A Restricted Neighbourhood Tabu Search for Storage Location Assignment Problem

Jing Xie [1], Yi Mei[1], Andreas T. Ernst[2], Xiaodong Li[1], Andy Song[1]
[1]School of Computer Science and IT, RMIT University, Melbourne, 3000, Australia
Email: {jing.xie, yi.mei, xiaodong.li, andy.song}@rmit.edu.au
[2] Commonwealth Scientific and Industrial Research Organisation (CSIRO), Melbourne,3169, Australia
Email: andreas.ernst@csiro.au

Abstract—**The Storage Location Assignment Problem (SLAP) is a significant optimisation problem in warehouse management. Given a number of products, each with a set of items with different popularities (probabilities of being ordered), SLAP is to find the best locations for the items of the products in the warehouse to minimise the warehouse operational cost. Specifically, the operational cost is the expected cost of picking the orders. Grouping constraints are included to take the practical considerations into account in the problem. That is, the items belonging to the same product are more desirable to be placed together. In this paper, the SLAP with Grouping Constraints (SLAP-GC) is investigated, and an efficient Restricted Neighbourhood Tabu Search (RNTS) algorithm is proposed to solving it. RNTS adopts the problem-specific search operators to maintain solution feasibility, and the tabu list to prevent searching back and forth. RNTS was empirically compared with the mathematical programming method and a previously designed Genetic Programming method, which is demonstrated to be the state-of-the-art algorithm for SLAP-GC. The experimental results on the real-world data show that RNTS outperforms the state-of-the-art algorithms for SLAP-GC in terms of solution quality and speed. It managed to achieve optimal solutions for most of the small-scale instances much faster and outperformed the Genetic Programming method in terms of both solution quality and running time on all the test instances.**

Keywords*—*Warehouse Optimisation, Storage Location Assignment Problem, Tabu Search, Meta-heuristics

I. Introduction

Warehouse plays an essential role in supply chain. The optimisation of warehouse system is a complex real-world problem. There are four types of activities in a warehouse: receiving, storage, picking and delivering. These four activities are correlated with each other. Previous study indicates that order picking is the most labour-intensive task, and it consumes about 50%–60% of the total operational effort [1]. The efficiency of order picking is affected by many factors. For example, the layout of a warehouse, allocation of the products and the routing strategies determine the travelling distance of the pickers when they pick the orders. Order-batching policy is one of the factors to be considered when choosing the routing and picking strategies. Storage Location Assignment Problem (SLAP) [2] is the problem that consider all these factors.

SLAP aims to find the best layout arrangement of a warehouse (the locations to assign the given products) under a number of practical considerations. The overall goal of this problem is to minimise the warehouse operational cost. In literature, the operational cost could refer to the space utilization efficiency [3] or the peak load [4]. Relocation cost may also be considered when solving this problem [5]. So far, there is no unified definition of operational cost. Different costs may be defined in different real-world scenarios, such as the demand of the industry partner or the bottleneck existing in the warehouse. The most widely used estimation is the total picking distance [6]. When minimising the total picking distance, the popularity of products or the correlation of products are typically considered [7]. Specifically, the products with higher demand should be expected to be picked more frequently. These products are more desirable to be assigned to locations that are closer to the loading zone (Pick up/Delivery point, or simply P/D point). The total picking distance may also be affected by the routing strategies that have been used [8] or the availability of resources in the warehouse [4]. With all these factors, the problem could be extremely hard to solve. Hence, only part of these aspects may be found in a study.

In practice, one may consider that the items belonging to the same product are more desirable to be placed together. For example, a T-shirt product may have different sizes (e.g., XS, S, M, L, and XL). Different size may have dramatically different popularities. However, one still prefers to put all of them together to facilitate management when possible. Taking this concern into account, grouping constraints for products are defined and a variant of SLAP called SLAP with Grouping Constraints (SLAP-GC) was proposed in [9]. In this paper, we focus on the investigation of SLAP-GC.

There are plenty of approaches to solving SLAP. Comprehensive surveys are presented in [2] and [10]. The majority of the work in this area is focused on finding the exact locations for products in a warehouse. Deterministic methods such as branch-and-bound can guarantee the optimality of solutions but not suitable for big instances due to its poor scalability. Stochastic approaches such as simulated annealing, tabu search, and genetic algorithms are more widely used in real-world scenarios to obtain a reasonably good solution within the given limited time budget. For example, a branch-and-bound algorithm is used to solve SLAP to minimise the sum of storage space cost and handling cost in a warehouse using class-based storage strategy in [11]. Simulated annealing is employed to find the allocation of products to classes to minimise the sum of the storage space cost and order picking cost in a warehouse using class-based storage policy in [12]. Tabu search is used to handle the relocation problem in a warehouse with dynamic operating policy in [5]. Genetic algorithm is used for a two-stage iterative approach to solving a paper reel layout problem in [13]. These optimisation techniques usually find good trade-off between solution quality and computational budget. However, they all require careful design of the operators so that they can adapt to problem-specific features.

In addition to the meta-heuristic approaches for SLAP, another trend in the broader field of combinatorial optimisation is the use of hyper-heuristic approaches [14]. These methods search in the space of heuristics instead of direct solutions for the problem. Many related works can be found for Job Shop Scheduling (JSS) [15] [16]. A Genetic Programming (GP)-based hyper-heuristic approach was proposed to search for the best allocation rules for a variant of the SLAP model in our previous study [9]. It includes the grouping constraints of the products. The optimisation performance of this GP method was compared with a traditional branch-and-cut [17] approach in smaller instances in terms of solution quality and running time. Also, a sampling method is proposed for this GP method to improve the scalability in [18].

Despite the intensive study of SLAP, only the GP-based method [9] has been proposed for SLAP-GC. Furthermore, the problem has not been fully explored in larger instances, of which the size of the search space become prohibitive for branch-and-cut approach. In this case, it is unclear whether the GP-based method performs well on large real-world instances. This paper investigates SLAP-GC more deeply and designs a more effective Hybrid Tabu Search (HTA) algorithm using problem-specific information to improve the solution quality especially for the large-scale instances.

The introduction of the grouping constraints raises a significant feasibility issue. It increases the probability of the occurrence of infeasible solutions which violate the shelf capacity constraint, i.e., the number of items placed on some shelves exceed their capacity. This issue makes it difficult for the existing scheduling operators to be directly applied to SLAP-GC, and problem-specific search operators to tackle feasibility issue need to be designed. In this paper, four local search operators and two exploring operators are developed, all of which can guarantee the feasibility of solutions. The designed operators are then embedded into the tabu search framework, which is a meta-heuristic optimisation method proposed in [19] and [20]. Tabu search has been successfully applied to many real-world combinatorial optimisation problems [21] [22]. The advantage of tabu search is mainly attributed to its unique process, which consists of both short-term and long-term memories. The short-term memory enables the exploration to be closer to the local optimum, and the long-term memory, implemented by the tabu list, helps the search to escape from the vicinity of a local optimum. The resultant tabu search, called the Restricted Neighbourhood Tabu Search (RNTS), addresses the challenging feasibility issue by adopting the problem-specific search operators, which guarantee solution feasibility by restricting the neighbourhood during the search process. Experimental studies are carried out to compare the proposed RNTS with the branch-and-cut and the GP-based method [9] on the real-world test instances. The results demonstrate that RNTS outperforms both of the compared algorithms in terms of solution quality and running time for solving SLAP-GC.

The rest of the paper is organized as follows: Section II presents the brief description of SLAP-GC. The operators and the main procedure of the proposed RNTS are described in Section III. The experimental studies are conducted, and the results are shown in Section IV. The conclusion in Section V discusses the future research directions that can be extended from this study.

TABLE I. EXAMPLE DATA

Item	Details
A	FL04 63 PINK 12
B	FL04 51 GREY 16
C	FL04 14 GREY 18
D	PS56 161 BLK 14
E	PS56 27 PINK 10

II. BACKGROUND

This study is conducted in a warehouse storing garments such as schoolwear and sportswear. This section presents a brief description of the problem. We start with the definitions of the key terms that will be used in the problem description:

1) *Item*: In the investigated warehouse, an item represents a set of clothes with the same colour, size, and Stocking Keeping Unit (SKU) number. Each item has a unique index i ranging from 1 to N. For example, given a data shown as “Item A: FL04 63 PINK 12”, the SKU number of the item is “PS031”, the size is “XXL”, the colour is black, and the picking frequency P_i is 1234.
2) *Product*: A product is a set of items sharing the same SKU number in the warehouse. Each product has a unique index t ranging from 1 to T. They may have different colour or size. For example, in Table I, item A, B and C belong to the same product.
3) *Group*: A group is a subset of a product, which contains a set of items with the same SKU number.
4) *Row*: A row represents the arrangement of the items on a shelf. It contains a set of groups in order. All groups in a row should be in descending order in terms of average picking frequency of the groups. The total number of items in a row should not exceed the shelf capacity C.
5) *Solution*: A solution is a representation of the warehouse layout. It is a vector of rows. The total number of items in each row in a feasible solution should not exceed the maximum capacity of the corresponding shelf. The length of the vector is a fixed number, which is equal to the number of shelves M.

The problem is to assign a set of products to a set of locations (with index $l = 1, ..., N$). Each product is allowed to be split into at most two groups and assigned to different areas. The Integer Linear Programming (ILP) [23] model of the problem is presented in Eqs. (1) – (9). The objective is to minimise the total picking frequency-weighted distance. Detailed description of the problem can be found in [9].

$$\min \ \varsigma(x) = \sum_{i=1}^{N} \sum_{l=1}^{N} 2\, P_i \,(V_l + H_l)\, x_{il} \tag{1}$$

$$s.t. \quad \sum_{i=1}^{N} x_{il} = 1, \quad l = 1, ..., N \tag{2}$$

$$\sum_{l=1}^{N} x_{il} = 1, \quad i = 1, ..., N \tag{3}$$

$$\sum_{l=1}^{N} y_{tl} \leq 2, \quad t = 1, ..., T \tag{4}$$

$$x_{il} \leq \sum_{t=1}^{T} A_{it} \left(y_{tl} + \sum_{j=1}^{N} A_{jt} x_{j,l-1} \right), \ \ i, l = 1, ..., N \tag{5}$$

$$\sum_{t=1}^{T} y_{tl} = 1, \quad \forall \ l \ mod \ C = 1 \tag{6}$$

$$\sum_{t=1}^{T} B_{it} x_{il} \leq \sum_{t=1}^{S} B_{iT} \left(\sum_{t=1}^{T} A_{it} y_{tl} \right), \quad i = 1, ..., N \tag{7}$$

$$\sum_{i=1}^{N} A_{it} x_{il} \geq y_{tl}, \quad t = 1, ..., T, \quad l = 1, ..., N \tag{8}$$

$$x_{il}, y_{tl} \in \{0, 1\}. \tag{9}$$

where V_l and H_l are the vertical and horizontal distance of the location l to the P/D point. A_{it} equals 1 if item i is in the product t, and 0 otherwise. B_{it} takes 1 if the item i is the most popular item in product t, and 0 otherwise. The decision variable x_{il} equals 1 if the item i is assigned to the location l, and $y_t l$ takes 1 if the location l is the starting point of the product t.

III. Methodologies

In this section, the proposed Restricted Neighbourhood Tabu Search (RNTS) for solving SLAP-GC is described, including the solution representation, evaluation, search operators and tabu list structure.

A. Solution Representation

In RNTS, a solution S is directly represented as the layout of of the warehouse, i.e., $S = (S_{ij})_{M \times C}$, where M is the number of shelves, and C is the capacity of each shelf. S_{ij} stands for the item placed on the i^{th} shelf at the j^{th} location. An example is given in Eq. (10), in which $M = 5$ and $C = 4$, and there are 20 items to be allocated. The matrix clearly shows the layout of the warehouse. For example, items 1, 3, 6 and 11 are placed in shelf M_1, as presented in the following matrix.

$$S = \begin{array}{c} \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \end{array} \begin{array}{c} \begin{array}{cccc} C_1 & C_2 & C_3 & C_4 \end{array} \\ \begin{pmatrix} 1 & 3 & 6 & 11 \\ 13 & 16 & 2 & 7 \\ 9 & 10 & 15 & 14 \\ 20 & 8 & 4 & 5 \\ 12 & 17 & 18 & 19 \end{pmatrix} \end{array}. \tag{10}$$

The above representation naturally satisfies the capacity constraint, since there is no item placed outside the shelves (the index cannot exceed C). Then, only the grouping constraints need to be examined and tackled for this representation.

B. Solution Evaluation

The objective function of a solution S can be directly calculated as follows:

$$\varsigma(S) = \sum_{i=1}^{M} \sum_{j=1}^{C} 2(i + j) P(S_{ij}), \tag{11}$$

where $2(i + j)$ is the Hamming distance between the j^{th} location of the i^{th} shelf and the P/D point, and $P(S_{ij})$ is the picking frequency of the item S_{ij}.

In terms of feasibility evaluation regarding the grouping constraints, the examination is rather inefficient, since one needs to examine the number of groups of each product in the given layout. Therefore, we only do such examination in the initialization phase, and prevent time-consuming feasibility check during the search process by designing restricted neighbourhood search operators that can guarantee the solution feasibility. The restricted neighbourhood search operators are described next.

C. Search Operators

Given a feasible solution, four sorting-based search operators are first designed to improve it without violating the feasibility. The design of these operators is based on the specific features of the problem. Besides, the sorting mechanism adopted by the operators enables them to reach a nearby local optimum quickly. The prerequisite for designing these operators is the feasibility of the solutions. Specifically, each operator must take a feasible solution as input and guarantees to output another feasible solution. The feasibility of the solution is determined by the following two aspects. First, each product cannot be divided into more than two groups. Second, the total number of items in each row must be equal to the capacity of the corresponding shelf.

1) `rowSort`: The `rowSort` function sorts all the groups of items placed in the same row in descending order in terms of the average picking frequency of these groups. It is the optimal assignment of given groups on a shelf. Otherwise, there exists two adjacent groups g_1 and g_2 on the shelf that has $P_{g_1} < P_{g_2}$ and $D_{g_1} < D_{g_2}$ (P_g and D_g are the average picking frequency and the starting distance of a group g correspondingly). A better solution can be found by swapping these two groups without violating the constraints. This is consistent with the intuition that groups with higher picking frequency should be more likely to be placed in the locations that are closer to the P/D point. Given a feasible solution, in which the total number of items equals the size of the shelf, the resultant solution is still feasible after applying the `rowSort` function since the number of items is not changed.

2) `allRowSort`: The `allRowSort` function sorts all the rows (shelves) in descending order in terms of the average picking frequency. After the sorting, the row with the highest average picking frequency is moved to the closest shelf to the P/D point, while the row with the lowest average picking frequency is moved to the farthest shelf. Similarly, this is the minimum ordering of the shelves when groups on the shelves are given. Obviously, such an operation does not change the feasibility since there is no movement of items cross shelves.

3) `groupSort`: Unlike the `rowSort` and `allRowSort` functions, the `groupSort` function allows exchange of items between different shelves. This function does not change the split of products. It moves one or several groups from one shelf to another, and move equal number of items back. This strategy maintains the feasibility of the solutions. The sorting mechanism is similar to that of `rowSort`. Given a set of equal-sized groups, `groupSort` sorts the groups in descending order in terms of average picking frequency.

4) `itemSort`: Given a solution, for each item i, let P_i and D_i be the picking frequency of item i and the distance from the location of item i to the P/D point, respectively. Then, for any two items i_1, i_2, if $P_{i_1} > P_{i_2}$ and $D_{i_1} > D_{i_2}$, then a better solution can always be obtained by swapping the location of these two items. In order to maintain feasibility in terms of the grouping constraints, each item can only be exchanged with the items belonging to the same product. This way, the number of groups is not changed after the item swap. Given a feasible solution, in which each product is divided into no more than two groups, the maximal number of groups in the resultant solution after the item swap is still no more than two. Based on the above item swap operation, the `itemSort` function, sort all the items in each product in the decreasing order of their picking frequency without changing the location-allocation of the product. For example, assuming that we are given the two groups of a product $G_1 = (i_1, i_2, i_3)$ and $G_2 = (i_4, i_5, i_6)$, where $P(i_1) > P(i_2) > \cdots > P(i_6)$. G_1 is placed in a 3-location block in which the locations have the distance of 3, 4, and 5 to the P/D point, and G_2 is placed in another block whose distances to the P/D points are 4, 5, and 6, respectively. Then, after the item sorting, $G_1 = (i_1, i_2, i_4)$ and $G_2 = (i_3, i_5, i_6)$.

The above four search operators retains the solution feasibility by keeping the grouping structure, i.e., the size of the groups for each product. Therefore, the best possible solution that can be obtained by the above four search operators mainly depends on the given grouping structure. In other words, if the given grouping structure is poor, then the four operators can by no means obtain promising solutions to the problem. To address this issue, two additional operators that can modify the size of groups without violating the feasibility constraints are designed. They are described as follows:

5) `resizeGroupsInSameRow`: This function changes the split of products of which the two groups are in the same row of the original solution. Specifically, given two groups g_1 and g_2 belonging to the same product, this function can only be called when g_1 and g_2 are placed in the same row. Without loss of generality, one can assume that g_1 is placed at locations that are closer than the P/D point than g_2. In order not to change the whole structure of the shelf too much, only the following two modifications are allowed to be conducted in g_1 and g_2: (1) remove the last item of g_1 and add it to the head of g_2; (2) remove the first item of g_2 and add it to end of g_1. As a consequence, this function will return at most two new solutions. More formally, the `resizeGroupsInSameRow` function can be called when all the following conditions are met:

1) Groups g_1 and g_2 are from the same product.
2) Groups g_1 and g_2 are in the same row of the original solution.
3) At least one of the group in g_1 and g_2 has more than 1 item. Otherwise, no new split can be generated.

It is obvious that the feasibility of solutions can be maintained by the `resizeGroupsInSameRow` function. Given a solution, a set of neighbouring solutions can be generated by applying the `resizeGroupsInSameRow` function to each row and possible pair of groups in the row. Preliminary studies showed that the conditions to apply the `resizeGroupsInSameRow` function is too restricted so that there is no neighbouring solution generated by this method in some cases. Hence, a more general function called *resizeGroupsOnDiffRow* is designed to expand the neighbourhood.

6) `resizeGroupsInDiffRow`: This function relaxes the condition that the two groups of items must be in the same row, which does not happen frequently in practice. In other words, the function focuses on dealing with groups that are placed in different rows. In order to retain solution feasibility, only two simple scenarios are considered in this function. Assuming that two distinct rows r_1 and r_2 are given, which consist of groups $g_{11}, \ldots, g_{1k_1}$ and $g_{21}, \ldots, g_{2k_2}$, respectively. Among the groups, one may find pairs of groups (g_{1i_1}, g_{2j_1}), $(g_{1i_2}, g_{2j_2}), \ldots$, that belong to the same product but are placed in different rows. Then, the two scenarios are described as follows:

In the first scenario, only one pair of such groups is found. For the sake of convenience, let the group be denoted as (g_{1i_1}, g_{2j_1}). Similar to the `resizeGroupsInSameRow` function, the change of split for the groups (g_{1i_1} and g_{2j_1}) can be accomplished by moving the last item of g_{1i_1} to g_{2j_1}, or moving the first item of g_{2j_1} to g_{1i_1} (suppose g_{1i_1} is placed at locations that are closer to the P/D point than g_{2j_1}). However, Such a modification will cause infeasibility for both rows. Assuming that one item is moved from the group g_{2j_1} in r_2 to the group g_{1i_1} in row r_1 . Then r_1 will have one more item, and r_2 will have one vacant location. To make the resultant solution feasible again, a simple alternative is to examine whether there are groups in r_1 with only one item and if so, move one of them to the vacant location of r_2.

In the second scenario, at least two pairs of groups are found. For the sake of simplicity, we assume that there are two pairs of groups found, namely (g_{1i_1}, g_{2j_1}) and (g_{1i_2}, g_{2j_2}). Group g_{1i_1} and g_{1i_2} are in row r_1, and g_{2j_1} and g_{2j_2} are in row r_2. Then, the feasibility of solutions can be maintained by moving the items of both pairs simultaneously, but in the opposite way. That is, when moving an item from g_{1i_1} to g_{2j_1}, an item in g_{2j_2} must be moved to g_{1j_2} at the same time so that both rows still have a valid number of items.

When combining the `resizeGroupsInSameRow` and `resizeGroupsInDiffRow` functions with the four sorting-based search operators, the search process is expected to be able to explore different split of products and achieve the local optimum for each split quickly.

D. Tabu List

In RNTS, the tabu list is an array of moves, and each move is identified by the name of product and the sizes of groups it is split into. This is a relatively simple structure. When checking a move is tabu or not, the name of the product to be resized is check first first. If the name is in the list, the sizes of new split are verified. Only those splits not in the tabu list are allowed. And since `itemSort` will change the assignment of items to these groups, we only need to care about the size of the groups. This tabu list structure guarantees that a recently tried split of a product is not retried in a certain number of iterations.

E. Tabu Search Framework

The standard tabu search framework is described in Algorithm 1, where $\mathcal{N}(S_{curr})$ indicates the neighbourhood of the solution S_{curr}, defined by some move operator(s). First, a solution is initialized randomly or heuristically. Then, in each iteration, the best non-tabu neighbouring solution is identified and replace the current solution, and the tabu list is updated by adding the new solution and removing the solutions whose tabu duration reach the tabu tenure.

Algorithm 1 The standard tabu search framework

```
Initialize a solution $S_{init}$;
Set the best-so-far solution $S^* \leftarrow S_{init}$;
Set the current solution $S_{curr} \leftarrow S_{init}$;
Initialize the tabu list tabuList $\leftarrow \emptyset$;
while Stopping criteria are not met do
    Set the next solution $S_{next} \leftarrow null$;
    for each $S' \in \mathcal{N}(S_{curr})$ do
        if $S'$ is not tabu then
            if $S'$ is better than $S_{next}$ or $S_{next} = null$ then
                $S_{next} \leftarrow S'$;
            end if
        end if;
    end for
    if $S_{next}$ is better than $S^*$ then
        $S^* \leftarrow S_{next}$;
    end if
    $S_{curr} \leftarrow S_{next}$;
    Add $S_{next}$ into tabuList, and remove the tabu elements whose duration reach the tabu tenure;
end while
return $S^*$;
```

In SLAP-GC, feasibility is a challenging issue to address. In this paper, the issue is resolved by a simple strategy that maintains the feasibility of solutions throughout the search process. To this end, a feasible initial solution is required. In order to generate a feasible initial solution, the following initialization is proposed.

1) Initialisation: The initialization method used in RNTS is described in Algorithm 2. First, a split method is randomly selected from a set of candidate methods. Four candidate split methods are defined, namely `splitRandom`, `splitMaxDiff`, `splitMinCost` and `splitRandomFixed`. `splitRandom` simply splits a product randomly into two groups. `splitMaxDiff` splits a product at the point where two items have the biggest difference on picking frequency. `splitMinCost` splits a product so that two groups have the lowest fitness value assuming that they are assigned to the best location in the warehouse. `splitRandomFixed` splits a product at an instantly generated random point. After obtaining the groups for all the products, they are sorted in the decreasing order of average picking frequency and insert into the shelves in turn. When inserting each group, the best available location is identified to place the group. Note that such an insertion procedure does not always generate a feasible solution. Therefore, the split and insertion trial is repeated until a feasible solution is generated.

Algorithm 2 Initialization in RNTS

```
Set $S_{init} \leftarrow null$;
while $S_{init} = null$ do
    Randomly select a split method split from the predefined set of candidate methods candidates;
    Split the products into a set of groups groups by split;
    Insert the groups into the warehouse by a greedy heuristic to obtain a layout $S$;
    if $S$ is feasible then
        $S_{init} \leftarrow S$;
    end if
end while
return $S$;
```

In RNTS, given the six search operators defined in Section III-C, the neighbourhood of the current solution is defined by `resizeGroupsInSameRow` and `resizeGroupsInDiffRow`, and `rowSort`, `allRowSort`, `groupSort`, and `itemSort` are used for fine tuning. In other words, before evaluating each new split of products generated by `resizeGroupsInSameRow` and `resizeGroupsInDiffRow`, it is first improved by a local search process with move operators of `rowSort`, `allRowSort`, `groupSort` and `itemSort`. This way, the search process can converge faster. The framework of RNTS is described in Algorithm 3, where the function `LocalSearch()` is described in Algorithm 4.

Algorithm 3 The RNTS framework

```
Initialize a solution $S_{init}$;
Set the best-so-far solution $S^* \leftarrow S_{init}$;
Set the current solution $S_{curr} \leftarrow S_{init}$;
Initialize the tabu list tabuList $\leftarrow \emptyset$;
while Stopping criteria are not met do
    Set the next solution $S_{next} \leftarrow null$;
    for each $S'$ generated by resizeGroupsInSameRow or resizeGroupsInDiffRow do
        if $S'$ is not tabu then
            $S'' \leftarrow$ LocalSearch(S');
            if $S''$ is better than $S_{next}$ or $S_{next} = null$ then
                $S_{next} \leftarrow S''$;
            end if
        end if;
    end for
    if $S_{next}$ is better than $S^*$ then
        $S^* \leftarrow S_{next}$;
    end if
    $S_{curr} \leftarrow S_{next}$;
    Update tabuList;
end while
return $S^*$;
```

In Algorithm 4, in each iteration, the four sorting operators are randomly shuffled before being applied to the current solution in turn. This shuffling can increase the diversity of the local search. Based on the preliminary studies, the maximum number of iterations is configured to 32 to save computational effort.

Algorithm 4 $S' \leftarrow$ `LocalSearch(S)`

```
$S' \leftarrow S$;
$IT \leftarrow 32$;
$FQ \leftarrow \{itemSort, rowSort, groupSort, allRowSort\}$;
$LP \leftarrow 0$;
$Prev \leftarrow 0$;
while $LP < IT$ and $fitness(S') > Prev$ do
    $Prev = fitness(S')$;
    Randomly shuffle $FQ$;
    for $f \in FQ$ do
        Apply $f$ to $S'$;
        $Prev \leftarrow fitness(S')$;
    end for
    $LP \leftarrow LP + 1$;
end while
return $S'$;
```

IV. Experimental Studies

The proposed RNTS was implemented based on the JAVA system used in [24]. For comparison, the branch-and-cut approach was coded in Java using the Gurobi 5.5.0 library [25]. The GP system was implemented based on ECJ21 [26]. All the experiments were run on a PC with Intel Core i7-3770 CPU and 8G RAM. The compared algorithms were tested on 20 instances of different problem size, which were generated from real-world data. For each instance, both RNTS and the GP approach were run 30 times independently, and the best, worst and average fitness over the 30 independent runs were recorded. The stopping criterion of the GP method was set to 30 generations while two stop criteria were set for RNTS. One is the maximum iteration, which was configured to n^2, where n is the total number of locations in the warehouse. The other is the maximum running time, which was set to $1.3^C s$. Such a setting was based on a rough estimation of the running time for the GP approach according to the previous experimental results. The size of tabu list is set to 8 in the experiment.

Table II and Table III show the experimental results for the 30 independent runs. In Table II, the column "Min", "Max", "Avg" and "Std" stand for the minimal fitness, maximal fitness, average fitness and standard deviation of the fitnesses obtained over the 30 runs. "Time" indicates the computational time by second. In Table III, $\Delta_{GP_{Min}}^{RNTS_{Min}}$, $\Delta_{GP_{Max}}^{RNTS_{Max}}$, and $\Delta_{GP_{Avg}}^{RNTS_{Avg}}$ are the percentage difference between its results and that of the GP method in terms of the minimal fitness, maximal fitness, average fitness respectively. Similarly, $\Delta_{Gurobi}^{RNTS_{Min}}$ and $\Delta_{Gurobi}^{RNTS_{Avg}}$ represent the percentage difference between its average fitness and minimal fitness and the Brach-and-cut result, which is the global optimum or a lower bound.

From the tables, we can see that RNTS performed better than the GP method on all the 20 test instances. The advantages of RNTS over the GP method become significant when the problem size increases. It performed the same as the GP method on the first 4 instances (both achieved the global optima). For the remaining larger instances, RNTS outperformed the GP method by 0.0%–0.74% in terms of the best performance and 0.01%–1.03% in terms of average performance. RNTS consistently achieved the global optimal solution for 8 out of the first 12 instances (0.0% difference to the Gurobi result), while the GP method only managed to obtain the global optimum in the smallest 4 instances. In addition, RNTS has a much smaller standard deviation than the GP method, which implies that the performance of RNTS is much more stable than the GP method. This tendency is more obvious when the problem size becomes larger. MannWhitney U [27] test of the GP method and RNTS shows that the distributions of the results for instances $5-20$ differed significantly ($p-value < 0.05$, two-tailed). Those two methods both consistently obtained optimal solutions in instances $1-4$ for 30 runs. Hence, the corresponding results is shown as NaN.

To make a more comprehensive comparison between the performance of RNTS and the GP method, we plot the converge curves of the best cases for instances 17–20, where the average running time of RNTS was longer than GP. The results show that RNTS was able to obtain better solutions, and the reason was not it ran for longer time. We can see from Fig. 1 that for instances 18 and 19, the initial solution of RNTS was better than the GP method and it consistently performed better during the whole search process. For instance 17 and 20, the GP method started with a much better result, but the converge curve were very flat, and it begun to fall behind at very early stage of the evolution. Overall, RNTS can converge much faster than the GP method, and its performance was little affected by the quality of the initial solution.

V. Conclusion

This study proposed an efficient Restricted Neighbourhood Tabu Search (RNTS) Algorithm for a Storage Location Assignment Problem with Grouping Constraints (SLAP-GC). The proposed method is compared with state-of-art GP-based method and Branch-and-cut approach to solving the SLAP-GC. The experimental results show that RNTS outperforms the other two algorithms in terms of both solution quality and running time on all the test instances.

Several further extensions can be explored based on this study. The GP method can evolve reusable allocation rules that can generate a complete solution in a short time, while the proposed RNTS can efficiently search the local and global optimum solutions. We may be able to design a more efficient algorithm by hybridising these two methods. Furthermore, the application of the proposed method in more complex scenarios, especially in dynamic environments, may also be investigated.

References

[1] JP van den Berg and WHM Zijm. Models for warehouse management: Classification and examples. *International Journal of Production Economics*, 59(1):519–528, 1999.

[2] Jinxiang Gu, Marc Goetschalckx, and Leon F McGinnis. Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, 177(1):1–21, 2007.

[3] Lu Chen, André Langevin, and Diane Riopel. The storage location assignment and interleaving problem in an automated storage/retrieval system with shared storage. *International Journal of Production Research*, 48(4):991–1011, 2010.

TABLE II. THE COMPARISON OF FITNESSES FOR THE PROPOSED RNTS METHOD PREVIOUSLY PROPOSED METHODS FOR SLAP-GC

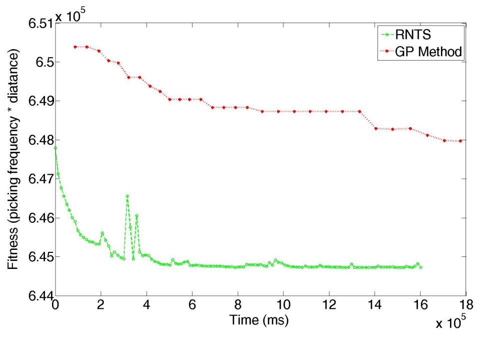
No.	**Size**	**Gurobi**		**Corresponding Fitness Values Shown in $GP/RNTS$ Format**				
	$(M \times C)$	OPT/LB	$Time(s)$	Min	Max	Avg	Std	$Time(s)$
1	5×5	1685	2	1685/1685	1685/1685	1685.00/1685.00	0.00/0.00	2.75/0.60
2	5×5	2554	< 1	2554/2554	2554/2554	2554.00/2554.00	0.00/0.00	4.10/0.09
3	5×5	2280	< 1	2280/2280	2280/2280	2280.00/2280.00	0.00/0.00	3.42/0.16
4	5×5	2350	< 1	2350/2350	2350/2350	2350.00/2350.00	0.00/0.00	3.82/0.19
5	8×8	7522	10	7522/7522	7531/7522	7523.43/7522.00	1.76/0.00	10.61/8.16
6	8×8	9506	7	9506/9506	9512/9506	9507.00/9506.00	1.66/0.00	14.46/6.16
7	8×8	9231	4	9233/9231	9241/9231	9236.73/9231.00	2.41/0.00	11.64/8.16
8	8×8	11280	91	11286/11280	11328/11280	11301.37/11280.00	13.63/0.00	9.13/4.56
9	10×10	31342	4	31364/31342	31401/31342	31392.50/31342.00	12.31/0.00	25.20/13.79
10	10×10	15781	27	15784/15781	15798/15781	15789.43/15781.00	3.91/0.00	23.05/13.79
11	10×10	32997	10	33056/32997	33122/32997	33071.33/32997.00	17.74/0.00	20.46/13.80
12	10×10	15561	7200	15639/15561	15767/15564	15689.10/15563.27	25.64/1.05	19.38/13.79
13	20×20	195046	3600	196662/195511	197912/195536	196962.03/195519.70	311.59/5.86	344.61/190.11
14	20×20	133017	3600	134354/133373	135464/133392	134762.73/133383.00	295.19/4.61	303.97/190.11
15	20×20	57876	3600	58471/58194	58813/58210	58610.17/58202.40	86.77/3.81	253.04/190.10
16	20×20	288651	3600	290122/289316	291116/289398	290501.40/289342.00	213.97/19.36	282.18/190.13
17	30×30	643104	34251	647971/644725	650553/644737	648817.43/644726.27	485.68/2.77	1762.67/1605.75
18	30×30	592862	28812	599269/594850	600980/594886	600505.53/594857.60	519.89/8.62	1277.46/1606.86
19	30×30	603347	31690	609826/605992	612064/606019	611047.80/605998.00	533.07/7.11	1682.31/1607.17
20	30×30	614520	28884	620579/616885	622363/616900	622026.13/616886.00	477.89/2.75	1576.85/1607.51

1. OPT/LB is the optimal solution or a lower bound calculated by Gurobi using branch-and-cut method. Details of the experiments can be found in [9].
2. Min, Max, Avg, Std are the minimum, maximum, average fitness and their standard deviation of 30 runs for GP and RNTS methods. $Time$ shows the elapsed time in seconds.

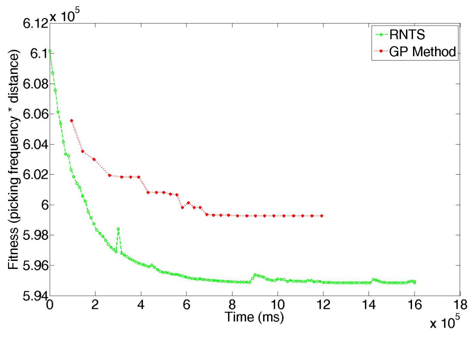
TABLE III. STATISTICAL TEST OF THE RESULTS FOR RNTS AND PREVIOUSLY PROPOSED METHODS FOR SLAP-GC

No.	**Size**	**Difference of the Fitness for different Methods**					**MannWhitney U-test**	
	$(M \times C)$	$\Delta_{GP_{Min}}^{RNTS_{Min}}$	$\Delta_{GP_{Max}}^{RNTS_{Max}}$	$\Delta_{GP_{Avg}}^{RNTS_{Avg}}$	$\Delta_{Gurobi}^{RNTS_{Min}}$	$\Delta_{Gurobi}^{RNTS_{Avg}}$	$p-value$	$z-value$
1	$5 * 5$	0.00%	0.00%	0.00%	0.00%	0.00%	NaN	NaN
2	$5 * 5$	0.00%	0.00%	0.00%	0.00%	0.00%	NaN	NaN
3	$5 * 5$	0.00%	0.00%	0.00%	0.00%	0.00%	NaN	NaN
4	$5 * 5$	0.00%	0.00%	0.00%	0.00%	0.00%	NaN	NaN
5	$8 * 8$	0.00%	−0.12%	−0.02%	0.00%	0.00%	$1.46E-11$	6.7523
6	$8 * 8$	0.00%	−0.06%	−0.01%	0.00%	0.00%	$3.06E-04$	3.6102
7	$8 * 8$	−0.02%	−0.11%	−0.06%	0.00%	0.00%	$9.27E-13$	7.141
8	$8 * 8$	−0.05%	−0.43%	−0.19%	0.00%	0.00%	$1.11E-12$	7.1165
9	$10 * 10$	−0.07%	−0.19%	−0.16%	0.00%	0.00%	$1.06E-12$	7.1225
10	$10 * 10$	−0.02%	−0.11%	−0.05%	0.00%	0.00%	$9.91E-13$	7.1318
11	$10 * 10$	−0.18%	−0.38%	−0.23%	0.00%	0.00%	$1.05E-12$	7.1236
12	$10 * 10$	−0.50%	−1.30%	−0.81%	0.00%	0.01%	$1.67E-11$	6.7326
13	$20 * 20$	−0.59%	−1.22%	−0.74%	0.24%	0.24%	$2.97E-11$	6.6481
14	$20 * 20$	−0.74%	−1.55%	−1.03%	0.27%	0.27%	$2.88E-11$	6.6523
15	$20 * 20$	−0.48%	−1.04%	−0.70%	0.55%	0.56%	$2.86E-11$	6.6536
16	$20 * 20$	−0.28%	−0.59%	−0.40%	0.23%	0.24%	$3.00E-11$	6.6466
17	$30 * 30$	−0.50%	−0.90%	−0.63%	0.25%	0.25%	$1.26E-11$	6.7731
18	$30 * 30$	−0.74%	−1.02%	−0.95%	0.33%	0.34%	$2.84E-11$	6.6548
19	$30 * 30$	−0.63%	−1.00%	−0.83%	0.44%	0.44%	$2.75E-11$	6.6595
20	$30 * 30$	−0.60%	−0.89%	−0.83%	0.38%	0.38%	$1.53E-11$	6.7455

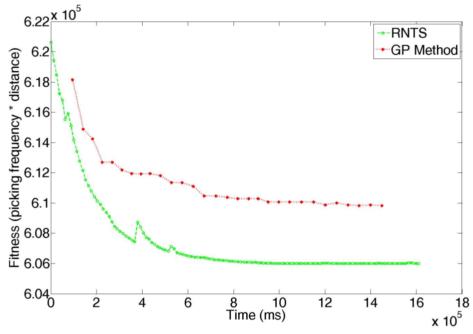
1. Δ_B^A is calculated by $\frac{A-B}{A} \times 100\%$.
2. NaN is presented for cases that are not applicable.



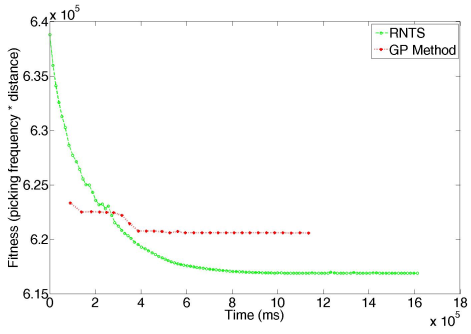
(a) Instance 17



(b) Instance 18



(c) Instance 19



(d) Instance 20

Fig. 1. The Comparison of the Converge Curve for the Best Cases of RNTS and the GP Method on Different Instances

[4] Pierre Montulet, André Langevin, and Diane Riopel. Minimizing the peak load: an alternate objective for dedicated storage policies. *International journal of production research*, 36(5):1369–1385, 1998.

[5] Lu Chen, André Langevin, and Diane Riopel. A tabu search algorithm for the relocation problem in a warehousing system. *International Journal of Production Economics*, 129(1):147–156, 2011.

[6] Semih Önüt, Umut R Tuzkaya, and Bilgehan Doğaç. A particle swarm optimization algorithm for the multiple-level warehouse layout design problem. *Computers & Industrial Engineering*, 54(4):783–799, 2008.

[7] EA Frazele and Gunter P Sharp. Correlated assignment strategy can improve any order-picking operation. *Industrial Engineering*, 21(4):33–37, 1989.

[8] Che-Hung Lin and Iuan-Yuan Lu. The procedure of determining the order picking strategies in distribution center. *International Journal of Production Economics*, 60:301–307, 1999.

[9] Jing Xie, Yi Mei, Andreas T Ernst, Xiaodong Li, and Andy Song. A genetic programming-based hyper-heuristic approach for storage location assignment problem. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 3000–3007. IEEE, 2014.

[10] Bart Rouwenhorst, B Reuter, V Stockrahm, GJ Van Houtum, RJ Mantel, and WHM Zijm. Warehouse design and control: Framework and literature review. *European Journal of Operational Research*, 122(3):515–533, 2000.

[11] Gajendra Kumar Adil et al. A branch and bound algorithm for class based storage location assignment. *European Journal of Operational Research*, 189(2):492–507, 2008.

[12] Gajendra Kumar Adil et al. Efficient formation of storage classes for warehouse storage location assignment: a simulated annealing approach. *Omega*, 36(4):609–618, 2008.

[13] GQ Zhang, J Xue, and KK Lai. A genetic algorithm based heuristic for adjacent paper-reel layout problem. *International Journal of Production Research*, 38(14):3343–3356, 2000.

[14] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer, 2010.

[15] Su Nguyen, Mengjie Zhang, Mark Johnston, and Kay Chen Tan. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *Evolutionary Computation, IEEE Transactions on*, 17(5):621–639, 2013.

[16] Su Nguyen, Mengjie Zhang, Mark Johnston, and Kay Chen Tan. Automatic programming via iterated local search for dynamic job shop scheduling. 2014.

[17] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[18] Jing Xie, Yi Mei, Andreas T Ernst, Xiaodong Li, and Andy Song. Scaling up solutions to storage location assignment problems by genetic programming. In *Simulated Evolution and Learning*, pages 691–702. Springer, 2014.

[19] Fred Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[20] Fred Glover. Tabu searchpart ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[21] Jun-Qing Li, Quan-Ke Pan, PN Suganthan, and TJ Chua. A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. *The international journal of advanced manufacturing technology*, 52(5-8):683–697, 2011.

[22] Kathryn A Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European journal of operational research*, 106(2):393–407, 1998.

[23] Laurence A Wolsey. *Integer programming*, volume 42. Wiley New York, 1998.

[24] RW Harder, RR Hill, and JT Moore. A java universal vehicle router for routing unmanned aerial vehicles. *International Transactions in Operational Research*, 11(3):259–275, 2004.

[25] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2013.

[26] David R White. Software review: the ecj toolkit. *Genetic Programming and Evolvable Machines*, 13(1):65–67, 2012.

[27] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 18(1):50–60, 1947.