Chapter 5

A Tabu Search solution algorithm

The TS examines a trajectory sequence of solutions and moves to the best neighbor of the current solution. To avoid cycling, solutions that were recently examined are forbidden, or tabu, for a number of iterations (Gendreau et al., 2002). Section 3.3.1 reviews the basic structure of the TS.

Taillard (1993) introduces a feature whereby the main problem is decomposed into independent subproblems so that the algorithm can be parallelized on multiple processors. Each subproblem is solved on a different processor before the tours are grouped together to construct a solution to the original problem. The new solution is then decomposed, and the process repeats itself for a given number of times. A random selection of components in the decomposition process ensures the algorithm produces different solutions from one execution to the next. In this thesis an approach similar to that of Taillard (1993) and Rochat and Taillard (1995) is followed, albeit on a single processor. The approach can be parallelized through the coding structure in future research, but recent software technology, i.e. cluster scheduling such as the *MATLAB Distributed Computing* system, provides the software the ability to automatically determine which segment of an algorithm can be parallelized on multiple clustered processors without adapting the code.

The chapter starts with a brief discussion of the main elements of a TS algorithm, followed by the TS proposed in this thesis, and a detailed discussion of each phase of the TS. The chapter concludes with an analysis of the algorithm's results for problems based on integrated data sets of Solomon (1987), Homberger and Gehring (1999), and Liu and Shen (1999a,b).

5.1 Elements of the tabu algorithm

Tabu list A list of the last few moves (or solutions). The *memory* of moves can be recency or frequency-based. Short-term recency-based memory forbids cycling around a local neighborhood in the solution space through setting the last T moves as Tabu. Recently made moves are stored in a mechanism that is referred to as the *Tabu-Move* list. The number of moves in the list is determined by the tabu list size, denoted by T. The list operates on a first-in-first-out principle. Other recency information that is stored in the Tabu list is the solution configurations. The larger the value of T, the longer the moves and solutions stay tabu. The *Tabu-Solution* list is a set of solutions that have been created recently by exchanging segments between routes. The solutions are coded into an integer string. The total cost of the solution is also attached to the string.

Long-term frequency-based memory allows searches to be conducted in the most promising neighborhoods. The frequency-based memory provides additional information of how many times a tabu move have been attempted. To alleviate time and memory requirements, it is customary to record an attribute of a tabu solution, and not the solution itself.

Candidate list TS makes use of a candidate list that provides a list of moves to evaluate. One move of the candidate list is chosen to proceed with the search. The candidate list plays an important role in the performance of TS.

Intensification and diversification Two memory-based strategies that form a fundamental principle of TS. Gendreau (2003) claims diversification to be the single-most important issue in designing a TS. With the use of the intensification strategy regions around attractive solutions are more thoroughly searched, and typically operates by restarting a search from a solution previously found to yield good results. The restart is achieved through the candidate list representing attractive regions. Diversification, on the other hand, encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from previous solutions. The probabilistic diversification and intensification introduced by Rochat and Taillard (1995) is also referred to as the Adaptive Memory Procedure (AMP).

Penalized objective function The objective function of a solution s is denoted by $f_1(s)$ and is calculated by (5.1) as the sum of the travel times of all routes and tours, and the total lateness at all customers (Ichoua et al., 2003).

$$f_1(s) = \sum_{\text{Tours}} \sum_{\text{Routes}} t + \sum_{\text{Customers}} \alpha_i y_i \tag{5.1}$$

In the calculation α_i denotes the lateness penalty for customer i, while $y_i = \max\{0, a_i - l_i\}$. The actual arrival time at customer i is denoted by a_i, while l_i denotes the latest allowed arrival time at customer i. The design of the algorithm ensures that $a_i \leq l_i + L_i^{\max}$, where $L_i^{\max}$ is the maximum allowed lateness at customer i. The objective function is artificially adapted to incorporate a significant penalty for any unrouted customers, referred to as *orphans*. The artificial objective function, $f_2(s)$, is expressed in (5.2),

$$f_2(s) = f_1(s) + \beta o \tag{5.2}$$

where β is a nonnegative penalty factor, and o the number of orphans in the final solution. Orphans are only created if the time window of the customer is completely incompatible with that of the depot, even if it is serviced by a dedicated vehicle.

Stopping criteria The search is terminated once a preset maximum number of iterations of the main TS algorithm have been reached. An alternative stopping criteria could be a predetermined number of attempts being made to set the same solution in the Tabu-Solution list as the new current solution. This indicates that the search has been caught in a local optimum, hence terminating the search.

5.2 Tabu algorithm

The phased approach of the TS algorithm, similar to the implementation of Taillard et al. (1997) and Gendreau et al. (1999), is illustrated in Algorithm 5.1. Data structures are indicated with sans serif font, while functional routines are indicated with `typewriter font`.

5.2.1 Initialization

The initial solution algorithm proposed in Chapter 4 forms the basis of the initialization phase, but generates only a single initial solution, s. As I, preferably *different*, initial solutions are required, the routine in Algorithm 5.2 is proposed. For each initial solution required, a random node $I_i^\star$ is identified and removed from the problem set P. The remaining nodes in P' are used to create an initial solution using the improved initial solution algorithm proposed in Chapter 4. After the nodes in P' have been routed, the identified node $I_i^\star$ is reinserted into the first feasible position. The result is a set of initial solutions

Algorithm 5.1: Tabu Search (TS) Overview

```
   Input: stopping criteria
   Input: Adaptive Memory size, $M$
1  begin Initialization (Section 5.2.1)
2      construct $I$ unique initial solutions $\boldsymbol{s} = \{s_1, s_2, \ldots, s_I\}$
3      $\hat{x} \leftarrow \min_{i \in \{1,\ldots,I\}} \{s_i\}$
4      decompose $\boldsymbol{s}$ into independent tour set $T$
5      store $M$ best tours of $T \cup$(Adaptive Memory) in the Adaptive Memory
6  end
7  begin Optimization (Section 5.2.2)
8      while stopping criteria is not met do
9          construct a biased solution, $x$ from the tours in Adaptive Memory
10         $x^{\text{current}} \leftarrow x$
11         for $W$ iterations do
12             $x^\star \leftarrow$ locally optimized $x^{\text{current}}$
13             $x^{\text{current}} \leftarrow x^\star$
14             if $x^{current} < \hat{x}$ then
15                 $\hat{x} \leftarrow x^{\text{current}}$
16             endif
17         endfor
18     endw
19     decompose $x^{\text{current}}$ into independent tour set $T$
20     store $M$ best tours of $T \cup$(Adaptive Memory) in the Adaptive Memory
21 end
22 report incumbent $\hat{x}$
```

Algorithm 5.2: Tabu Search (TS) Initialization

```
    Input: Problem set $P$, with $|P| = n$ nodes
    Input: Number of initial solutions required, $I$
1   identify $I^\star \subset P$, a randomly identified subset with $I$ nodes from problem set ;
2   foreach $I_i^\star \in I^\star$ do
3       $P' \leftarrow P \backslash \{I_i^\star\}$;
4       find initial solution $s$ by executing Initial solution heuristic with $P'$ ;
5       re-insert $I_i^\star$ into initial solution to create $s_i$
6   endfch
```

$\boldsymbol{s} = \{s_1, s_2, \ldots, s_I\}$. Each initial solution's tours are stored in the adaptive memory, and associated with it the objective function value of the initial solution from which the tour originates. All tours consisting of only a single node are removed from the adaptive memory.

5.2.2 Optimization

The TS optimization routine listed in Algorithm 5.3 terminates after executing a predefined number of local optimization iterations, denoted by $I^{\max}$. A partially constructed tour is created through iteratively selecting tours from the adaptive memory, and removing all tours from the adaptive memory that share nodes with the selected tour. The probability of selecting any tour is based on the objective function associated with the tour, which in turn is taken from the solution from which the tour originates. Glover (1990) notes that the use of probabilities, based on past performance, as an underlying measure of randomization yields efficient and effective means of diversification. The better a solution, the higher the probability of selecting a tour from that solution. Once a tour is selected from the adaptive memory, all tours sharing nodes with the selected tour are removed from memory. Removing tours from the adaptive memory ensures each node is represented only once in the partially constructed tour. The selection of tours from the adaptive memory, and the removal of tours with common nodes, is repeated until no more tours remain in the adaptive memory. As not all nodes are represented, the partially constructed tour denoted by s, is completed by inserting the remaining unrouted nodes into feasible positions of s. The resulting tour, denoted by $s^\star$, is achieved through either identifying positions on a current route, creating a new route on a current tour, or creating a new tour with its associated vehicle.

Algorithm 5.3: Tabu Search (TS) Optimization

```
   Input: Incumbent solution, $\hat{x}$
   Input: Iteration limit for local optimization, $I^{\max}$
   Input: Frequency parameter, $\zeta$
1  $s = \{\cdot\}$
2  assign set of tours, $A \leftarrow$ Adaptive Memory
3  repeat
4      select $a \in A$
5      $s \leftarrow s \cup a$
6      $A \leftarrow A \ominus (a \cap A)$
7  until $A = \{\cdot\}$
8  $s^{\star} \leftarrow s \oplus (\{1, 2, \ldots, N\} \ominus s)$
9  $i \leftarrow 0$
10 repeat
11     $i \leftarrow i + 1$
12     if $\left\lfloor \frac{i}{\zeta} \right\rfloor = \frac{i}{\zeta}$ then
13         exchange heuristic $j = \{1, 2\}$
14     else
15         select exchange heuristic $j \in \{1, 2\}$ with probability $p_j$
16     endif
17     ${s'}_j \leftarrow e_j(s^{\star})$
18     $s' \leftarrow \min_j \left\{ s'_j \right\}$
19     $x' \leftarrow f(s')$
20     if $x' < \hat{x}$ then
21         $\hat{s} \leftarrow s'$
22         $s^{\star} \leftarrow s'$
23     endif
24 until $i > I^{\max}$
```

Two exchange operators are considered. The first operator removes a randomly selected node from one tour and inserts the node into the best possible position in another tour that has the same vehicle type. The second operator also removes a randomly selected node from an origin tour, but selects the best insertion position for the node on a tour having a different vehicle type than the origin tour.

Initially the probability of selecting either of the operators is equal. A frequency parameter, ζ, ensures that every ζ iterations *both* operators are used to create perturbations. The probability of the operator producing the best solution is then increased relative to its current probability. Consider, during a general iteration, the first operator having a weight of $\alpha = 30$ and the second operator having a weight of $\beta = 60$. If both operators are executed, and the first operator yields a better solution, its weight will be increased by a factor γ. In this thesis γ is arbitrarily set to 2. The new probability of selecting the first operator is

$$\begin{aligned} p_1 &= \frac{\gamma\alpha}{\gamma\alpha+\beta} \\ &= \frac{2\times 30}{2\times 30+60} \\ &= 0.5\dot{0}, \end{aligned}$$

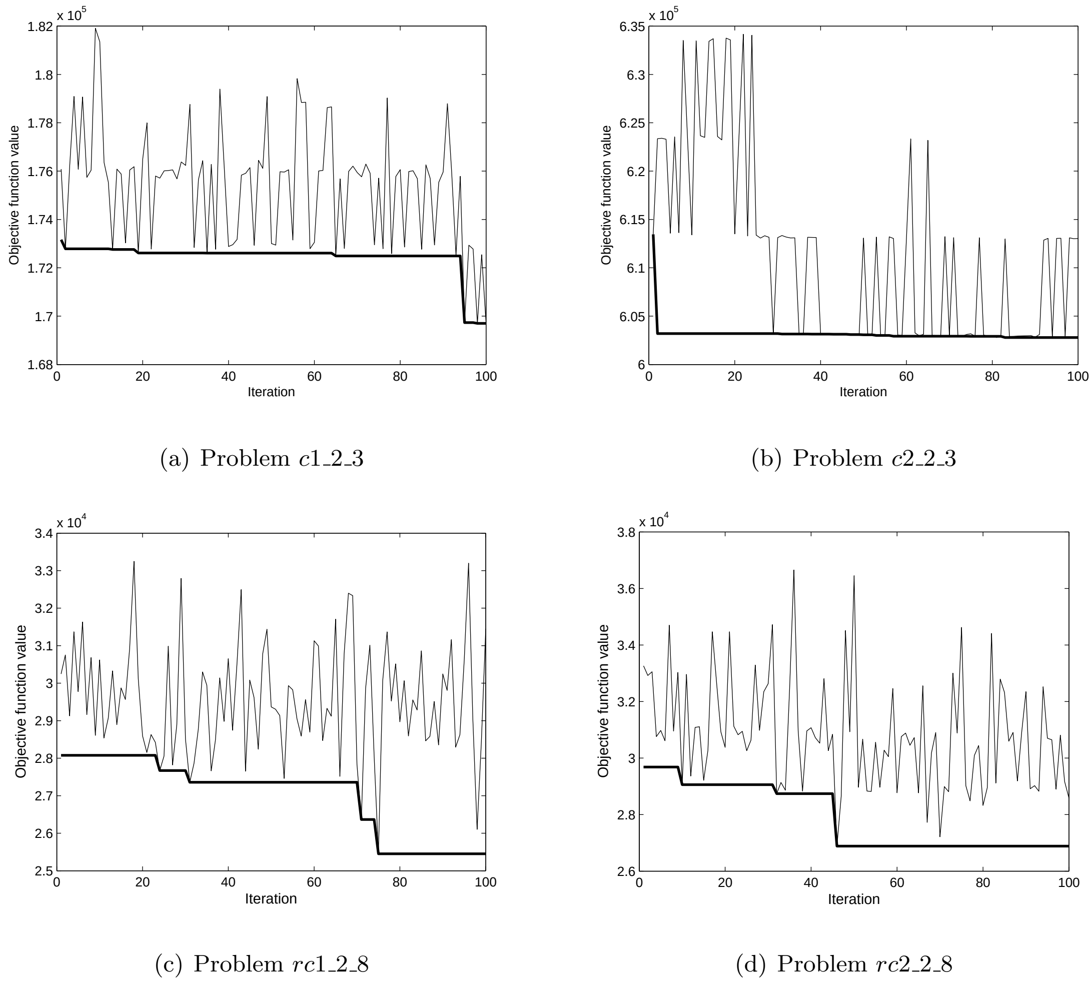
and the probability of selecting the second operator is calculated as

$$p_2 = 1 - p_1.$$

5.3 Results and analysis

The TS algorithm proposed in this thesis contains a random component similar to the algorithm proposed by Rochat and Taillard (1995). This means that two runs of the algorithm will generally produce two different solutions. Figure 5.1 provides graphs for a random selection of problems. Each graph indicates the iteration number on the x-axis, while the objective function value is represented on the y-axis. The thinner of the two lines on each graph represent the actual objective function value of the solution for the given iteration, while the thick line represents the incumbent — the best solution found thus far, at that iteration.

It is noticeable that the incumbent for the first iteration is frequently lower than the actual iteration value. This is the result of the incumbent being represented by one of the ten initial solutions created for the TS, whereas the first iteration's solution is created through the solution-building mechanism that selects tours from the adaptive memory. The incumbent,



(a) Problem $c1\_2\_3$ (b) Problem $c2\_2\_3$

(c) Problem $rc1\_2\_8$ (d) Problem $rc2\_2\_8$

Figure 5.1: Selection of TS result graphs

furthermore, is never improved by more than 10% over the 100 iterations, reflecting on the high quality initial solution proposed in Chapter 4.

Because of the randomness inherent in the structure of the proposed TS, the results presented in Appendix B sees four independent runs executed, with Tables B.1(a) through B.1(f) providing the objective function values for each of the runs, as well as the average objective function value obtained. The last column of the result tables provide the average time (in seconds) required to obtain a solution. The average time is provided under the assumption that time-dependent travel time matrices are not available, and that such matrices have to be established once, and adhere to the triangular inequality

$$t_{ik} + t_{kj} \geq t_{ij} \qquad \forall i,j,k \in \{1,2,\ldots,N\}. \tag{5.3}$$

Although Toth and Vigo (2002b) interpret the triangular inequality as being inconvenient to deviate from the direct link between nodes i and j, it may be practical to adjust the link from i to j to rather pass via node k without actually visiting node k. This occurs when the direct link is heavily congested during peak times. Adjusting the route selection in combination with time-dependent travel times are highly dependent on an accurate GIS.

5.4 Conclusion

A Tabu Search (TS) algorithm is proposed that generates a number of initial solutions as input, from where tours are added to an Adaptive Memory Procedure (AMP). During each consecutive iteration, tours are selected from the AMP in a biassed manner to construct a new solution. Non-tabu, feasible solutions are generated in an attempt to escape local minima.

The algorithm is coded in *MATLAB*, and tested on 60 benchmark data sets adapted from literature. The sets are adapted to accommodate multiple routes per tour, as well as a heterogeneous fleet in an environment where time dependent travel times occur. The results are promising, yielding solutions between 670 and 4762 seconds on a standard *Intel Pentium Centrino* laptop computer with a 1.5GHz processor and 512MB of RAM. Four independent runs are executed for each of the 60 problems. The Absolute Mean Deviation (AMD) of the solution quality between the 240 runs is 3.6%, indicating an algorithm that produces consistent solutions between runs.

In the next chapter, the GA is investigated as an alternative to the TS.