Journal of Global Optimization **8:** 35–48, 1996.

Tabu Search for the Planar Three-Index Assignment Problem*

D. MAGOS*
Department of Informatics, Athens School of Economics and Business, 76 Patission Str., Athens 104 34, Greece. Tel: 01-8225268, E-mail: nnk@aueb.ariadne-t.gr

(Received: 8 April 1993; accepted: 23 June 1995)

Abstract. Tabu Search is a very effective method for approximately solving hard combinatorial problems. In the current work we implement Tabu Search for solving the Planar Three-Index Assignment Problem. The problem deals with finding the minimum cost assignment between elements of three distinct sets demanding that every pair of elements, each representing a different set, appears in the same solution exactly once. The solutions of the problems correspond to Latin squares. These structures form the basis of the move generation mechanism employed by the Tabu Search procedures. Standard Tabu Search ideas such as candidate move list, variable tabu list size, and frequency-based memory are tested. Computational results for a range of problems of varying dimensions are presented.

Key words: Combinatorial optimization, three-index assignment, tabu search, Latin square.

1. Introduction

The planar three-index assignment problem (PP3) can be thought of as a variation of the min-sum assignment problem with three sets of elements instead of two: given three disjoint n-sets $I = \{1, ..., n\}, J = \{1, ..., n\}, K = \{1, ..., n\}$ consider the set $A = \{(i, j, k) | (i, j, k) \in I \times J \times K\}$ and a cost c_{ijk} attached to each element of A. An *assignment* is defined as a set A_f, $A_f \subseteq A$, with the property that each pair of indices, each representing a different set, appears in exactly one triplet of A_f. **PP3** deals with finding the assignment of minimal total cost. **PP3** can be stated as a 0–1 programming problem as

$$\min \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} c_{ijk} x_{ijk}$$

$$s.t.$$

$$\sum_{i \in I} x_{ijk} = 1 \forall j \in J, k \in K \qquad (1)$$

* Preferred corresponding address: 30 Theodorou Geometrou Str., Neos Kosmos, Athens 117 43, Greece

$$\sum_{j \in J} x_{ijk} = 1 \forall i \in I, k \in K \tag{2}$$

$$\sum_{k \in K} x_{ijk} = 1, \forall i \in I, j \in J \tag{3}$$

$$x_{ijk} \in \{0,1\}, \forall (i,j,k) \in I \times J \times K$$

The closest relative of **PP3** is the axial three-index assignment problem (**AP3**) (Balas and Saltzman, 1991). Both problems minimize over the same function but in the case of **AP3** the assignment is defined with respect to the property that each element of the sets I, J, K appears in exactly one triplet of A_f.

PP3 is closely related to the *solid* (*multi-index*) transportation problem (Haley, 1962) in which the right-hand sides of the constraints are positive constants, the sets I, J, K are not equal in size and the integrality constraints on the variables are relaxed. There are additional constraints, imposed on the right-hand side constants, that correspond to the constraint that equates total demand and total supply for the transportation problem.

It is obvious that **PP3** is a special case of the *set partitioning* problem (**SPP**): $\{\min \mathbf{cx} | \mathbf{Ax} = \mathbf{e}, \mathbf{x}$ binary$\}$, where $\mathbf{A}$ is a matrix of zeros and ones and $\mathbf{e}$ is a vector of ones.

PP3 has several applications with respect to scheduling and timetabling problems. Consider n teachers, n classes and n time periods. A *schedule* is a set of triplets (i,j,k) that assigns in each time period exactly one teacher to one class and vice versa, so that after n periods every teacher has taught in every class. Let c_{ijk} be a *satisfaction value* associated with the assignment of teacher i to class j during time period k. We seek the schedule that maximizes/minimizes the total satisfaction. A special case where c_{ijk} equals 0 or 1 is proven *NP*-complete in (Frieze, 1983).

Another application concerns the schedule of meetings between buyers and sellers during conventions. A tourism industry convention scheduling problem of such kind is presented in Gilbert and Hofstra (1987).

Only a limited amount of literature refers to **PP3**. Exact algorithms for the problem have been developed by M. Vlach (1967) and recently by D. Magos and P. Miliotis (1994). Theoretical work includes the paper of Burkard and Frolich on admissible transformations (1980), whereas some classes of facet-defining inequalities for the problem are presented in Euler *et al.* (1986).

In the current work we present algorithms for **PP3** based on Tabu Search technique (**TS**). The method was first introduced by Glover as an intelligent search technique to overcome local optimality (1989, 1990). **TS** has been successfully employed in a large collection of applications and has been proven very effective for handling difficult problems (Glover and Laguna, 1993). It has also been successfully combined with other search techniques in the framework of hybrid algorithms (Areibi and Vannelli, 1994, Laguna and Glover, 1993b, Osman, 1993).

To our knowledge this is the first time that **TS** is applied to **PP3**. The basic ideas of **TS** are sketched in Section 2. In Section 3 we examine the structure of feasible solutions of **PP3** and discuss moves that transition between solutions. The specifics of the procedures implemented are discussed in Section 4. Computational results are illustrated and discussed in Section 5.

2. The Tabu Search Technique

Tabu Search is an iterative method used for finding, in a set X of feasible solutions, the solution that minimizes an objective function f. Before describing the procedure we introduce the necessary definitions.

A move s consists of a mapping defined on a subset $X(s)$ of $X : s : X(s) \rightarrow X$ (Glover, 1989). In practise s can be considered as a sequence of operations that, if applied, transition from one solution to another. We implicitly refer to the neighbor solution produced by applying move s to a solution x as $s(x)$. We denote $S(x)$ the set of moves that can be applied to solution x. The move that cancels s is noted $s^{-1}(s^{-1}(s(x)) = x)$. The value of a move, noted $v(s)$, is defined as $f(s(x)) - f(x)$.

TS starts with an initial solution. At each iteration a set of moves is considered. The move with the smallest value is applied to the current solution. The new solution is adopted and the algorithm proceeds to the next iteration. In order to avoid being trapped at a local optimum only a subset of $S(x)$ is considered at each iteration. More precisely, a so-called *tabu* list T of length $\mid T \mid = m$ (fixed or variable) is used to store moves that are forbidden (i.e. tabu). T stores the s^{-1} of the last m moves applied. The search is confined to selecting a move from the set $S(x) - T$, preserving thus the effects of the m most recent moves applied. T is updated circularly. When a move s is selected s^{-1} enters the list, while the oldest entry is removed.

The tabu mechanism may sometimes be very restrictive. This may have undesirable effects, such as missing local optima or confining the search to solutions of great structural similarity. Situations like these are dealt with through the use of *aspiration* criteria. When satisfied, an aspiration criterion cancels the tabu status of a move, thus making the move available for selection. Evidently, such a criterion must be used with caution so as to minimize the possibility of cycling. Several aspiration criteria have been proposed in the literature (Glover and Laguna, 1993). We consider the following two as part of any tabu search procedure. The first selects the "least tabu" move if all candidate moves are tabu at an iteration. The term "least tabu" usually refers to the move that is scheduled to leave the tabu list first. The second criterion selects a tabu move s if $f(s(x)) < f(x^{best})$, where x^{best} denotes the incumbent (i.e. the best solution found so far).

The procedure in mock Pascal follows. Text in {} is commentary.

TABLE I. A Feasible Solution

	Latin square				Variables set-to-one			
i/j	1	2	3	4				
1	2	3	4	1	x_{112}	x_{123}	x_{134}	x_{141}
2	4	2	1	3	x_{214}	x_{222}	x_{231}	x_{243}
3	3	1	2	4	x_{313}	x_{321}	x_{332}	x_{344}
4	1	4	3	2	x_{411}	x_{424}	x_{433}	x_{442}

2.1. TABU SEARCH

find an initial solution x;

$x^{best} = x$;

while stopping criterion not satisfied *do*
begin
solve (min $v(s)$, s.t :$s \in S(x) - T$ or $f(s(x)) < f(x^{best})$);
{let s^*} denote the solution to the previous problem}
update T;
if $f(s^*(x)) < f(x^{best})$ *then* $x^{best} = s^*(x)$;
$x = s^*(x)$;
end

The stopping criterion is usually implemented as an elapsed number of iterations either in total or since x^{best} was last improved.

The tabu list constitutes a type of short-term memory which stores information on the m most recent moves. However, this does not always achieve a thorough exploration of the solutions' space. In many **TS** implementations short-term memory is complemented with intermediate- and/or long-term memory. The longer-term memory structures accumulate information on solutions visited over a long period of iterations. The search is conducted in stages. Initially, only short-term memory is used. Longer-term memory structures collect information that is utilized in the next stage, where the search is diversified to regions where improved solutions can be found. Additionally, longer-term memory can be used to intensify the search in regions where high quality local optima lie. After the termination of the intensification and/or diversification phase the search continues making use of the short-term memory only. The procedure is repeated until a stopping criterion is satisfied. Practical experience shows that long-term memory plays an important role in obtaining best solutions for hard problems.

3. The Structure of a Feasible Solution

Every solution of **PP3** corresponds to a Latin square and vice versa (Euler *et al.*, 1986). The square consists of n^2 cells, each with a pair of indices as co-ordinates

TABLE II. Known values of l_n

n	2	3	4	5	6	7
l_n	1	1	4	56	9408	16,942,080

TABLE III.

	The new Latin Square				Variables	
i/j	1	2	3	4	One	Zero
1	2	3	4	1	x_{221}	x_{222}
2	4	1(2)	2(1)	3	x_{322}	x_{321}
3	3	2(1)	1(2)	4	x_{331}	x_{332}
4	1	4	3	2	x_{232}	x_{231}

(row,column) and the index of the third set as content. Without loss of generality we consider i as the row index, j as the column index, and k as the content index. A feasible solution to an order-4 problem, both in variable-set-to-one and Latin-square format, is presented in Table I.

This one-to-one correspondence between Latin squares and feasible solutions allows us to calculate the number of feasible solutions. The number of Latin squares of order-n is $l_n \cdot n! \cdot (n-1)!$. The known l_n values are illustrated in Table II (Ryser, 1963).

The Latin-square structure of the solutions facilitates the production of the neighbor solutions. Moves take as input a Latin square (current solution) and generate other Latin squares (neighbor solutions). More specifically, a move consists of changing the content of a cell and then restoring the Latin-square structure by imposing further changes on the contents of other cells. For the Latin square of Table I, namely $L1$, consider changing the content of cell (2,2), noted $k(2,2)$, from 2 to 1. Clearly the solution is now infeasible. Feasibility can be restored through a unique *chain* of changes in the contents of certain cells. We search rowwise to change a 1 to 2 and columnwise to change a 2 to 1. The move is completed when we reach a cell in the same row with the starting cell. The cells affected by the move are ((2,2), (3,2), (3,3), (2,3)). In Table III the new Latin square ($L2$) is illustrated. The old cell contents are included in brackets. The corresponding variables that are set to one and zero by the move are also illustrated.

The value of a move $v(s)$ is calculated with respect to the c_{ijk} co-efficients of the variables affected by the move. The rest of the variables maintain the same value in both solutions, therefore they do not contribute to $v(s)$. The value of the move described is

$$v(s) = c_{222} + c_{321} + c_{332} + c_{231} - c_{221} - c_{322} - c_{331} - c_{232}.$$

The collection of cells affected uniquely identifies the move. The cell whose k index is changed first is of no importance as long as the cell belongs to the collection in question. Note that the same move is generated regardless of whether

TABLE IV. The Three Configurations of a Solution

i/j	1	2	3	4
1	2	3	4	1
2	4	2	1	3
3	3	1	2	4
4	1	4	3	2

j/k	1	2	3	4
1	4	1	3	2
2	3	2	1	4
3	2	3	4	1
4	1	4	2	3

k/i	1	2	3	4
1	4	3	2	1
2	1	2	3	4
3	2	4	1	3
4	3	1	4	2

the move starts from cell (2,2), or (3,2), or (3,3), or (2,3). Therefore the move is completely defined by the pair of indices swapped and one of the cells it affects. For example, the move that changes 1 to 2 and vice versa starting from cell (4,1) is distinct from the move we described because it affects a different configuration of cells regardless of the fact that the same pair of k indices are swapped. The cardinality of the set $S(x)$ ($|\ S(x)|$) of all distinct moves that can be applied to x can be approximated by an upper and a lower bound. For each cell there are $n-1$ possible substitutions between its current content and the elements of set $K \setminus k(i,j)$. The total number of substitutions is $n^2(n-1)$. The number of cells affected by a move is between 4 and $2n$. Therefore

$$\frac{n^2(n-1)}{4} \geq |\ S(x)\ | \geq \frac{n(n-1)}{2} \tag{4}$$

It is not always necessary to re-evaluate all moves when transitioning to a new solution. With respect to $L1$ consider the move that swaps 3 and 4 by setting $k(1,2) = 4$. Feasibility is restored by setting $k(4,2) = 3$, $k(3,4) = 4$, $k(1,3) = 3$. The same set of variables are led to zero and one regardless of $L1$ or $L2$. Therefore the value of the move is valid for both solutions. It is easy to see that only moves that swap k and 1, and k and 2 (for every $k \in K \setminus \{1,2\}$) need be re-evaluated for $L2$. All other moves are valid for both solutions. The only exception is the move that leads from $L1$ to $L2$, which must be replaced by its reverse move. Note that other moves that swap 1 and 2 are valid for both solutions since they refer to different variables than the ones affected by the application of the move. To take advantage of the fact that only a limited number of moves need re-evaluating at each iteration a list of candidate moves is maintained throughout the search. For the starting solution all moves are evaluated and enter the list. At each iteration, after a move is selected, the list is updated to provide the new candidate moves for the new solution.

We have expressed every solution of **PP3** as a Latin square whose cells have as row and column co-ordinates the elements of the sets I and J, and as cell contents the elements of set K. We denote this configuration $K(I, J)$. By circularly exchanging the sets two more configurations are derived. According to the notation used these are $I(J, K)$ and $J(K, I)$. Therefore a solution of **PP3** corresponds to three Latin squares, each belonging to a different configuration. Table IV illustrates these three Latin squares for the solution of Table I.

The three configurations constitute three different "views" of the same solution. Moves can now be evaluated for all three Latin squares, thus enlarging the set $S(x)$.

Most importantly, the search becomes more flexible and sensitive because moves that swap pairs of indices of all three sets are considered at every iteration. Therefore the search can arrive at a solution by applying a move that swaps i_1 and i_2 (where $i_1, i_2 \in I$) and then continue with a move that swaps elements of another set. The search is now able to reveal solutions that could not be reached otherwise given the starting solution. Additionally, certain solutions considered remote when a single type of Latin square is used can now be found in few iterations. The performance of our algorithm increased dramatically when moves for the three Latin-square configurations were considered. Unfortunately the computational burden increased as well. However, because only few moves need re-evaluating at each iteration, and because the evaluation of a move is done much faster in a manner to be described next, the increase in computing time was small.

Most of the time the move evaluation routine searches along rows and columns of the Latin square for the cells that contain the indices to be swapped. This can be avoided by exploiting the information supplied by the three alternative configurations of a solution. We define $i(j, k)$ the content of the cell (j, k) belonging to the configuration $I(J, K)$. $j(k, i)$ is defined accordingly for configuration $J(K, I)$. We consider again the move that leads from $L1$ to $L2$. The move consists of swapping 1 and 2 and initially sets $k(2,2) = 1$. According to the procedure described above, we search along the same column ($j = 2$) for a cell with content index $k = 1$ to be changed to $k = 2$. The index that gives the row co-ordinate of the cell can be found from the configuration $I(J, K)$. It is $i(j = 2, k = 1) = 3$. We set $k(3,2) = 2$ and search along the same row ($i = 3$) for a 2 to be changed to 1. The j index in question can be found in the cell $(k = 2, i = 3)$ of $J(K, I)$. This is $j(2, 3) = 3$. We set $k(3, 3) = 1$ and continue locating i and j indices from configurations $I(J, K)$ and $J(K, I)$ respectively. The move is completed when a cell with the same i index as the starting cell is reached. The same procedure is used to evaluate moves of any configuration.

When moves for a single Latin-square configuration are evaluated, we have shown that only a part of $S(x)$ needs re-evaluating at each iteration. The same goes for the case of the three configurations. Let $x1$, $x2$ denote two neighbor solutions, and s the move that transitions from $x1$ to $x2$ ($s(x1) = x2$). Then some of the moves applicable for $x1$ are valid for $x2$ as well. We denote $K(s)$ the set of k indices of the variables affected by the application of s. The sets $I(s)$ and $J(s)$ are defined accordingly. Moves that swap k_1 and k_2 (for every $k_1, k_2 \in K \setminus K(s)$), need not be re-evaluated for $x2$. Similarly we need not re-evaluate moves that swap i_1 and i_2 (for every $i_1, i_2 \in I \setminus I(s)$), and j_1 and j_2 (for every $j_1, j_2 \in J \setminus J(s)$). The rule embedded in our algorithm re-evaluates all other moves. This is done with the knowledge that there are further moves that need not be evaluated at $x2$. For example, if s swaps a and b ($a, b \in Z$, where Z denotes any of I, J, K) all other moves that swap the same elements remain valid for $x2$. However, because in practise the moves that are unnecessarily evaluated are few, we have chosen not to adopt a more elaborate approach.

4. Tabu Search for PP3

The implementation of the tabu notion is of primary importance to every **TS** procedure. We have experimented with three alternative definitions of the tabu move. All these definitions are given with respect to $x^0(s)$, which denotes the set of variables led to zero by the application of move s to the solution x.

(a) A move is tabu if it sets one or more variables of $x^0(s)$ to one.

(b) Let $x^1(s)$ denote the set of variables of x^0(s) that have the value of one at the current iteration. These variables are set to one by a move/moves applied after move s. A move is tabu if it sets all variables of the set $x^0(s) - x^1(s)$ to one. Note that if at least one variable of $x^0(s)$ has the value of zero we cannot return to solution x.

(c) A move is tabu if it sets all variables of $x^0(s)$ to one at the same iteration. This definition refers to the move that reverses s directly (at one iteration). This is a special case of rule (b) with $x^1(s)$ empty.

Rule (c) is a rather weak implementation of the tabu notion. It can be easily shown that it cannot prevent cycling. Let $x1$ denote the solution at iteration t. The move s^t is selected and yields the solution $x2$ ($s^t(x1) = x2$) at iteration $t + 1$. The move selected at iteration $t + 1$ cannot lead back to $x1$. Let $x3$ denote the solution at iteration $t + 2$ ($s^{t+1}(x2) = x3$, $x3 \neq x1$). If none of the variables of $x1^0(s^t)$ is set to one at $x3$ then s^{t+2} cannot yield $x1$. However, if at least one of the variables of $x1^0(s^t)$ is present at $x3$ then the rule does not prevent s^{t+2} from setting the rest of the variables of $x1^0(s^t)$ to one as well. Therefore it is possible that $x1 \equiv x4$ ($s^{t+2}(x3) = x4$). Even if $x1 \neq x4$, revisiting of $x1$ may occur at a future iteration, especially if $x1$ is a "deep" local optimum. A situation like this can never occur if (a) or (b) are used. Rule (b) ensures that not all variables of $x^0(s)$ will take the value of one during the next m iterations since the application of s. Rule (a) is more restrictive, not allowing any variable of $x^0(s)$ to take the value of one at the future m iterations.

Rule (b) proved superior to the rest in terms of the quality of the solutions produced. Rule (a) renders many moves tabu at each iteration, confining the search to a poor exploration of the solutions' space. On the other hand, rule (c) in a great number of cases proved incapable of directing the search away from a local optimum.

A very important parameter of the **TS** procedure is the size of the tabu list (m). We have repeatedly solved the same set of problems using each time a different value for m. The results showed that the procedure was highly sensitive to different m values. To make the procedure more robust we adopted a strategy that modifies the tabu list size throughout the course of the search. The value of m is systematically varied every $2n$ iterations since the last change or $2.5n$ iterations since x^{best} was last updated, following the sequence $\{1.15n, n, 0.85n\}$. The sequence is repeated as many times as necessary until the end of the search.

Tabu lists of variable size have been efficiently employed in the framework of several **TS** applications (Hertz and de Werra, 1990; Tailard, 1990).

In addition to the short-term memory our algorithm incorporates a frequency-based memory used to diversify the search to regions where improved solutions can be found. Frequency-based memory functions have been employed in a variety of **TS** applications. Laguna and Glover, (1993a) and Chakrapani and Skorin-Kapov (1993) are two such applications where the frequency-based memory has been adopted in rather different settings for the purpose of diversifying the search.

Our algorithm consists of three phases which are iteratively executed. In the first phase the moves are selected with respect to the short-term memory only. Then an intensification loop is entered (second phase) during which the first phase is executed again using x^{best} as the initial solution. The loop is exited if no improvement of x^{best} is achieved. During the first two phases for every variable x_{ijk} a counter m_{ijk} is maintained, indicating the number of solutions this variable occured to as a percentage of the total number of solutions visited by the search. In the third phase, if no admissible (i.e. non-tabu) improving moves exist, moves that introduce variables of low m_{ijk} value (frequency) are favored. More precisely, for every admissible non-improving move the average frequency of the variables it sets to one is calculated. Only ten percent of the moves, the ones with the smaller average frequencies, are considered. In this way a move that introduces a high frequency variable into the solution is not excluded by default. However, in order to be considered for selection the move must also introduce variables of low frequency so as to achieve a low average frequency. The move that degrades the objective function the least is selected. The procedure is repeated until a stopping rule is satisfied. The number of iterations spent at each phase of the search is given to the algorithm as input. Alternatively, the number of iterations can be considered as a function of the problem's size.

5. Computational Experience

Our algorithm was programmed in C. Computational results demonstrated in this chapter were obtained from an "IBM 386-sx" class computer running at 16 MHz with no mathematic co-processor. The code was compiled using *Borland Turbo C v2.00* compiler and run under *Microsoft DOS v5.00*. Unfortunately there are no test problems available from the literature. We have randomly generated two sets of problems with costs integers drawn from a uniform distribution in the interval [1,100]. Results illustrated were obtained with a cut-off limit of 1500 iterations for every **TS** implementation.

The first problem set consists of twenty-five problems, five problems for each value of n, with n = 5,6,...9. The problems were solved optimally (column **OPT**, Table V) by the algorithm described in Magos and Miliotis (1994). Two versions of our algorithm were tested, namely **TS1** and **TS2**. **TS1** is based on short-term memory only. **TS2** embeds the intensification and diversification strategies described

TABLE V. Computational Experience (I)

N	OPT	LS		TS1		TS2		TS1	TS2
		% GAP	SECS	% GAP	SECS	% GAP	SECS	% GAP	% GAP
5									
	633	27.96	2	0.00	52	0.00	70	2.52	2.52
	811	7.76	2	3.94	49	3.94	69	0.00	0.00
	863	13.78	1	3.24	48	3.24	71	0.00	0.00
	663	21.56	2	6.18	47	6.18	70	0.00	0.00
	800	8.37	1	0.12	81	0.12	71	0.00	0.00
6									
	782	19.95	2	0.00	80	0.00	112		
	934	14.56	3	0.00	83	0.00	115		
	1058	11.72	2	0.00	84	0.00	113		
	898	9.24	1	0.00	82	0.00	119		
	955	7.64	1	0.00	85	0.00	120		
7									
	1319	29.11	1	14.03	116	13.87	173	0.00	0.00
	1268	24.36	1	21.37	118	20.90	175	0.00	0.00
	1195	30.62	2	19.16	115	19.16	149	0.58	0.00
	1121	51.38	3	25.25	117	20.61	146	0.00	0.00
	1252	39.85	2	20.93	115	15.58	176	0.00	0.00
8									
	1528	21.40	2	1.57	172	0.13	216		
	1467	10.16	3	0.00	171	0.00	215		
	1486	13.39	2	0.00	175	0.00	217		
	1474	22.25	2	0.00	160	0.00	273		
	1484	12.13	2	1.68	173	1.68	215		
9									
	1685	27.18	2	1.78	234	1.36	298		
	1882	22.64	3	2.71	237	0.63	311		
	1907	9.49	3	1.31	235	0.73	300		
	1748	25.40	1	0.00	237	0.00	306		
	1671	21.66	3	0.83	240	0.83	391		

in the previous section. After some experimentation with the problems at hand we have allowed 350 iterations for the first phase of the procedure. Each time the search is intensified starting from a new x^{best} = 175 (350/2) additional iterations are allowed. The diversification phase is given 75 (350/5 $\approx$ 75) iterations. For comparison purposes an implementation of the local search method (**LS**) is also tested. Results are summarized in Table V.

Columns **SECS** illustrate the CPU times in seconds. For each algorithm column **%GAP** illustrates the quantity $(100\times(z^* - z^{opt})/z^{opt})$, where z^* denotes the function value of the solution produced by the algorithm, and z^{opt} the value of

TABLE VI. Computational Experience (II)

N	LS		TS1		TS2	
	FV	SECS	FV	SECS	FV	SECS
10						
	2735	6	2185	320	2253	414
	3047	4	2089	319	2052	540
	2539	6	2061	320	1964	433
	2553	6	2081	314	2055	456
	2754	4	1940	321	1940	459
11						
	4360	4	3796	399	3520	462
	4279	6	3838	393	3786	451
	4023	3	3415	395	3415	527
	4314	3	3621	399	3751	460
	4140	4	3689	400	3689	549
12						
	3544	14	2770	734	2770	1003
	3668	14	2856	574	2867	1008
	3886	12	2957	545	2895	711
	3834	12	2683	556	2641	867
	3788	13	2768	551	2717	856
13						
	5815	6	4887	643	4887	751
	5463	8	5194	637	5025	805
	5652	5	4791	647	4895	744
	5531	9	5144	644	5144	878
	5782	6	5133	649	5133	893
14						
	4722	21	3366	880	3431	1545
	5333	19	3684	893	3598	1407
	4967	26	3620	873	3649	1390
	4709	25	3481	899	3258	1546
	4583	26	3260	895	3294	1543

the optimum. Columns **LS**, **TS1**, **TS2** show results produced by the corresponding algorithms when starting from the random solution

$$k(i,j) = \begin{cases} (i+j-1) \bmod n & \text{if } (i+j-1) \bmod n > 0 \\ n & \text{otherwise} \end{cases} \tag{5}$$

The problems of the second set are greater in size. The set consists of twenty-five problems, five problems for each value of n, with $n = 10,11,...14$. Unfortunately, because optimum solutions could not be obtained within a reasonable time limit, the problems were solved only suboptimally. The solutions obtained by **LS**, **TS1** and **TS2** are illustrated in Table VI (columns **FV**). The starting solutions were produced by (5).

In general, it is to be noted that **TS** procedures are by far superior to the **LS** with respect to the quality of the solutions obtained. The difference in CPU times is justified, since **LS** terminates as soon as it cannot find an improving move. In the case of **TS** local optimality is not a barrier, and the procedures exhaust all 1500 iterations. Comparing the two **TS** procedures, **TS2** produces better solutions than **TS1**, thus confirming the important role of intensification and diversification strategies. Better solutions come with an increase in CPU times mainly due to the diversification phase of **TS2**, where most of the computation is carried out with real numbers (calculation of average frequencies). We note that for some problems of the second set **TS1** managed to find a better solution than **TS2**. This is mainly due to the small cut-off limit adopted (1500 iterations), which terminated the search rather prematurely for problems of bigger dimensions. For these problems additional diversification of the search is needed for a fuller exploration of the solutions' space.

Conspicuous exceptions to the good performance of **TS** are problems of size 5 and 7. For every problem of size 5 and 7 we tested we found that the same solutions kept on appearing again and again. Increasing the size of the tabu list only increased the period of appearance of these solutions. After some unsuccessful experimentation with the values of the procedures' parameters (tabu list size, number of iterations given to each search phase of **TS2**) we focused on the structure of the solutions visited. We observed that the number of candidate moves remained constant at each iteration. What is more interesting, the number of variables affected by any candidate move was always $2n$. That is, the search only visited solutions which differed from all their neighbor solutions in exactly $2n$ variables. We notice that the starting solutions for problems of size 5 and 7 described by (5) have the same property. As pointed out in Section 3, the number of variables (cells) affected by a move ranges between 4 and $2n$. Therefore a great number of solutions remain out of reach when starting from solutions with this particular property.

We ran into the same situation when we dealt with problems of size 11 and 13. This is reflected on the results in Table VI. For problems of size 11 and 13 **LS**, **TS1** and **TS2** produce solutions that are worse in terms of function value than solutions produced by the same procedures for problems of bigger size. The fact that this situation occurs in problems of prime size merits of further theoretical study.

Such a situation is a signal in **TS** for the necessity to apply a diversification strategy that drives the solution away from the entrapping region. In our case, it sufficed simply to avoid the entrapping structure at the start of the search. After some experimentation we were able to construct solutions, one of size 5 and one of 7, that did not differ by exactly $2n$ variables from every neighbor solution. These were used as initial solutions for all problems of size 5 and 7. Results are illustrated in columns **TS1*** **TS2***. We see that for all problems but one **TS2** managed to find the optimum solution. For the first problem of size 5 the optimum is discovered when starting from the solution described by (5). This indicates that solutions with the property previously analyzed form a "closed" set. If the search starts from a

solution of the set then all solutions discovered belong to this set. If the starting solution does not belong to the set then no solution of the set is visited. This explains the repetition of the same solutions. Given that only solutions of the set are visited, the search repeats the best such solutions as soon as the moves that produce them lose their tabu status. An interesting area of future research is the study of a move mechanism that will destroy and rebuild the feasibility of the solutions in such a way as to overcome the entrapment of the search in regions that exclusively consist of solutions of the particular structure identified here. The strategic oscillation approach of **TS**, which destroys and rebuilds selected parts of a solution in alternating waves (or similarly drives the solution infeasible and feasible in coordinated alternation) provides a natural basis for pursuing this goal.

In summary, **TS** is a most promising tool for obtaining near optimal solutions for **PP3**. The core of the **TS** procedures presented is the move generation mechanism which exploits the Latin-square structure of the solutions. Computational experience indicates that the intensification and diversification strategies contribute to the effectiveness of the approach. Due to hardware limitations, this is more obvious for problems of smaller size. As in similar studies the values of the search parameters are determined with the help of some preliminary experimentation.

References

Areibi S. and Vannelli A. (1994), Advanced Search Techniques for Circuit Partitioning, in Pardalos P. and Wolkowicz H. (eds.), Quadratic Assignment and Related Problems, *DIMACS Series* **16**, American Mathematical Society, 77–98.

Balas E. and Saltzman M.J. (1991), An algorithm for the three-index assignment problem, *Operations Research* **39**(1), 150–161.

Burkard R.E. and Frolich K. (1980), Some remarks on 3-dimensional assignment problems, *Methods Operations Research* **36**, 31–36.

Chakrapani J. and Skorin-Kapov J. (1993), Connection Machine Implementation of a Tabu Search Algorithm for the Travelling Salesman Problem, *Journal of Computing and Information Technology* **1**(1), 29–36.

Euler R., Burkard R.E., and Grommes R. (1986), On Latin Squares and the facial structure of related polytopes, *Discrete Mathematics* **62**, 155–181.

Frieze A.M. (1983), Complexity of a 3-dimensional assignment problem, *European Journal of Oper. Res.* **13**, 161–164.

Gilbert K.C. and Hofstra R.B. (March 1987), An algorithm for a class of three-dimensional assignment problems arising in scheduling applications, *IIE Transactions*, 29–33.

Glover F. (1989), Tabu Search – Part I, *ORSA Journal on Computing* **1**, 190–206.

Glover F. (1990), Tabu Search – Part II, *ORSA Journal on Computing* **2**, 4–32.

Glover F. and Laguna M. (1993) Tabu Search, in Colin Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publishing, 70–141.

Haley B. (1962), The solid transportation problem, *Operations Research* **10**, 448–463.

Hertz A. and De Werra D. (1990), The Tabu Search Metaheuristic: How We Used It, *Ann. Math. and Artificial Intelligence* **1**, 111–121.

Laguna M. and Glover F. (1993a), Bandwidth Packing: A Tabu Search Approach, *Management Science* **39**(4), 492–500.

Laguna M. and Glover F. (1993b), Integrating Target Analysis and Tabu Search for Improved Scheduling Systems, *Expert Systems with Applications* **6**, 287–297.

Magos D. and Miliotis P. (1994), An Algorithm for the Planar Three-index Assignment Problem, *European Journal of Oper. Res* **77**, 141–153.

Osman I.H. (1993), Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem, *Annals of Operations Research* **41**, 421–451.

Ryser J.H. (1963), *Combinatorial Mathematics – The Carus Mathematical Monographs* **14**, Wiley, New York.

Tailard E. (1990), Robust Tabu Search for the Quadratic Assignment Problem, *Parallel Computing* **17**, 443–445.

Vlach M. (1967), Branch and bound method for the three-index assignment problem, *Ekonomicko-Matematicky Obzor* **3**, 181–191.