoff time		Normal cutoff time			Relaxed cutoff time	
	f_{best}	f_{avg}	t_{avg}	f_{best}	$t(s)$	f_{best}	f_{avg}	t_{avg}	f_{best}	$t(s)$
p310	99,529,965	99,532,406.4	1658.2	99,529,350	20675	**99,528,408**	**99,530,164.6**	2615.2	**99,528,408**	96751
p320	108,737,431	108,739,789.2	2003.1	108,737,281	59649	**108,737,085**	**108,738,522.4**	2192.1	**108,736,979**	2823
p330	113,536,558	113,536,665.0	1740.0	113,536,517	14358	**113,536,498**	**113,536,614.4**	2646.8	**113,536,422**	15263
p340	130,119,157	130,119,568.9	2168.4	130,118,943	29045	**130,118,946**	**130,119,296.9**	2781.8	**130,118,929**	4829
p350	142,089,722	142,091,475.3	1905.9	142,089,276	51001	**142,089,171**	**142,090,468.7**	2057.8	**142,089,169**	33286
p360	155,414,925	**155,416,967.4**	2915.2	155,415,318	28548	**155,414,562**	155,417,613.2	3673.2	**155,414,558**	6997
p370	172,403,126	172,403,521.6	2130.5	172,403,080	145811	**172,403,103**	**172,403,409.6**	4303.4	**172,402,930**	38398
p380	**184,233,263**	**184,235,600.2**	2715.6	184,230,214	78816	184,234,412	184,236,844.3	3205.0	**184,229,058**	148063
p390	185,318,447	**185,322,050.8**	3317.1	185,318,069	93098	**185,317,425**	185,323,757.6	4187.0	**185,317,344**	31239
p400	208,908,971	208,913,084.0	4118.3	208,909,374	112218	**208,908,761**	**208,912,600.8**	3107.1	**208,908,967**	158130
Avg.	150,029,157	150,031,112.9	2467.2	150,028,742.2	63321.9	**150,028,837**	**150,030,929.3**	3076.9	**150,028,276.4**	53577.9

Table 6 summarizes the results of the proposed AMA algorithm compared to each reference algorithm under the above-mentioned cutoff time, together with the p-values of the non-parametric Wilcoxon signed-rank test. p-values smaller than 0.05 are shown in bold and indicate statistically significance differences. From these results, one observes that AMA significantly dominates each reference algorithm on at least one category in terms of the best or/and average results.

Table 6
Summary of comparative results between AMA and five reference algorithms under the normal cutoff time

Algorithm pair	Category/Instance	Indicator	Better	Equal	Worse	p-value
AMA vs. SA	I/20	f_{best}	2	18	0	3.71E-1
		f_{avg}	19	1	0	**1.43E-4**
	II/20	f_{best}	15	4	1	**8.52E-4**
		f_{avg}	20	0	0	**1.91E-6**
AMA vs. SA-VNS-0-0	I/20	f_{best}	4	16	0	1.00E-1
		f_{avg}	20	0	0	**1.43E-4**
	II/20	f_{best}	17	3	0	**3.21E-4**
		f_{avg}	20	0	0	**1.91E-6**
AMA vs. SA-VNS-0-1	I/20	f_{best}	0	20	0	1.0
		f_{avg}	10	10	0	**5.92E-3**
	II/20	f_{best}	14	5	1	**1.62E-3**
		f_{avg}	19	1	0	**1.43E-4**
AMA vs. SA-VNS-1-0	I/20	f_{best}	1	19	0	1.0
		f_{avg}	16	4	0	**4.82E-4**
	II/20	f_{best}	13	5	2	**2.37E-3**
		f_{avg}	19	1	0	**1.43E-4**
AMA vs. SA-VNS-1-1	I/20	f_{best}	0	20	0	1.0
		f_{avg}	14	6	0	**1.10E-3**
	II/20	f_{best}	11	8	1	**8.56E-3**
		f_{avg}	15	2	3	**7.98E-4**
	III/15	f_{best}	3	12	0	1.81E-1
		f_{avg}	15	0	0	**6.10E-5**
	IV/10	f_{best}	9	0	1	6.45E-2
		f_{avg}	7	0	3	5.57E-1

Finally, as in [11], to assess the long-run behavior of the algorithm, we run our AMA algorithm under relaxed conditions on four sets of bench-

mark instances. Table 7 summarizes AMA's results under the relaxed cutoff time (AMA_relaxed) compared with the results under the normal cutoff time (AMA) (detailed results in the Appendix), together with the p-values of the non-parametric Wilcoxon signed-rank test. One observes that AMA_relaxed can further improve its best results for 7 instances among the 55 conventional benchmarks and for 8 instances among the 10 new large instances. In addition, 4 new BKV results (p180, p280, p290, and p300) are obtained (Table 8). Among these 4 new BKV results, two of them (p180 and p290) were also reached by AMA under the normal cutoff time, as shown in Table 3, while for p280 and p300, AMA_relaxed further improves the new BKV results established by AMA under the normal cutoff time.

Table 7
Summary of comparative results between AMA under the relaxed cutoff time (AMA_relaxed) and AMA under the normal cutoff time (AMA)

Algorithm pair	Category/Instance	Indicator	Better	Equal	Worse	p-value
AMA_relaxed vs.	I/20	f_{best}	0	20	0	1.0
AMA	II/20	f_{best}	6	14	0	**3.60E-2**
	III/15	f_{best}	1	14	0	1.0
	IV/10	f_{best}	8	1	1	8.50E-2

Table 8
Improved upper bounds discovered by AMA under the relaxed cutoff times

Instance	SA-VNS-1-1	AMA_relaxed
	f_{best} (BKV)	f_{best}
p180	17,879,424	**17,879,398**
p280	69,343,736	**69,343,678**
p290	80,334,743	**80,334,587**
p300	89,325,779	**89,325,647**

To sum, the hybrid AMA algorithm competes very favorably with the state-of-the-art reference algorithms under identical stopping conditions. Unlike the reference algorithms, which are all based on local optimization, AMA combines the population-based approach and local optimization approach. As such, AMA benefits from the advantages of both search approaches to achieve a better balance between search exploration (visiting new and promising regions by the use of crossovers) and exploitation (locating high-quality solutions in a given region by the 3-phase local search).

4 Additional investigations

Additional experiments are performed on a selection of 15 instances to analyze two features of the proposed AMA algorithm: the adaptive crossover selection and the 3-phase local search. Then, we perform a sensitivity analysis to show the impact of parameters.

Table 9
Impact of the adaptive crossover selection

Instance	BKV	MA-OPOX			MA-LOX			MA-OBX			MA-Random			AMA		
		f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
sko_100_1	288,678	**288,678**	288,683.7	27.3	**288,678**	288,691.9	33.0	**288,678**	288,682.4	25.4	**288,678**	**288,678.0**	19.1	**288,678**	**288,678.0**	20.1
sko_100_2	1,806,738	**1,806,738**	**1,806,873.2**	26.7	**1,806,738**	1,806,881.7	25.7	**1,806,738**	1,806,897.9	31.0	**1,806,738**	1,806,904.7	23.0	**1,806,738**	1,806,991.9	18.9
sko_100_3	14,871,217	**14,871,217**	14,871,799.6	32.3	**14,871,217**	**14,871,219.8**	28.2	**14,871,217**	14,871,609.9	16.6	**14,871,217**	14,871,599.2	30.1	**14,871,217**	14,871,596.4	24.1
sko_100_4	2,980,012	**2,980,012**	2,980,051.7	28.2	**2,980,012**	**2,980,019.8**	15.9	**2,980,012**	2,980,031.5	20.6	**2,980,012**	2,980,067.2	30.3	**2,980,012**	2,980,051.6	22.8
sko_100_5	879,038	**879,038**	879,129.2	26.6	**879,038**	879,129.3	34.4	**879,038**	879,116.7	28.9	**879,038**	**879,104.1**	19.9	**879,038**	879,177.7	22.7
p210	28,275,043	**28,275,043**	28,275,064.2	815.8	**28,275,043**	28,275,093.9	759.8	**28,275,043**	28,275,258.5	801.3	**28,275,043**	**28,275,060.9**	843.8	**28,275,043**	28,275,238.2	988.2
p220	34,618,625	**34,618,625**	34,618,933.9	893.6	**34,618,625**	34,618,834.4	756.4	**34,618,625**	**34,618,699.6**	827.2	**34,618,625**	34,618,833.9	705.9	**34,618,625**	34,618,754.8	834.0
p230	42,559,657	**42,559,657**	42,559,737.2	787.2	**42,559,657**	**42,559,711.3**	736.7	**42,559,657**	42,559,727.6	861.8	**42,559,657**	42,559,779.5	818.1	**42,559,657**	42,559,756.3	792.0
p240	43,876,137	43,876,357	43,877,665.0	934.7	43,876,138	43,877,108.1	840.0	43,876,184	43,876,902.5	833.0	**43,876,137**	43,877,175.9	717.5	**43,876,137**	**43,876,729.1**	740.8
p250	50,981,493	**50,981,493**	**50,981,609.6**	990.8	**50,981,493**	50,981,777.0	741.9	50,981,555	50,981,725.4	806.7	50,981,496	50,981,743.7	785.2	50,981,496	50,981,655.8	919.8
p260	58,694,312	**58,694,312**	58,694,467.2	1293.4	**58,694,312**	**58,694,372.2**	1276.6	**58,694,312**	58,694,641.6	1243.4	**58,694,312**	58,694,495.5	1323.5	**58,694,312**	58,694,567.0	1036.9
p270	64,033,556	64,033,576	64,033,790.9	1246.5	64,033,576	64,033,851.6	1383.2	64,033,563	64,033,809.8	1446.5	64,033,565	**64,033,766.2**	1329.8	64,033,558	64,033,985.5	1182.0
p280	69,343,736	69,343,722	69,344,098.7	1182.7	69,343,722	69,343,975.6	1300.6	**69,343,678**	69,344,023.0	1166.3	**69,343,678**	**69,343,810.5**	1334.4	69,343,717	69,343,872.2	1198.3
p290	80,334,743	80,334,729	80,336,870.7	1224.5	80,335,159	80,337,267.2	1063.0	80,334,625	80,336,135.7	1072.1	80,335,160	80,336,580.7	1137.2	**80,334,587**	**80,335,870.3**	1288.8
p300	89,325,779	89,325,808	89,327,931.0	1089.2	**89,325,660**	89,328,196.8	1182.6	89,325,929	89,328,444.4	1094.3	89,325,797	**89,327,708.7**	1307.6	89,325,729	89,328,162.0	1253.1

4.1 Impacts of adaptive crossover selection

To assess the influences of the adaptive crossover selection mechanism (Section 2.3.2), we compare the AMA with four AMA variants: MA-OPOX, MA-LOX, MA-OBX, which respectively use only one crossover, and MA-Random, which randomly selects one crossover among OPOX, LOX and OBX at each generation. For this experiment, each algorithm is run 10 times to solve each instance under the normal cutoff time given in Section 3.2.3. The best results among the compared values are indicated in bold.

Table 9 presents the results of this experiment with the same information as before. Compared with MA-OPOX/MA-LOX/MA-OBX/MA-Random, the AMA algorithm obtains respectively 5/4/5/3 better results, 9/9/9/11 equal results, and 1/2/1/1 worse results. Thus, the adaptive crossover selection globally contributes to the performance of the AMA algorithm.

4.2 Influences of 3-phase local search

To evaluate the usefulness of the 3-phase local search, we compare AMA with three algorithmic variants (GA, MA-SA and MA-IDS). These three variants are generated from AMA by respectively: 1) replacing the 3-phase local search with a perturbation procedure as mutation; 2) adopting only the simulated annealing algorithm in the local search; 3) adopting the iterated descent search in the local search.

Table 10
Influences of the joint use of three local search phases

Instance	BKV	GA			MA-SA			MA-IDS			AMA		
		f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
sko_100_1	288,678	289,642	290,736.2	0.0	**288,678**	288,698.1	22.3	**288,678**	288,695.1	13.6	**288,678**	**288,678.0**	20.1
sko_100_2	1,806,738	1,816,234	1,820,686.8	0.0	**1,806,738**	1,807,046.4	32.8	**1,806,738**	1,807,033.9	27.1	**1,806,738**	**1,806,991.9**	18.9
sko_100_3	14,871,217	14,889,764	14,993,863.2	0.0	**14,871,217**	14,872,443.4	29.3	**14,871,217**	14,871,786.1	26.0	**14,871,217**	**14,871,596.4**	24.1
sko_100_4	2,980,012	2,995,385	3,004,584.7	0.0	**2,980,012**	2,980,175.7	34.7	**2,980,012**	**2,980,012.0**	23.3	**2,980,012**	2,980,051.6	22.8
sko_100_5	879,038	882,889	886,493.7	0.0	**879,038**	879,201.1	26.6	**879,038**	**879,125.2**	20.7	**879,038**	879,177.7	22.7
p210	28,275,043	28,375,121	28,407,437.0	1.4	28,275,068	**28,275,201.4**	604.4	28,275,062	28,286,792.7	906.7	**28,275,043**	28,275,238.2	988.2
p220	34,618,625	34,719,083	34,756,591.5	2.1	34,618,741	34,618,877.9	557.0	34,619,795	34,632,949.3	1033.6	**34,618,625**	**34,618,754.8**	834.0
p230	42,559,657	42,616,455	42,700,031.0	2.1	42,559,875	42,560,136.2	500.3	42,560,511	42,568,651.6	1037.6	**42,559,657**	**42,559,756.3**	792.0
p240	43,876,137	43,958,923	44,035,782.6	2.4	43,876,498	43,877,118.8	335.7	43,876,647	43,882,131.9	965.9	**43,876,137**	**43,876,729.1**	740.8
p250	50,981,493	51,160,801	51,194,360.8	3.1	50,981,731	50,981,935.5	468.3	50,981,723	51,013,323.7	935.4	50,981,496	**50,981,655.8**	919.8
p260	58,694,312	58,839,808	58,913,836.0	2.8	58,694,367	58,694,676.8	704.8	58,694,451	58,710,273.5	1344.1	**58,694,312**	**58,694,567.0**	1036.9
p270	64,033,556	64,235,847	64,266,080.6	4.1	64,033,746	64,034,052.4	784.0	64,046,374	64,064,100.9	1457.3	64,033,558	**64,033,985.5**	1182.0
p280	69,343,736	69,537,155	69,614,181.9	4.4	69,343,862	69,344,654.8	783.7	69,373,064	69,397,124.9	1342.2	**69,343,717**	**69,343,872.2**	1198.3
p290	80,334,743	80,601,581	80,645,518.4	6.2	80,335,088	80,336,052.6	837.8	80,350,442	80,370,742.6	1139.4	**80,334,587**	**80,335,870.3**	1288.8
p300	89,325,779	89,556,940	89,621,990.3	8.7	89,326,143	**89,327,108.3**	752.5	89,334,570	89,376,313.6	1234.2	**89,325,729**	89,328,162.0	1253.1

Table 10 shows the results of AMA compared with the variants GA, MA-SA, and MA-IDS. It can be seen that AMA outperforms GA with 15 better results, MA-SA and MA-IDS with 10 better results, 5 equal results, and 0

worse results. These observations confirm the benefits of the 3-phase local search strategy of the AMA algorithm.

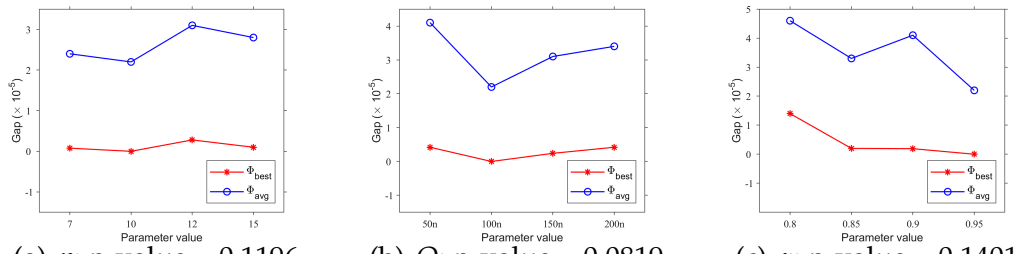
4.3 Sensitivity analysis of the parameters

As shown in Table 1, six parameters are required in the AMA algorithm: the population size p, the number of iterations Q at which the temperature is kept constant, the cooling factor α, the search depth L of iterated descent search, the threshold of applying random perturbation β, and the maximum length of random perturbation η_{max}.

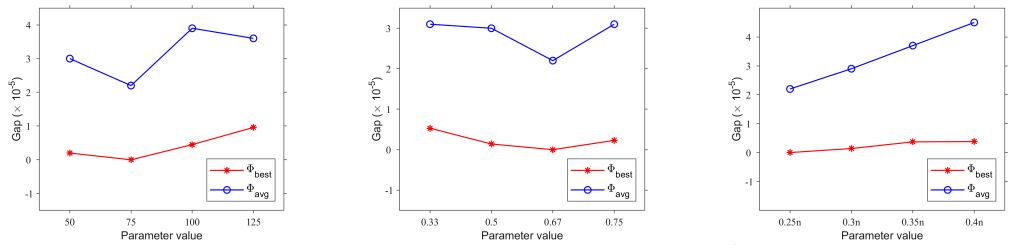
Firstly, the interaction among these six parameters is examined through a 2-level full factorial experiment [43] on 7 randomly selected instances adopted in Section 3.2.1. Each parameter has a high level and a low level, which are the largest and smallest values in Table 1. Since each parameter has two levels, this leads to 64 ($2^6 = 64$) combinations for the six parameters. Each tested instance is independently solved 10 times under the normal cutoff time limit for each parameter combination. Then, for each combination, we consider the average of the best objective values obtained on the 7 instances. The Friedman test shows no statistically significant difference (p-values > 0.05) in terms of the mentioned average, which means that there are no statistically significant interaction effects among these six parameters.

Secondly, the influence of each parameter on the performance of AMA is analyzed through a one-at-a-time sensitivity analysis [44], and the most suitable value of each parameter is determined. In the analysis, each parameter value within the range of possible values as presented in Column 4 of Table 1 is tested while the values of other parameters are fixed to their default values (Column 5) in Table 1. For each parameter value, we run the AMA algorithm independently 10 times under the normal cutoff time limit. The results of the best and average objective values (denoted by Φ_{best} and Φ_{avg}) over 10 runs on the 7 instances are presented in Figure 7, where the value of each parameter is on the X-axis and the best/average gap to the best-known results is on the Y-axis, which presents the best/average gaps to the best-known results over the 7 instances. Figure 7 shows that this calibration experiment leads to the same parameter values as suggested by the irace tool.

Moreover, the Friedman test is carried out to determine whether different values of a given parameter have statistically significant differences in solution quality. The test results indicate that the AMA algorithm is only sensitive to the setting of η_{max} (p-value = 0.0465). The reason for its sensitivity



(a) p: p-value = 0.1196 (b) Q: p-value = 0.0819 (c) α: p-value = 0.1401



(d) L: p-value = 0.0851 (e) β: p-value = 0.0547 (f) η_{max}: p-value = 0.0465

Fig. 7. Sensitivity analysis of the parameters with the significance level of 0.05

to η_{max} is probably that a too small/large η_{max} may lead to slight/strong modification for the solution in the perturbation procedure. Hence, suitable values of η_{max} are critical to the performance of the AMA algorithm. To sum up, the six parameters required by AMA have no significant interaction effects. AMA is sensitive to the settings of η_{max}. Therefore, when it is necessary to tune the parameters, the focus should be on η_{max}.

5 Conclusions and perspectives

The bidirectional loop layout problem is a variant of the well-known quadratic assignment problem with a number of practical applications. This paper introduced an effective adaptive memetic algorithm for the problem. The proposed algorithm is characterized by its adaptive crossover selection strategy and 3-phase local search, whose combination ensures a suitable balance between intensification and diversification of the search process.

Extensive computational assessments on 65 benchmark instances showed that the AMA algorithm competed favorably with the five state-of-the-art algorithms in the literature [11] by reporting equal or improved results.

Given the NP-hard nature of the studied problem, it is no surprise that the proposed algorithm (and any other algorithm) could fail to solve satisfactorily some particular instances. In the future, more studies can be carried out from the following aspects. First, the local search of the proposed algorithm relies only on the insert neighborhood. To further improve the exploitation

capacity of the algorithm, it is worth testing other neighborhoods, in addition to the insertion neighborhood applied in this work. Second, AMA ensures search diversification by the use of three crossovers and diversified local search. However, the algorithm could still stagnate in deep local optima. Therefore, it is worth investigating other diversification strategies including those guided by the learned information. Third, there are few studies on exact algorithms in the literature. Therefore, it would be useful to fill this gap. For this, AMA can be used to generate high quality initial bounds, or to obtain estimates of upper bounds for subproblems during the exact search. Finally, it will be interesting to examine the proposed framework for other layout problems, such as the quadratic assignment problem [1], unidirectional loop layout problem [45], single row facility layout problem [46], AGV scheduling problem [47,48], and the tool indexing problem [7].

Acknowledgments

We are grateful to the reviewers for their valuable comments and suggestions, which helped us improve the manuscript. This work is supported by the National Natural Science Foundation of China [Grant No. 62101125, 72122006], Natural Science Foundation of Jiangsu Province [Grant No. SBK2020040023], and Fundamental Research Funds for the Central Universities [Grant No. 2242022R40067, 2242022k30007].

References

[1] T. C. Koopmans, M. Beckmann, Assignment problems and the location of economic activities, Econometrica: Journal of the Econometric Society (1957) 53–76.

[2] A. B. Mohamed, M. Gunasekaran, R. Heba, A. Zaied, A comprehensive review of quadratic assignment problem: variants, hybrids and applications, Journal of Ambient Intelligence & Humanized Computing (2018) 1–24.

[3] Y. A. Bozer, R. Suk-Chul, A branch and bound method for solving the bidirectional circular layout problem, Applied Mathematical Modelling 20 (5) (1996) 342–351.

[4] Z. Kalita, D. Datta, A constrained single-row facility layout problem, The International Journal of Advanced Manufacturing Technology 98 (5) (2018) 2173–2184.

[5] Z. Kalita, D. Datta, The constrained single-row facility layout problem with repairing mechanisms, in: Nature-Inspired Methods for Metaheuristics Optimization, Springer, 2020, pp. 359–383.

[6] M. Saravanan, S. G. Kumar, Different approaches for the loop layout problems: a review, The International Journal of Advanced Manufacturing Technology 69 (9) (2013) 2513–2529.

[7] T. Dereli, İ. H. Filiz, Allocating optimal index positions on tool magazines using genetic algorithms, Robotics and Autonomous Systems 33 (2-3) (2000) 155–167.

[8] D. Ghosh, Allocating tools to index positions in tool magazines using tabu search, Iima Working Papers (2016).

[9] S. Atta, P. R. S. Mahapatra, A. Mukhopadhyay, Solving tool indexing problem using harmony search algorithm with harmony refinement, Soft Computing 23 (16) (2019) 7407–7423.

[10] V. Liberatore, Circular arrangements and cyclic broadcast scheduling, Journal of Algorithms 51 (2) (2004) 185–215.

[11] G. Palubeckis, An approach integrating simulated annealing and variable neighborhood search for the bidirectional loop layout problem, Mathematics 9 (1) (2021) 5.

[12] U. Benlic, J. K. Hao, Memetic search for the quadratic assignment problem, Expert Systems with Applications 42 (1) (2015) 584–595.

[13] L. X. Tang, Z. C. Li, J. K. Hao, Solving the single row facility layout problem by k-medoids memetic permutation group, IEEE Transactions on Evolutionary Computation (2022).

[14] J. Y. Mao, Q. K. Pan, Z. H. Miao, L. Gao, S. Chen, A hash map-based memetic algorithm for the distributed permutation flowshop scheduling problem with preventive maintenance to minimize total flowtime, Knowledge-Based Systems (2022) 108413.

[15] P. Moscato, Memetic algorithms: A short introduction, New Ideas in Optimization (1999) 219–234.

[16] A. Misevicius, Genetic algorithm hybridized with ruin and recreate procedure: application to the quadratic assignment problem, in: Research and Development in Intelligent Systems XIX, Springer, 2003, pp. 163–176.

[17] U. Benlic, J. K. Hao, Memetic search for the quadratic assignment problem, Expert Systems with Applications 42 (1) (2015) 584–595.

[18] Y. L. Lu, U. Benlic, Q. H. Wu, A hybrid dynamic programming and memetic algorithm to the traveling salesman problem with hotel selection, Computers & Operations Research 90 (2018) 193–207.

[19] Y. Y. Zhu, Y. Q. Chen, Z. H. Fu, Knowledge-guided two-stage memetic search for the pickup and delivery traveling salesman problem with fifo loading, Knowledge-Based Systems 242 (2022) 108332.

[20] G. Gong, Q. Deng, R. Chiong, X. Gong, H. Huang, An effective memetic algorithm for multi-objective job-shop scheduling, Knowledge-Based Systems 182 (2019) 104840.

[21] R. Li, X. Zhao, X. Zuo, J. Yuan, X. Yao, Memetic algorithm with non-smooth penalty for capacitated arc routing problem, Knowledge-Based Systems 220 (2021) 106957.

[22] C. Prins, S. Bouchenoua, A memetic algorithm solving the vrp, the carp and general routing problems with nodes, edges and arcs, in: Recent Advances in Memetic Algorithms, Springer, 2005, pp. 65–85.

[23] J. K. Hao, Memetic algorithms in discrete optimization, in: Handbook of Memetic Algorithms, Springer, 2012, pp. 73–94.

[24] I. Sghir, J. Hao, I. B. Jaâfar, K. Ghédira, A multi-agent based optimization method applied to the quadratic assignment problem, Expert Systems with Applications 42 (23) (2015) 9252–9262.

[25] Y. Lu, J. K. Hao, Q. H. Wu, Hybrid evolutionary search for the traveling repairman problem with profits, Information Sciences 502 (2019) 91–108.

[26] T. Murata, H. Ishibuchi, Positive and negative combination effects of crossover and mutation operators in sequencing problems, in: T. Fukuda, T. Furuhashi (Eds.), Proceedings of 1996 IEEE International Conference on Evolutionary Computation, Nayoya University, Japan, May 20-22, 1996, IEEE, 1996, pp. 170–175.

[27] E. Falkenauer, S. Bouffouix, A genetic algorithm for job shop, in: Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, CA, USA, 9-11 April 1991, IEEE Computer Society, 1991, pp. 824–829.

[28] G. Syswerda, Schedule optimization using genetic algorithms, in: Handbook of Genetic Algorithms, 1991, pp. 332–349.

[29] Y. Lu, J. Hao, Q. Wu, Hybrid evolutionary search for the traveling repairman problem with profits, Information Sciences 502 (2019) 91–108.

[30] J. Ren, J.-K. Hao, F. Wu, Z.-H. Fu, An effective hybrid search algorithm for the multiple traveling repairman problem with profits, European Journal of Operational Research (2022).

[31] K. S. Narendra, M. A. Thathachar, Learning automata: an introduction, Courier Corporation, 2012.

[32] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[33] Y. L. Lu, U. Benlic, Q. H. Wu, A memetic algorithm for the orienteering problem with mandatory visits and exclusionary constraints, European Journal of Operational Research 268 (1) (2018) 54–69.

[34] Y. Xue, H. k. Zhu, J. Y. Liang, A. Słowik, Adaptive crossover operator based multi-objective binary genetic algorithm for feature selection in classification, Knowledge-Based Systems 227 (2021) 107–218.

[35] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem, ORSA Journal on Computing 2 (1) (1990) 33–45.

[36] M. F. Anjos, G. Yen, Provably near-optimal solutions for very large single-row facility layout problems, Optimization Methods & Software 24 (4-5) (2009) 805–817.

[37] H. Zhang, F. Liu, Y. Y. Zhou, Z. Y. Zhang, A hybrid method integrating an elite genetic algorithm with tabu search for the quadratic assignment problem, Information Sciences 539 (2020) 347–374.

[38] P. Hungerlaender, Single-row equidistant facility layout as a special case of single-row facility layout, International Journal of Production Research 52 (5) (2014) 1257–1268.

[39] H. Ahonen, A. G. de Alvarenga, A. R. S. Amaral, Simulated annealing and tabu search approaches for the corridor allocation problem, European Journal of Operational Research 232 (1) (2014) 221–233.

[40] R. Kothari, D. Ghosh, An efficient genetic algorithm for single row facility layout, Optimization Letters 8 (2) (2014) 679–690.

[41] M. Dahlbeck, A. Fischer, F. Fischer, Decorous combinatorial lower bounds for row layout problems, European Journal of Operational Research 286 (3) (2020) 929–944.

[42] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58.

[43] D. C. Montgomery, Design and analysis of experiments, John Wiley & Sons, 2017.

[44] D. M. Hamby, A review of techniques for parameter sensitivity analysis of environmental models, Environmental Monitoring and Assessment 32 (2) (1994) 135–154.

[45] A. C. Nearchou, Meta-heuristics from nature for the loop layout design problem, International Journal of Production Economics 101 (2) (2006) 312–328.

[46] M. Rubio-Sánchez, M. Gallego, F. Gortázar, A. Duarte, Grasp with path relinking for the single row facility layout problem, Knowledge-Based Systems 106 (2016) 1–13.

[47] W. Q. Zou, Q. K. Pan, L. Wang, An effective multi-objective evolutionary algorithm for solving the agv scheduling problem with pickup and delivery, Knowledge-Based Systems 218 (2021) 106881.

[48] W. Q. Zou, Q. K. Pan, L. Wang, Z. H. Miao, C. Peng, Efficient multiobjective optimization for an agv energy-efficient scheduling problem with release time, Knowledge-Based Systems 242 (2022) 108334.

A Detailed results on the 65 instances

We present the results on 65 instances, including 55 classical benchmark instances and 10 new large instances. Table A.1 shows the comparison results of the proposed AMA algorithm under normal and relaxed time limits on the 55 classical benchmark instances. One observes that AMA under the relaxed time improves the best results of AMA under the normal cutoff time for 7 instances, while matching the best results of AMA under the normal cutoff time for the remaining instances.

Table A.1: Comparison of AMA under the relaxed cutoff time with AMA under the normal cutoff time for the 55 benchmark instances in the literature

Instance	AMA_relaxed		AMA		
	f_{best}	t	f_{best}	f_{avg}	t_{avg}
sko_64_1	**74,067**	6	**74,067**	74,067.0	1.9
sko_64_2	**573,458**	6	**573,458**	573,458.0	9.2
sko_64_3	**363,994**	2	**363,994**	363,994.0	2.1
sko_64_4	**243,966**	4	**243,966**	243,966.0	2.8
sko_64_5	**430,063**	9	**430,063**	430,086.0	14.4
sko_72_1	**107,431**	< 1	**107,431**	107,431.0	6.3
sko_72_2	**609,044**	1	**609,044**	609,044.0	9.8
sko_72_3	**1,009,747**	6	**1,009,747**	1,009,747.0	3.8
sko_72_4	**853,106**	12	**853,106**	853,112.9	7.7
sko_72_5	**351,489**	1	**351,489**	351,489.0	5.8
sko_84_1	**155,730**	< 1	**155,730**	155,730.0	1.5
sko_84_2	**447,633**	5	**447,633**	447,633.0	5.1
sko_84_3	**848,904**	11	**848,904**	848,904.0	11.8
sko_84_4	**1,768,175**	22	**1,768,175**	1,768,175.0	10.9
sko_84_5	**1,175,705**	12	**1,175,705**	1,175,705.0	1.5
sko_100_1	**288,678**	26	**288,678**	288,678.0	20.1
sko_100_2	**1,806,738**	16	**1,806,738**	1,806,991.9	18.9
sko_100_3	**14,871,217**	34	**14,871,217**	14,871,596.4	24.1
sko_100_4	**2,980,012**	21	**2,980,012**	2,980,051.6	22.8
sko_100_5	**879,038**	13	**879,038**	879,177.7	22.7
p110	**4,121,976**	21	**4,121,976**	4,121,976.0	9.8

Continued on next page

Table A.1 – continued from previous page

Instance	AMA_relaxed		AMA		
	f_{best}	t	f_{best}	f_{avg}	t_{avg}
p120	**5,712,477**	11	**5,712,477**	5,712,477.0	36.3
p130	**7,163,273**	379	**7,163,273**	7,163,340.3	155.2
p140	**8,531,830**	144	**8,531,830**	8,531,884.9	148.7
p150	**10,223,765**	5340	10,223,814	10,223,815.8	105.4
p160	**13,831,552**	230	**13,831,552**	13,831,603.5	320.4
p170	**15,765,986**	1885	**15,765,986**	15,766,148.5	343.5
p180	**17,879,398**	887	**17,879,398**	17,879,702.1	474.5
p190	**22,570,679**	180	**22,570,679**	22,570,680.1	326.2
p200	**25,948,640**	705	25,948,647	25,948,777.5	466.2
p210	**28,275,043**	764	**28,275,043**	28,275,238.2	988.2
p220	**34,618,625**	2298	**34,618,625**	34,618,754.8	834.0
p230	**42,559,657**	5398	**42,559,657**	42,559,756.3	792.0
p240	**43,876,137**	17073	**43,876,137**	43,876,729.1	740.8
p250	**50,981,493**	1297	50,981,496	50,981,655.8	919.8
p260	**58,694,312**	6387	**58,694,312**	58,694,567.0	1036.9
p270	**64,033,556**	17203	64,033,558	64,033,985.5	1182.0
p280	**69,343,678**	10097	69,343,717	69,343,872.2	1198.3
p290	**80,334,587**	28013	**80,334,587**	80,335,870.3	1288.8
p300	**89,325,647**	5323	89,325,729	89,328,162.0	1253.1
p100-1	**3,003,926**	38	**3,003,926**	3,003,926.9	15.2
p100-2	**3,148,061**	71	**3,148,061**	3,148,061.2	27.5
p100-3	**2,995,405**	31	**2,995,405**	2,995,407.6	24.3
p100-4	**2,709,226**	120	**2,709,226**	2,709,226.2	26.9
p100-5	**3,081,967**	49	**3,081,967**	3,082,064.7	25.6
p150-1	**11,209,802**	446	**11,209,802**	11,210,503.8	198.6
p150-2	**9,592,147**	563	**9,592,147**	9,592,173.8	182.6
p150-3	**10,363,199**	345	**10,363,199**	10,363,269.9	216.7
p150-4	**10,306,319**	71	**10,306,319**	10,306,319.0	147.9
p150-5	**10,345,363**	21	**10,345,363**	10,345,363.0	71.9
p200-1	**26,003,404**	3456	**26,003,404**	26,004,820.2	392.3
p200-2	**25,812,802**	820	**25,812,802**	25,812,802.8	357.0
p200-3	**25,047,165**	205	**25,047,165**	25,047,267.1	340.4
p200-4	**24,982,077**	9843	**24,982,077**	24,983,098.1	406.6
p200-5	**26,576,831**	971	26,576,846	26,577,011.2	392.8

Table A.2 shows the comparison results of the proposed AMA algorithm under the normal and relaxed time limits on the 10 new instances. AMA under relaxed time limits performs better than AMA under normal time limits by reporting 8 better results, 1 equal result and 1 worse result.

Table A.2: Comparison of AMA under the relaxed cutoff time with AMA under the normal cutoff time for the 10 new instances

Instance	AMA_relaxed		AMA		
	f_{best}	t	f_{best}	f_{avg}	t_{avg}
p310	**99528408**	96751	**99528408**	99530164.6	2615.2
p320	**108736979**	2823	108737085	108738522.4	2192.1
p330	**113536422**	15263	113536498	113536614.4	2646.8
p340	**130118929**	4829	130118946	130119296.9	2781.8
p350	**142089169**	33286	142089171	142090468.7	2057.8
p360	**155414558**	6997	155414562	155417613.2	3673.2
p370	**172402930**	38398	172403103	172403409.6	4303.4
p380	**184229058**	148063	184234412	184236844.3	3205
p390	**185317344**	31239	185317425	185323757.6	4187
p400	208908967	158130	**208908761**	208912600.8	3107.1