2008

Tabu Search Heuristics for the Crane Sequencing Problem

Alan McKendall

Tabu Search Heuristics for the Crane Sequencing Problem

Alan R. McKendall Jr.*

Department of Industrial and Management Systems Engineering,
West Virginia University,
325A Mineral Resources Building,
PO BOX 6070,
Morgantown, WV 26506, USA
E-mail: armckendall@mail.wvu.edu
*Corresponding author

Jin Shang

Quality Planning Corporation,
388 Market Street, Suite 750,
San Francisco, CA 94111, USA
E-mail: jshang@qualityplanning.com

Abstract: Determining the sequence of relocating items (or resources) moved by a crane from existing positions to newly assigned locations during a multi-period planning horizon is a complex combinatorial optimisation problem, which exists in power plants, shipyards, and warehouses. Therefore, it is essential to develop a good crane route technique to ensure efficient utilisation of the crane as well as to minimize the cost of operating the crane. This problem was defined as the Crane Sequencing Problem (CSP). In this paper, three construction and three improvement algorithms are presented for the CSP. The first improvement heuristic is a simple Tabu Search (TS) heuristic. The second is a probabilistic TS heuristic, and the third adds diversification and intensification strategies to the first. The computational experiments show that the proposed TS heuristics produce high-quality solutions in reasonable computation time.

Keywords: Crane Sequencing Problem; diversification; heuristic; intensification; Tabu Search.

Reference to this paper should be made as follows: McKendall, A.R. and Shang, J. (xxxx) 'Tabu Search Heuristics for the Crane Sequencing Problem', *Int. J. Operational Research*, Vol. 3, No. 4, pp.412–429.

Biographical notes: Alan McKendall Jr. is an Associate Professor in the Department of Industrial and Management Systems Engineering at West Virginia University. He received his PhD degree in Industrial Engineering from the University of Missouri in Columbia, MO, USA. His main research interest is in developing efficient algorithms for Logistics and Scheduling Systems. He has published in journals such as *Computers & Operations Research, Information Sciences, International Journal of Industrial Engineering*, and *International Journal of Production Research*.

Jin Shang is a Software Engineer at Quality Planning Corporation in San Francisco, CA, USA. He earned his PhD and MS degrees in Industrial Engineering from West Virginia University. Currently he conducts research on the use of algorithms in database keyword search. Also he has an experience in using different heuristics to solve real industrial problems such as the dynamic facility layout and Crane Sequencing Problems. His research interests are developing heuristics for various combinatorial problems.

1 Introduction

In many industrial environments such as power plants, shipyards, and warehouses, multiple items (resources) need to be reassigned and moved to new locations. Relocating items, especially heavy bulky items, is costly and may represent a significant portion of the overall relocation budget. Therefore, it is necessary to develop efficient techniques to determine the sequences in which a vehicle (e.g. crane) moves items to their newly assigned locations (i.e. destination) during a multi-period planning horizon. More importantly, crane routes need to be constructed such that total travel cost of the crane is minimised. This problem is defined as the Crane Sequencing Problem (CSP) and was presented in McKendall et al. (2006).

The CSP is related to the well-known Travelling Salesman Problem (TSP) defined in Laporte (1992) and other related problems such as the dial-a-ride problem. The dial-a-ride problem is a TSP with precedence relation where a vehicle transport a number of passengers, and each passenger should be picked up from a specific location and delivered to a specific destination (Hunsaker and Savelsbergh, 2002). If a single capacity vehicle is considered in the dial-a-ride problem, the resulting problem is known as the Stacker Crane Problem (SCP) as mentioned in Hernandez-Perez and Salazar-Gonzalez (2004). SCP is a modified TSP which requires that a salesman or a vehicle visits a set of ordered pairs of locations (Frederickson et al., 1978). Each pair of locations corresponds to a pickup and delivery location. In SCP, the crane may start from an initial position, perform a set of moves, and return to a terminal position, and the objective is to find the sequence of these moves such that total tour length is minimised. In Frederickson et al. (1978), the authors described some practical applications of SCP such as operating a crane or a forklift, or driving a pickup and delivery truck. Frederickson and Guan (1992) also discussed the pre-emptive case of SCP (P-SCP) where items can be temporarily stored at available locations and picked up and delivered to their destination locations later. CSP considered in this paper is similar to P-SCP, since the crane starts from an initial position and performs a set of moves such that total cost is minimised while allowing pre-emption to occur. However, CSP differs from P-SCP because of the following.

1 In CSP, the crane does not return to a terminal position as in P-SCP.

2 In CSP, a multi-period planning horizon is considered whereas a single period is considered in P-SCP.

3 In CSP, locations have limited capacities. However, the locations have unlimited capacity in P-SCP.

4 In CSP, the objective is to minimise the total cost which is the sum of the loading/unloading costs and tour costs. In P-SCP, the objective is to minimise total tour length.

As a result, CSP is a more general problem than P-SCP. Therefore, CSP is computationally intractable and only small-size problems can be solved optimally in reasonable computation time.

As mentioned earlier, CSP was first defined by McKendall et al. (2006) where the authors presented a simulated annealing heuristic for their problem. Their problem was developed from the problem of relocating resources (items) during outages at electric power plants. In other words, items (or resources) need to be relocated from existing locations to their newly assigned locations by using a single-capacity overhead crane. More specifically, the problem was to determine the sequences (or the orders) in which items are to be moved for a single overhead crane, during a multi-period planning horizon, such that the total distance travelled by the crane is minimised. However, in this paper, the objective is modified to minimise the sum of the crane travel cost as well as the loading/unloading costs.

Besides determining the order in which items are relocated during outages at electric power plants, CSP has many potential applications. Another application of CSP occurs in the context of warehouse rearrangement. Since the demand of products is dynamic and some products may become obsolete, warehouse managers may rearrange the layout of their products in warehouses. For example, products with high demand are located close to the input/output locations. As a result, the locations of products may change. Therefore, CSP can be used to determine the sequence in which a vehicle moves the products reassigned to new locations in a warehouse. This problem is called the warehouse rearrangement problem and was presented by Christofides and Colloff (1973). Also, CSP can be used to determine the order in which a crane removes/loads containers from/onto ships.

In this paper, Tabu Search (TS) heuristics, which include a basic TS, a Probabilistic TS (PTS), and a TS with intensification and diversification strategies, are proposed for CSP. The contributions of the paper and how it is organised are as follows. Section 2 gives the definition, assumptions, and solution representation of CSP. Construction algorithms, a simple TS, PTS, and TS with intensification and diversification strategies are presented in Section 3. In Section 4, the computational results are given, and Section 5 provides conclusions and future research directions.

2 The crane sequencing problem

2.1 Introduction

CSP is the problem of determining the sequences in which a single overhead crane moves items (or resources) to their newly assigned locations in multiple periods with respect to minimising total costs. However, temporary storage locations can be used to store items temporarily when the destination location of an item currently being moved is at full capacity or cost of crane routes can be reduced if temporary storage space is used. Therefore, pre-emption is allowed as discussed earlier. An item stored in temporary

storage can be delivered to its destination location or another temporary storage space when:

1 the crane is at its temporary location or

2 after all the other items have been moved to their assigned locations.

Other important assumptions of CSP include the single overhead crane can move only one item at a time and, without loss of generality, the initial position of the crane at the beginning of a period is the last position of the crane in the previous period.

The inputs for CSP are the assignments of items to locations for multiple periods (which is obtained by solving the dynamic space allocation problem presented in McKendall et al. (2005)), the distances between locations (i.e. $D = [d_{ij}]$, $i, j = 1, 2, \ldots, L$ where d_{ij} is the distance from location i to j, and L is the total number of locations), capacities of the locations, initial location of the crane, costs of moving items per distance unit, and the costs of loading/unloading items. The outputs of CSP are the crane routes for each period and the total cost of the routes. More specifically, the set of sequences of items to be moved during a multi-period planning horizon is obtained. Based on these sequences of items to be moved, the crane will load an item, move, and unload it at either its destination location or a temporary storage. As stated earlier, a temporary location is used if its destination location is at full capacity or total cost can be reduced. In other words, the actual movements of the crane (e.g. moving items to/from temporary storage locations) are determined using a serial method (given the sequences of items to be moved). The serial method presented in McKendall et al. (2006) is used to construct the crane routes. CSP is difficult due to the large number of possible permutations of items to be moved at each period and the large number of possible crane routes that can be generated.

During operation, the crane travels while either moving an item to its destination location or moving empty to retrieve an item. Thus, the status of the crane can be defined as either loaded or not loaded. Hence, there are two corresponding types of moves: empty moves and non-empty moves (see Figure 1). More specifically, an empty move is a move where the crane does not carry any items (i.e. crane is not loaded); whereas, a non-empty move is a move where the crane carries an item (i.e. crane is loaded). An empty move may be necessary for the crane to obtain an item to be moved. For instance, if there is no item to be moved at the current location of the crane, the crane needs to perform an empty move to arrive at a location of an item which needs to be moved.

Based on the move types of the crane, the total costs of CSP should include the costs related to empty moves and non-empty moves of the crane as well as the cost related to loading/unloading items (see Figure 2). The distances the crane travels can be used as a criterion to measure costs when the crane moves empty or non-empty. As a result, the travel cost of the crane is the product of the distances the crane travels (empty or non-empty) and the cost per distance unit. In contrast, the loading/unloading cost is considered only for non-empty crane moves. That is, the loading/unloading cost of an item is the product of the number of non-empty moves for the item and the sum of the unloading and loading costs for the item. In other words, the cost of a non-empty move is the cost of loading an item onto the crane and the cost of unloading the item once it reaches its destination location. This is called the loading/unloading cost. The objective of CSP is to minimise the sum of the crane travelling costs and loading/unloading costs.

Figure 1 Status of crane and move types

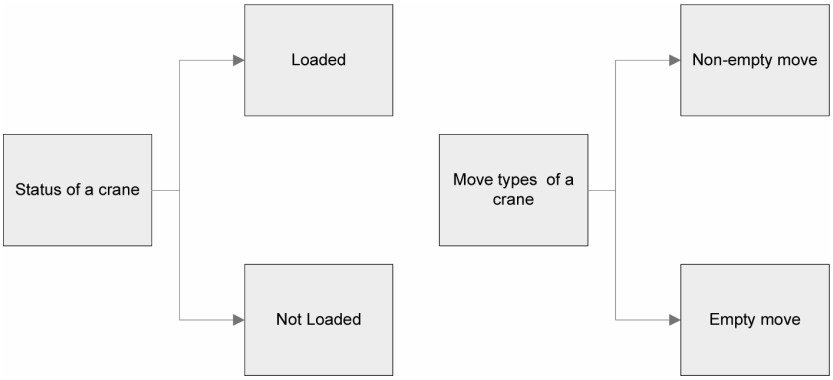
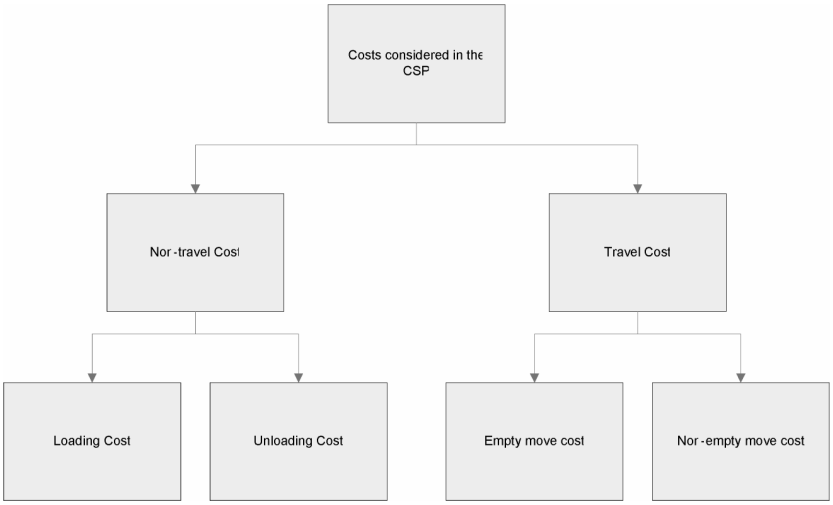


Figure 2 Costs considered in CSP



2.2 Crane routes

Since temporary storage locations may be used, an item may be moved multiple times (i.e. item is moved to temporary storage spaces one or more times). Therefore, the sequences of items to be moved may not give the crane routes. As discussed previously, in order to determine the crane routes (i.e. a set of ordered locations to be visited by the crane) for relocating items to be moved, the serial method developed in McKendall et al. (2006) is used. In other words, by using the serial method, once the sequence of items to be moved is obtained, the crane route and its total cost are generated.

2.3 Solution representation

The sequences of items to be moved can be represented as $\pi = \{\pi_2, \ldots, \pi_T\}$ where each π_t in π represents an ordered list of items (i.e. a sequence of items) to be moved by the crane

at the beginning of each period t (where $t = 2,\ldots,T$). More specially, $\pi_t = (\pi_{1t}, \pi_{2t},\ldots, \pi_{n_t t})$, for $t = 2,\ldots,T$, where π_{it} is the ith item moved by the crane in period t, and $\pi_{n_t t}$ is the last item to be moved by the crane in period t. Therefore, the entire solution π includes multiple permutations of items to be moved for T–1 periods and $\pi = \{\pi_2,\ldots,\pi_T\} = \{(\pi_{12},\pi_{22},\ldots, \pi_{n_2 2}), (\pi_{13}, \pi_{23},\ldots, \pi_{n_3 3}),\ldots,(\pi_{1T}, \pi_{2T},\ldots, \pi_{n_T T})\}$. Once the sequences of items to be moved are obtained, the serial method discussed above is used to obtain the crane routes and the total cost of the routes.

3 Solution techniques

3.1 Construction algorithms

In order to obtain diverse solutions for CSP, three construction algorithms are proposed in this paper. The first is a very simple algorithm, which lists the items to be moved in ascending order for each period. For example, if items 1, 2, 4, and 8 need to be reassigned to new locations in period t, then $\pi_t = \{1, 2, 4, 8\}$. This construction algorithm is called CAI.

The second construction algorithm, called CAII, is a nearest neighbour heuristic. In other words, the order in which the items are moved is based on the distances between the current location of the crane and the locations of the items to be moved. For instance, if items 1, 2, and 4 are reassigned to locations in period t and the distances between the current location of the crane and the locations of the items are 3, 2, and 1, respectively, then item 4 is assigned to the first position of the move sequence. If a tie exists, the item with the least number of items in its destination location is selected (used to reduce the use of temporary storage locations). Next, the item assigned to the second position of the sequence is the item closest to either the destination location or the temporary storage location of item 4. Nevertheless, the item closest to the current location of the crane is selected, say for instance item 1. As a result, $\pi_t = \{4, 1, 2\}$. It is obvious that this heuristic attempts to minimise crane travel cost.

In the third construction algorithm, CAIII, the location of the first item to be moved is selected such that the location has the most items to be moved. If a tie exists between one or more locations, the location closest to the crane is selected (used to reduce crane travel cost). Once this location is determined, the item in this location with the least number of items in its destination location is selected first. This process is repeated for all items needed to be moved. This heuristic attempts to minimise the use of temporary storage locations such that loading/unloading costs are minimised.

3.2 Tabu Search

The TS heuristic was first presented by Glover (1986). Also, see Glover (1989, 1990a,b). The basic idea of TS is to improve a solution iteratively, using some guiding rules such as recency (short-term) memory as well as intensification and diversification strategies to obtain good solutions in complex solution spaces. The basic components of the proposed TS heuristic are discussed below.

The TS heuristic uses a steepest descent local search heuristic. The steepest descent heuristic starts from an initial solution π and explores its entire neighbourhood, $N(\pi)$.

More specifically, all possible pairwise exchanges between items to be moved are considered for each period t, and the best exchange is performed. That is, all the neighbouring solutions in the neighbourhood of π, $N(\pi)$, is considered for each period t and the best neighbour $\pi' \in N(\pi)$ (i.e. the best move, $move^* = (t, u, v)$, which exchange the locations of items u and v in period t) is selected such that $f(\pi') < f(\bar{\pi})$ for $\forall\ \bar{\pi} \in N(\pi)$. The corresponding solution is the current solution at the next iteration (i.e. $\pi = \pi'$). When a local optimum is obtained (i.e. no π' exist such that $f(\pi') < f(\bar{\pi})$ for $\forall\ \bar{\pi} \in N(\pi)$), the heuristic terminates. Therefore, the heuristic accepts only improved solutions and do not accept non-improving solutions as with other simple local search techniques such as the first improvement local search heuristic. As a result, the steepest descent often converges to a poor local optimum, usually depending on the quality of the initial solution π. Therefore, components of the TS heuristic (e.g. short-term memory, aspiration criterion) are used to overcome these drawbacks of the simple steepest descent local search technique in search of the global optimum.

The proposed TS heuristic uses the steepest descent heuristic with short-term memory (or recency based memory) to accept uphill moves. In other words, the steepest descent heuristic converges to a local optimum; however, short-term memory is used to forbid the recent moves so that the heuristic can climb out of the valley which contains the local optimum (i.e. accept non-improving moves) in search of better local optima. In CSP, if the best solution in the neighbourhood of the current solution π is π' (i.e. $f(\pi') < f(\bar{\pi})$ for $\forall\ \bar{\pi} \in N(\pi)$) and π' is obtained by $move^* = (t, u, v)$, which exchanges the locations of items u and v in period t, then this move is tabu restricted for a certain duration (*tabusize*), called tabu list size. For CSP, the tabu list size $tabusize_t$ are unique for different periods. The tabu status and tabu list size of each move are maintained in the lower half of the tabu list structure $tabu[t][u][v]$, where $u > v$. Sometimes a move which is tabu restricted may give the best solution found thus far. Therefore, the aspiration criterion is used to override the tabu restriction of a move when the move improves the best found solution thus far. For example, if at the current iteration (*iter*), items 2 and 6 exchange positions in period 3 (i.e. $move^* = (3, 6, 2)$) where $\pi = \{\pi_2, \pi_3\} = \{(1, 4), (1, 2, 5, 3, 6)\}$, then $\pi' = \{\pi_2, \pi_3'\} = \{(1, 4), (1, 6, 5, 3, 2)\}$ and $tabu[3][6][2] = tabusize_3 + iter$. Therefore, the move, which considers exchanging items 2 and 6 in period 3, is tabu until $iter = tabusize_3 + iter$. In other words, the move, which considers exchanging items 2 and 6 in period 3, can be performed again when $iter = tabusize_3 + iter + 1$. Also, if $move^*$ had been performed recently, is tabu restricted, and it improves the best solution found thus far, then the aspiration criterion is used to override its tabu restriction. Any move which is acceptable (i.e. non-tabu move and tabu move overridden by aspiration criterion) is defined as an admissible move. Hence, $move^*$ is defined as the best admissible move. A simple TS heuristic for CSP is outlined below.

Step 1: Initialise parameters and counters:
T is the total number of periods;
$Tabu[][][]$ is the tabu list structure;
$tabusize_t$ is the tabu tenure length for period t;
iter is iteration number where $iter = 0$;
TRT is the Total Running Time before terminating the heuristic;

Step 2: Obtain an initial solution π using each of the construction algorithms presented above (i.e. CAI, CAII, and CAIII) and determine the objective function value $f(\pi)$ for each π using the serial method presented in McKendall et al. (2006); then perform the following steps for each initial solution π.

Step 3: Set $\pi^* = \pi$, where π^* is the best solution found thus far and set $f(\pi^*) = f(\pi)$;

Step 4: Set $iter = iter + 1$;
Find the best admissible move $move^* = (t, u, v)$ with respect to objective function value, which gives π';

Step 5: Set $\pi = \pi'$ and $f(\pi) = f(\pi')$;
If $f(\pi) < f(\pi^*)$,
Set $\pi^* = \pi$ and $f(\pi^*) = f(\pi)$;

Step 6: Update tabu list as $tabu[t][u][v] = iter + tabusize_t$;

Step 7: Stopping criteria
If heuristic running time < TRT, go to Step 4.
Else, terminate the heuristic and return solution π^* and total cost $f(\pi^*)$;

3.3 Probabilistic Tabu Search

The PTS heuristic for CSP is an extension of the TS heuristic presented in the previous section. Although the TS heuristic accepts uphill moves to escape from poor local optima, it is a deterministic heuristic which may not explore a large portion of the solution space far away from the initial solutions. Therefore, the PTS heuristic is used to add randomness so that diverse solutions may be obtained. The main difference between the PTS and the TS heuristic is how *move** is selected at each iteration. Before, in the TS heuristic, *move** is defined as the best admissible move. However, in the PTS heuristic, *move** is selected randomly among the best G admissible moves. Similarly, *move** is used to generate the new solution π', which becomes the current solution π at the next iteration. More specifically, for the PTS heuristic, *move** is selected from the best G admissible moves according to their probabilities. In other words, the best G admissible moves are sorted based on their corresponding Objective Function Value (OFV). The probability to accept the first (i.e. best) move from the G moves is p. In this research, G is set to 10 and p is set to 0.4. If the first move is rejected, the second move is accepted with a probability $p(1-p)$. This process is repeated until a move is selected. However, if no move is selected after considering all G moves, the first move will be selected. An efficient technique for selecting *move** is to use the cumulative probability proposed in Chiang and Chiang (1998). More specifically, the cumulative probability table below is created using the following equations.

$$CP(i) = \begin{cases} 0, & \text{for } i > G \text{ or } i < 1, \\ p(1-p)^{G-1}, & \text{for } i = G, \\ AP(i+1) + p(1-p)^{i-1}, & \text{for } 2 \le i < G, \\ 1, & i = 1 \end{cases}$$

Then, a random number x between 0 and 1 is generated. If $CP(i + 1) < x \leq CP(i)$, then the ith move out of the G moves is selected as *move**. In Table 1 below are the cumulative probabilities where G = 10 and p = 0.4. For instance, a random number x, say $x = 0.25$ is generated. Since $CP(4) = 0.2099 < x \leq CP(3) = 0.3539$, the third move from the list of G moves is defined as *move**. Therefore, the PTS heuristic contains steps 1–7 of the proposed TS heuristic presented earlier. However, *move** in step 4 is obtained using the cumulative probability table and the equations (i.e. $CP(i)$) defined above.

Although the PTS heuristic usually out-performs the basic TS heuristic, only a few researchers actually applied the PTS heuristic to solve combinatorial optimisation problems in the literature. Chiang and Chiang (1998) applied the PTS heuristic to the quadratic assignment problem, and Lim et al. (2004) presented a similar PTS heuristic to solve a crane scheduling problem with spatial constraints.

Table 1 Cumulative probabilities with $G = 10$ and $p = 0.4$

i	*P*	*AP*
0	0	0
1	0.4	1
2	0.24	0.5939
3	0.144	0.3539
4	0.086	0.2099
5	0.0518	0.1235
6	0.0311	0.0717
7	0.0186	0.0406
8	0.0111	0.0219
9	0.0067	0.0107
10	0.0040	0.0040
>10		0

3.4 Tabu Search with intensification and diversification strategies

The main idea of the proposed TS and PTS heuristics presented above for CSP is as follows. The use of short-term (recency-based) memory to keep track of the most recent moves is used to avoid getting trapped at a poor local optimum by allowing uphill (non-improving) moves. More specifically, the steepest descent heuristic with recency-based memory is used to obtain the best admissible move *move** in the TS heuristic. In the PTS heuristic, the same components are used to obtain the best G admissible moves, as in the TS heuristic. However, the admissible move selected, *move**, may not be the best move but is in the top G admissible moves. More specifically, *move** is selected based on cumulative probabilities. In short, the proposed TS heuristic is a deterministic heuristic in which the solution quality may depend on the quality of the initial solutions. However, the PTS heuristic adds randomness to the TS heuristic such that the solution quality does

not depend on the quality of the initial solutions provided. Adding cumulative probabilities to the basic TS heuristic is a diversification strategy used to better explore different areas of the solution space. Other diversification techniques such as dynamic tabu list size and frequency-based (long-term) memory may be used to improve the performance of the basic TS heuristic. Also, intensification strategies may be used to explore promising regions within the solution space more thoroughly. Next, diversification and intensification strategies used to improve the basic TS heuristic for CSP are presented.

The diversification strategies (i.e. dynamic tabu list sizes and frequency-based memory) presented in Chiang and Kouvelis (1996) are modified for the CSP and used in this research. With dynamic tabu list size, the tabu list size $tabusize_t$ is not fixed as in the basic TS procedure but is dynamic in order to diversify the search. In other words, tabu list size $tabusize_t$ is dynamic and updated at each iteration. More specifically, the $tabusize_t$ varies between a Lower Bound LB_t and an Upper Bound UB_t. To illustrate the dynamic tabu list size, first z is obtained to determine if the cost of the solution obtained by performing the best admissible move in period t (π') is an improvement over the cost of the current solution π such that

$$z = 1 - \frac{f(\pi')}{f(\pi)}.$$

Then, LB_t and UB_t are defined as:

$$LB_t = n_t * T / 3 \quad \forall t > 1$$

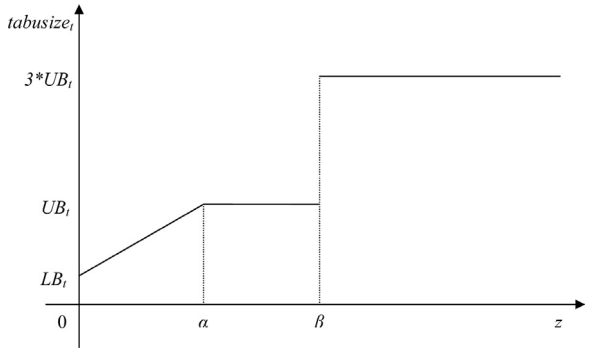
$$UB_t = n_t * T * 2 \quad \forall t > 1$$

where n_t is the total number of items to be moved at each period t and T is the total number of periods in CSP. Since n_t may change between periods, LB_t, UB_t and $tabusize_t$ may not be the same for different periods. Initially, the tabu list size $tabusize_t$ in period t is set to LB_t. Then, at the end of each iteration, tabu list size $tabusize_t$ will be updated as follows.

$$f(tabusize_t) = \begin{cases} tabusize_t & \text{if } z \le 0 \\ LB_t + \dfrac{UB_t - LB_t}{\alpha} * z & \text{if } 0 < z \le \alpha \\ UB_t & \text{if } \alpha < z < \beta \\ 3 * UB_t & \text{if } z \ge \beta \end{cases}$$

where α and β are pre-defined parameters. The graph of this function for $z > 0$ is shown in Figure 3. In short, if the solution obtained in the current iteration is worse than the solution obtained in the previous iteration $tabusize_t$ remains the same. However, if the solution is much better than the solution obtained at the previous iteration (say $z > \beta$), $tabusize_t = 3*UB_t$.

Figure 3 Dynamic tabu list size $tabusize_t$ when $z > 0$



Therefore, the basic TS heuristic can be easily modified to consider dynamic tabu list sizes by replacing step 6 with the following.

Step 6: Update $tabusize_t = f(tabusize_t)$;
Update tabu list as $tabu[t][u][v] = iter + tabusize_t$;

Another diversification technique is to use frequency-based (long-term) memory. This diversification strategy is added to the TS heuristic to diversify the search space, forcing the search into unexplored regions of the solution space. It is employed only when no improving admissible move exists and a penalty is given for each non-improving move. In order to apply this diversification strategy to the TS heuristic, a long-term memory structure is needed. Unlike the short-term (recency) memory discussed above, where only the recent moves are given, the long-term memory structure keeps track of the frequencies of all moves during the search process and shows the distribution of each move. For CSP, the long-term memory is used to keep track of the number of exchanges between pairs of items in each period. The frequencies of the moves are kept in the upper half of the tabu list structure $tabu[t][u][v]$, where $u < v$. For instance, if items u and v exchanged positions in period t (i.e. $move^* = (t, u, v)$), then long-term memory is updated as:

$$tabu[t][u][v] = tabu[t][u][v] + 1 \text{ for } u < v;$$

See a tabu list structure instance in Figure 4 below. The tabu structure shows that in period 3, 5 items were relocated (i.e. items 1, 2, 3, 5, and 6) and after several iterations, a number of exchanges have been performed. For instance, $tabu[3][3][5] = 13$, which means that items 3 and 5, in period 3, exchanged locations 13 times thus far during the execution of the heuristic.

The long-term memory structure defined above shows the distribution of moves in each period and is used in the diversification strategy to penalise non-improving moves by giving a larger penalty to the moves with greater frequency counts. Notice, in Step 4 presented above, the basic TS heuristic selects the best admissible move $move^* = (t, u, v)$ and the corresponding solution π' is obtained by considering $f(\pi') < f(\bar{\pi}) \; \forall \, \bar{\pi} \in N(\pi)$). However, if the penalty for non-improving moves is added, then a modified objective function is used. Thus, step 4 is modified as follows.

Step 4: Set *iter* = *iter* + 1;

Find the best admissible move *move** = (*t*, *u*, *v*) which gives π' according to a revised OFV where the revised OFV is

$rf(\pi') = f(\pi') + \lambda * tabu(t, u, v)$ (for $u < v$)

$$\lambda = \begin{cases} 0 & \text{if } f(\pi') < f(\pi); \\ \Lambda & \text{Otherwise;} \end{cases}$$

Λ is a pre-defined parameter for penalty. For each non-improving move, a penalty is given based on the frequency of the move performed thus far. Then, *move** = (*t*, *u*, *v*) will be selected based on this revised evaluation value for each move.

Figure 4 A tabu structure for a CSP instance

Tabu $t = 3$	Item				
Item	1	2	3	5	6
1	0	5	2	9	0
2	140	0	0	0	0
3	47	0	0	13	0
5	212	102	320	0	17
6	99	0	0	401	0

Unlike the diversification strategies presented above, the intensification strategy explores promising areas of the solution space more thoroughly. In the literature, intensification typically operates by restarting from relatively high quality solutions or modifying a solution to favour some attributes. In this research, the intensification method described in Chiang and Kouvelis (1996) is modified for CSP and integrated into the proposed TS heuristic. This intensification strategy is implemented by fixing two items after exchanging their locations, if this exchange reduces the total cost of the current best solution π^* by at least α, where α is the same value as the parameter used in the diversification strategy as used in Chiang and Kouvelis (1996). Also, experimental tests were performed which yielded the same results. Similar to the aspiration criterion, a fixed item can be freed to exchange its location with other items if the exchange results in an improvement better than the best solution found so far. Intensification is employed after a certain number (η) of iterations have been performed, since there are relatively higher probabilities of having more solution improvements at earlier iterations.

The above diversification and intensification strategies are added to the basic TS heuristic which yields the so called TS heuristic with strategies (TS/S). The steps for the TS/S heuristic are as follows.

Step 1: Initialise parameters and counters:

T is the total number of periods;

Tabu[][][] is the tabu list structure;

$tabusize_t$ is the tabu tenure length for period *t*;

iter is iteration number where *iter* = 0;

TRT is Total Running Time before terminating the heuristic;

LB_t is the Lower Bound for $tabusize_t$;

UB_t is the upper bound for $tabusize_t$;

α is the parameter for diversification and intensification strategy;
β is the parameter for diversification strategy;
Λ is a parameter for penalty
η is the number of iterations performed before invoking the intensification strategy;
F[][] stores all items fixed during the intensification process;

Step 2: Obtain an initial solution π by using a construction algorithm and determines its objective function value $f(\pi)$;

Step 3: Set $\pi^* = \pi$, where π^* is the best solution found thus far and set $f(\pi^*) = f(\pi)$;

Step 4: Set $iter = iter + 1$;
Find the best admissible move $move^* = (t, u, v)$ which gives π' with respect to a revised OFV where the revised OFV is
$rf(\bar{\pi}) = f(\bar{\pi}) + \lambda * tabu(t, u, v)$ (assume $u < v$) for

$$\lambda = \begin{cases} 0 \text{ if } f(\bar{\pi}) < f(\pi); \\ \Lambda \text{ Otherwise;} \end{cases}$$

If $iter > \eta$,
The items u and v in selected $move^*$ cannot belong to the set F (i.e. $move(t, u, v)$ is not fixed); or
If either (t, u) or (t, v) belong to the set F,
The $move(t, u, v)$ is selected as the $move^*$ only when this move results in a new best solution found.

Step 5: $z = 1 - f(\pi')/f(\pi)$ and set $\pi = \pi'$, $f(\pi) = f(\pi')$;
If $f(\pi) < f(\pi^*)$,
If $iter > \eta$ and $z' = 1 - (f(\pi)/f(\pi^*)) \geq \alpha$,
Set $F = F \cup (t,u) \cup (t,v)$;
Set $\pi^* = \pi$ and $f(\pi^*) = f(\pi)$;

Step 6: Update $tabusize_t = f(tabusize_t)$ where

$$f(tabusize_t) = \begin{cases} tabusize_t & \text{if } z \leq 0 \\ LB_t + \dfrac{UB_t - LB_t}{\alpha} * z & \text{if } 0 < z \leq \alpha \\ UB_t & \text{if } \alpha < z < \beta \\ 3 * UB_t & \text{if } z \geq \beta \end{cases}.$$

Update tabu list as $tabu[t][u][v] = iter + tabusize_t$ for u > v and
tabu[t][u][v] = tabu[t][u][v] + 1 for u < v;

Step 7: Stopping criteria: if heuristic running time < TRT, go to Step 4.
Else, terminate the heuristic and return solution π^* and total cost $f(\pi^*)$;

4 Computational results

Two sets of test problems are used in this paper in order to test the performances of the proposed heuristics. In all experiments, a Pentium IV 2.4 GHz PC was used to solve CSP instances in data sets I and II using the proposed heuristics. Data set I was generated to test a mathematical model developed for CSP, and data set II was taken from McKendall et al. (2006). The heuristics were coded using the C++ programming language. In order to make fair comparisons, the proposed heuristics ran for the same amount of computational time for each CSP problem instance.

For the basic TS and PTS heuristics, $tabusize_t$ is set to $n_t * T/2$. Moreover, $p = 0.4$ and $G = 10$ are the parameters setting for the PTS heuristic. The intensification strategy parameters α, η and the diversification parameters Λ, α, β are set by statistical techniques. More specifically, for each CSP problem, a certain numbers of initial solutions were generated randomly. Then, for each of these solutions, a steepest descent heuristic is used to improve the solution until no further improvement can be found (i.e. local optimum is reached). Next, the statistical information for these improved solutions are gathered, such as Average Percent Improvement (API) at each iteration, Average Iterations (AI) before reaching the local optimum, Average Maximal Percent Improvement (AMPI) at each iteration, Average Median Objective Function Value (AMOFV), and Average Minimal Objective Function Value (AOFV). Then, the parameters α is set to API, η is equal to AI, Λ is set to AMOFV/AOFV and β is set to AMPI. In addition, LB_t and UB_t are defined as:

$$LB_t = n_t * T/3 \quad \forall t > 1 \text{ and } UB_t = n_t * T * 2 \quad \forall t > 1.$$

Table 2 shows the results for the first data set obtained from the proposed heuristics (TS, PTS, TS/S) and the SA heuristic presented in McKendall et al. (2006). The optimal solutions were obtained for all of the test problems in this set and were obtained using a mathematical model and the CPLEX solver version 6.6. Also, their computational times are listed. All times in the tables are given in minutes. As a result, all of the proposed heuristics obtained the optimal solutions for all 24 test problems. Notice the run times for the heuristics are much less than the run times for solving the problems using a mathematical model and the CPLEX solver. The main reason for using this data set was to test the mathematical model and also for testing the heuristics on smaller problems where the optimal solutions are obtainable.

The second data set is used to further verify the speed and robustness of the proposed heuristics. Compared to data set I, data set II is much larger and has much larger problems, which cannot be solved by using exact methods (i.e. mathematical model and CPLEX) in reasonable time. The TS and TS/S heuristics, which are deterministic techniques, ran once with each of the three solutions obtained using the construction algorithms for each test problem. SA and PTS ran five times with each of the three solutions obtained from the construction algorithms. Data set II is divided into four groups, each of which has 24 test problems. From group 1 to group 4, the sizes of the test problems are increased. The test problems in group 1 are relatively small, groups 2 and 3 have medium size test problems, and the test problems in group 4 are the largest.

Table 3 shows the number of best solutions obtained by each of the proposed heuristics. For the first group (i.e. test problems 01–24), the test problems have nine resources, six locations, and three to five periods, which are the smaller size problems.

All proposed heuristics performed well for this group of test problems. More specifically, TS obtained the best found solution 22 times, PTS 24 times, TS/S 24 times, and SA 23 times.

Table 2 Computational results for data set I

	Optimal solution						
Pb no	*Final*	*Time*	*SA*	*TS*	*PTS*	*TS/S*	*Heuristic time*
P01	57	0.01	57	57	57	57	0.00
P02	83	0.05	83	83	83	83	0.00
P03	62	0.03	62	62	62	62	0.00
P04	88	0.11	88	88	88	88	0.01
P05	65	0.08	65	65	65	65	0.01
P06	119	7.60	119	119	119	119	0.01
P07	72	0.02	72	72	72	72	0.01
P08	91	0.23	91	91	91	91	0.02
P09	121	0.33	121	121	121	121	0.02
P10	150	19.42	150	150	150	150	0.02
P11	118	2.02	118	118	118	118	0.02
P12	150	19.53	150	150	150	150	0.01
P13	104	0.10	104	104	104	104	0.01
P14	189	1527.36	189	189	189	189	0.02
P15	114	0.12	114	114	114	114	0.02
P16	229	8.48	229	229	229	229	0.02
P17	145	7.05	145	145	145	145	0.02
P18	278	1947.32	278	278	278	278	0.02
P19	157	60.97	157	157	157	157	0.02
P20	246	1014.73	246	246	246	246	0.03
P21	169	12.93	169	169	169	169	0.03
P22	320	2915.87	320	320	320	320	0.04
P23	197	318.12	197	197	197	197	0.04
P24	347	2755.59	347	347	347	347	0.03

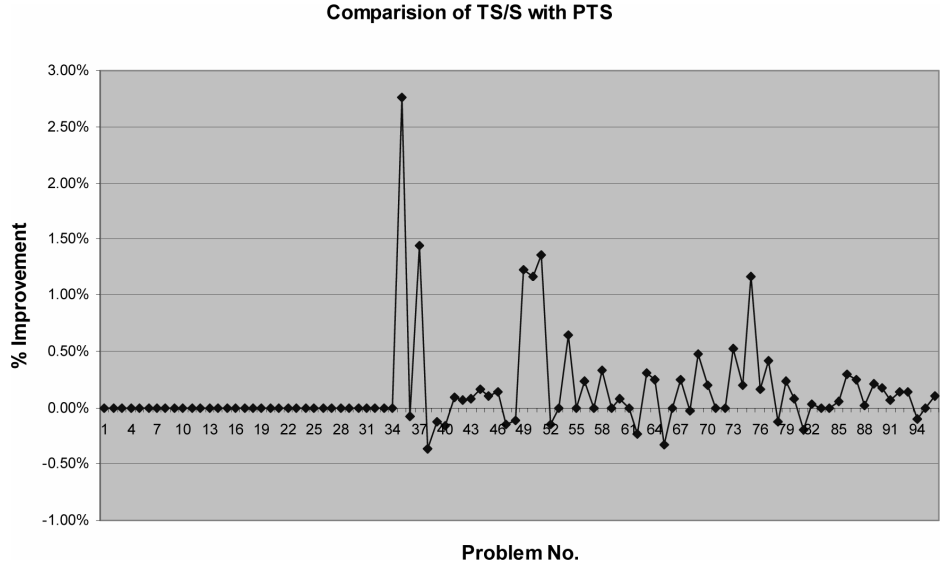
Table 3 Computational results (no. of best solutions obtained) for data set II

No.	*TS*	*PTS*	*TS/S*	*SA*
P01–P24	22	24	24	23
P25–P48	5	14	17	15
P49–P72	1	11	17	3
P73–P96	0	6	16	4
Total	28	55	74	45
Percent	29.2	57.3	77.1	46.9

However, the proposed heuristics perform differently for the other three groups of test problems. In other words, as the problem size increases, the performances of the TS, PTS, and SA decrease. More specifically, TS obtained the best found solutions 5, 1, 0 times, PTS obtained 14, 11, 6 times, TS/S obtained 17, 17, 16 times, and SA obtained 15, 3 and 4 times. Clearly, TS/S outperformed all other heuristics since it obtained 50 (74) best solutions of 72 (96) test problems, which is the most. In other words, TS/S obtained the best solution for 74 (77.1%) of the 96 test problems. In contrast, the simple TS heuristic only found 28 (29.2%) best solutions overall.

Based on the previous analyses, it is obvious that both proposed PTS and TS/S heuristics outperformed the basic TS heuristic. See the comparisons of TS/S and PTS in Figure 5, which shows the percent improvement of TS/S heuristic over the PTS heuristic for data set II. More specifically, when comparing the TS/S heuristic with the PTS heuristic, TS/S heuristic obtained better solutions than PTS 38 out of 96 test problems, and PTS heuristic outperformed TS/S heuristic 13 times. In addition, they obtained the same results 45 times, which include all test problems from P01 to P34. Therefore, TS/S heuristic performed better than PTS heuristic for medium and large size test problems. Also, it is important to note that the PTS heuristic ran 15 times for each test problem (i.e. five runs for each of the three solutions obtained from the construction algorithms) and TS/S heuristic only ran three times for each problem (i.e. single run for each of three solutions generated from the construction algorithms). Therefore, TS/S heuristic total run time was 1/5 of the PTS total run time.

Figure 5 Percentage improvement of TS/S heuristic over PTS heuristic



Moreover, computational experiments show that the solution qualities of the TS heuristic really depend on the qualities of the initial solutions. When comparing the construction algorithms, CAI, CAII, and CAIII obtained better solutions 2, 69, 66 times, respectively. When only considering the solutions obtained from the basic TS with CAI, CAII, and CAIII used to construct the initial solutions, TS/CAI performed best 32 times, TS/CAII

76 times, and TS/CAIII 73 times. Because the qualities of the constructed solutions by using CAII and CAIII are relatively the same, both of them are much better than the solutions constructed by CAI. It is obvious that the initial solution quality affects the performance of the TS heuristic. However, the performances of the other TS heuristics (i.e. PTS and TS/S) were not dependent on the quality of the initial solutions. It is obvious that the performance of the stochastic heuristic (i.e. SA) is not dependent on the quality of the initial solutions. Therefore, when applying these improvement heuristics to improve the solutions obtained from construction algorithms, more emphasis should be place on the improvement heuristics themselves instead of the construction algorithms.

5 Conclusion

In this paper, three TS heuristics were presented to solve the CSP. The first heuristic (i.e. TS heuristic) is a simple TS heuristic which uses static tabu list sizes and multiple initial solutions as a diversification strategy. The second heuristic (i.e. PTS heuristic) is a PTS heuristic. More specifically, the admissible move is randomly selected from a list of the top G moves (to diversify the search), unlike in TS heuristic where the best admissible move is selected. The third heuristic (i.e. TS/S) uses an intensification strategy and different diversification strategies from TS and PTS. More specifically, it uses dynamic tabu list sizes, frequency based memory, and a modification of the intensification strategy presented in Chiang and Kouvelis (1996). The proposed heuristics performed well on two data sets: data set I (generated in this paper) and data set II generated by McKendall et al. (2006). More importantly, two of the three proposed heuristics (PTS and TS/S) out-performed the heuristic (SA heuristic) available in the literature for CSP. Although the TS/S heuristic performed many more operations per iteration than the other heuristics, it obtained the best solution for 77.1% of the problems in the larger data set (data set II). In other words, the TS/S heuristic performed less iterations and obtained higher quality solutions using the same average run time. Therefore, TS/S is much more computationally efficient than the other heuristics. However, a tremendous amount of effort was required in setting the heuristic parameters for TS/S, since it has many more heuristic parameters. The following recommendations are given for future research:

- develop heuristics for CSP which require less heuristic parameters than the proposed TS/S heuristic but perform equally as well or better
- develop hybrid heuristics for CSP.

References

Chiang, W-C. and Chiang, C. (1998) 'Intelligent local search strategies for solving facility layout problems with the quadratic assignment problem formulation', *European Journal of Operational Research*, Vol. 106, pp.457–488.

Chiang, W-C. and Kouvelis, P. (1996) 'An improved tabu search heuristic for solving facility layout design problems', *Int. J. Production Research*, Vol. 34, pp.2565–2585.

Christofides, N. and Colloff, I. (1973) 'The rearrangement of items in a warehouse', *Operations Research*, Vol. 21, pp.577–589.

Frederickson, G.N. and Guan, D.J. (1992) 'Preemptive ensemble motion planning on a tree', *SIAM Journal on Computing*, Vol. 21, pp.1130–1152.

Frederickson, G.N., Hecht, M.S. and Kim, C.E. (1978) 'Approximation algorithms for some routing problems', *SIAM Journal on Computing*, Vol. 7, pp.178–193.

Glover, F. (1986) 'Future paths for integer programming and links to artificial intelligence', *Computers & Operations Research*, Vol. 13, pp.533–549.

Glover, F. (1989) 'Tabu search Part I', *ORSA Journal on Computing*, Vol. 1, pp.1900–2006.

Glover, F. (1990a) 'Tabu search Part II', *ORSA Journal on Computing*, Vol. 2, pp.4–32.

Glover, F. (1990b) 'Tabu search: A tutorial', *Interfaces*, Vol. 20, pp.74–94.

Hernandez-Perez, H. and Salazar-Gonzalez, J-J. (2004) 'Heuristics for the one-commodity pickup-and-delivery traveling salesman problem', *Transportation Science*, Vol. 38, pp.245–255.

Hunsaker, B. and Savelsbergh, M. (2002) 'Efficient feasibility testing for dial-a-ride problems', *Operations Research Letters*, Vol. 30, pp.169–173.

Laporte, G. (1992) 'The traveling salesman problem: an overview of exact and approximate algorithms', *European Journal of Operational Research*, Vol. 59, pp.231–247.

Lim, A., Rodrigues, B., Xiao, F. and Zhu, Y. (2004) 'Crane scheduling with spatial constraints', *Naval Research Logistics*, Vol. 51, pp.386–406.

McKendall Jr. A.R., Noble, J.S. and Klein, C.M. (2005) 'Simulated annealing heuristics for managing resources during planned outages at electric power plants', *Computers & Operations Research*, Vol. 32, pp.107–125.

McKendall Jr. A.R., Shang, J., Noble, J.S. and Klein, C.M. (2006) 'A simulated annealing heuristic for a crane sequencing problem', *Int. J. Industrial Engineering*, Vol. 13, pp.90–98.