McMaster University

Advanced Optimization Laboratory

Title:

TABU SEARCH APPROACHES FOR MULTIPLE-LEVEL WAREHOUSE LAYOUT PROBLEM WITH ADJACENCY CONSTRAINTS

Author:

Guoqing Zhang

AdvOl-Report No. 2001/6
August 2001, Hamilton, Ontario, Canada

TABU SEARCH APPROACHES FOR MULTIPLE-LEVEL WAREHOUSE LAYOUT PROBLEM WITH ADJACENCY CONSTRAINTS

G Q Zhang [1]
Department of Computing and software
McMaster University, Hamilton, Canada L8S 4L7

[1]Corresponding author (Tel: 905 525-9140 Ext. 27739, Fax: 905 524-0340, E-mail: gzhang@mcmaster.ca).

Abstract

In this paper, we investigate a new multiple-level warehouse layout problem, *Multiple-level warehouse layout problem with adjacency constraints* (MLWLPAC). An IP model is proposed to formulate the problem. We present a three-phase assignment method to construct feasible layout for each sequence code of items. A tabu search (TS) approach is presented to solve the MLWLPAC problem. Some improed versions, such as the greedy TS and the dynamic neighborhood (NB) search strategies are proposed. The standard TS, greedy TS and dynamic neighborhood strategy are evaluated with many tests. The results show that the proposed approaches can reduce the transportation cost drameticly.

0.1 Introduction

Warehouse layout is one of the key issues of warehousing, which involves all logistical components. An effective warehouse layout can distinctly reduce operation costs. A number of results for warehouse layout problem have been reported according to different storage policies. The reviews of warehouse models and layout policies can be found in Cormier and Gunn (1992), Francis et al. (1992), and Zhang (2000c). In the literature, the majority of the proposed approaches have considered single-level warehouse layout problem. Very little is reported about multiple-level warehouse layout problems, although these are popular in the manufacturing and service sector. Zhang et. al (2000b) addressed a multiple-level warehouse layout problem and proposed a heuristic to assign multiple-level space. However, the inventory of items in the problem is limited to be equal or less than the capacity of a cell. When inventory requirement of a item type exceeds the capacity of a cell, it is not avoidable to split the type for storing in cells. The adjacency constraint for the split item is of practical requirement in most of applications.

In this paper, under a dedicated storage policy, we study a multple-level warehouse layout problem with adjacency constraints. The problem is described simply as follows:

- There are multiple-level storage areas. Each area consists of identical cells.
- There is one elevator to transfer items from the storage floors to the ground. The elevator capacity constraints are not considered.
- There are different item types and each type is characterized by demand, inventory, horizontal unit transportation cost (HUTC) and vertical unit ransportation cost (VUTC). The inventory for some types may exceed the cell capacity.
- Only the item types whose inventory requirement exceeds cell capacity can be split in a special way: at most have one odd part and be put in adjacent cells.
- One cell can store more than one item type, and one cell can hold at most two odd parts of item types as defined in the next section.
- The objective is to minimize the total cost of horizontal and vertical transportation.

An early method in the assignment of space to different items is the cube-per-order index (COI) rule by Hestkett (1963). Malmborg and Krishnakumar (1989) extended COI to a multiple command warehouse operation environment and proved that under certain conditions, COI produces the shortest traveling cycle-time. The most of previous research focused on warehouse layout where each cell holds at most one item type. A linear programming (LP) model to mix item types in the storage cell was proposed by Wilson (1977). The LP model permits that any item type can be split and put into more than one cell, even the item types whose inventory requirement is less or equal to the capacity of a cell. Zhang et al. (2000a) proposed an integer programming model where only item types whose inventory requirement exceeds the capacity of a cell can be split. A two-stage heuristic is proposed for the single-level warehouse layout problem with adjacency constraint. Different from the problem, the warehouse layout problem in this paper is of multiple-level. Vertical transportation cost needs to be considered.

The earliest work about multiple-level layout problems was done by Johnson (1982). He investigated the problem of relative location of facilities in a multiple-floor building. A computerized heuristic solution procedure was offered. This approach is in part based on the CRAFT heuristic (Francis et al. 1992). Kaku et al. (1988) studied a multi-story office building layout problem.

Meller and Bozer presented several algorithms for multiple-level facility layout problem (MLFLP). MULTIPLE (Bozer and Meller, 1994) is a method that extends a well-known facility layout algorithm (CRAFT) to multiple floors problem using a spacefilling curves. MULTIPLE is similar to CRAFT except for the exchange procedure and layout formation. Their objective function is identical to that of CRAFT, i.e., a objective based on rectilinearly distance between department centers. SABLE (Meller and Bozer, 1996) uses a SA based heuristic combined with spacefilling curves.

Although the MLFLP is related to MLWLPAC, the above approaches for MLFLP cannot apply to our problem because MLWLPAC has its special characteristics. The four obvious features of MLWLPAC that are different from MLFLP are:

- The objective of our problem. It is to minimize total transportation cost between the facility to the I/O port instead of transportation cost among facilities.
- The sub-problem of grouping item types for a cell.
- The horizontal and vertical travel unit cost is item type dependent.
- The adjacency constraints for split item types.

In the rest of the paper, we first give the description of the problem in the section 2. Section 3 proposes the assignment method based on sequence coding. Tabu search approach is developed in section 4. Two improved approaches are given in section 5 and 6 respectively. Section 7 tests the approaches and makes comparision for different parameters and approaches. Section 8 concluse our results.

0.2 Problem Formulation

To formulate the problem, the following indices and notations are used:

- $j = 1, 2, \ldots, J$: indices for item types
- $l = 1, 2, ..., L$: indices for warehouse levels
- $k = 1, 2, ..., K_l$: indices for storage cells of level l
- $Z_{k_1 k_2}$ equals 1 if cells k_1 and k_2 are adjacent; 0 otherwise
- D_{lk}: horizontal distance from cell k of level l to I/O port or elevator
- Q_j: demand of item type j
- U_j: inventory requirement of item type j
- C_j^h: horizontal unit transportation cost of type j
- C_{jl}^v: vertical unit transportation cost of type j to level l
- A: the capacity of a cell

If an arbitrary item type j splits into j_R ($j_R = 1 + \lfloor (U_j - 1)/A \rfloor$) parts, then its demand Q_j and inventory requirement U_j will split proportionally into Q_{j_p} and U_{j_p}, $p = 1, \ldots, j_R$ ($j_R \geq 1$, for a non-spit type j, $j_R = 1$), satisfying:

- each part can be stored in a single cell.
- j_R is minimal. That is, the inventory requirements $U_{j_p} = A$ for all j_p, $1 \le p \le j_R - 1$, and $U_{j_{j_R}} \le A$.

Note: $U_{j_{j_R}}$can be equal to A. In this case, $\lfloor U_j/A \rfloor = U_j/A$.
A *split type*, *odd part*, and *integral part* are defined as follows:

Definition 0.2.1 *Item type j is a split type if $j_R > 1$. Otherwise, it is a non-split type. For a split type j, if $U_{j_{j_R}} < A$, the j_R-th part of type j is called an odd part, the j_p-th part of type j, $1 \le p \le j_R - 1$, is called an integral part.*

The following variables are introduced:

$$X_{j_plk} = \begin{cases} 1, & \text{if the } p\text{-th part of type } j \text{ is assigned to cell } k \text{ of level } l, \\ 0, & \text{otherwise.} \end{cases}$$

Now the definition of the adjacency relationship of two cells and of a set of cells is given as follows.

Definition 0.2.2

- *Two cells k_1 and k_2 are adjacent if and only if they are in the same level, share a common boundary, and $k_1 \neq k_2$.*
- *A set of cells S are adjacent if and only if there exists a path $P = (k_1, \ldots, k_{|S|})$ ($k_j \in S$, with no repeated cells) such that k_j and k_{j+1} are adjacent, for $j = 1, \ldots, |S| - 1$. The adjacency of set S is defined by P).*

In the MLWLPAC, the adjacency constraints require the parts of a split type to be assigned to adjacent cells.

Definition 0.2.3 *Let a split type j be* assigned *to the adjacent cells defined by a path $P = (k_1, \ldots, k_{j_L})$. This assignment is* feasible *if the j_L-th part of type j is assigned to either cell k_1 or cell k_{j_L}.*

A mathematical formulation of the MLWLPAC is as follows:

$$(P): \quad \begin{aligned} \text{Min} \quad & \sum_{l=1}^{L}\sum_{k=1}^{K_l}\sum_{j=1}^{J}\sum_{p=1}^{j_R} Q_{j_p}(D_{lk}C_j^h + C_{jl}^v)X_{j_plk} && (1) \\ \text{s.t.} \quad & \sum_{l=1}^{L}\sum_{k=1}^{K_l} x_{j_plk} = 1, \text{ for } j = 1, ..., J, \ p = 1, ..., \mathrm{J}R && (2) \\ & \sum_{j=1}^{J}\sum_{p=1}^{j_R} U_{j_p}X_{j_plk} \le A, \quad \text{for } l = 1, ..., L, \ k = 1, ..., K_l && (3) \\ & X_{j_pl_1k_1} + X_{j_{p+1}l_2k_2} \le Z_{k_1k_2} + 1, \ \forall j, \ \ p, \ l_1, \ l_2, \ k_1, \ k_2 && (4) \\ & \sum_{j=1,\, j_R>1}^{J} X_{j_Rlk} \le 2, \quad \text{for } l = 1, ..., L, \quad k = 1, ..., K_l && (5) \\ & x_{j_plk} = 0, 1, \ \forall j, p, l, k. \end{aligned}$$

where the constraints (1) to (5) correspond to:

- Each (part of an) item type is assigned to exactly one cell.
- The cell capacity is not violated.
- Parts of a split type must be stored in adjacent cells.
- A cell can hold at most two odd parts.

Like classical warehouse layout problem, the low transportation cost can be expected by putting the item type with high throughput per unit to cells close to the I/O port. But, different from the obvious priority of cells in single-level warehouse layout problem, the priority of cells in different levels of MLWLPAC is dependent on item types since HUTC and VUTC of the item types are not same.

Problem P is an NP-hard problem even when the adjacency constraints are relaxed and $L = 1$, as proved in Lai (1999). Due to complexity of the problem, solution methods for large-scale problems are limited to heuristics. In this paper, a TS based heuristic is presented to solve MLWLPAC.

0.3 Assignment Procedure Based on Sequence Coding

0.3.1 Encoding Problem

The sequence of item type index is used as the representation of a solution. For example, a sequence of 6 types is given as 2-3-6-1-4-5. The key problem is to establish the relationship between the sequence and the feasible layout of item types under the constraints. A three-phase assignment approach is proposed, which includes grouping, assignment, and level re-assignment.

- The grouping phase is to group types for a cell according to the sequence by considering the cell capacity and two odd part constraints.
- Assignment operation uses comparison-based method to lay the groups to cells in different levels of the warehouse. That is, first try to assign a group to the cell that is empty and closest to the I/O port or elevator at each level, then compare the transportation costs in each level, finally put the group in the cell with the smallest cost.
- Level re-assignment forms a feasible layout to meet adjacency constraints and try to optimize the layout of each level.

The details of the above operations are discussed in the following subsection.

0.3.2 Grouping Approach

To assign items to cells, item types need to be grouped for a cell according to the sequence of the type index. The following notations are used in the grouping method:

g	:	the index of a group, which corresponds to a cell
N_odd_g	:	the number of odd parts in group g
U_g	:	inventory volume of group g
k_l	:	the first available cell k in level l
l_0	:	the lowest level which has at least one available cell

The grouping steps are:

1. Initialization: without loss of generality, assume a sequence of item types is $\{j|\ j = 1, ..., J\}$.
   Let $O = \{j|$ *type* j *has not been grouped and sorted as the sequence*$\}$.
   Let $j^i =$ *the* i*th member of* O.
   Set group $g = 1,\ U_g = 0,\ N\_odd_g = 0,\ i = 1$.
2. If the inventory volume $U_{j^i} > A$, go to step 4.
3. If $U_g + U_{j^i} \leq A$, set $U_g = U_g + U_{j^i}$, delete the type j from the list O, go to step 5.
4. If $U_g + U_{j^i_{j_R}} \leq A$ and $N\_odd_g < 2$, set $U_g = U_g + U_{j^i_{j_R}}$, delete j from the list O, $N\_odd_g = N\_odd_g + 1$.
5. If the set O is empty, stop.
6. If $U_g = A$ or $i = |O|$, let $g = g + 1,\ U_g = 0,\ N\_odd_g = 0$, i=1, go to step 2. Otherwise, let i=i+1, go to step 2.

The demand of a group g, denoted as Q_g, is defined as $\sum_{j \in g} C^h_j Q_{j_{j_R}}$. Average demand $\overline{Q}_g$ of the group g is defined as:

$$\overline{Q}_g = \begin{cases} Q_g & \text{if } N\_odd_g = 0 \\ \frac{\sum_{j \in g, j \neq j'} C^h_j Q_j + C^h_{j'} Q_{j'}}{j'_R} & \text{if } N\_odd_g = 1 \\ \frac{\sum_{j \in g, j \neq j', j \neq j''} C^h_j Q_j + C^h_{j'} Q_{j'} + C^h_{j''} Q_{j''}}{j'_R + j''_R - 1} & \text{if } N\_odd_g = 2 \end{cases}$$

where j' and j'' are split types in the group g.

0.3.3 Assignment Approach

Assume the result of grouping is: $g = 1, 2, \cdots, G$, an assignment approach is given as follows:

---

1. Initialization: let $l = 1,\ g = 1,\ k_l = 1,\ l_0 = 1$.
2. If $N\_odd_g = 0$ (the group does not have an odd part), proceed as follows:
   (a) Assign the group g to k_l, compute the transportation cost $cost_{gl}$ of group g in k_l.
   (b) Let $l = l + 1$. If $l = L + 1$, find l^* to make $cost_{gl^*} = \min_l\ cost_{gl}$. Otherwise, go to step (a).
   (c) $k_{l^*} = k_{l^*} + 1$. If $k_{l^*} = K_{l^*} + 1$, set $l_0 = l + 1$.
3. If $N\_odd_g = 1$ (the group has one odd part of type j'), proceed as follows:
   (a) If the number of available cells in level l is less than j'_R ($j'_R - 1 + k_l > K_l$), go to step (c). Otherwise, assign the group g and other parts of j' as follows:
   If $Q_g > \overline{Q}_g$, assign the group g to k_l, the integral parts of j' to $k_l + i, i = 1, ..., j'_R - 1$.
   Otherwise, assign the group g to cell $k_l + j'_R - 1$, the integral parts of j' to $k_l + i, i = 0, ..., j'_R - 2$.
   (b) Compute the transportation cost $cost_{gl}$ of the above assignment.

(c) Let $l = l + 1$. If $l = L + 1$, find l^* to make $cost_{gl^*} = min_l\ cost_{gl}$. Otherwise go to step (a).

(d) $k_{l^*} = k_{l^*} + j'_R$. If $k_{l^*} = K_{l^*} + 1$, set $l_0 = l + 1$.

4. If $N\_odd_g = 2$, (the group g has two odd parts of type j' *and* j''. Without loss of generality, assume $C^h_{j'} Q_{j'_1} > C^h_{j''} Q_{j''_1}$), proceed as follows:

   (a) If the number of available cells in level l is less than $j'_R + j''_R - 1$ ($j'_R + j''_R - 2 + k_l > K_l$), go to step (c). Otherwise, assign the group g and other parts of j' and j'' as follows:
   If $Q_g > C^h_{j'} Q_{j'_1}$, assign the group g to k_l, the integral parts of j' to $k_l + i, i = 1, ..., j'_R - 1$ and the other parts of j'' to $k_l + j'_R + i, i = 0, ..., j''_R - 2$.
   Otherwise, assign the group g to $k_l + j'_R - 1$, the integral parts of j' to $k_l + i, i = 0, ..., j'_R - 2$ and the other parts of j'' to $k_l + j'_R + i, i = 0, ..., j''_R - 2$.

   (b) Compute the transportation cost $cost_{gl}$ of above assignment.

   (c) Let $l = l + 1$. If $l = L + 1$, find l^* to make $cost_{gl^*} = min_l\ cost_{gl}$. Otherwise, go to step (a).

   (d) $k_{l^*} = k_{l^*} + j'_R + j''_R - 1$. If $k_{l^*} = K_{l^*} + 1$, set $l_0 = l + 1$.

5. Let $g = g + 1$. If $g \leq G$, *let* $l = l_0$ go to step 2. Otherwise, compute the total transportation cost, then stop.

In the above assignment method, the adjacency requirement is not considered, since the layout will be changed by following level re-assignment, which will meet the requirement.

0.3.4 Level Re-Assignment

A re-assignment approach of each level is developed to meet the adjacency constraint. The approach not only provides a feasible layout, but also provides a near optimal solution for each level under the grouping result.

LevelReAssign

1. Set $l = 1$,

2. Sort the cells of level l in ascent sequence in terms of distance to the I/O port or elevator. Without loss of generality, assume the sequence of cells is $k = 1, \cdots, K_l$,
   Let $C_l = \{k | k\ is\ a\ available\ cell\ in\ level\ l, and\ D_{lk} < D_{lk+1}\}$
   Let $k^i = the\ ith\ number\ of\ C_l$, set $i = 1$,

3. Sort groups in level l in descending order of $\overline{Q}_g$. Without loss of generality, assume the sequence is $g = 1, ..., G_l$. Let g=1.

4. Re-assign the groups as following:

   (a) If $N\_odd_g = 0$ (the group has no odd part of any type), assign the group g in k^i, delete k^i in C_l.

   (b) If $N\_odd_g = 1$ (the group has one odd part of type j'),

- Find a $feasible$ assignment for type j' beginning with k^i. If such an assignment does not exist in the cell k^i, let $i = i + 1$, find again until $k^i > K_l$ or a $feasible$ assignment is found. If $k^i > K_l$, go to step 8.
- If $Q_g > \overline{Q}_g$, put the group g in the cell k^i, the integral parts of the type j' in other cells of the path.
  Otherwise, put the group at the end of the path, put the integral parts of the type j' in other cells of the path from k^i.
- delete the cells in the path from C_l.

(c) If $N\_odd_g = 2$, (the group g has two odd parts of type j' *and* j'', without loss of generality, assume $C^h_{j'}Q_{j'_1} > C^h_{j''}Q_{j''_1}$), do the following:

If $Q_g > C^h_{j'}Q_{j'_1}$,

- Find two $feasible$ assignments for types j' *and* j'' beginning with k^i. If any of the two assignments do not exist in the cell k^i, let $i = i + 1$, find again until $k^i > K_l$ or the two paths are found. If $k^i > K_l$, go to step 8.
- Put the group g in the cell k^i, put the integral parts of the type j' in other cells of one path. Do the same for j''. Delete the cells in the paths from C_l.

Otherwise,

- Find a path of cells which begins with k^i and for which the path length is equal to $j'_R + j''_R - 1$. If such a path does not exist for the cell k^i, let $i = i + 1$, find again until $k^i > K_l$ or a path is found. If $k^i > K_l$, go to step 8.
- First, put the integral parts of the type j' in cells of the path, then put group g, finally, put the integral parts of the type j'' in the other cells of the path. Delete the cells in the path from C_l.

(d) Compute the transportation cost of group g as in the above assignment.

5. Let $g = g + 1$. If $g \leq G_l$, *set* $i = 1$, go to step 4.
6. Compute the new horizontal transportation cost in level l and total transportation cost in level l.
7. Set $l = l + 1$. If $l \leq L$, go to step 2. Otherwise, compute total transportation cost, stop.
8. There is no feasible assignment.

0.4 Tabu Search for MLWLPAC

Tabu search has had considerable success as a generalized solution approach for many complex optimization problems. It was proposed and formulated by Glover in 1986. Unlike SA and GA, TS is not a method based on Monte Carlo or probability. This method is a memory-based search method and explores the solution space by moving from a solution to its best neighbor. A structure of the standard TS is given in Glover and Laguna (1997).

An application of the TS approach includes the following issues:

- solution representation and evaluation

- initialization
- neighborhood definition, or candidate list
- tabu data structure and tabu tenure (size)

In the next discussion, a TS based heuristic is developed for the MLWLPAC. Some improved techniques are presented for the problem.

0.4.1 The Solution Representation and Layout

As discussed in section 3, the sequence of item type index is used to represent the solution. The assignment method based on the sequence is given. The transportation cost of each sequence (solution) is computed according to the assignment.

0.4.2 Initialization

Some reports have suggested that a good initial solution can provide more chances to get better results for the TS approach.

Definition 0.4.1 *The horizontal throughput per unit of a type j is defined as $C_j^h D_j/U_j$, denoted as HTP_j. The vertical throughput per unit of a type j is defined as $C_{1j}^v D_j/U_j$, denoted as VTP_j. The mixed throughput per unit is defined as $(HTP_j + VTP_j)/2$, denoted as MTP_j.*

SHTP are used to express the sequence of type index in the descending order of HTP_j, SVTP for the sequence of type index in the descending order of VTP_j, and SMTP for the sequence of type index in the descending order of MTP_j.

In the implementation, SVTP are used as the initial solution. Then, the three-phase assignment method is applied to item layout and the transportation cost is computed. In the experiment discussed in section 7.1, two other methods also are observed to produce an initial solution .

0.4.3 Neighborhood Search

Neighborhood definition has a large impact on the effectiveness of a tabu search (Glover, 1995). The pair-wise exchange of two item types is a popular way of obtaining a neighborhood solution. For J types, the number of neighborhoods based on pair-wise exchange is $J(J-1)/2$. When J is large, the computational effort for the best neighbor search is enormous. This is particularly so if the result of a pair exchange needs a lot of computation. To reduce the considerable computation time and effort, context information can be used to limit the neighbor search. However, this depends on the problem in question. For example, James and Buchanan (1998) compared several candidate selection strategies for a scheduling problem. In fact, the space of neighbor search should include chances of improvement. In this problem, the close throughput per unit should be put in the same cell or nearby cells. So the exchange of types with close throughput per unit provides more chances for improvement. The neighborhood $NB(j)$ of a type j is defined as follows.

Definition 0.4.2 *The horizontal neighborhood $NB^h(j)$ of a type j is defined as a set of such types that meet the following conditions:*

- *The type is not in the same group of j after a grouping phase.*
- *The type is after type j and the distance to type j is less than J/θ_h in SHTP sequence.*

Similarly, the *vertical neighborhood* $NB^v(j)$ and *mixed neighborhood* $NB^m(j)$ of a type j can be defined based on parameters θ_v and θ_m.

For each type j, the horizontal neighbor solutions include pair-exchange with the types in $NB^h(j)$. The total number of pair-exchanges are at most $J \times (J/\theta_h - 1)$.

For example, let the SHTP sequence be 1-2-3-4-5-6-7-8-9-10, the $\theta_h = 2$. After grouping, there are four groups, $group_1$: 1, 7, and 9, $group_2$: 2 and 5, $group_3$: 3 and 4, $group_4$: 6, 8, and 10. So, for type 2, the $NB^h(j)$ will be 3, 4, and 6. (Note that type 5 is in the same group as type 2.)

The parameters θ_h, θ_v and θ_m have an impact on the solution quality. It is obvious that the smaller the parameters, the bigger the neighborhood of a type. A big neighborhood needs considerable computational effort. Specially, if setting θ_h or θ_v and θ_m to 1, the neighborhood will include all pair-wise exchanges. In implementation, the algorithm must make a trade-off between the solution quality and running time.

Another kind of random neighborhood strategy, namely RRW-CL, is also employed in one of present algorithms. The strategy is obtained by revising the RW-CL candidate list, which is proposed for the scheduling problem by James and Buchanan (1998). The neighborhood solutions in the RRW-CL strategy are:

1. choose the fixed number (J/θ_s) of types, namely *source types*,
2. randomly produce a source type j,
3. randomly produce J/θ_d types which are not j, namely *destination types*,
4. make a pair-exchange for type j and each destination types which are not in the same group as j,
5. go to step 2 until a total of J/θ_s source types are produced.

where, the θ_s and θ_d are parameters to control the size of neighborhood solutions. So the total number of pair-exchange are at most $J/\theta_s \times J/\theta_d$. The RRW-CL is the same as RW-CL except for step 3: destination types are generated in random way instead of using a fixed position. This experiment shows the revision to RW-CL can significantly improve the solution quality.

0.4.4 Tabu Data Structure

Recency-based memory

Short term memory is to keep track of solutions attributes that have changed during the recent past, and are also called $recency-based\ memory$. Recency-based memory tries to prevent the recurrence of the same solutions from the recent past. There are several basic kinds of move attributes which serve as memory function. One of the common methods is to keep track of the accepted pair-exchange, for example, in iteration 1, if accepting the move of exchange of types 2 and 5, the exchange of types 2 and 5 will be prohibited in the next several iterations (*tabu tenure*). This is called tabu active.

In the implementation, a two-dimensional array *tabu* is used to record tabu information. As in Chiang and Chi (1998), recency-based memory uses the upper triangle of the array *tabu*. (The lower triangle of array is used for frequency-based memory.) For recency-based memory, if the exchange of types i and j is accepted, the tabu information is given as follows:

If $i < j$, $tabu[i][j] = itera_{current} + tenure$;

otherwise, $tabu[j][i] = itera_{current} + tenure$.

Frequency-based memory

Many researchers have reported that *long term memory*, call *frequency-based memory*, can improve the solution quality. The lower triangle of array *tabu* is used to count the frequency of pair exchange, the number of exchange of two types. That is, if the exchange of types i and j is accepted, then the lower triangle of the *tabu* is updated as follows:

If $i < j$, $tabu[j][i] = tabu[j][i] + 1$;

otherwise, $tabu[i][j] = tabu[i][j] + 1$.

The frequency-based information can be used intensively to search in the area or divert from the area.

Parameter setting

Based on the experiment, the tabu tenure is set equal to 6.

0.4.5 Tabu Search Algorithm

The following notations are used in the algorithm:

$cost_{iter}$	:	the lowest travel cost in the current iteration
$cost_{min}$	:	the lowest travel cost so far found in all iterations
$s_{j'}$	:	the sequence of type indices after exchange of types j and j'
$cost_{s_{j'}}$	:	the objective value of the solution with sequence $s_{j'}$
$freq_{iter}$	:	the frequency-based memory $tabu[j'][j]$ when getting $cost_{iter}$
$freq_{min}$	:	the frequency-based memory $tabu[j'][j]$ when getting $cost_{min}$

The main steps of tabu search algorithm for MLWLPA are given in Figure 1.

Note that the neighborhood NB(j) is one of $NB^h(j)$, $NB^v(j)$, and $NB^m(j)$. Section 7.1 will investigate the performance of these different neighborhoods.

In this implementation, the algorithm will stop when the transportation cost of the best solution is kept unchanged for 100 iterations.

An extension of the above algorithm is to use the frequency-memory information as diversification. The algorithm is the same as in Figure 1 except for steps 7 and 8,which are given as follows:

step 7. If $cost_{s_{j'}} + tabu[j'][j] < cost_{min} + freq_{min}$, then let $cost_{min} = cost_{s_{j'}}$, $cost_{iter} = cost_{s_{j'}}$, record $best\_pair = [j', j]$, go to step 9.

step 8. If $cost_{s_{j'}} + tabu[j'][j] < cost_{iter} + freq_{iter}$ or $cost_{iter} = 0$ and $tabu[j][j'] < iteration\_No$, record $best\_pair = [j', j]$, $cost_{iter} = cost_{s_{j'}}$.

0.5 Greedy Tabu Search

Since inventory requirements of item types are different and there is an adjacency constraint, it is difficult to update the layout directly by exchanging the types. The layout and objective value after pair-exchange needs to be re-computed and this costs a lot of computational time and effort if the neighborhood is defined as above.

To speed up the algorithm, two greedy TS approaches are proposed as follows:

Method 1: move to the first improvement

In contrast with the standard TS, which searches all neighborhood spaces and then selects the best to move, the greedy TS will take the move once the solution is better than the current best. If there is no such solution, move to the best of the NB. The greedy tabu search algorithm for the MLWLPAC is the same as in Figure 1 except for steps 7, which is given as follows:

1. Initialization.
2. Produce the initial solution by the sequence of SVTP.
3. Use the three-phase assignment approach to layout and compute cost $cost_0$.
4. Set $j = 1,\ cost_{iter} = 0,\ cost_{min} = cost_0,\ j' = j + 1$.
5. If $j' \in NB(j)$, exchange j and j' to obtain a new sequence $s_{j'}$. Otherwise go to step 9.
6. Use a three-phase assignment to compute cost $cost_{s_{j'}}$ as follows:
   - group the item types according the sequence $s_{j'}$,
   - assign the groups to cells in different levels,
   - make level re-assignment and compute the objective value: $cost_{s_{j'}}$.
7. If $cost_{s_{j'}} < cost_{min}$, then let $cost_{min} = cost_{s_{j'}},\ cost_{iter} = cost_{s_{j'}}$, record $best\_pair = [j', j]$, go to step 9.
8. If $cost_{s_{j'}} < cost_{iter}$ or $cost_{iter} = 0$ and $tabu[j][j'] < iter_{current}$, then record $best\_pair = [j', j]$, $cost_{iter} = cost_{s_{j'}}$.
9. Let $j' = j' + 1$. If $j' > j + J/\theta$, go to the next step. Otherwise go to step 5.
10. Let $j = j + 1$, $j' = j + 1$. If $j = J$, go to the next step. Otherwise go to step 5.
11. Select the $best\_pair = [j', j]$ as the accepted move, and update the tabu list
12. If the stop condition is met, stop, the best solution is the solution of the problem. Otherwise, go to step 4.

Figure 1: TS Procedure for MLWLPAC

Step 7: If $cost_{s_{j'}} < cost_{min}$, let $cost_{min} = cost_{s_{j'}}$, $best\_pair = [j', j]$, go to step 11.

Method 2: select the best after group comparison

This greedy TS compares the best pair exchanges among NB(j) for each type j and accepts the best exchange, if the solution of the exchange improves the best solution obtained in the previous search. This method is a trade-off between the standard TS and the greedy TS.

Based on the experiment, the greedy TS with method 1 is obviously better than that with method 2. In the rest of this paper, the greedy TS refers to method 1.

0.6 Dynamic NB Search

To improve solution quality, several strategies on neighborhood space are investigated. The strategies dynamically adjust the NB search space according to the solution. They are given as follows:

Alternative NB space The three kinds of neighborhood definition, $NB^h(J)$, $NB^v(j)$, and $NB^v(j)$, are alternately used during the search process. The approach is:

Let *n=the consecutive iteration number of the non-improved solution.*

The neighborhood NB is changed as follows during the process:

$$NB(j) = \begin{cases} NB^h(j), & \text{if } n/3 = \lfloor n/3 \rfloor, \\ NB^v(j), & \text{if } n/3 = \lfloor n/3 \rfloor + 1, \\ NB^m(j), & \text{if } n/3 = \lfloor n/3 \rfloor + 2. \end{cases}$$

With the above alternative NB definition, the new TS procedure is obtained by modifying steps 5 and 9 in Figure 1: the NB(j) in step 5 is changed as the above during the search process according to n, and θ in step 9 needs to keep right to NB(j).

Shifting NB space Another way to explore more space is to shift to another neighborhood space while solution stays unchanged. That is, the NB range are moved from [j+1, j+NB(j)] to [j+nβ, j+NB+nβ]. β is set equal to 2 in the implementation. Note that the alternative NB technique is also employed when adopting shifting NB search.

Extension of NB The search space is extended when the solution is trapped in a local one, that is, extend NB from [j+1, j+NB] to [j+1, j+NB+nλ]. λ is set equal to 2 in the implementation. Similarly, the alternative NB technique is also employed when realizing the extended NB search.

0.7 Results and Comparison

This experiment first uses standard tabu search to solve the problems and observes the effort of the different parameters and techniques. Then the performance of the greedy TS are evaluated by comparing it with the standard TS. Finally, the effectiveness of the TS with different dynamic NB search strategies are investigated. To make a comparison with different algorithms, the stop condition will be changed to a CPU time limit (e.g., $2 \sim 4$ minutes), which is longer than the time when the algorithm reaches the original stop condition.

0.7.1 Test Problem Design

The research generates a set of test problems which are similar to the practical paper reel layout problem given in Lai (1999). The following parameters are used in generating the MLWLPAC problems:

- The number of item types (J), their demand (Q_j), horizontal transportation unit cost (C_j^h), vertical transportation unit cost (C_{jl}^v), and inventory requirements (U_j).
- The number of levels of warehouse (L).
- The number of cells (K_l) in every level, their capacities (A), their distance (d_{lk}) to the I/O port, and their adjacency relationships.

The above parameters are set as in Zhang (2000b) except for inventory requirements (U_j) and cell adjacency relationships, which are given as follows:

- 70% of item types have their U_j less than half of the cell capacity. 20% of item types have their U_j between half and the total cell capacity. 10% of item types have their U_j between capacities of one and three cells.
- A warehouse structure is given in Lai et al. (1999). It provides the cell adjacency relationship, for example, the adjacent cells of cell 30 are cells 16, 17, 39, and 40.

For each size (J, L), five instances of MLWLPAC are randomly generated. A total of 80 problem instances are generated with type number $J = 150 \sim 300$, level $L = 2 \sim 5$. All TS algorithms are coded in C++ and run on a Pentium II/333 computer.

0.7.2 Results of Standard TS

Comparison of different neighborhood definitions

As introduced in section 4.3, three kinds of neighborhood definitions are proposed in the standard TS. The results are given in Tables 1 and 2. TSH, TSV and THM denote the TS heuristics corresponding to the neighborhoods NB^h, NB^v, and NB^m respectively. The algorithms are with the same parameters except for the different NB definition. In Table 1, the value of TSH is the average transportation cost of five instances and the others are the average relative gap (percentage) to TSH. Negative values mean that the cost is reduced. Table 2 shows the average percentage gap to the best solution. As shown in Table 1, the TSM and TSV can reduce the cost by an average of 3.62% and 1.86% from TSH. Furthermore, TSM can improve the solution quality for all sizes of problem. From Table 2, it is shown that the average relative gaps of TSM and TSV are also reduced to 3.56% from 7.57% and to 5.43% from 7.57% respectively.

Comparison of different parameter θ_m settings

The solution quality are also compared with different values of θ_m=5, 7, or 20 respectively. As shown in Tables 1 and 2, from the view of average cost reduction, the TSM with $\theta_m = 7$ is best. But, for $J = 300$, the solution quality is a little worse than TSH. The reason for this is that the NB(j) of $\theta_m = 7$ is too big for J=300 and it therefore takes a lot of computational effort to obtain a new solution.

Comparison of different initializations

Tables 1 and 2 also show the solution quality with different initial solutions. Three kinds of initial solution of TSM are investigated:

- Use SHTR as an initial solution. The results are given in column $\theta_m = 7$ of TSM.
- Use SVTR as an initial solution, called VI. The results are given in column VI of TSM.
- Randomly produce a sequence of indices as an initial solution, called RI. The results are given in column RI of TSM.

The algorithms given in the three columns are the same except for the initial solution. As shown in Table 1, the TSM can reduce the cost by an average of 3.99% from TSH. But, the cost of RI and VI increases by an average of 16.95% and 0.29% from TSH respectively. The relative gap of TSM is 3.00%, which is much smaller than 26.00% of TSM-RI and 7.84% of TSM-VI in Table 2. So, the initial solution selection obviously has an impact on solution quality, and TSM with initial sequence of SHRT promises a better solution than the other two.

Also an algorithm named NFM without frequency based memory is investigated and the results are reported in column NFM of TSM in Tables 1 and 2. The algorithm is the same as TSM with $\theta_m = 20$ except it has a frequency based memory. The average cost reduction of the algorithm with the memory is 3.62% compared with 3.48% of the algorithm NFM. Compared with NFM, TSM's average relative gap to the best solution is also reduced to 3.56% from 3.68%.

0.7.3 Comparison of Greedy TS with Standard TS

The comparison of results between greedy TS and standard TS is reported in Tables 3 and 4. The standard TS is a TSM algorithm with $\theta_m = 7$ as shown in Table 1. The random neighborhood strategy, RRW-CL , is also used for solving the problem with θ_s=$\theta_d = 7$.

Table 3 gives the average transportation cost of five instances of each size. The value of the first column is the cost of the standard TS algorithm. The other columns are the relative cost to TSM (percentage). Negative values mean the cost is reduced. From Table 3, it can be seen that the average cost reduction of the greedy TS is 6.19% with the maximal cost reduction of 15.76% from TS. During all 16 size problems, 13 sizes are improved by the greedy TS. From Table 4, it can be seen that the average relative gap to the best solution is reduced to 3.15% from 10.37% with the maximal to 9.18% from 19.95%. The tables also show that the greedy TS is better than the RRW-CL method in both the objective value and relative gap to the best solution for the test problems, though the RRW-CL method also improves the standard TS method.

0.7.4 Performance of TS with Dynamic NB Search

Three kinds of dynamic NB strategies are also tested and the results are reported in Tables 3 and 4. The algorithms are the same as the greedy method except for the dynamic neighborhood definition. Tables 3 and 4 show that the alternative NB method can improve the solution performance. Compared with the greedy method, the transportation cost is reduced to 7.36% from 6.17%. The relative gap is further reduced to 1.85% from 3.15% with the maximal to 4.88% from 9.18%.

The solution performance of the shifting NB and the extended NB strategy are also reported in Tables 3 and 4. It can be seen that the solution quality of all kinds of problems are improved with an average of 7.61% and 7.66% respectively. The average relative gaps to the best are reduced to 1.59% and 1.55% respectively. Simultaneously, the greedy algorithm with extended NB strategy can contribute 34 best solutions among 80 instances, and the greedy algorithm with shifting NB strategy can contribute 25 best solutions among 80 instances. Meanwhile standard TS only gets five best solutions.

Table 1: Comparison of Different Settings: Objective Value

method		TSH	TSV	TSM					
J	L	$\theta_h = 20$	$\theta_v = 20$	$\theta_m = 20$	$\theta_m = 5$	$\theta_m = 7$	RI	VI	NFM
150	2	4637595.6	-4.85	-4.23	-8.09	-9.10	13.33	-3.29	-6.07
	3	4228773.8	-3.08	-3.20	-10.55	-9.94	14.10	1.31	-1.70
	4	3986857.8	-4.12	-7.91	-11.81	-9.92	8.40	-0.74	-5.86
	5	3741975.2	-3.18	-3.55	-9.33	-11.59	12.10	-0.14	-2.54
200	2	6738947.6	-2.16	-3.57	-5.07	-0.96	19.35	0.67	-3.94
	3	6611334.4	-0.45	-3.22	-5.70	-6.30	14.12	-0.89	-1.38
	4	6764537.4	-2.11	-1.36	-6.70	-6.69	19.63	3.74	-3.36
	5	6905415.4	-7.68	-9.24	-12.26	-11.22	5.79	-2.71	-9.45
250	2	13004296.8	0.71	-0.38	0.81	-0.11	20.81	2.23	-1.79
	3	10644705.8	0.64	-2.58	0.39	-0.94	19.16	1.11	-2.45
	4	10670526.8	0.27	-2.73	-0.11	0.93	21.61	3.68	-3.38
	5	10686271.2	-2.38	-5.80	-4.62	-5.16	22.99	-0.23	-6.39
300	2	17653006.6	0.41	-1.79	4.69	2.48	24.53	1.37	-1.29
	3	16157384.8	1.61	-3.41	6.21	2.48	22.16	1.44	-2.17
	4	16861241.6	-1.84	-3.04	2.82	0.73	7.20	-1.85	-2.25
	5	15148201.0	-1.54	-1.93	5.13	1.52	25.84	-1.04	-1.60
average			-1.86	-3.62	-3.39	-3.99	16.95	0.29	-3.48

Table 2: Comparison of Different Settings: Relative Gap to the Best

method		TSH	TSV	TSM					
J	L	$\theta_h = 20$	$\theta_v = 20$	$\theta_m = 20$	$\theta_m = 5$	$\theta_m = 7$	RI	VI	NFM
150	2	10.70	5.31	5.89	1.57	0.22	26.24	7.07	4.11
	3	11.69	8.48	8.37	0.00	0.70	28.12	13.45	9.82
	4	13.38	8.61	4.37	0.00	2.16	23.87	12.62	6.50
	5	13.63	9.96	9.63	2.87	0.30	27.26	13.46	10.75
200	2	6.62	4.28	2.51	0.90	5.65	27.44	7.30	2.18
	3	7.14	6.60	3.53	0.70	0.03	22.28	5.95	5.51
	4	8.93	5.90	7.27	0.81	1.04	29.17	12.26	4.38
	5	15.98	7.02	5.04	1.69	2.87	25.35	12.73	5.04
250	2	1.92	2.75	1.59	2.89	1.91	23.45	4.27	0.13
	3	3.67	4.23	0.90	3.95	2.64	23.34	4.79	1.00
	4	4.07	4.40	1.20	3.95	5.08	26.52	7.92	0.56
	5	8.38	5.64	1.97	3.62	2.78	33.34	8.31	1.39
300	2	3.16	3.57	1.31	8.08	5.82	28.57	4.79	1.81
	3	4.13	5.77	0.51	10.50	6.57	28.63	5.42	1.83
	4	4.24	2.48	1.37	7.45	5.31	12.06	2.69	2.08
	5	3.43	1.83	1.45	8.65	4.92	30.35	2.34	1.81
Min_Gap		1.92	1.83	0.51	0.00	0.03	12.06	2.34	0.13
Max_Gap		15.98	9.96	9.63	10.50	6.57	33.34	13.46	10.75
Ave_Gap		7.57	5.43	3.56	3.60	3.00	26.00	7.84	3.68
NO_best		2	3	15	21	19	1	6	14

Figures 2 and 3 show the relative gap of the algorithms at $L = 2$ and $L = 5$. It seems the relative gaps of the standard method TSM and the RRW-CL method have increased when the scale of the problem increases. However, the relative gaps of the greedy TS method and the dynamic TS approaches are reduced when the scale of problem increases.

Finally, as a comparison, we extend COI rule to solve the test problems by combineing the three-phase assignment approach. Compared with the results of extended COI procedure, the greedy TS with extending NB reduces the transportation cost from 32.65% to 40.61% with an average of 37.25%, while the computation time ($2 \sim 4$ minutes) of TS is acceptable.

Table 3: Comparison of TS Algorithms: Objective Value

J	L	Standard TS	RRW-CL	Greedy TS	Alter. NB	Shift NB	Extend NB
150	2	4215391.2	1.62	0.84	-0.01	-1.15	-1.95
	3	3808307.4	2.28	1.11	0.37	-0.08	-0.57
	4	3591167.4	-4.52	-6.59	-7.18	-8.25	-8.18
	5	3308187.0	4.94	1.83	-2.07	-2.27	-2.82
200	2	6673961.0	-5.10	-7.23	-8.23	-8.24	-9.26
	3	6194776.2	0.69	-1.21	-4.43	-4.78	-2.48
	4	6312308.4	2.58	-2.43	-4.38	-5.28	-4.36
	5	6130298.6	0.93	-2.12	-6.04	-6.15	-5.24
250	2	12990303.8	-2.32	-6.13	-6.35	-6.30	-5.10
	3	10544358.0	-1.22	-8.06	-8.25	-8.15	-8.39
	4	10769726.6	-5.44	-11.58	-11.92	-11.83	-11.87
	5	10134374.6	-2.99	-11.23	-12.46	-12.60	-13.39
300	2	18090593.0	-3.55	-10.62	-10.65	-10.61	-11.42
	3	16557647.2	-9.18	-9.07	-9.25	-9.05	-9.68
	4	16984105.4	-3.91	-10.47	-11.00	-11.18	-11.45
	5	15377777.2	-6.96	-15.76	-15.90	-15.85	-16.47
average			-2.01	-6.17	-7.36	-7.61	-7.66

Table 4: Comparison of TS Algorithms: Relative Gap to the Best

J	L	Standard TS	RRW-CL	Greedy TS	Alter. NB	Shift NB	Extend NB
150	2	4.82	6.91	5.87	4.88	3.59	3.06
	3	1.86	4.17	2.87	2.13	1.60	1.05
	4	10.99	5.56	3.37	2.72	1.57	1.40
	5	7.18	12.35	9.18	4.79	4.49	4.02
200	2	10.74	4.77	2.29	1.18	1.17	0.00
	3	5.56	6.67	4.27	0.97	0.59	2.94
	4	7.29	9.34	3.71	1.72	0.94	2.40
	5	7.58	8.51	5.34	1.15	1.07	2.15
250	2	7.08	4.57	0.46	0.24	0.27	1.58
	3	9.59	8.16	0.73	0.52	0.63	0.34
	4	14.55	8.26	1.16	0.77	0.89	0.80
	5	16.48	13.18	3.15	1.63	1.48	0.51
300	2	13.54	8.97	1.02	0.98	1.03	0.11
	3	15.10	4.53	4.24	4.05	4.28	3.53
	4	13.58	9.10	1.64	1.06	0.87	0.71
	5	19.95	11.65	1.06	0.88	0.95	0.19
Min_Gap		1.86	4.17	0.46	0.24	0.27	0.00
Max_Gap		19.95	13.18	9.18	4.88	4.49	4.02
Ave_Gap		10.37	7.92	3.15	1.85	1.59	1.55
NO_best		5	6	2	17	25	34

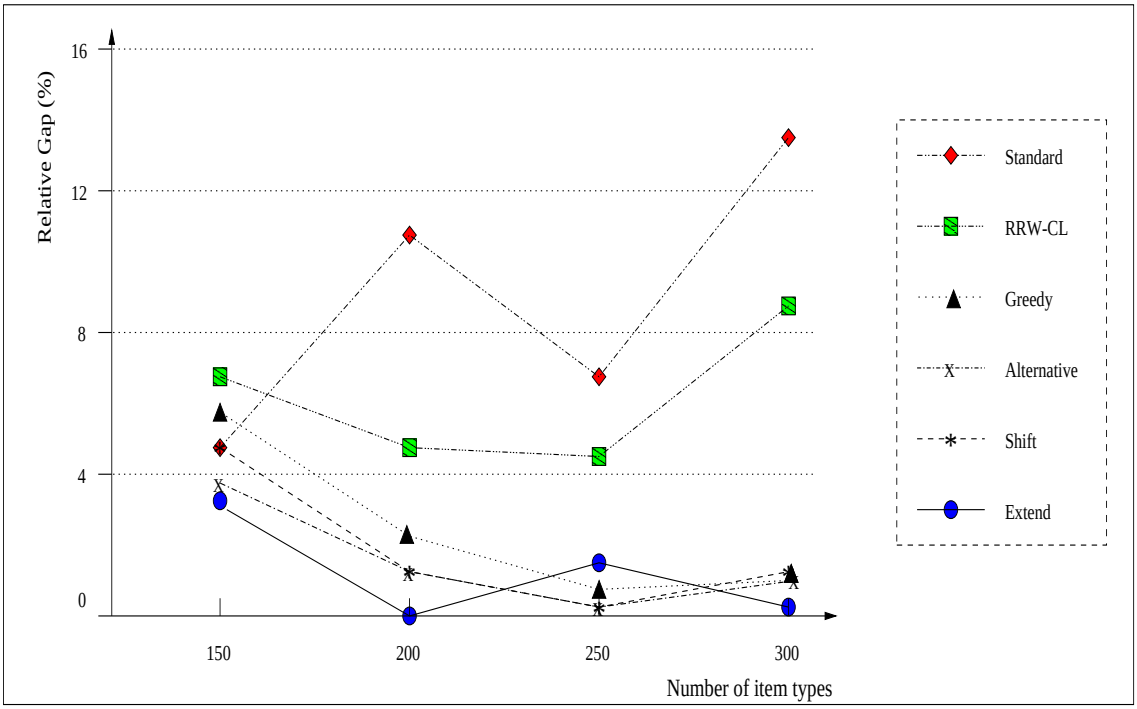

Figure 2: Comparison of Relative Gap of TS Algorithms (L=2)

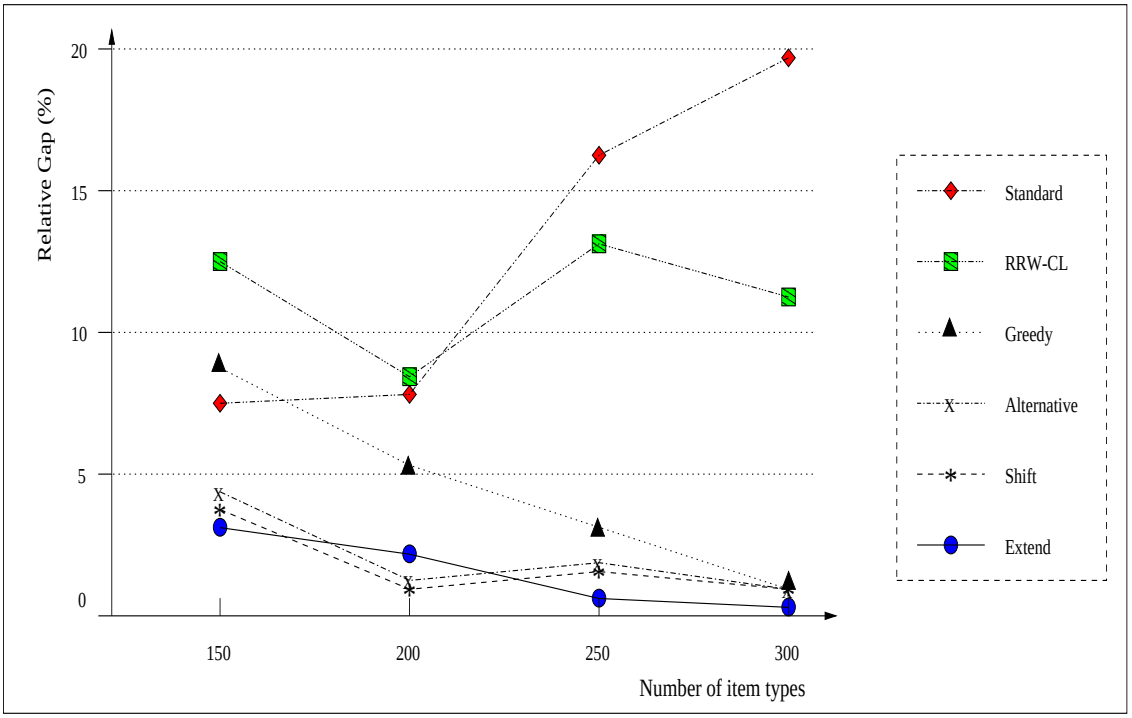

Figure 3: Comparison of Relative Gap of TS Algorithms (L=5)

0.8 Conclusions

In this paper, a new kind of warehouse layout problem with multiple-level areas and adjacency constraints, MLWLPAC, is investigated. A TS approach is developed based on three-phase assignment method to provide effective layout with low transportation cost. Some improved techniques for the approach are developed. The experiment shows that the greedy TS and dynamic neighborhood techniques can improve solution quality significantly.

Bibliography

[1] Bozer, Y. A., Meller, R. D., and Erlebacher, S. J., An improvement-type layout algorithm for single and multiple-floor facilities., *Management Science*, 40(1994), 918-932

[2] Chiang, W. C., and Chi, C., (1998), Intelligent local search strategies for solving facility layout problems with the quadratic assignment problem formulation, *European Journal of operational Research*, 106, 457-488.

[3] Cormier,G. and Gunn,E.A., A review of warehouse models. European Journal of Operational Research 58 (1992), 3-13.

[4] Francis, R. L., McGinnis, Jr. L. F. and While, J. A., *Facility layout and location: an analytical approach*, Prentice-Hall, 1992

[5] Glover, F., (1995) Tabu search fundaments and uses, Working paper, Graduate School of Business, University of Colorado, Boulder, Colorado.

[6] Glover F., and Laguna M., (1997), Tabu search, Boston : Kluwer Academic Publishers.

[7] Harmatuck, D. J., A comparison of two approaches to stock location, *The Logistics and Transportation Review*, 12: 282-284, 1976

[8] Heskett, J. L., Cube-per-order — a key to warehouse stock location, *Transportation and Distribution Management*, 3: 27-31, 1963

[9] James, R. J. W., and Buchanan, J. T., (1998), Performance enhancements to tabu search for the early/tardy scheduling problem, *European Journal of operational Research*, 106(2-3), 254-265.

[10] Johnson, R. V., 1982, SPACECRAFT for multi-floor layout planning, *Management Sciences*, **28**, 407-417.

[11] Kaku, B. K., Thompson, G. L., and Baybars, I., A heuristic method for the multi-story layout problem, *European Journal of operational Research*, 37(1988), 384-397

[12] Lai, K. K., Xue, J. and Zhang G. Q., A Two-Stage Heuristic for a Paper Reel Layout Problem –A Revised Simulated Annealing Approach, to appear in Int. J. Prod. Econ., 1999.

[13] Malmborg C. J. and Krishnakumar B., 1989, Optimal storage assignment policies for Multi-address warehousing systems, *IEEE Trans. on Syst. Man. and Cybernetics*, **19**, 197-204.

[14] Meller., D., and Bozer, Y.A., 1996, A new simulated annealing algorithm for the facility layout problem, *Int. J. Prod. Res.*, **34**(6), 1675-1692.

[15] Wilson H. G., 1977, Order quantity, product popularity, and the location of stock in a warehouse, *AIIE Transactions*, **9**, 230-237.

[16] Zhang G. Q., Xue, J., and Lai, K. K., 2000a, An improved GA for layout problem with adjacent constraints, *International Journal of Production Research*, 38(14) 3343-3356.

[17] Zhang G. Q., Xue, J., Lai, K. K. and Leung J., 2000b, A class of genetic algorithms for multiple-level warehouse layout problem, to appear.

[18] Zhang, G. Q., A study of warehouse layout problems: meta-heuristics approaches, Ph. D thesis, City University of Hong Kong, Hong Kong, 2000.