4. TWO-STAGE HEURISTIC METHOD

Although the pure Traveling Salesman Problem is proved as an NP-hard problem, both the exact methods and the heuristics of TSP with excellent performances are extensively and deeply studied in the past 50 years. However, with the time window constraints and probabilistic characteristic, the problem we faced here is much more complicated than the pure TSP so that the exact methods are not appropriate. A two-stage heuristic algorithm to solve PTSPTW is introduced in this chapter. In the first stage, the request probabilities of regular customers are neglected and the goal is to construct a feasible initial TSPTW solution. This initial route is improved in the second stage. We will first introduce two local search techniques, 2-p-opt and 1-shift, which will be applied in the proposed algorithms, in Section 4.1. The detail of initial feasible route construction will be presented in Section 4.2. And in Section 4.3., we will show the heuristic of route improvement.

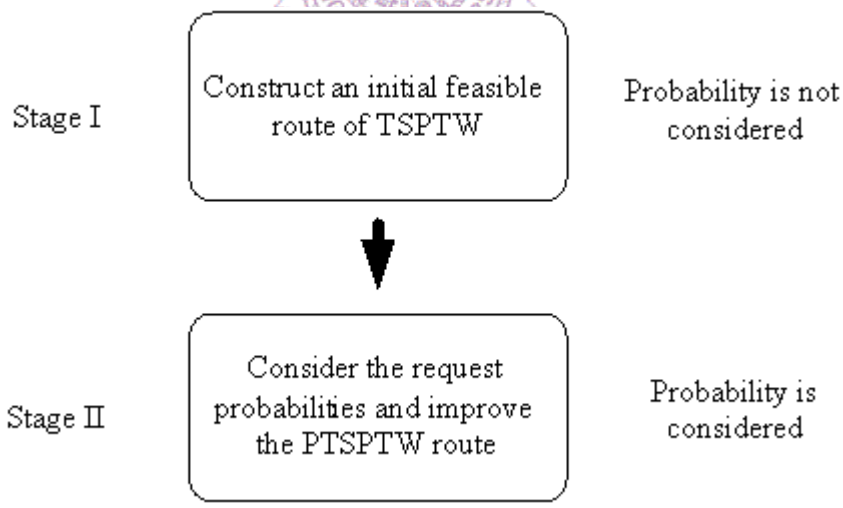


Fig. 4.1 The two-stage heuristic method

4.1 The Local Search Techniques

Here, we introduce two local search techniques called 2-p-opt and 1-shift first and these two techniques will be applied in both initial feasible route construction and *a priori* route improvement. The 2-p-opt and the 1-shift local search techniques were

both first proposed by Bertsimas and Howell [3]. In the TSP with time window constraints, the time windows of consecutive customers in a feasible route should be very close even overlap. In other words, the consecutive nodes in the problems with time constraints have stronger relationship than which without time constraints. Therefore, the 2-p-opt, which mainly reverses partial route to find neighborhoods, may perform better in TSPTW or PTSPTW. In addition, we also use the 1-shift, which is a kind of insertion techniques, to find the solution which cannot be found by reversing.

4.1.1 The 2-p-opt local search

The main idea of the 2-p-opt local search is as follows. Given a tour τ , its 2-p-opt neighborhood is the set of tours obtained by reversing a section of τ (that is, a set of consecutive nodes) and adjusting the arcs adjacent to the reversed section, as the example shown in Fig. 4.2. According to this idea, we implement the 2-p-opt local search here and the procedure is as follows:

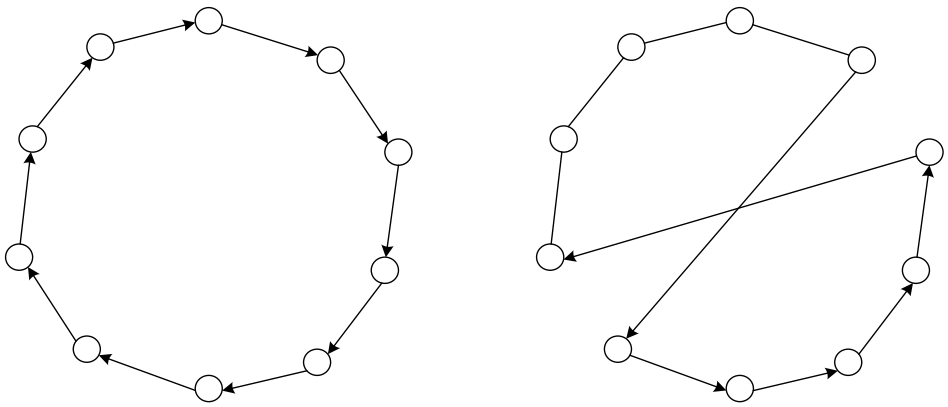


Fig. 4.2 A simple example of 2-p-opt neighborhood [6]

Step 1: Let *best* = the objective value of current best route τ_c.

Step 2: Given a positive integer *p*, choose a node *i* in τ_c randomly and reverse the part *i*, *i* + 1,…,*i* + *p*, and a new route τ is obtained.

Step 3: If *best* is better than the objective value of τ, let *best* = the objective value of τ and $\tau_c = \tau$. Let *t* = 0, where *t* represents the times of consecutive operations without improvements. Otherwise, *t* = *t* + 1.

Step 4: If t < 1000, go to step 2. Otherwise the process is terminated and τ_c is the output.

The flowchart of the 2-p-opt local search is as Fig. 4.3. In this procedure, we search for 2-p-opt neighborhoods of current best route randomly and continuously until *t* equals to 1000, which represents 1000 consecutive operations without bringing any improvements.

4.1.2 The 1-shift local search

The main idea of the 1-shift local search is as follows. Given a tour τ, its 1-shift neighborhood is the set of tours obtained by moving a node which is at position *i* to position *j* of the tour, with the intervening nodes being shifted backwards on space accordingly, as the example shown in Fig. 4.4. The procedure of our 1-shift local search is the same as 2-p-opt except Step 2:

Step 2: Choose a node *i* and another node pairs (*j*, *j* + 1) in τ_c randomly. Remove *i* and reinsert *i* into the position between *j* and *j* + 1.

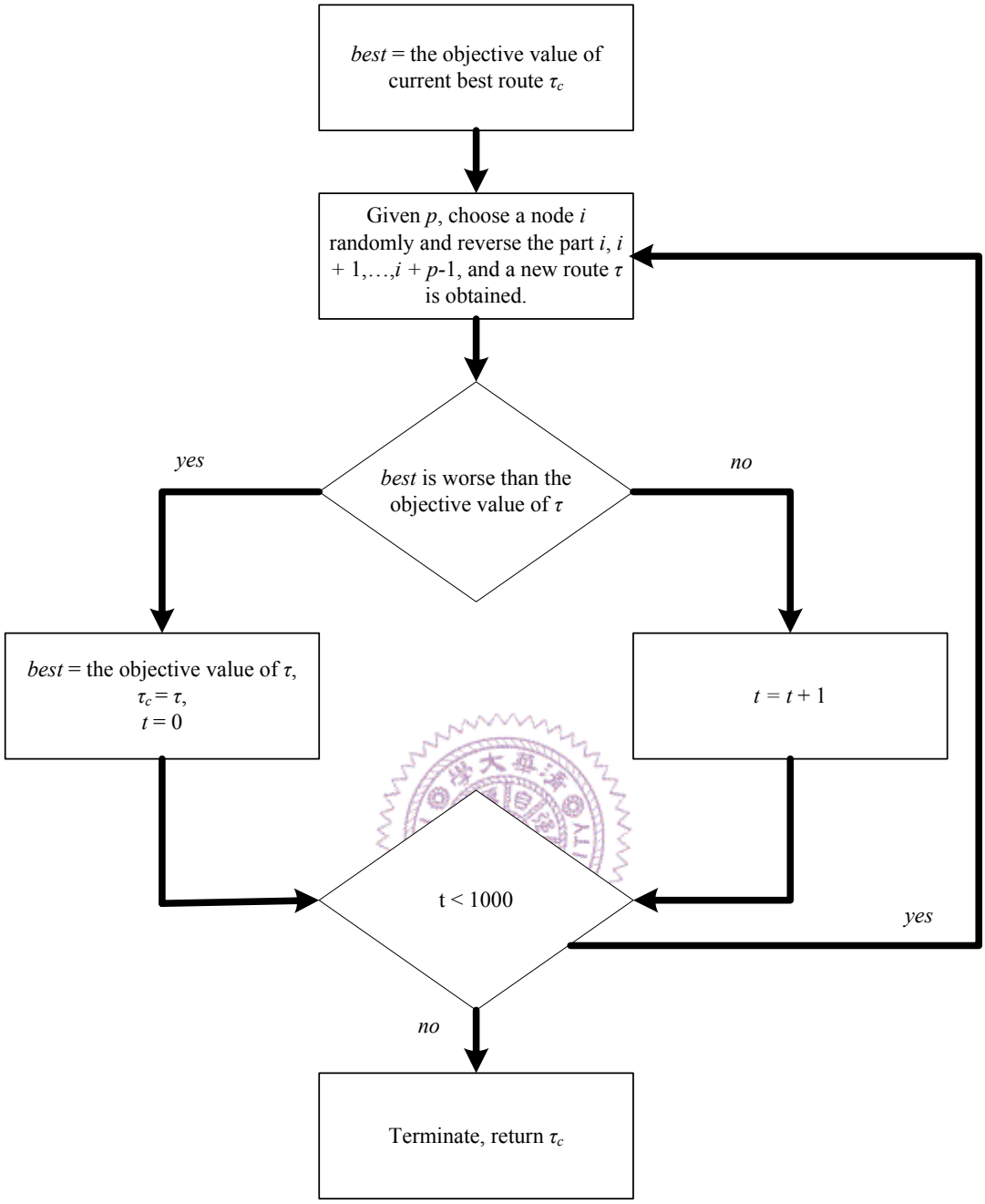


Fig. 4.3 The flowchart of the 2-p-opt local search

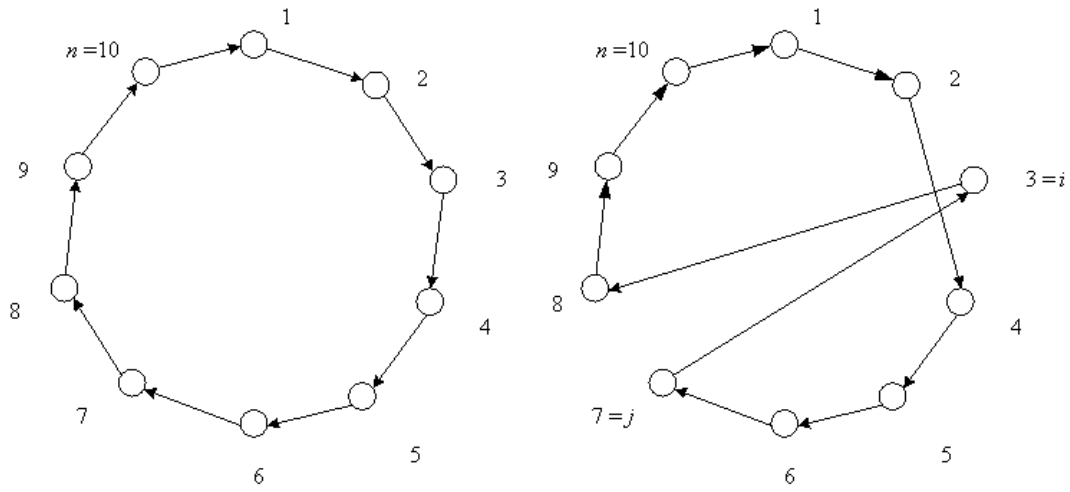


Fig. 4.4 A simple example of 1-shift neighborhood [6]

4.2 Stage I - Initial Feasible Route Construction

We propose two methods here to construct a feasible initial solution, which are the Slack-time Maximizing (STM) method and the Backtracking method. The STM method is a heuristic approach for deciding a feasible solution. It works efficiently but may not find a feasible solution in some special cases. The Backtracking method is an approach which guarantees to find a feasible TSPTW solution, but it may take much more time. Advantages of both techniques will be taken in this stage. The ideas and procedures of these two methods will be introduced as in the following sections.

4.2.1 The Slack-time Maximizing (STM) Method

The main concept of the Slack-time Maximizing (STM) method is to maximize total slack time in the route by using the 2-p-opt and the 1-shift local search techniques. In a feasible TSPTW route, the slack time of each customer must be positive. The slack time of customer *i* can be defined by

$$h_i = \begin{cases} l_i - eb_i, & l_i \geq eb_i \\ -M, & l_i < eb_i \end{cases}$$

where l_i is the late time window (deadline) of customer i, eb_i is the possible earliest service beginning time of customer i and can be computed as

$$eb_i = \max\left(eb_{i-1} + t_{i-1,i}, e_i\right)$$

where e_i is the early time window (ready time) of customer i and $t_{i-1,i}$ is the traveling time needed from customer i -1 to the consecutive customer i. M denotes a big number. If the total slack time of the route we obtained is positive, the route is a feasible one. By this idea and the 2-p-opt and 1-shift local search techniques proposed in previous section, the STM method can be implemented as follows:

Step 1: Sorting all daily and regular customers according early time windows in ascending order. They form a TSPTW route τ_c.

Step 2: Let *best* = the total slack time of τ_c and the best route $\tau_b = \tau_c$. If best > 0, terminate the process and τ_b is the output.

Step 3: Under the slack time maximization criteria, improving τ_b by series of 2-p-opt and 1-shift local search. If the route is improved, update *best* and τ_b. Stop when the total slack time cannot be improved.

Step 4: If best > 0, return the signal “feasible” and a feasible route τ_b. Otherwise return the signal “infeasible” and current infeasible route τ_b.

The flowchart of the STM method is shown as in Fig. 4.5, which is a heuristic approach to decide a feasible TSPTW solution. In most cases, this method is effective and we can obtain a feasible route quickly. However in some other cases it doesn’t work. Therefore, we develop the Backtracking method, which guarantees to obtain a feasible route, as supplement.

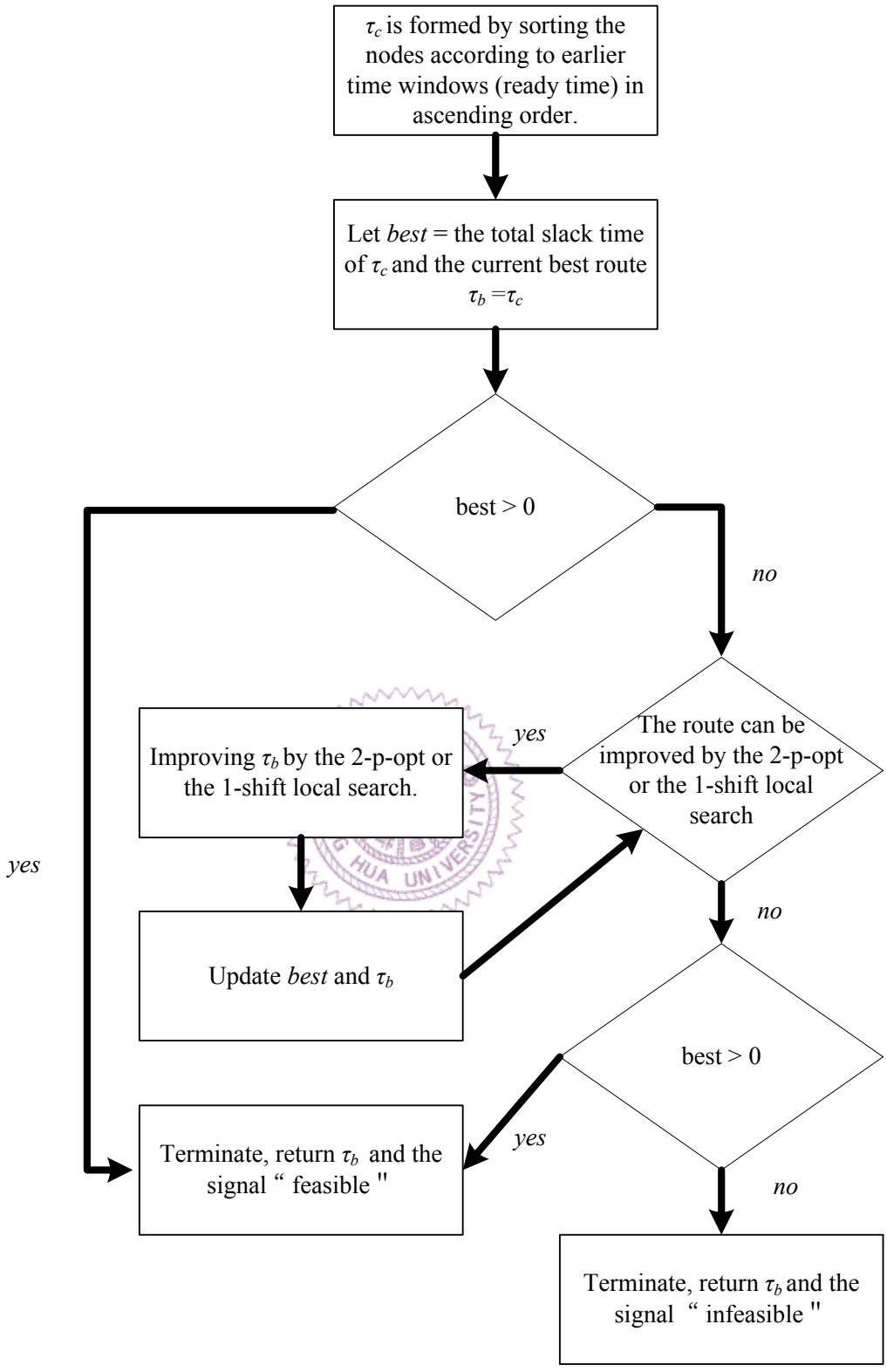


Fig. 4.5 The flowchart of the STM method

4.2.2 The Backtracking Method

The Backtracking method was often used to solve mazes or other complicated problems. The main concept is to record all successful operations in a stack. Once we reach a blind alley, call the newest record from the stack (backtracking) and go forward in another direction. The procedure of the Backtracking method we proposed is as follows and the flowchart is shown as in Fig.4.6:

Step 1: Form a customer (node) list which sequence is same as route τ_b obtained from STM method. Create a stack *routeStack* for recording successful insertion and a list *banList* for recording banned node sequences.

Step 2: Choose the first uninserted node, node *j*, from the node list.

Step 3: Inserted the node into the k_{th} possible place (counted from the end) of τ_p. If τ_p is feasible and not banned, go to step 6. Otherwise, go to Step 4.

Step 4: If all possible places in τ_p for inserting are tried, go to Step 5. Otherwise, $k = k + 1$ and go to Step 3.

Step 5: Put τ_p into *banList*. Pop out the top route of *routeStack*. Let τ_p be the top route of *routeStack*. Label all nodes which are not in τ_p as uninserted. Go to Step 2.

Step 6: Push τ_p into *routeStack*. Label node *j* as inserted. If all nodes are inserted, terminate the process and τ_p is a feasible solution. Otherwise, go to Step 2.

Here we have a simple example as shown in Fig. 4.7. Considered a five nodes instance (including depot, denote as node 0), the partial route 0 -> 1 -> 0 can be easily formed without any problem. Iteration 1 shows the operation of node 2 insertion. Node 2 is inserted into the first possible position counted from the end of τ_p and a

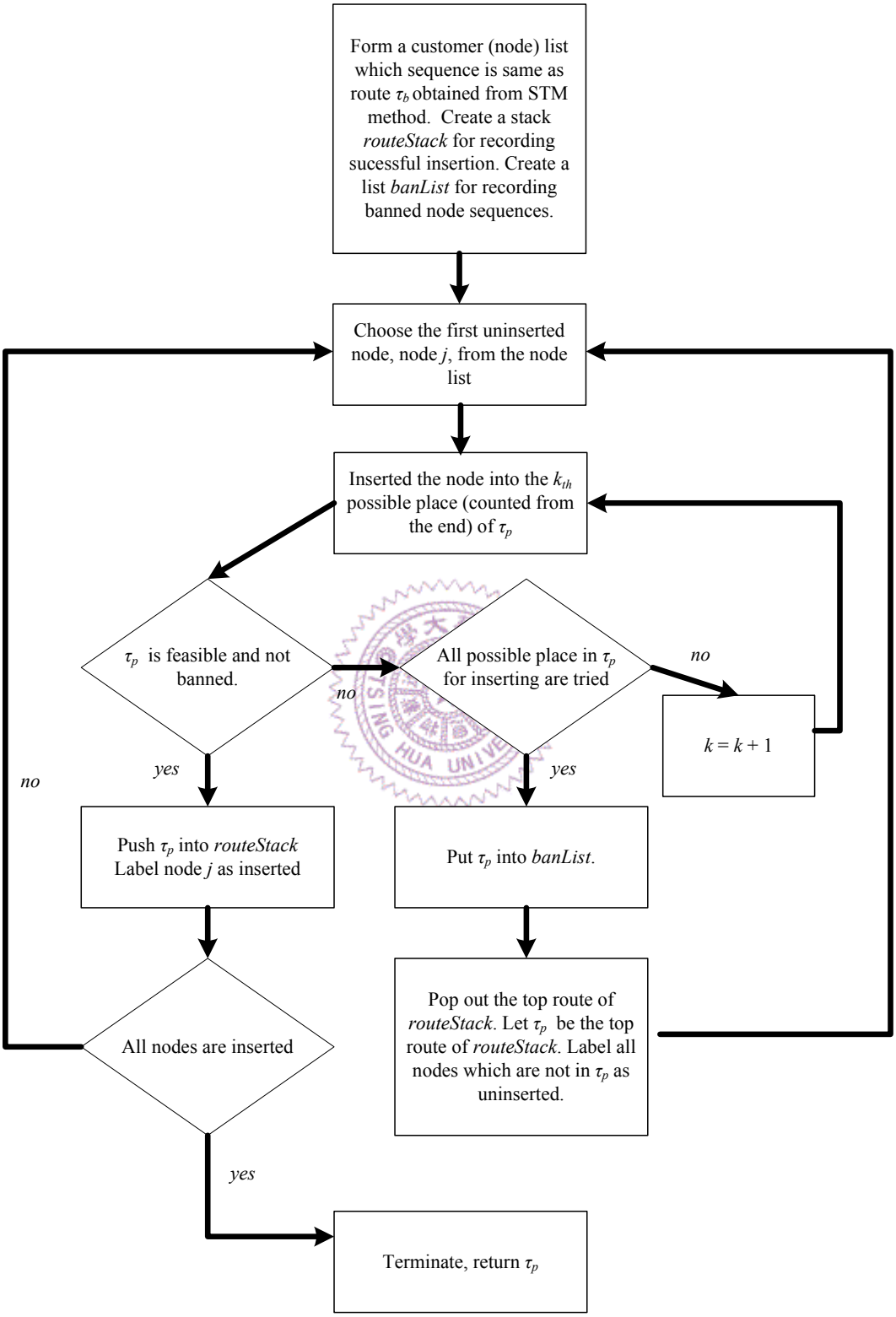


Fig. 4.6 The flowchart of the Backtracking method

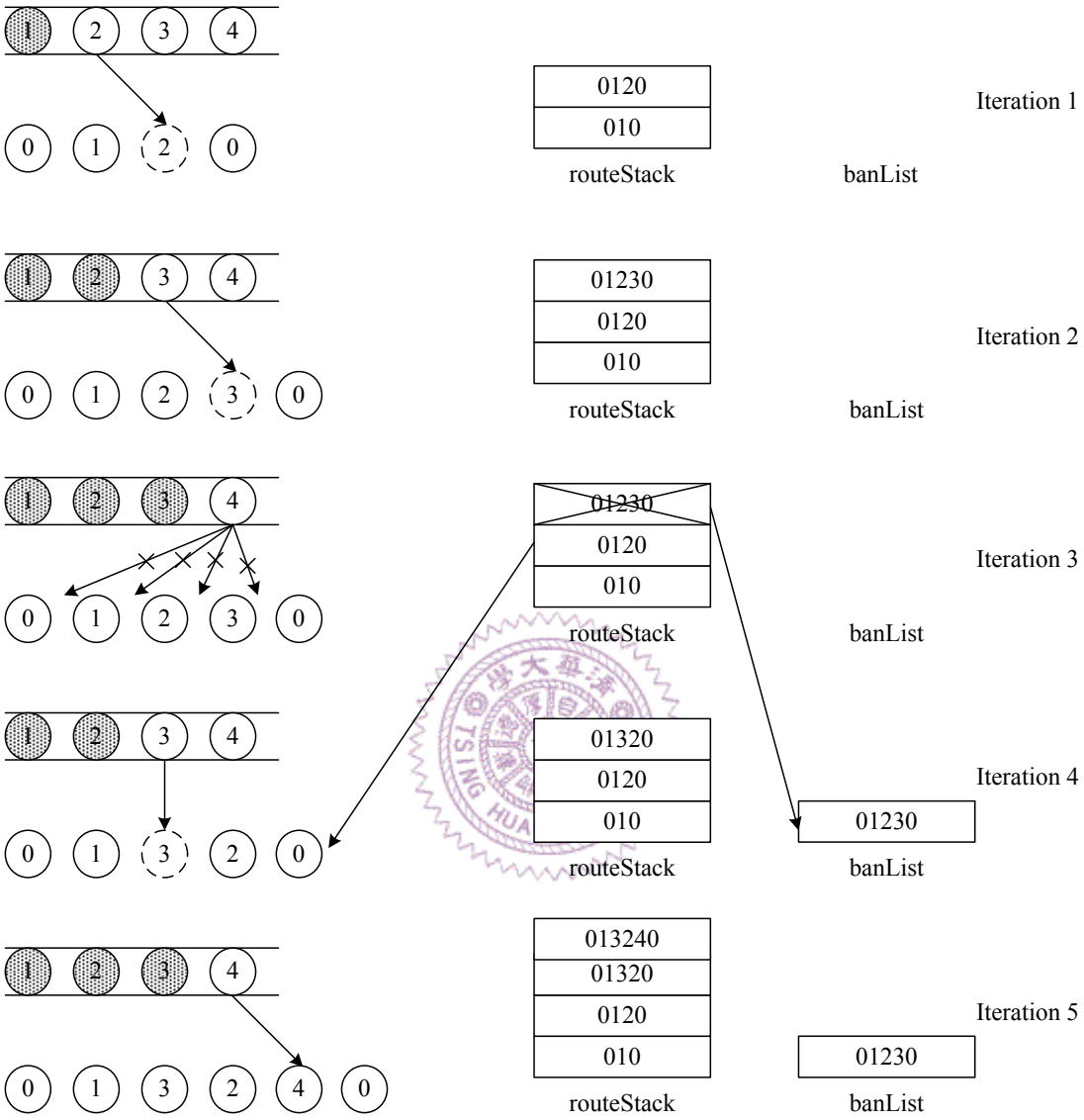
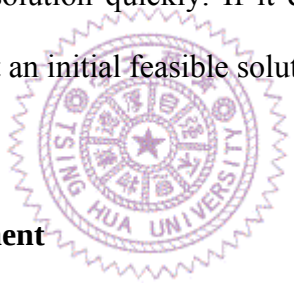


Fig. 4.7 A simple illustration of the Backtracking method

new τ_p , 0 -> 1 -> 2 -> 0, is formed. This new partial route must be push into *routeStack* for recording. Now we can find a position for inserting node 3, which is the first uninserted node. Now we can find a position for inserting node 3, which is the first uninserted node. As Iteration 2 shows, node 3 can be successfully inserted into the tail of τ_p except the deopt, and this new partial route is also pushed into *routeStack*. However, when we want to insert node 4, suppose because of the conflict

on time windows, we cannot find any position for inserting. That means, the sequence 0 -> 1-> 2-> 3-> 0 is inappropriate. Therefore, this partial route is banned and popped out from *routeStack*, as Iteration 3. Let τ_p be the new top of *routeStack*, 0 -> 1 ->2 -> 0, and node 3 is label as uninserted. At Iteration 4 node 3 is tried to insert node 3 into the partial route again. Because 0 -> 1 -> 2 -> 3 -> 0 is banned, we try a feasible partial route 0 -> 1 -> 3 -> 2-> 0 this time. And finally we can insert node 4 after node 2 and obtained a feasible complete route 0 -> 1 -> 3 -> 2 -> 4 -> 0.

The Backtracking method can assure us that a feasible initial solution is found. However, because it is a kind of branch-and-bound method, it takes us much more time. Therefore, considering advantages of both methods, we use the STM method first to find a feasible initial solution quickly. If it doesn't work, the Backtracking method is applied to assure that an initial feasible solution can be found.



4.3 StageII - Route Improvement

After obtaining a feasible initial solution, we can go forward to the next stage – Route Improvement. The request probabilities of regular customers must be considered in this stage and the TSPTW initial route is transformed into a PTSPTW one. Therefore, the way to evaluate the objective value is very different with traditional TSPTW. As mentioned in the end of Chapter 3, we can compare two different *a priori* routes by their expected weighted sum of total traveling time and average slack time. To improve the PTSPTW initial route, we develop a heuristic which core is also formed by the 2-p-opt and the 1-shift local search. The concept of this improvement heuristic is quite similar as the STM method mentioned in previous section. The 2-p-opt and the 1-shift local search techniques are used continuously to reduce the expected weighted sum of total traveling time and average slack time as

possible until the *a priori* route cannot be improved. The procedure of the improvement heuristic is as follows:

Step 1: Let τ_c be the *a priori* route obtained from the first stage. Let current best route $\tau_b = \tau_c$ and current best objective value *best* be the objective value of τ_b.

Step 2: Improving τ_b by series of the 2-p-opt and the 1-shift local search. If the *a priori* route is improved, update *best* and τ_b. Stop when the objective value cannot be improved by any local search.

Step 3: Return τ_b. Terminate the process.

The flowchart of the Stage II - route improvement heuristic is shown as in Fig. 4.8. By this two-stage algorithm, we can obtain a near-optimum PTSPTW *a priori* route.

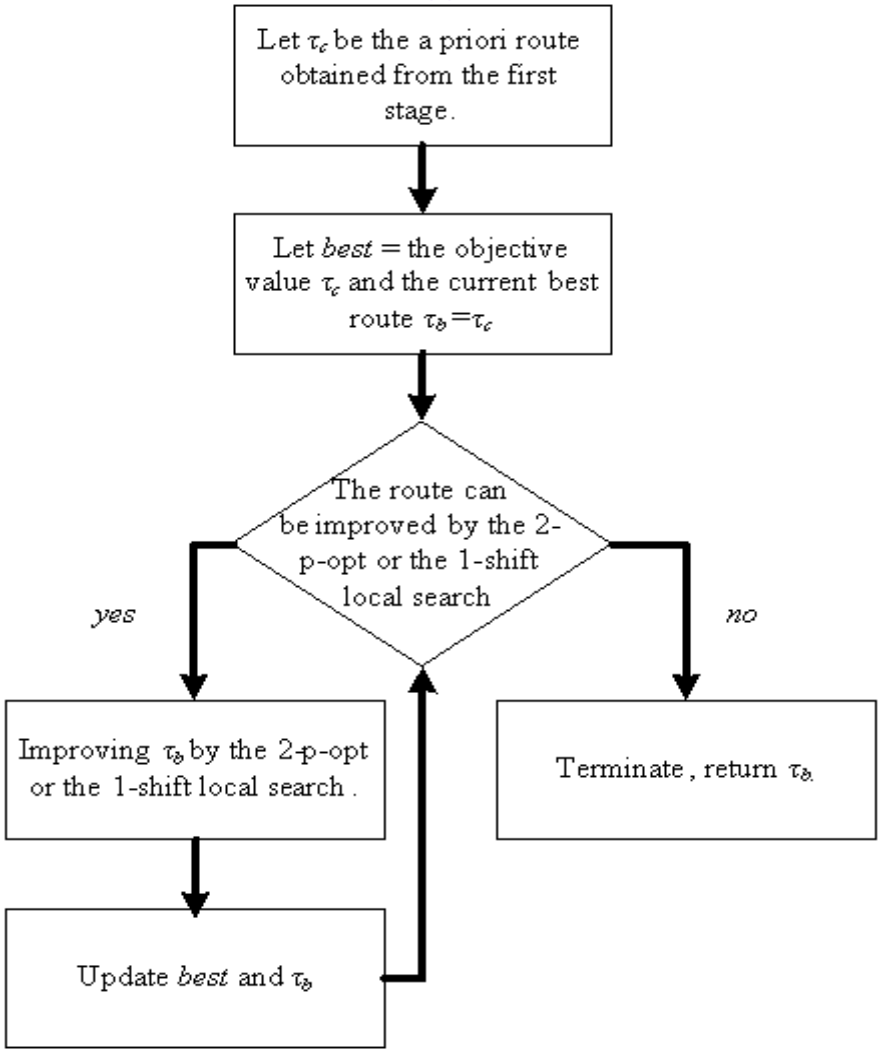


Fig. 4.8 The flowchart of route improvement (Stage II) heuristic