CAPACITATED VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

by

EKIM OZAYDIN

Submitted to the Graduate School of Engineering and Natural Sciences

in partial fulfillment of

the requirements for the degree of

Master of Science

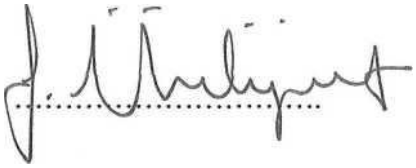
Sabancı University

July 2003

CAPACITATED VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

APPROVED BY:

Assis. Prof. Dr. Tonguç Ünlüyurt
(Thesis Advisor)



Assis. Prof. Dr. Berrin Yanıkoğlu

Assis. Pmf. Dr. Bülent Çatay

DATE OF APPROVAL: 07.04.2003

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Assis. Prof Dr. Tonguç Ünlüyurt for his encouragement, motivation and considerable time he spent from beginning to end of my thesis.

I thank to graduate committee members of my thesis. Assis. Prof. Dr. Bülent Çatay and Assis. Prof Dr. Berrin Yanıkoğlu for their worthwhile suggestions and remarks.

I would also thank to many people who have encouraged me during my study. Especially my office-mates Murat Kılıç, N. Mehmet Gökhan, G. Arzu İnal, A. Volkan Vural, O. Serdar Basmacı, and H. Murat Özdemir for the enjoyable days spent together.

I am grateful to all the graduate students, especially şilan Hun and Bilge Küçük, and faculty members for providing a peaceful environment in the department.

I wish to thank Ömer Karagoz for his support and concern. I'm so fortunate for knowing him.

Lastly to Niliifer Aydmoglu and Burcu Anıl ... For their endless friendship that makes my life that beautiful.

ABSTRACT

Vehicle Routing Problem with Time Windows (VRPTW) is an extension of the Capacitated Vehicle Routing Problem. The objective is to design optimal routes that satisfy all of the constraints.

In this study, a linear IP model and hybrid heuristics for the VRPTW are proposed. The objective function considered in the model is the total distance traveled by all vehicles. Vehicles are identical, capacities of the vehicles are finite and the time window constraints are assumed to be strict.

The proposed hybrid heuristics are combined by two parts. The first part, which has both parallel and sequential versions, finds an initial solution. Both parallel and sequential initial solution algorithms are based on the idea of clustering the customers while doing the insertion. Second part is an improvement heuristic, which is a combination of three procedures: *Inter-route exchanges, inter-route moves* and *intra-route exchanges*. In the proposed heuristics, these operators are used nested with each other. There are two improvement heuristics proposed that use these operators in different ways. The improvement algorithms are supported with a restart mechanism called *diversification* in order to escape the local optima and widen the search space. In this study, two diversification methods are proposed.

The hybrid algorithms in this study are the combinations of the initial solution, improvement and diversification methods proposed.

The algorithms have been tested on the 56 benchmark problem instances of Solomon (1987), which were used widely in the literature. The hybrid algorithms are proven to give better results when compared to not only some known metaheuristics, but also to the best known results in the literature.

ÖZET

Zaman Kısıtlı Araç Rotalama Problemi, Araç Rotalama Problemi'nin bir uzantısıdır. Problemde amaç, tüm kısıtları sağlayan optimal rotalar oluşturmaktır.

Bu çalışmada Zaman Kısıtlı Araç Rotalama Problemi için bir doğrusal tamsayılı programlama modeli ve problemin çözümü için hibrid sezgisel yaklaşımlar önerilmiştir. Modeldeki amaç fonksiyonu, araçlar tarafından kat edilen toplam mesafenin en küçüklenmesidir. Tüm araçlar aynı özelliklere sahiptir ve araçların kapasiteleri göz önünde bulundurulmaktadır.

Önerilen sezgisel algoritmalar iki bölümden oluşmaktadır. Birinci bölüm daha sonra geliştirilebilmek üzere bir başlangıç çözümü oluşturmaya yöneliktir. Başlangıç çözümü algoritmasının paralel ve sıralı versiyonları önerilmiştir. Her iki yaklaşım da müşterileri rotalara atarken onları gruplandırma esasına dayanmaktadır.

İkinci kısım ise üç farklı prosedürü içeren bir çözüm geliştirme algoritmasıdır. Bu üç prosedür *rotalar arası değişim, rotalar arası taşıma* ve *rota içi değişim'dir.* Önerilen sezgisel algoritmalarda bu prosedürler iç içe, birbirine bağlanmış şekilde kullanılmıştır. Bu üç prosedürü farklı şekillerde birbirine bağlayarak kullanan iki farklı yerel arama algoritması geliştirilmiştir.

Ayrıca, çözüm geliştirme algoritmaları *dağıtma* adlı yeniden başlatma algoritmasıyla desteklenmiştir. Çalışmada iki farklı dağıtma metodu önerilmiştir. Bunlardan biri çözümü maliyet değişimini göz önünde bulundurmadan bozmakta, diğeri ise daha kötü çözümlere sabit ve belirli bir olasılıkla gitmektedir.

Çalışmada önerilen hibrid sezgisel algoritmalar, tüm başlangıç çözümü, çözüm geliştirme ve dağıtma algoritmalarının farklı kombinasyonlarıdır.

Algoritmalar, Solomon'un 1987 yılında oluşturduğu ve araç rotalama için geliştirilen algoritmaların karşılaştırılmasında çok yaygın olarak kullanılan 56 problem ile test edilmiştir. Hibrid algoritmalar hem bilinen bazı sezgisel-ötesi yaklaşımlarla, hem de problemlerin literatürdeki bilinen en iyi çözümleri ile karşılaştırıldığında, genel anlamda iyi sonuçlar vermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The problem of transportation of people, goods or information is commonly denoted as routing problem. Routing problems are not restricted to the logistics companies itself but also others. Optimization of the transportation has become an important issue, as the world economy turns more and more global.

The basic routing problem is Traveling Salesman Problem (TSP) where a number of cities have to be visited by a salesman who must return to the same city where he started. The Vehicle Routing Problem (VRP) is the *m-TSV* (TSP with *m* vehicles) where a demand is associated with each city and the system has various constraints. VRP was first formulated by Dantzig and Ramser in 1959. The problem can be defined as *"the design of a set of minimum-cost vehicle routes, originating and terminating at a central depot, for a fleet of vehicles that services a set of customers with known demand."* (Dantzig and Ramser, 1959). VRP is concerned with the determination of the optimal routes by a fleet of vehicles, based at one or more depots, to serve a set of customers (Toth and Vigo, 2002).

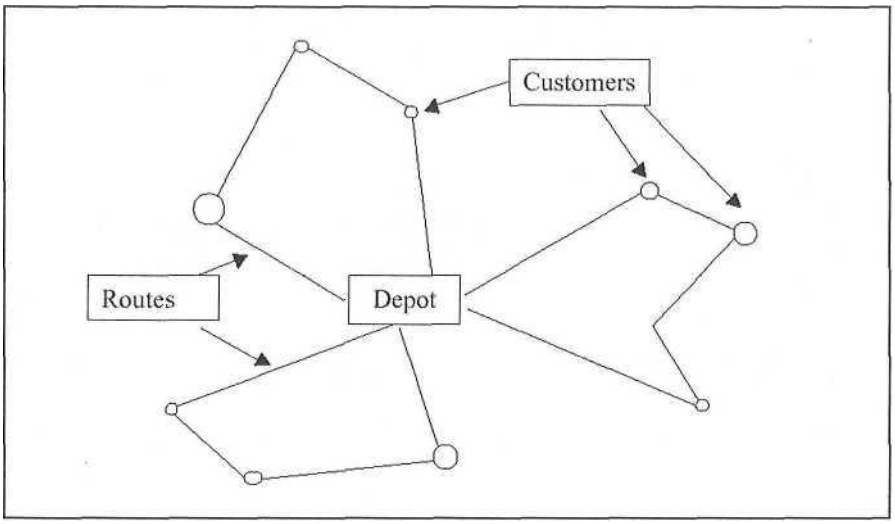


Figure 1.1 General representation of the Vehicle Routing Problem

The problem is to plan least costly routes for the vehicles with the given constraints. The cost definition may vary from one case to another, but in the most basic case, the total distance traveled by the vehicles is considered as the objective function.

In a pure routing problem, there is only a geographic component, while more realistic problems also include a scheduling part, that is, a time component. The problems studied are often simpler than real-life problems. But even though a number of constraints are left out, the research models typically model the basic properties and there by provide the core results used in the analysis and implementation of systems in real-life problems.

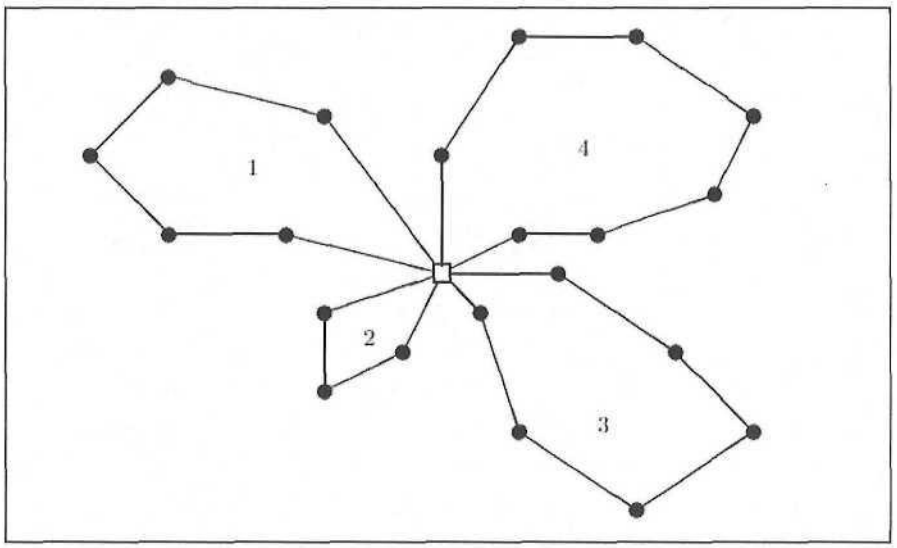


Figure 1.2 A typical solution to a VRP instance (4 routes). The square denotes the depot and the nodes are the customers.

In the literature, VRP is commonly formulated with capacity constraints, so the Vehicle Routing Problem generally has the same meaning with Capacitated Vehicle Routing Problem (CVRP). Capacitated Vehicle Routing Problem with Time Windows (VRPTW) has been a research area that has attracted many researchers in the last 25 years.

As a generalization of the VRP, the VRPTW includes time windows defined for each customer and the depot. The time window constraint may occur due to product restrictions (such as usable dates of products) or some production constraints, or it may be forced by the customer because of her inventory policy. In addition to the time windows for the customers, there are travel times between all customers or a customer and the depot. These travel times are associated with the distances. There may exist service times for the customers. The vehicles have to serve the customers within a predefined time window at minimum cost. A vehicle is allowed to arrive at a customer before the beginning of the time window, but it must wait until the time the window "opens." It is not allowed for a vehicle to arrive after the time window ends. With the given constraints, the VRPTW is proven to be NP-Hard. (Kohl, 1995)

There have been many papers proposing exact algorithms for solving the VRPTW. These algorithms are based on three methods: dynamic programming, Lagrangean relaxation and column generation. The first exact algorithm was proposed by Kolen *et al.* in 1987 where they used dynamic programming. Following this, there had been many papers published that use dynamic programming or other methods, but since the VRPTW is known to be NP-hard, exact algorithms are not capable of solving problems for big numbers of customers.

Non-exact algorithms (heuristics) are thought to be more efficient for complex VRPTW problems and have become very attractive and popular for researchers. The non-exact algorithms in the literature are of two types: construction algorithms, which are used for building an initial solution or initial solutions for the problem, and improvement algorithms which are used to improve the initial solution(s) found. Classical local search techniques have been used as an improvement technique for many years, but in the recent years, another type of non-exact algorithms has become very popular for solving the VRPTW: metaheuristics. Metaheuristic approaches are mostly a combination of construction and improvement algorithms and are very efficient for escaping local optimum values while searching for better solutions. Classical improvement algorithms are not very effective at escaping the local optima unless they include a restart mechanism. Since metaheuristics are designed to overcome this situation, they give competitive results. That is why the recent publications are all based on metaheuristic approaches such as genetic algorithms, tabu search, simulated annealing.

In this thesis, hybrid heuristics that use different initial solution and improvement algorithms are proposed. Since it is a known fact that the algorithm must not stick into local optima, a methodology called "diversification" is used.

Chapter 2 includes a comprehensive literature review on the VRPTW where a detailed definition of the problem is given and major studies on this subject are explained.

Chapter 3 describes the proposed linear integer programming model, initial solution and improvement algorithms, and the methodology used for escaping local optimum values. The IP models in the literature are in non-linear forms in general. In this thesis, a new representation of the constraints is used where they are formulated linearly.

Chapter 4 reports the computational study on the proposed algorithms where the results of the hybrid heuristics that are generated from the proposed initial solution and improvement algorithms are illustrated and a benchmark study between the proposed heuristics, some other competing heuristics and the best known results in the literature based on the test problems of Solomon (1987).

A conclusion of the study is provided in the last chapter that includes the interpretations and the summary of the study and results achieved.

2. LITERATURE REVIEW

In this chapter, first we will give an example of a linear integer programming formulation of the VRP. Then we will describe variations of the VRP problem. Finally, a detailed review of the VRPTW from the literature is given.

2.1. Vehicle Routing Problem Formulation

The VRP is described by a set of homogenous vehicles (denoted by V), a set of customers C and a directed graph *G.* The graph consists of | *C*| *+2* vertices where the customers are denoted *1,2,...,n* and the depot is represented by the vertex *0* and *n+1.* The set of vertices *0,1,...,n+1* is denoted as *N.* The set of arcs (denoted by *A)* represents connections between the depot and the customers and among the customers. No arc terminates at vertex *0* and no arc originates from vertex *n+1.* With each arc *(i,j),* where /'≠ *j,* we associate a cost (distance) C_{ij}

Each vehicle has a capacity *q* and each customer *i* has a demand d_i. It is assumed that *q,* d_i ,c_{ij} are non-negative integers.

For each arc *(i,j),* where *i*≠*j; i*≠*n* +1; *j*≠0, and each vehicle *k,* X_{ijk} is defined as

$$x_{ij} = \begin{cases} 1 & \text{if vehicle } k \text{ is not using arc(i,j)} \\ 0 & \text{otherwise} \end{cases}$$

We want to design a set of minimal cost routes, one for each vehicle, such that each customer is serviced exactly once and every route originates at vertex *0* and ends at vertex *n + 1*

We can state the VRP mathematically as: (Larsen,1999)

$$\min \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} \qquad (2.1)$$

subject to:

$$\sum_{k \in V} \sum_{i \in N} x_{ijk} = 1 \qquad \forall i \in C \qquad (2.2)$$

$$\sum_{i \in C} d_i \sum_{j \in N} x_{ijk} \leq q \qquad \forall k \in V \qquad (2.3)$$

$$\sum_{j \in N} x_{ojk} = 1 \qquad \forall k \in V \qquad (2.4)$$

$$\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} = 0 \qquad \forall h \in C \ , \forall k \in V \qquad (2.5)$$

$$\sum_{i \in N} x_{in+1k} = 1 \qquad \forall k \in V \qquad (2.6)$$

$$x_{ijk} \in \{0,1\} \qquad \forall i,j \in N \ , \forall k \in V \qquad (2.7)$$

The objective function (2.1) minimizes of the total distance traveled. The constraint (2.2) states that each customer is visited exactly once and (2.3) means that no vehicle is loaded more than its capacity allows it to. The next three equations (2.4, 2.5 and 2.6) ensure that each vehicle leaves the depot *0,* after arriving at a customer the vehicle leaves that customer again and finally arrives at the depot *n+1.* Constraints (2.7) are the integrality constraints.

2.2. Categories of Vehicle Routing Problems

2.2.1. Time Constraints:

If we add a "time window" constraint for each customer, we obtain the Vehicle Routing Problem with Time Windows (VRPTW). Time constraints ensure that a vehicle visits a customer within a certain time frame. The vehicle may arrive before the time window "opens," but the customer cannot be serviced until the time window "opens." It is not allowed to arrive after the time window "is closed." Some models allow for early or late servicing but with some form of additional cost or penalty. These models are denoted as "soft" time window models. Most research has been done on "hard" time window models.

2.2.2. Multiple Depots:

There might be more than one depot in the VRPs. Each depot can have its own fleet of vehicles or the vehicles may be based on different depots. For the first case, it is usually assumed that the vehicles must return to the same depot. For the second case, there can be a constraint such as the number of vehicles that arrive at a depot must be equal to the number of vehicles that leave the depot.

2.2.3. Pick-Up and Delivery:

In the delivery VRP case, we are concerned with the distribution of the goods, but we can also pick-up goods from customers. These type of vehicle routing problems are known to be pick-up and delivery VRP problems. In the simple case of this problem, the customers are divided into two classes: a set of delivery customers and a set of pick-up customers. Two capacity labels, one for delivery and one for pick-up, must be handled for these types of VRPs.

2.2.4. Multiple Compartments:

When the vehicles transport several commodities, which must remain separate during the transportation, multiple compartment vehicles are used. The multiple compartment case has no influence on the main problem structure, but capacity constraints should be revised. If vehicle has m compartments, the capacity constraints must be modeled by m states instead of one.

2.2.5. Multiple Time Windows:

For the Vehicle Routing Problem with time constraints, there is one time window defined for each customer. This time window includes the earliest and latest arrival time information. Allowing customers to have multiple time windows in which they can be serviced is handled in VRP with multiple time windows.

2.2.6. Non-Identical Vehicles:

VRP can be modeled with non-identical vehicles. The typical variability that disturbs the homogeneity is the capacity of the vehicles, but there can be other factors such as different travel times, different costs or time windows for the vehicles. In the non-identical (or multiple vehicle-type) VRP, the vehicles can vary or there may exist categories of vehicles where an upper limit on the number of vehicles in each category is given in most cases.

2.2.7. Soft Time Windows:

Violating the time constraints is sometimes allowed by adding some penalty terms to the objective function. With this allowance, time constraints are said to be soft. VRP with soft time windows with a general objective function is not efficiently solvable.

2.2.8. Other Additional Constraints:

In addition to the generalizations given above, there may be some different constraints, such as:

- Upper or lower limits on the length of the routes,
- Upper or lower limit on the number of customers that can be served,
- Upper or lower limit on the total travel time of each vehicle,
- Time dependent travel speed for vehicles,
- Upper or lower limit on the variability of the service times of the vehicles, etc.

2.2.9. Different Objective Functions

The objective function may also differ in VRPs. Below are some types of these objective functions. It should be noted that a combination of these can also be used.

- Minimum number of vehicles,
- Minimum total distance,
- Minimum total travel time,
- Maximum number of customers served with a given number of vehiples,
- Minimum total waiting time of the vehicles,
- Minimum variability in the travel times of the vehicles,
- Efficient loading of the vehicles,
- Minimum variability in the total distance traveled by the vehicles.

2.3. Vehicle Routing Problem with Time Windows

In this section, VRPTW is explained, a detailed definition is given and complexity of the problem is discussed.

2.3.1. Problem Definition

The vehicle routing problem with time windows (VRPTW) is a well-known NP-hard problem, which is an extension of normal VRP, encountered very frequently in making decisions about the distribution of goods and services (Tan et al., 2000). VRPTW can be stated as follows: given a central depot, a fleet of vehicles and, a set of customers with known demands (e.g., some quantity of goods to be delivered), find a set of closed routes, originating and ending at the depot, that service all customers at minimum cost. Moreover, each route must satisfy capacity and time window constraints (Potvin *et al.,* 1995). In VRPTW, a set of decision variables is added to the model to specify the times that services begin are the decision variables based on customers.

Allowable delivery times of the customers add complexity to the VRP because of the time feasibility check for each customer. In the VRPs with time constraints, the service of a customer, involving pick up or delivery of goods or services, can start within the time window defined by the earliest and the latest times when the customer permits the start of service.

We can define the time window given for a customer as follows: for customer *i, (e_i, l_i)* means that the vehicle must not arrive at customer *i* after $/_i$ and service at customer *i* must not start before e_i. If the vehicle arrives at customer before e_i then it should wait until e_i

In some cases, vehicles are allowed to start service just at the time they arrive to the customer site, so in these types of problems, there are no waiting times for the vehicles at the customer sites..

2.3.2. Complexity of VRPTW

Being one of the most important problems in Operations Research literature, the VRP is one of the most difficult problems to solve. The problem is quite close to the Traveling Salesman Problem. TSP is a well-known NP-Hard problem, where only one vehicle or person visits all the customers. As an m-TSP, VRP, even for small fleet sizes

and moderate number of transportation requests, is more complicated than TSP. Adding time windows to the VRP results in with a more complicated problem VRP without time windows. Furthermore, Savelsbergh (1985) had shown that, even finding a feasible solution to the VRPTW when the number of vehicles is fixed is itself an NP-Complete problem. Therefore, the development of approximation algorithms or heuristics for this problem is of primary interest to many researchers.

2.4. Review of Optimal Algorithms for VRPTW

There have been many papers proposing exact algorithms for the VRPTW. The first of these papers was published by Kolen *et al.* (1987). Since then, many people have used exact algorithms for finding an optimal solution for the VRPTW. These exact algorithms can be classified in three groups:

1. Dynamic Programming
2. Lagrangean Relaxation-based Methods
3. Column Generation

2.4.1. Dynamic Programming

The first paper on dynamic programming for the VRPTW is the publication of Kolen *et al.* (1987) It is inspired from Christofides *et al.* (1981) which used Dynamic Programming for the VRP for the first time.The algorithm of Kolen *et al* uses branch-and-bound approach in order to retrieve optimal solutions. There are three nodes in the branch-and-bound algortihm, which corresponds to three sets:

F (a) : The set of fixed feasible routes starting and finishing at the depot.

P(a) : Partially built route starting at the depot.

C(a) : Customers that are not allowed to be next on *P(a)*

Branching is done by selecting a customer that is not forbidden and that does not appear in any route. At each branch-and-bound node, Dynamic Programming is used to

calculate a lower bound on all feasible solutions defined by *F (a), P (a)* and *C (a).* Kolen *et al.* solved problems up to 15 customers by this method in this paper.

2.4.2. Lagrangean Relaxation-Based Methods

There exist many papers that use Lagrangean Relaxation-based Methods using different approaches. Variable splitting followed by Lagrangean Relaxation was used by Jornsten *et al.* (1986), Madsen (1988), Halse (1992), and Fisher *et al.* (1997). Fisher *et al.* (1997) used *K-tree* approach followed by Lagrangean Relaxation. Finally Kohl and Madsen.(1997) applied shortest path with side constraints approach followed by Lagrangean Relaxation.

Variable splitting (or cost splitting) was first presented in the technical report of Jornsten *et al.* (1986), but no computational results were given in the paper. Madsen *et al.*(1988) used four different splitting approaches but they are not tested either. Halse (1992) offered three approaches and he had implemented and tested one of these approaches.

Fisher *et al.* (1997) presents an optimal algorithm where the problem is formulated as a *K-tree* problem with degree *2K* on the depot. The VRPTW can be formulated as finding a *K-tree* with degree *2K* on the depot, degree 2 on the customers and subject to capacity and time constraints. This representation becomes equal to *K* routes. This algorithm was able to solve many of the clustered Solomon test problems but it could not solve any of the random given test problems.

Kohl and Madsen (1997) relax the constraints that ensure every customer must be visited exactly once, that is;

$$\sum_{k \in V} \sum_{\substack{j \in N \\ j \neq i}} x_{ijk} = 1 \qquad \forall\, i \in C \tag{2.8}$$

is relaxed and a penalty term is added to the objective function as below:

$$\min \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} (c_{ij} - \lambda_j) x_{ijk} + \sum_{j \in C} \lambda_j \qquad (2.9)$$

Here λ_j is the Lagrangean Multiplier associated with the constraint that ensures that customer *j* is served. The model is decomposed to one sub-problem for each vehicle but since vehicles are assumed to be identical, all the sub-problems are identical. The resulting problem is a shortest path problem with time window and capacity constraints. Kohl *et al.* were able to solve some of the Solomon (1987) instances.

2.4.3. Column Generation

Column generation has turned out to be an efficient method for a range of vehicle routing and scheduling problems. This approach is implemented previously by Desrochers *et a.l* (1992) and Kohl (1995). Column generation is based on the idea of initializing the linear program with a small subset of variables (by setting all other variables to 0) and computes a solution of this reduced linear program. Column generation used together with branch-and-bound is denoted as branch-and-price.

Desrochers *et al.* (1992) add feasible columns as needed by solving a shortest path problem with time windows and capacity constraints using dynamic programming. The LP solution obtained provides a lower bound that is used in a branch-and-bound algorithm to solve the integer set-partitioning formulation.

By column generation, problems up to 25 customers can be solved optimally, but only few of the problems with 50 and 100 customers can be solved.

2.5. Review of Approximation Algorithms and Heuristics for the VRPTW

Since the VRPTW is proven to be NP-hard, non-exact algorithms are very popular for finding solutions. There are many papers that propose heuristic algorithms to the VRPTW. These algorithms can be classified into three groups:

1. Construction algorithms
2. Improvement algorithms
3. Metaheuristic algorithms

Heuristic algorithms that build a set of routes are known as construction algorithms and are used to build an initial feasible solution for the problem. The algorithms that try to find an improved solution by using the initial solution are called improvement algorithms. Metaheuristic approaches are mostly a combination of these two and are based on different things.

2.5.1. Construction Algorithms

The construction (route-building) algorithms are used to generate good feasible solution(s). These algorithms can be classified as sequential and parallel algorithms. In a sequential algorithm one route is constructed initially and others are constructed when necessary, while in a parallel algorithm, many routes are constructed simultaneously.

2.5.1.1. Sequential Construction Algorithms

The first sequential construction (route-building) algorithm was proposed by Baker and Schaffer (1986.) This algorithm can be interpreted as an extension of the *Savings Heuristic* of Clarke and Wright (1964). The algorithm starts with all possible single customer routes and at each iteration; two routes with the maximum saving are combined. The saving between customers *i* andy *is* calculated as:

$$s_{ij} = d_{i0} + d_{0j} - G.d_{ij} \tag{2.10}$$

where *G* is the *route form factor (weight)* and d_{ij}- is the distance between nodes *i* and *j*.

Baker and Schaffer developed a sequential algorithm by defining the savings as a combination of distance and time feasibility. Then Solomon (1987) used a similar heuristic where the time feasibility aspect is not included in the savings function. The

arcs that can be used are limited by the magnitude of the waiting times. It is required to check the violation of time windows when two routes are integrated. Reasonable results were reported for *Savings Heuristic* and the adopted versions.

Landeghem (1988) also presents a sequential construction heuristic based on the *Savings Heuristic*. It devolops a *bi-criteria* algorithm for obtaining a measurement of how good a connection between customers is in terms of timing.

Time Oriented Nearest Neighborhood Heuristic is another sequential heuristic proposed in Solomon (1987). Every route is initialized by selecting the customer closest to the depot. The closeness may be geographical or temporal closeness. Insertion of unassigned customers is done by selecting the customer that is closest to the last customer added at each iteration. When there is no feasible point for the insertion of any customer, a new route is initialized and insertion continues until all unassigned customers are added. This paper also proposes *Insertion Heuristics* which are based on different insertion criteria. As in *the Time Oriented Nearest Neighborhood Heuristic,* if no feasible insertion is possible, a new route is initialized.

One of the sequential route-building algorithms proposed by Solomon in Solomon (1987) is *Time-Oriented Sweep Heuristic*. It has two phases: a clustering phase which assigns customers to different clusters and a scheduling phase which solves a TSPTW problem by using the TSPTW heuristics proposed by Savelsbergh (1985,1990).

Another cluster-first route-second algorithm for VRP is proposed by Gillet and Miller (1974). Here, generating strict clusters may cause some customers to be unscheduled due to time window constraints. In order to schedule these customers, the previously scheduled ones are removed from the routes and the process is repeated.

2.5.1.2 Parallel Construction Algorithms

Building routes sequentially may cause latterly constructed routes to be poor quality since there are only few alternative points of insertion at the latter iterations of the process. This can be overcome partially by constructing parallel routes. Potvin and

Rousseau (1993) proposed a parallelization of the *Insertion Heuristics* by creating many routes simultaneously. For the initialization of each route, the customer that is farthest from the depot is selected as a "center customer." Then, the customers are inserted to the best feasible insertion place. Russell (1995) also adapted parallel insertion approach.

In the parallel algorithm in Antes and Derigs (1995) offers comes to the customers from the routes, unrouted customers send a proposal to the route with the best offer, and each route accepts the best proposal.

As a route-first schedule second algorithm, Solomon (1987) proposes a *Giant-Tour Heuristic*. In this heuristic, customers are routed on a big route and then this route is divided into number of routes. Building an initial tour is a TSPTW problem.

Foisy and Potvin (1993) also presented a parallel algorithm which is an *Insertion Heuristic* building routes simultaneously using the Solomon's heuristic to generate the initial center customers.

2.5.2. Route- Improving Heuristics

2.5.2.1. Neighborhood Search

Almost every route-improving algorithm is based on the concept of neighborhood. Neighborhood concept has been used for at least 40 years for combinatorial optimization problems. One of the earliest references is Croes (1958) which used the idea to improve the solutions of the Traveling Salesman Problem.

Checking some or all the solutions in a neighborhood might reveal better solutions. This idea can be repeated starting at the better solution. At some point, no better solution can be found and a local optimum has been reached. This algorithm is called local search. A schematic representation of the local search is given in Figure 2.1

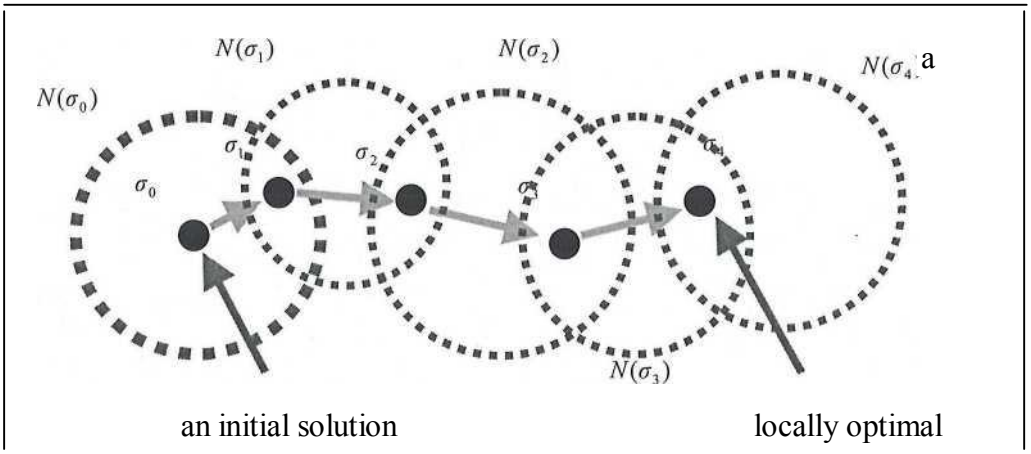


Figure 2.1 Schematic representation of the local search procedure.

Metaheuristics are also based on local search methods but they use additional methods for escaping local optimum in order to search other parts of the solution space for better solutions.

In the neighborhood search, it is assumed that a solution is specified by a vector *x,* where the set of all feasible solutions is denoted by *X.* The cost of the solution is denoted by *c(x),* which is often called the *objective function.* Each solution $x \in X$ has an associated set of *neighbors, N (x) s X,* which is the *neighborhood of x.* Figure 2.2 shows the neighborhood search procedure (Smith *et ah,* 1996)

In this point, it should be noted that there are two strategies for selecting the neighborhoods during the search:

1. The first-best (FB) strategy: It selects the first solution in *N(x)* that results in improvement.
2. The global-best (GB) strategy: It searches all solutions in *N(x),* and selects the one that brings maximum improvement on the solution.

1. *(Initialization)*

   1.1. Select a starting solution $x^{now} \in X$

   1.2. Record the current best known solution by setting $x^{best} = x^{now}$ and define $best\_cost = c(x^{best})$.

2. *(Choice and termination)*

   2.1 .Choose a solution $x^{next} \in N(x^{now})$. If the choice criteria used cannot be satisfied by any member of $N(x^{now})$ (hence no solution qualifies to be x^{next}), or if other termination criteria apply (such as a limit on the total number of iterations), than the method stops.

3. *(Update)*

   3.1.Re-set $x^{now} = x^{next}$, and if c $(x^{now}) < best\_cost$, perform Step 1.2. Then return to Step 2.

Figure 2.2 Neighborhood Search, (Smith *et al,* 1996)

It is obvious that local search terminates at local optimum values with high probability. Thus, metaheuristics use techniques for escaping local optima.

2.5.2.2. Neighborhoods of VRPTW

Most of the improvement algorithms for the VRPTW use an exchange neighborhood to obtain a better solution. Two classical algorithms which were originally proposed for Traveling Salesman Problem are *k*-Opt and Or-Opt heuristics. In the TSP, there exists a single route and exchange operations can be done on the nodes or arcs within the same route. These heuristics are modified for VRPTW, which is the multiple-route case of TSPTW.

The *k*-opt heuristic replaces a set of links in the route by another set of *k* links. The complexity of the heuristic is mostly affected by the size of *k*. For larger *k* values, the heuristic tends to give better results, but the computational time increases. Lin and Kernighan (1973) propose a heuristic that determines the size of A: dynamically. Time window constraints of the problem may result with infeasible solutions for this heuristic

since it changes the orientations of the customers within routes. The adaptation of this heuristic can easily be interpreted as having the opportunity of exchanges not only within the route, but also between the routes.

The Or-Opt exchange, which was originally proposed by Or (1976) for traveling Salesman Problem, removes a chain of at most three consecutive customers from the route and tries to insert this chain at all feasible locations in the routes. This heuristic is slightly modified by allowing the chain to be inserted in the same route and other routes. An example of Or-opt can be seen in Figure 2.6. Generally, the size of the neighborhood of Or-Opt is $O(n^1)$.

Potvin and Rousseau (1995) present two variants of 2-Opt and Or-Opt that maintain the direction of the route. For the 2-Opt, every pair of links is considered for removal with only one restriction: the two links must be in different routes. An example of this neighborhood is illustrated in Figure 2.4. For the Or-Opt, every sequence of three customers, two customers and one customer (in this order) is considered and, for each sequence, all insertion places are also considered.

The relocate operator moves a customer from one route to another as it can be seen in Figure 2.5. The exchange operator exchanges customers in different routes. An example can be seen in Figure 2.6.

The *k*-node interchange, was initially proposed by Christofides and Beasley (1984), and then modified by several authors. In this heuristic, each customer *i* is considered sequentially, and sets M_1 and M_2 *are identified.* M_1 is defined as the customer *i* and its successor *j*. M_2 consists of the two customers that are closest to *i* and *j,* but not on the same route with *i* and *j*. Removing the elements of the sets M_1 and M_2 and inserting them in any other possible way define a neighborhood. Since this neighborhood is very large, *k* most promising candidates are selected.

λ -interchange local search method is another local search algorithm, which was first introduced by Osman and Christofides (1994). The local search procedure is conducted by interchanging customers between routes. The search order for the customers is defined for a pair of routes either systematically or randomly. Parameter

means that maximum λ customer nodes can be interchanged between routes. For the case $\lambda = 2$, there are eight operators defined: *(0,1), (1,0), (1,1), (0,2), (2,0), (2,1), (1,2)* and *(2,2)*. For example, the operator *(1,2)* means that, on a route pair (R_p, R_q) a shift of one customer from R_p to R_q and a shift two customers from R_q to R_p will be done. Only improved solutions are accepted during the interchanges.

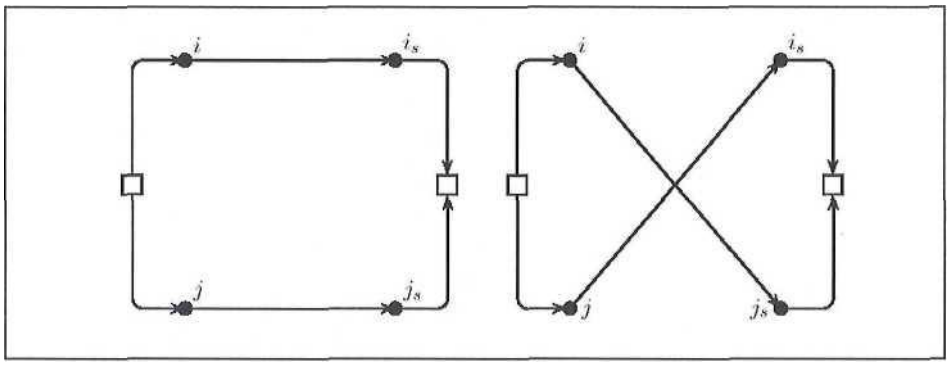


Figure 2.3 An example of Or-Opt exchange. The figure on the left presents the route before the operation is performed, and the one on the right is the route after the operation.

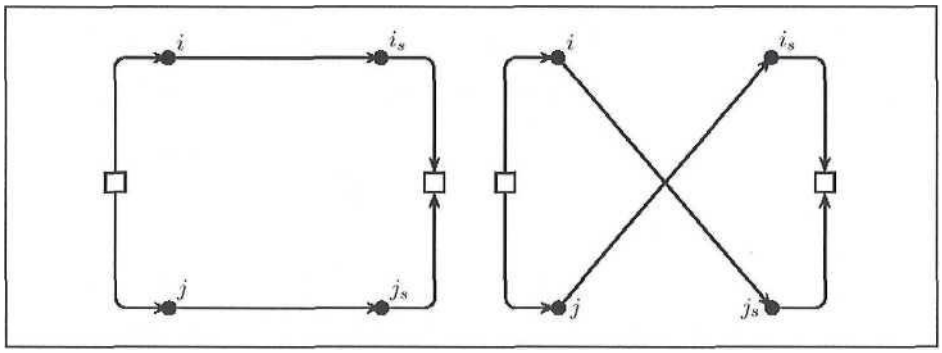


Figure 2.4 An example of 2-Opt exchange. The figure on the left presents the route before the operation is performed, and the one on the right is the route after the operation.

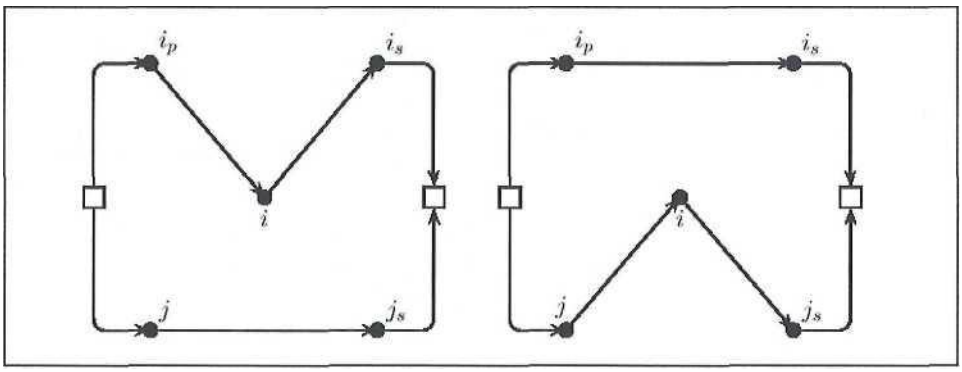


Figure 2.5 An example of relocate operator. The figure on the left presents the route before the operation is performed, and the one on the right is the route after the operation.

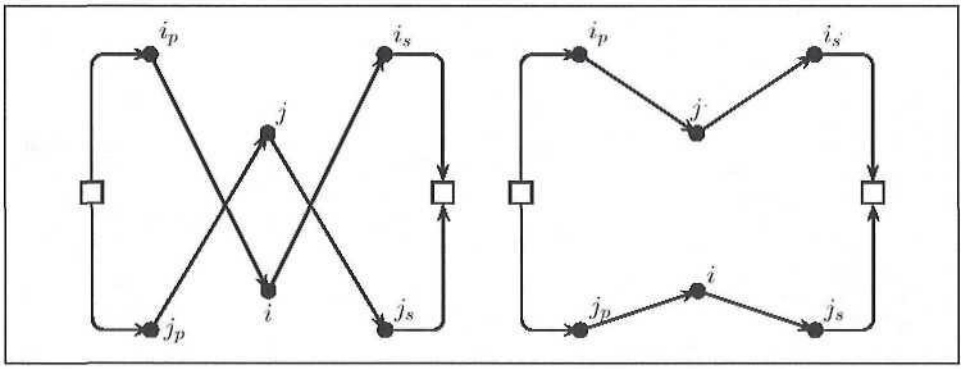


Figure 2.6 An example of exchange operator. The figure on the left presents the route before the operation is performed, and the one on the right is the route after the operation.

Schulze and Fahle (1999) propose a neighborhood called shift-sequence where a customer is moved from one route to another checking all possible insertion positions. If an insertion is feasible by removing another customer, it is removed and inserted in another route.

There had been many modified versions of the neighborhoods used in different heuristics proposed by many authors. For instance, Russell (1995) modified the *k*-node interchange operator and Potvin and Rousseau the Or-Opt operators.

2.5.3. Metaheuristics for the VRPTW

Since they only provide local optimal solutions, local search methods fail to give promising results for the RPTW. In order to escape local optima and enlarge the search space, metaheuristic algorithms such as simulated annealing, tabu search and genetic algorithm have been used to solve VRPTW.

2.5.3.1. Simulated Annealing

Simulated annealing (SA) is a stochastic relaxation technique that finds its origin in statistical mechanics (Metropolis et al., 1953). Its methodology is similar to the annealing processing of solids. In order to avoid the meta-stable states produced by

quenching, metals are often cooled very slowly, which allows them time to order themselves into stable, structurally strong, low energy configurations. This process is called annealing (Tan et al., 2000). The states of solids correspond to feasible solutions, the energy of each state to objective function value at each feasible solution, and the minimum energy to the optimal solution in the combinatorial optimization problems. During the SA process, the temperature is gradually reduced. It can be generalized that the system is often first heated and then cooled.

At each step of the simulated annealing process, a new state of the system is reached from the current state by giving a random displacement to a randomly selected particle. If the energy of the new state is lower than the current state, the new solution is accepted. In other words, SA works by searching the set of all possible solutions, reducing the chance of getting stuck in a poor local optimum by allowing moves to inferior solutions under the control of a randomized scheme. A move to the solution x' from the solution x which results in a change Δc is accepted if

$$\exp(-\Delta c / T) < R \qquad (2.11)$$

where T is a control parameter and R is a uniform random number. In order to allow many inferior moves to be accepted, the parameter T is set high at initial steps. At the latter steps, this parameter is reduced up to point where nearly all inferior values are rejected. Then the temperature T *is* reset to a high value after the occurrence of a special neighborhood without accepting any moves.

Mentioned above, the ideas that form the basis of SA was first published by Metropolis et al. (1953). Based on this idea, Kirkpatrick et al. (1983) suggests that this simulation technique could be applied to the search of feasible solutions of optimization problems. In this paper, the parameter T is used to generate a cooling schedule on the simulation. Davis (1991) had statistically approved that SA is capable of finding the optimal solution. However, SA has the possibility of getting caught in repetition of moves which results in cycling and high computational time.

Thangiah et al. (1994) uses a non-monotone probability function in their simulated annealing heuristic. They used the λ-*interchange* operator while decreasing the temperature after each iteration. In case the entire neighborhood has been explored without finding and accepting moves the temperature is increased. This is called a reset. After R resets, the algorithm terminates.

Chiang and Russell (1996) proposed three different SA methods. First one is modified version of the *k*-node interchange mechanism and second one is a modified version of λ-interchange method. The third one is based on the concept of tabu list, which is described on the next section of this chapter. Using SA with the λ-interchange method, the tabu list contained moves that are allowed for the time being.

Tan et al. (2001) proposed a SA heuristic defining a non-monotanic cooling schedule defined as

$$T_{k+1} = T_k /(1 - \rho\sqrt{T_k}) \qquad (2.13)$$

where the starting temperature is set to 50 and the time constant ρ is set to 0.3. When the temperature is high, the probability of accepting the worse is high, when the temperature is decreased according to function given above, the probability of accepting worse is low, which lets the search go into thermal-equilibrium point.

Finally, Li and Lim (2003) used a metaheuristic that proposes simulated annealing-like restarts. Proposed algorithm finds an initial solution using Solomon's insertion heuristic and then starts local search from initial solution using specified restart strategy.

2.5.3.2. Tabu Search

Tabu search (TS), like SA, is based on the neighborhood search with local optima avoidance, but in a deterministic way, which tries to model human memory processes.

TS has its antecedents in methods designed to cross boundaries of feasibility or local optimality standardly treated as barriers, and to systematically impose and release constraints to permit exploration of otherwise forbidden regions (Glover and Laguna, 1993) So the idea that lies under TS is systematically violating the feasibility conditions. At each iteration, the neighborhood of the current solution is explored and the best solution is selected as the new current solution. However, as opposed to a classical local search technique, the procedure does not stop at the first local optimum when no more improvement is possible. The best solution in the neighborhood is selected as the current solution even if it is worse than the current solution.

To prevent cycling, recently selected solutions are forbidden and are inserted into a tabu list. Often, the tabu list does not contain "illegal" solutions, but forbidden moves. It makes sense to allow the tabu list to be overruled if it leads to an improvement of the current overall best solution. Criteria such as this for overruling the tabu list are called aspiration criteria. The most used criterion for stopping tabu search is a constant number of iteration in all.

1. (Initialization)

1.1. Begin with the same initialization used by Neighborhood Search, and with the history record H empty.

2. (Choice and Termination)

2.1. Determine $Canditate\_N(x^{now})$ as a subset of $N(H, x^{now})$. Select x^{next} from $Canditate\_N(x^{now})$ to minimize $c(H, x)$ over this set. (x^{next} is called a highest evaluation element of $Canditate\_N(x^{now})$.)

2.2. Terminate by a chosen iteration cut-off rule

3. (Update)

3.1. Perform the update for the Neighborhood Search Method and additionally update the history record H.

Figure 2.7 Standard Tabu Search, (Smith *et al,* 1996)

H can be defined in terms of prohibition on re-visiting certain states in $N(x^{now})$. As it was mentioned before, such states are called tabu. In modified versions of TS, *H* may also include elements, which are not members *of* $N(x^{now})$.

Most early references to TS in its present form are Glover (1986) and Glover and McMillan (1986). Following this, there have been a number of contributions to the algorithm. First and the most important of them are Glover (1989 and 1990).

The study of Garcia *et al.* (1994) should be included in the tabu concept. In order to restrict the amount of work, not all possible neighborhoods are carried out. The exploration of the neighborhood is restricted to the exchange of arcs that are close in distance.

Thangiah et al. (1994) proposes a TS, which used λ-interchange method. They also use the simulated annealing parameters combined with tabu list, so their algorithm can be interpreted as a hybrid heuristic.

Badeau et al. (1997) use the strategy of decomposing the problem into sub-problems. After the generation of a solution at one processor, the solution is decomposed into groups of routes. The other processors try to improve the solution by improving the solution of each sub-problem. They first build a number of routes and group the routes according to some solution quality criteria. For each group, a TS is performed using the exchange operator. Their tabu list contains penalized exchanges that are frequently performed.

The TS of Potvin *et al.* (1995) uses the local search methods of Potvin and Rousseau (1995) and has similarities with Garcia et al. (1994).

There are several papers published on the parallelization of the TS, which try to partition the neighborhood or decompose the problem into sub-problems, each solved by parallel TS.

Garcia et al. (1994) use the strategy of partitioning the neighborhood to parallelize the TS. One processor is used for controlling the TS, while the other is used for exploring the neighborhood.

Garcia et al. (1996) apply TS strategy to the Or-Opt and 2-Opt neighborhood operators. In order to save computation time and focus on the most promising exchanges, the two neighborhoods are reduced by selecting a subset.of customers at each iteration and by considering only the exchanges that link the selected customers with customers that are close in distance.

Chiang and Russell (1997) use a reactive TS metaheuristic where the route improvement procedure is invoked each time another 10 percent of the customers are added to the emerging routes. They use the λ-interchange operator of Osman and Christofides.(1994)

Schulze and Fahle (1999) describe a parallel TS heuristic where the neighborhood structure is based on simple customer shifts. All routes generated are collected in a pool and to obtain a new initial solution for the TS heuristic, a set covering heuristic is applied to the routes in the pool. Furthermore, route elimination is used for the routes with few customers.

2.5.3.3. Genetic Algorithm

A genetic algorithm (GA) is a randomized search technique operating on a population of individuals, which form the solutions (Potvin and Bengio, 1996). A fitness value is calculated for each individual and the search is guided by this value. Basically, a genetic search consists of the following components:

1. Representation: Encode the characteristics of each individual in the population as a chromosome (typically, a chromosome is a bit string). Set the population to this initial population.

2. Reproduction: Select two parent chromosomes from the current population. This process is stochastic and a chromosome with high fitness value is more likely to be selected.
3. Recombination: Generate two offsprings from two parents by exchanging pieces (crossover).
4. Mutation: Apply a random mutation to each offspring with a small probability.

Steps 2,3,and 4 are repeated until the number of chromosomes in the new population is the same as in the old population. Then the current population is set to the new population of chromosomes.

Simple genetic operator such as crossover and mutation are used to construct new solutions from pieces of old ones, in such a way that the population steadily improves for many problems.

Crossover operator is based on a simple idea. Suppose there are 2 strings (parents) each consisting of 6 variables as below:

PARENT 1: (al, a2, a3, a4, a5, a6) and PARENT 2 :(bl,b2, b3, b4$_{Sl}$i4b6)

These two strings represent two solutions to a problem. A crossover point is chosen at random form the numbers 1 to 5, and a new solution is produced by combining the pieces of the original parents. If the crossover point is chosen as 2, the offsprings (children) will be as follows:

CHILD 1: (al,a2,b3,b4,b5,b6) and CHILD 2: (bl,b2, a3,a4, a5,a6)

Other most commonly used operator, mutation, provides the opportunity to reach parts of the search space, which perhaps cannot be reached by crossover alone. Each gene of a string is examined in turn and, with a small probability, its current position is changed. For example, a string of

(al, a2, a3,a4, a5,a6) may become (al,a2, a3,A6, a5,a4)

This procedure is repeated for a fixed number of generations or until convergence to a population of similar individuals is obtained. Then, the best chromosome generated during the search is decoded into the corresponding individual.

There have been many applications of GA for the VRPTW, These algorithms can be interpreted as the modifications of the standard GA.

In Thangiah (1993), GA is designed to find good clusters of customer, within a "cluster-first route-second" strategy. Once the clusters are identified by the genetic search, classical insertion and post-optimization procedures are applied to produce the final routes.

The GA in Blanton and Wainwright (1993), which was designed for multiple vehicle type VRPTW problems, describes a well-known approach to combinatorial optimization problems with side constraints. It is based on the hybridization of a GA with a greedy heuristic.

Potvin and Bengio (1996) describes a genetic algorithm that operates on chromosomes of feasible solutions. The selection of parent solutions is stochastic and is directed to the best solutions. Two types of crossover operators are used. The reduction of routes is obtained by two mutation operators. The routes are optimized by an Or-Opt based local search at every *k* iterations.

Berger et al. (1998) propose a method based on the hybridization of a GA with well-known construction heuristics. The initial population is created with nearest neighborhood heuristic of Solomon and the fitness values of the individuals are based on the number of routes and the total distance of the corresponding solution. Braysy (1999) extends the work of Berger et al. (1998) by proposing several new crossover and mutation operators, testing different forms of GAs.

Homberger and Gehring (1999) proposes two evolutionary metaheuristics based on the class of Evolutionary Strategy algorithms. The difference of their algorithm comes with the role of the mutation operator. The representation of the individuals also

includes a vector of evolutionary strategy in addition to the solution vector and both components are evolved by crossover and mutation operators.

2.5.3.4. Other Metaheuristics and Hybrid Algorithms

Rochat and Taillard (1995) uses a probabilistic local search method based on intensifying the solution, which is in some ways similar to the SA approach.

Kilby et al. (1999) uses a Guided Local Search (GLS) algorithm. GLS can be defined as a memory-based metaheuristic like TS. In GLS, the cost function is modified by adding penalty term encouraging diversification, that is, escaping form local optima is done by penalizing solution features. As local search neighborhoods, Kilby *et al.* uses 2-Opt exchanges.

In Potvin and Robillard (1999), a combination of a competitive neural network and a GA is described. They use a special type of neural network called competitive neural network to select the customers. Competitive neural network is frequently used to cluster the data. For every vehicle, a weight vector is defined. Initially, all weight vectors are placed randomly close to the depot. Then, customers are selected. For each cluster, the distance to all weight vectors are calculated and closest weight vector is updated by moving it closer to the customer.

Braysy et al. (2000) describes a two-step evolutionary algorithm based on the hybridization of a GA consisting of several local searches and route construction heuristics inspired form the studies of Solomon (1987). At the first step, a GA based on the studies of Braysy (1999) and Berger *et al.* (1998) is used. The second step consists of an evolutionary metaheuristic that picks every pair of routes in random order and applies randomly one out of the four local search operators or route construction heuristics.

Tan *et al.* (2001) proposes an artificial intelligence heuristic which can be interpreted as the hybrid combination of SA and TS. During the process, if a move is not a tabu and satisfies the SA criterion, it will be accepted and then the search is

restarted from the beginning of a new current solution after updating the tabu list and SA parameters.

Tan *et al.* (2001) also describes a hybrid GA. The major focus of this algorithm is about the grouping of the customer nodes. With different groupings, one chromosome may represent different solutions. A local search method is then used to search better grouping (better fitness value) for each chromosome.

2.6. Overview of Exact, Heuristic and Metaheuristic Algorithms for VRPTW

Since 1987, after the publication of Solomon (1987) on VRPTW, many researchers have been using the Solomon Instances to benchmark their algorithms, but the difference in the objective function and calculation of time and cost makes it hard to make direct comparisons. A detailed explanation of the Solomon's benchmark problems is given in Chapter 4. Table 2.1 gives a summary of the publications classified based on the objective functions formulated. As mentioned before, we have only focused on the papers the objective is minimizing the total distance/cost/time, minimizing the number of vehicles and a combination of them.

Exact methods are able to give optimal solutions for small and specific types of problems, but they cannot solve large problems in polynomial time.

A number of heuristics seemed to perform well, but in general, we can claim that whatever quality the initial solution is, there should be some methods defined for escaping the local optima. This is the reason why many researchers have focused on metaheuristics in the recent years.

Table 2.1 Overview of Objective Functions Chosen for the Reviewed Publications on the VRPTW

REFERENCE	OBJECTIVE
Solomon, 1987	■ Min Number of Vehicles ■ Min Schedule Time ■ Min Travel Distance ■ Min Waiting Time
Desrochers et al.,1992	■ Min Travel Distance
Halse,1992	■ Min Travel Distance
Fisher et al, 1994	■ Min Travel Distance
Potvin and Rousseau, 1993	■ Min Number of Vehicles ■ Min Route Time
Foisy and Potvin, 1993	■ Min Number of Vehicles ■ Min Travel Distance
Garcia et al, 1994	■ Min Number of Vehicles ■ Min Routing Cost
Thangiah et al, 1994	■ Min Number of Vehicles ■ Min Travel Distance '
Russell, 1995	■ Min Number of Vehicles ■ Min Schedule Time
Kohl, 1995	■ Min Travel Distance
Antes and Derigs, 1995	■ Min Number of Vehicles ■ Min Routing Cost
Badeau et al, 1995	■ Min Travel Distance
Rochar and Taillard, 1995	■ Min Number of Vehicles ■ Min Travel Distance
Schulze and Fahle, 1996	■ Min Number of Vehicles ■ Min Travel Distance
Potvin et al, 1995	■ Min Number of Vehicles " Min Route Time
Potvin and Bengio, 1996	■ Min Number of Vehicles ■ Min Route Time
Robillard et al, 1996	■ Min Number of Vehicles ■ Min Route Time
Chiang and Russell, 1996	■ Min Number of Vehicles ■ Min Schedule Time ■ Min Travel Distance
Kohl and Madsen, 1997	■ Min Travel Distance
Kilby et al, 1999	■ Min Travel Distance .
Homberger and Gehring, 1999	■ Min Travel Distance ■ Min Number of Vehicles
Potvin and Robillard, 1999	■ Min Number of Vehicles ■ Min Route Time
Braysy et al, 2000	■ Min travel distance
Tan et al, 2001	■ Min travel distance
Li and Lim, 2003	■ Min Number of Vehicles ■ Min Travel Distance

3. HYBRID HEURISTICS FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

In this thesis, hybrid heuristic algorithms for the cost (distance) minimization for the VRPTW are proposed. Vehicles are identical, capacities of the vehicles are finite and time window constraints are assumed to be strict.

3.1. Mathematical Formulation of the Problem

The VRPTW model is given by homogenous vehicles a set of customers and a directed graph *G*. The graph consists of $n+1$ vertices where the customers are denoted as $1,2,...,n$ and the depot is represented by the vertex 0. The set of vertices excluding the depot is denoted as *N'*. The set of arcs (denoted as *A)* represents connections between the depot and the customers and among the customers. All arcs originate from vertex 0 and terminate in vertex 0. With each arc (i, j), where $i \neq j$, we associate a cost c_{ij} (t_{ij}) .Each vehicle has a capacity Q and each customer i has a demand d_i

Notation.

$G = (N,A)$

N: Nodes

A: Arcs

0: Depot

$A = \{(i,j) : i,j \in N, i \neq j\}$

$N = \{0,1,...,n\}$

$N' = \{1,...,n\}$

c_{ij}: Cost (or time) of going from node i to node j (equal to t_{ij}) d_i: Demand at node i

s_i : Service time at node i

l_i : Latest arrival time at node i

e_i: Earliest arrival time at node i

Q: Vehicle capacity

Decision Variables:

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i, j) \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

$y_{ij} \in \{0,1\}$(used for the linearization of some constraints and takes value related with x_{ij})

w_i: waiting time at node i

t_i: arrival time at node i

z_i: vehicle's load coming to node i

Objective Function:

$$\min \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij} \qquad (3.1)$$

Subject to:

$$\sum_{i \in N} x_{ij} = 1 \qquad \forall j \in N' \qquad (3.2)$$

$$\sum_{j \in N} x_{ij} = 1 \qquad \forall i \in N' \qquad (3.3)$$

$$\sum_{j \in N'} x_{0j} - \sum_{i \in N'} x_{i0} = 0 \qquad \forall i \in N, \ \forall j \in N \qquad (3.4)$$

$$t_i + w_i \geq e_i \qquad \forall i \in N \qquad (3.5)$$

$$t_i + w_i \leq l_i \qquad \forall i \in N \qquad (3.6)$$

$$z_i - d_i - z_j \leq M y_{ij} \qquad \forall i \in N, \ \forall j \in N \qquad (3.7)$$

$$-z_i + d_i + z_j \leq M y_{ij} \qquad \forall i \in N, \ \forall j \in N \qquad (3.8)$$

$$x_{ij} \leq M(1 - y_{ij}) \qquad \forall i \in N, \ \forall j \in N \qquad (3.9)$$

$$z_j \leq Q \qquad \forall i \in N \qquad (3.10)$$

$$t_i + s_i + w_i + t_{ij} - t_j \leq M y_{ij} \qquad \forall i \in N,\ \forall j \in N \qquad (3.11)$$

$$-t_i - s_i - w_i - t_{ij} + t_j \leq M y_{ij} \qquad \forall i \in N,\ \forall j \in N \qquad (3.12)$$

The objective is the minimization of the total distance traveled by all vehicles. Constraints (3.2) and (3.3) ensure that each customer is visited exactly once and (3.4) ensures that the number of vehicles leaving the depot should be equal to the number of vehicles arriving at the depot. Constraints (3.5) and (3.6) are the time window constraints. No vehicle can arrive to a customer i after l_i. It can arrive before e_1 but it should wait until e_i. Constraints (3.7), (3.8) and (3.9) are the linear formulations of the following constraint:

$$If\ x_{ij} = 1, \quad then \quad Z_i - d_i - Z_j = 0\ , \qquad (3.13)$$

which ensures the demand equalities of the vehicles. A vehicle's load leaving customer i is equal to the vehicle's load coming to customer i - demand at customer i. Constraints (3.10) are the capacity constraints for each vehicle. Constraints (3.11), (3.12) with equation (3.9) are the linear formulations of the following constraint:

$$If\ x_{ij} = 1\ , \quad \text{then} \qquad t_i + S_i + w_i + t_{ij} - t_j = 0, \qquad (3.14)$$

which ensures the time equalities of the vehicles visiting customers,

3.2. Hybrid Heuristics for the VRPTW

The heuristic algorithms that are proposed in this thesis are composed of a number of strategies for both initial solution and improvement.

The heuristics proposed in the literature generally find a feasible and moderate initial solution mostly based on same strategies and focus on the improvement part of the algorithm. Finding a good initial solution would intuitively increase the probability of reaching less costly solutions easily (by spending less time). Considering this situation, emphasis is given to the initial solution and the algorithm is tried to be designed by considering more complex strategies in order to achieve good solutions.

All of the algorithms consist of two parts:

1. Initial Solution Heuristic
    - Parallel Version
    - Sequential Version
2. Improvement Heuristic (includes a restart mechanism)

As most heuristic strategies, the proposed heuristics involve finding an initial feasible solution and than improving that solution using local optimization techniques. On the other hand, the proposed algorithms have some different features like center selection mechanism in the initial solution and diversification mechanism in the improvement part.

3.2.1. Initial Solution Heuristics

The first part of all proposed heuristics is a construction heuristic that aims to find an as good as possible solution that is used in the improvement part. The initial solution algorithm has both parallel and sequential versions that are based on the same strategy. In the sequential algorithm, one route is constructed initially and others are constructed when necessary, while in parallel algorithm, many routes are constructed simultaneously.

3.2.1.1. Parallel Initial Solution Heuristic

The parallel version of the construction part is based on the strategy of clustering the customers while doing insertion so that minimum distance is traveled. First, a lower bound is calculated for the number of routes (vehicles) and lower bound number of centers are selected within the customers to construct routes as *Depot-Center-Depot* initially.

The algorithm has four steps:

1. Calculation of a lower bound for the number of routes (vehicles): The first step determines a lower bound in order to initialize a number of routes for the insertion of unassigned customers. The lower bound is calculated as below:

$$\left\lceil \frac{\text{Total Demand of All Customers}}{\text{Capacity of One Vehicle}} \right\rceil \qquad (3.16)$$

$m =$

The number of vehicles (routes) cannot be less than m, because all demand must be satisfied by the vehicles. If all vehicles are used in full capacity, there should be at least m vehicles.

2. Center selection: In the second step, the algorithm selects centers that are distant from each other (geographically). Centers are intuitively thought as a main point for a group of customers and are used for clustering the customers that are close to the selected centers. This strategy can be useful in clustering the customers because the customers that are geographically dispersed will have little possibility of appearing in the same routes (the centers selected will initialize different routes) and the customers close to the center will appear on the same route. At the beginning of the algorithm, since there are no previously selected centers, the first center is the customer that is farthest to the depot. The second center is the unassigned customer that is farthest to the depot and the previously selected center (total distance is used here). This step goes on until there are lower bound number of centers are selected. So, there are centers $c_1, C_2, \ldots, c_m$ selected at the end of this step.

3. Initialization of the routes: There are lower bound number of centers selected at the previous step. These centers are used to initialize different routes. The routes constructed are as follows:

0 - c_1 - 0

0 - c_2 - 0

0 - c_m - 0

where 0 is the depot. All routes start and end at the depot as it is defined it the model.

4. Insertion of customers to the routes: At the last step of the initial solution algorithm, all feasible locations in all routes are searched for inserting each unassigned customer. Each unassigned customer has a best feasible insertion point at this step of the algorithm. The whole unassigned customer set is searched then and the customer and related best feasible location that gives the minimum cost increase is chosen and the customer is inserted to that location. The minimum cost increase formulation is given in the pseudo-code in Appendix A and is based on Solomon (1987). This is an iterative procedure that goes on until all customers are assigned to the routes but it can be the case that the initial routes are not enough for inserting all customers in a feasible way. If there are no feasible points for any customer, a new route is constructed as a center by selecting the customer that is farthest from the depot.

3.21.2. Sequential Initial Solution Heuristic

Sequential initial solution is based on the same strategy with the parallel version: clustering the customers while doing insertion so that minimum distance is traveled.

Main differences from the parallel initial solution can be stated as follows:

- Initial routes are not generated simultaneously. One route is generated and others are added when necessary.
- Since there is no simultaneous route generation no lower bound is determined.
- Insertion is done on the current routes; not all the routes are searched for feasible insertion since the current route is used when there is no feasible insertion point on the previous route.

Sequential construction of the routes allows determining clusters within the algorithm, but parallel algorithm determines the clusters at the beginning. Selecting centers at the beginning can be disadvantageous sometimes because the selected centers are always in different clusters and it might be better for them to appear in the same

routes. This is not possible with the parallel algorithm. This situation is tried to be overcome with the sequential algorithm.

There are two steps of the algorithm:

1. Route Initialization: The first route is initialized by selecting the customer farthest from the depot. Other routes are initialized (when necessary) by selecting the customer who has the maximum total Euclidean distance to the previously selected customers and the depot.

2. Insertion: It is the same procedure with the parallel algorithm, but insertion is done over the current routes. There is no search done over many routes as it is in the parallel algorithm since a new route is constructed when a route cannot be used for insertion anymore.

The insertion procedure is as follows:

From the set of unassigned customers, each customer is tried to be inserted at all feasible locations in the current route and a cost increase is calculated for the insertion of that customer to the referred location at that route. So, each customer has a best feasible insertion point (some may not have). From these customers and their best feasible insertion cost, the minimum is selected and that customer is inserted to the related location that gives the calculated cost. The procedure goes on until all customers are assigned. In the case that there are no feasible points for any customer for insertion, a new route is constructed as (0 - *center* - 0) by selecting the customer that gives the maximum total Euclidean distance to the previously assigned customers and the depot.

For the details of the algorithm, the reader is referred to the pseudo-code of the algorithm in Appendix B.

3.2.2. Improvement Heuristics

In the second part, based on the initial solution found in the first part, the solution is tried to be improved with a search heuristic. The improvement part contains alternative local search algorithms that use three operators and a restart mechanism

applied in different ways. From the viewpoint of local search, the most important issue is designing an algorithm that does not stick into local optimum values. So, the diversification methodology lets nearly the whole solution space to be explored in order to get better solutions.

In this thesis, two improvement algorithms are proposed by making sure that the running times are reasonable.

3.2.2.2. Algorithm 1

The first improvement algorithm uses the neighborhoods below and is a combination of them.

- Inter-Route Exchange Neighborhood: Exchange of two customers between routes.
- Inter-Route Move Neighborhood: Moving one customer from one route to another
- Intra-Route Exchange Neighborhood: Exchange of two customers in the same route.

The proposed algorithm is a combination of three improvement procedures and a diversification procedure. The algorithm has three main steps:

1. Inter-route Exchange and Inter-Route Move Operators Applied Together: This step is a nested procedure that uses two operators together. As it can be seen in detail in the pseudo-code, the move operator is applied over routes for one tour of all routes on all customers sequentially and then exchange operator is applied for one tour on all routes. It can be explained step by step as follows:

- The search starts with the last route. The first customer of the last route is chosen. The depot is not included in the search since all routes begin and end with the depot.

- The selected customer is tried to be inserted to a feasible location in all routes sequentially (from last route to first) except its own route. Feasibility conditions have to be considered at this point. If a route's capacity is not enough to accept the selected customer, this route is skipped and search goes on with the next route. If capacity constraint is satisfied, then time feasibility conditions of the locations in that route is checked. If a location is not time feasible for the selected customer to be inserted, the next location in the same route is considered.

- If a feasible insertion point is found for the selected customer, it has to be checked whether there is a cost improvement by making this move. If a feasible location is not cost improving, next location on the same route is checked. If there are no feasible and cost improving locations in a route, next route is searched in the same way. The customer is inserted to the first feasible and cost improving location.

- Necessary updates are made and the algorithm continues with the customer next to the previously moved customer. If a customer cannot be inserted to any feasible and cost improving point, the next customer in the route is considered.

- This procedure is applied until all locations of all routes are considered for moves. One tour ends when all locations are searched.

- This is one iteration of the first step. After the completion of one tour, inter-route moves part is terminated.

- After the inter-route move tour, inter-route exchange tour is applied in the same way. All of the search methodology is same with the inter-route moves, but here, the customer at the location that is chosen for the insertion has to be considered since it is also moved form its place. Capacity and time feasibility checks are also applied to that customer. If a time feasible and cost improving location is found, the selected customer is moved to that location and the customer at that location is inserted to the place of the selected customer. Necessary updates are made, all procedure is the same with the move operator.

- One tour of exchanges ends when all routes and locations are considered for exchanges.

- This procedure is iterated until a local optimum value is caught. This means that the algorithm comes to the beginning of the inter-route moves unless there are no feasible assignments. If there are no feasible and cost improving assignments, a local optimum is reached and part 1 is terminated.

2. Intra-Route Exchange Operator Applied: Since the first step is a procedure that is designed for the search with exchange and moves between different routes, it is necessary to apply a procedure that searches for less costly solutions within the routes. When the first step reaches a local optimum, the second step starts and exchanges of customers in the same route are applied sequentially. The search can be explained as follows:

- The first customer of the first route is chosen and searched for feasible insertion points in the same route. The capacity constraint is not considered since the exchanges are applied in the same route. (Only the sequence of the customers in the routes is changed). If a time feasible location is found, the selected customer is moved to that location and the customer at that location is moved to the place of the selected customer.

- If a feasible and cost improving exchange is applied, the search continues with the customer next to the previously selected customer.

- One tour of intra-route exchanges ends when all routes and all locations are considered. This step iterates itself (several tours of search is applied) until a local optimum is reached. This means that the algorithm comes to the beginning of intra-route exchanges unless there are no feasible and cost improving exchanges. The details of the sequential search structure can be explored in the pseudo-code.

Using three operators and using two of them nested allow making different moves and this lets the algorithm search for various solutions. It is assumed that applying

different moves at the same time increases the search quality. When the first step reaches an optimum value, it is necessary to search for better solutions within the routes. Changing the sequence of customers in the same route may result with less costly solutions.

3. Diversification: Applying the first and second steps only is not sufficient for designing an algorithm that gives high quality solutions for the VRPTW. As it was mentioned before, a restart mechanism lets the algorithm search for more solutions and this increases the possibility of finding better ones.

When the local search method cannot produce any better solutions, it is believed that a local optimum has been reached. In order to escape the local optimum and explore a larger solution space, a diversification mechanism is proposed in the algorithm. The current local optimal solution is destroyed (diversified) by making some exchanges between the routes. The current solution jumps up to a neighbor and this is a new initial point. The new solution is used as an initial solution and the local search is restarted. The most important constraint of the diversification method is that during the process, only feasible exchanges are accepted, but it is not important whether they improve the solution (decrease the cost) or not.

The diversification method that is proposed in this thesis is one tour of inter-route exchanges that was explained in part 1. The only difference is that cost change is not considered. If capacity and time feasibility constraints are satisfied, exchange is applied. When one tour of exchanges ends, the diversification part is terminated. The diversification can be applied in two different ways by changing the sequence of the search.

The diversification mechanism is applied several times each time the local search reaches a local optimum. Solution is updated if the cost is decreased at the end of a local search. The number that the process is repeated is a parameter of the algorithm. The bigger the parameter is, the longer the computational time. Given enough time, this method can search sufficiently large solution space and find near optimal solutions.

Details of the algorithm 1 are given in the pseudo-code in Appendix C.

3.2.2.2.Algorithm 2

The second improvement algorithm proposed uses the same neighborhoods with the first algorithm, but there are differences in the use of these operators.

The main differences of two algorithms are as follows:

- Algorithm 1 uses the inter-route exchange and move operators together and apply intra-route exchanges after a local optimum is reached in the first step, but Algorithm 2 uses all these three operator together.

- Algorithm 1 has two stages and there are two local optima found; whereas Algorithm 2 reaches one local optimum.

In the design of the first algorithm, inter-route exchanges are considered after optimal customers are assigned to the routes. This may give good solutions but may have a side effect on the performance of the search. Therefore, another algorithm (Algorithm 2) is designed that uses three operators together that does not limit the search between routes and consider the sequence of the customer in the routes when applying inter-route exchanges and moves.

Algorithm 2 has two main steps:

1. Inter-Route Exchanges, Inter-Route Move and Intra-Route Exchange Applied Together: The algorithm starts with an initial solution and applies one sequential tour of inter-route moves. Then, search procedure in one tour is the same with algorithm 1. Here, as in the first algorithm, sequential means that the algorithm searches for feasible moves starting from the last route and ending at the first route. There are no random moves. After one sequential tour on inter-route moves, inter-route exchanges are applied in the same methodology. When the exchanges are finished (one tour over all routes and customers are applied), one tour of intra-route exchanges is performed. So, a complete tour of the algorithm is finished at this point. If there are no feasible and cost improving assignments at this point, the algorithm terminates.

Otherwise the search starts form the beginning and several tours are applied until a local optimum is reached.

2. Diversification: Diversification module of the second algorithm is the same as the first algorithm.

The algorithm is explained in Appendix D in detail.

3.2.2.3. Diversification with Probability

The diversification method that is proposed in both Algorithm 1 and Algorithm 2 is based on the idea of escaping local optima in order to continue the search. The solution is diversified by making feasible exchanges without considering the cost changes. By making a number of exchanges, the solution comes to a new initial point that is generally worse than the local optimum value found. If the solution is diversified so much, this may decrease the possibility of finding better solutions since the search jumps up to a point that is greatly different. Although it may be advantageous to jump up a very different point in order to search a wide area, another diversification method that limits the destruction of the local optimum is proposed.

Difference of the new diversification method is the use of probability for escaping local optima. The first method jumps to a feasible point without considering the cost increases, but the new method accepts more costly points with a probability. This probability must not be high in order to be meaningful. It is better to choose the probability parameter near 0, but high enough to escape the local optimum.

The method can be generalized s follows:

- If a capacity and time feasible exchange ends with cost decrease or does not affect the total cost, this exchange is accepted.

- If a capacity and time feasible exchange ends with a cost increase, it is accepted with probability p, which is close to 0.

Other parts of the probabilistic diversification are the same as the diversification with probability, which was explained earlier. The basic concept of this method has some similarities with simulated annealing, but there is a constant probability here. Simulated annealing goes on the search with probabilities that is controlled with a reducing temperature.

The pseudo-code for the probabilistic diversification part is provided in Appendix E.

4. COMPUTATIONAL STUDY

4.1. Benchmark Problems

In order to compare the proposed algorithms with other heuristics proposed in the literature, the six data sets of Solomon (1987) are used. These are commonly accepted as the main benchmark for the VRPTW algorithms.

There are 56 problems made up of 100 customers in the data set. Vehicle capacity, customer time windows, service time and coordinates vary so as to cover all configurations as thoroughly as possible. Thus, customers may be uniformly distributed (problem sets Rl and R2), clustered (problem sets Cl and C2), or a mix of the two patterns (problem sets RC1 and RC2). Problem sets Rl, Cl and RC1 have a narrow scheduling horizon, tight time windows and low vehicle capacity. Conversely, problem sets R2, C2 and RC2 have a large scheduling horizon, wide time windows and a high vehicle capacity. In these data sets, travel times correspond to Euclidean distances.

The test problems provide an opportunity to compare the solution quality of the algorithms with various heuristic approaches. Since it is an NP-Hard problem, only a few of the instances are solved optimally in the literature (See Appendix F for the optimal values).

4.2. Parameters Used in the Algorithms

In this section, parameters used in the algorithms are explained.

4.2.1, Diversification Number (*M*)

Number of the diversification is determined by considering two factors:

- Solution Quality: The bigger the diversification number is, the highet is the probability of reaching better (less costly) solutions in general.
- Computational Time: The bigger the diversification number is, the longer is the computational time.

So the diversification parameter has to be chosen in a way that solution quality is high but the computational time is not long. By applying several runs to various instances, it is seen that there is almost no improvement after the 200^{th} diversification and the computational times are efficient at this number. So, the diversification number is set as 200.

4.2.2. Diversification Probability *(p)*

As it is stated in the diversification with probability module, a small probability (close to 0) is used when diversifying solution. By applying experimental runs to different problems, it turns out that very small probabilities (like 0.0001 and 0.001) do not guarantee diversifying the solution to a new point. So the search may not escape from the local optima using small probabilities. On the other hand, the probabilities that are not close to 0 act as if there is no probability in the diversification. Based on the initial runs and the behavior of the algorithm, the diversification probability is set to 0.01.

4.3. Comparison of the Algorithms with Benchmark Heuristics

The computational study includes the combinations of the proposed algorithms illustrated in Table 4.1. All of the algorithms are coded in Visual C++ 6 and executed on a PC with Celeron 800 MHz. 128 RAM.

Table 4.1 Hybrid heuristics generated with the proposed algorithms

Hybrid Algorithm	Initial Solution Algorithm	Improvement Algorithm	Diversification Method
1	Parallel	Algorithm 1	Diversification without Probability
2	Parallel	Algorithm 1	Diversification with Probability
3	Parallel	Algorithm 2	Diversification without Probability
4	Parallel	Algorithm 2	Diversification with Probability
5	Sequential	Algorithm 2	Diversification without Probability
6	Sequential	Algorithm 2	Diversification with Probability
7	Sequential	Algorithm 1	Diversification without Probability
8	Sequential	Algorithm 1	Diversification with Probability

Since the hybrid algorithms that use parallel insertion heuristic do not give competitive results, these algorithms (1, 2, 3 and 4) are not presented here, but the results of the parallel initial solution can be seen in Appendix F. Although parallel initial solution heuristic finds better solutions for some of the benchmark problems, the results after improvement applied are not that good.

The results of the hybrid algorithms that use the sequential initial solution heuristic are presented in Appendix G in detail. Since algorithm 8 does not give good results compared to the other ones, these results are not presented either.

In order to test the performance of the algorithms, comparison with some competing heuristics is given in Appendix H. in detail. Also, Table 4.2 gives the

number of results that are better than and equal to these algorithms' results, and Table 4.3 shows the number of results that are better than these algorithms'.

Comparison is also done based on the results of the algorithms that are within a percentage deviation from the benchmark algorithms. Tables 4.4, 4.5, 4.6 and 4.7 show the number of instances that are within the indicated percentage deviation of the compared algorithm.

Benchmark heuristics are as follows (The details of these papers are given in Chapter 2):

- Potvin and Bengio (1996): Genetic algorithm
- Tan *et al.* (2001): Tabu search
- Li and Lim (2002): Simulated annealing-like restarts (best results of different algorithms are published.)
- Backer *et al.* (2000): Guided tabu search

Table 4.2 Number of instances and the related percentages that are better than or equal to the benchmark heuristics (out of 56 instances)

Algorithm	Tan *et al.* (2001)	Potvin & Bengio (1996)	Backeret *al.* (2000)	Li & Lim (2002)
Hybrid Algorithm 5	35 (62.5%)	29 (52%)	5 (8.9%)	7(12.5%)
Hybrid Algorithm 6	45 (80.4%)	33 (58.9%)	10(17.9%)	10(17.9%)
Hybrid Algorithm 7	40 (71.4%)	34 (60.7%)	10(17.9%)	9(16.1%)
Best of Three	46(82.1%)	37(66.1%)	14(25%)	14 (25%)

Table 4.3 Number of instances and the related percentages that are better than the benchmark heuristics (out of 56 instances)

Algorithm	Tan *et al.* (2001)	Potvin & Bengio (1996)	Backeref *al.* (2000)	Li & Lim (2002)
Hybrid Algorithm 5	34 (60.7%)	28 (50.0%)	5 (8.9%)	6(10.7%)
Hybrid Algorithm 6	44 (78.6%)	32(57.1%)	9(16.1%)	9(16.1%)
Hybrid Algorithm 7	37(66.1%)	30 (53.6%)	4 (8.9%)	5 (8.9%)
Best of Three	42 (75.0%)	32(57.1%)	10(17.9%)	10(17.9%)

Table 4.4 Number of instances and the related percentages that are within 1% deviation of the benchmark heuristics (out of 56 instances)

Algorithm	Tan *et al.* (2001)	Potvin & Bengio (1996)	Backeref *al.* (2000)	Li & Lim (2002)
Hybrid Algorithm 5	37(66.1%)	31 (55.4%)	8 (14.3%)	8 (14.3%)
Hybrid Algorithm 6	44 (78.6%)	35 (62.5%)	11 (19.6%)	13(23.2%)
Hybrid Algorithm 7	40 (71.4%)	37(66.1%)	11 (19.6%)	13(23.2%)
Best of Three	47 (83.9%)	40(71.4%)	16(28.6%)	18(32.1%)

Table 4.5 Number of instances and the related percentages that are within 2% deviation of the benchmark heuristics (out of 56 instances)

Algorithm	Tan *et al.* (2001)	Potvin & Bengio (1996)	Backeref *al.* (2000)	Li & Lim (2002)
Hybrid Algorithm 5	39 (69.6%)	34 (60.7%)	10(17.9%)	13(23.2%)
Hybrid Algorithm 6	44 (78.6%)	35 (62.5%)	15(26.8%)	16(28.6%)
Hybrid Algorithm 7	43 (76.8%)	39 (69.6%)	15(26.8%)	17(30.4%)
Best of Three	47 (83.9%)	41 (73.2%)	18(32.1%)	22 (39.3%)

Table 4.6 Number of instances and the related percentages that are within 5% deviation of the benchmark heuristics (from 56 instances)

Algorithm	Tan *et al.* (2001)	Potvin & Bengio (1996)	Backer *et al.* (2000)	Li & Lim (2002)
Hybrid Algorithm 5	43 (76.8%)	40(71.4%)	22 (39.3%)	24 (42.9%)
Hybrid Algorithm 6	46(82.1%)	41 (73.2%)	31 (55.4%)	34 (60.7%)
Hybrid Algorithm 7	47 (83.9%)	42 (75.0%)	29(51.8%)	29 (51.8%)
Best of Three	49 (87.5%)	44 (78.6%)	37(66.1%)	39 (69.6%)

Table 4.7 Number of instances and the related percentages that are within 10% deviation of the benchmark heuristics (from 56 instances)

Algorithm	Tan *et al.* (2001)	Potvin & Bengio (1996)	Backeref *al.* (2000)	Li & Lim (2002)
Hybrid Algorithm 5	48 (85.7%)	46(82.1%)	46(82.1%)	45 (80.4%)
Hybrid Algorithm 6	50 (89.3%)	48 (85.7%)	47 (83.9%)	47 (83.9%)
Hybrid Algorithm 7	48 (85.7%)	45 (80.4%)	42 (75.0%)	43 (76.8%)
Best of Three	52 (92.9%)	49 (87.5%)	49 (87.5%)	46(82.1%)

The results of all three algorithms proposed are obviously better than the genetic algorithm proposed by Potvin and Bengio and TS proposed by Tan *et al.* (2001) When the results are compared with the simulated annealing-like restarts of Li and Lim (2002) and guided tabu search Backer *et al.*(2000), they are relatively worse. Results for few cases are competing with the results of Li and Lim's and Backer *et al,* but there are a number of instances that are better than these algorithms. When the tables from 4.3 to 4.7 are examined, it can be seen that the results are very close to the results of the algorithms proposed by Backer *et al.* and Li and Lim. Most of the instances are within %10 deviation. It should also be noted that Li and Lim proposes many algorithms but gives only the best results. So, this should be taken into consideration when making a comparison.

Table 4.8 Average % divergence of the algorithms from the algorithm of Potvin and Bengio (1996) (Negative (-) divergence mean that the proposed algorithm gives better results.)

Problem Group	Hybrid Algorithm 5	Hybrid Algorithm 6	Hybrid Algorithm 7	Best of Three Algorithms
R1	-2.07	-3.59	-3.36	-4.28
R2	-10.36	-11.29	-9.41	-11.55
C1	16.84	11.34	17.72	10.86
C2	8.85	9.81	7.07	4.63
RC1	0.82	-0.97	0.38	-1.05
RC2	-13.87	-16.13	-12.87	-16.35

Table 4.9 Average % divergence of the algorithms from the algorithm of Tan *et al.* (2001) (Negative (-) divergence mean that the proposed algorithm gives better results.)

Problem Group	Hybrid Algorithm 5	Hybrid Algorithm 6	Hybrid Algorithm 7	Best of Three Algorithms
R1	-0.52	-2.08	-1.81	-2.77
R2	-8.49	-9.41	-7.48	-9.68
C1	12.79	7.38	13.53	6.93
C2	2.07	3.14	0.22	-1.92
RC1	-0.45	-2.19	-0.87	-2.27
RC2	-9.60	-11.99	-8.53	-12.25

Table 4.10 Average % divergence of the algorithms from the algorithm of Li and Lim (2002)

Problem Group	Hybrid Algorithm 5	Hybrid Algorithm 6	Hybrid Algorithm 7	Best of Three Algorithms
R1	3.95	2.32	2.58	1.59
R2	3.47	2.45	4.65	2.13
C1	18.21	12.64	19.10	12.15
C2	9.02	9.97	7.23	4.79
RC1	5.21	3.36	4.76	3.28
RC2	2.77	0.004	3.91	-0.23

Table 4.11 Average % divergence of the algorithms from the algorithm of Backer *et al.* (2000) (Negative (-) divergence means that the proposed algorithm gives better results.)

Problem Group	Hybrid Algorithm 5	Hybrid Algorithm 6	Hybrid Algorithm 7	Best of Three Algorithms
R1	4.93	3.30	3.56	2.57
R2	4.56	3.53	5.78	3.22
C1	18.06	12.49	18.94	12.01
C2	8.80	9.75	7.01	4.57
RC1	4.71	2.89	4.29	2.81
RC2	1.52	-1.11	2.74	-1.41

4.4. Comparison of the Proposed Algorithms with the Best Known

In order to test the performance and solution quality of the algorithms proposed, the results have been compared with the best known results of the Solomon instances in the literature.

The error percentages of the 5th, 6th, and 7th hybrid algorithms from the best known are also presented in Appendix G. The error percentages are the deviations from the best known. Table 4.12 shows the average deviations form the best known values of each instance. Table 4.13 gives the number of instances that are better than or equal to the best known values in the literature.

Table 4.12 Average deviations of the algorithms from the best known in the literature

Problem Group	Hybrid Algorithm 5	Hybrid Algorithm 6	Hybrid Algorithm 7	Best of Three Algorithms
R1	7.45	5.78	6.04	5.03
R2	8.58	7.50	9.81	7.18
C1	18.32	12.74	19.20	12.25
C2	9.02	9.97	7.23	4.79
RC1	8.57	6.64	8.07	6.57
RC2	11.50	8.74	12.73	8.39

When the three algorithms are compared with each other, it may be seen that hybrid algorithm 6, which uses probabilistic diversification, sequential initial solution and improvement algorithm 2, gives better results. Probabilistic diversification does not diversify the solutions to absolutely different points, but lets the algorithm escape from the local optima. This makes the search go to better solutions easily.

In general, the algorithms do not perform very well for the problem set Cl, but are very efficient for the problem set Rl. When we look at the overall best results of the three algorithms, we can see that the algorithms generated give good results when compared with the best known in the literature. Although there are few results that are better than the best known, they are very close to the best known values (The differences are not that large). Comparison with the best known can be explored in Appendix G.

Table 4.13 Number of instances that are better than, better than or equal to the best known in the literature, and within a percentage deviation form the best known.

Algorithm	better than	better than or equal to	within 1% deviation	within 2% deviation	within 5% deviation	within 10% deviation
Hybrid Algorithm	1 (1.8%)	1 (1.8%)	1 (1.8%)	2 (3.6%)	11 (19.6%)	39 (69.6%)
Hybrid Algorithm	2 (3.6%)	3 (5.4%)	4(7.1%)	5 (8.9%)	18(32.1%)	40(71.4%)
Hybrid Algorithm 7	0 (0%)	6(10.7%)	6(10.7%)	6(10.7%)	17(30.4%)	35 (62.5%)
Best of Three	4(7.1%)	8 (14.3%)	8 (14.3%)	8(14.3%)	27 (48.2%)	42 (75%)

It should be noted that, computational time is not the main focus of this study. In general, we can say that the proposed hybrid algorithms are not inefficient in terms of running time. Although the computational time changes from problem to problem and problem set to problem set, the range is 15-45 minutes and the average is 35 minutes approximately.

During the study, several algorithms are designed and various computational experiments are performed in order to test the efficiency of these algorithms. Some of them are not included in this thesis such as the ones that were mentioned in Section 4.3. Another group of algorithm that is not discussed here is SA. Standard SA procedure is applied by adding a temperature function and probabilities, but the results are not promising and skipped here.

5. CONCLUSION

The purpose of this study was to develop heuristic algorithms for solving the VRPTW. Since exact algorithms are not sufficient for solving large VRPTW problems, heuristics are generally used in the literature. Most of the heuristics first find an initial solution and then improve the solution by using local search techniques.

At the initial solution part, the aim is to find an as good as possible solution to be improved later. The initial solution heuristics proposed in the literature are generally based on the same strategies, that is, the initial solution part is not the main focus. In this study, two initial solution heuristics that use different and more complex strategies are proposed. Initial solution part gives good results, but the benchmark of the results is not possible since no results exist in the literature.

After finding an initial solution, local search techniques are used for improving the solution. The main concern at this point is to be able to escape the local optima. Classical local search algorithms generally stick into local optimum values unless they include a restart mechanism. There have been several methods for escaping the local optima. In the recent years, metaheuristics are popularly used instead of classical local search algorithms since they are capable of exploring wide areas in the search space.

The improvement algorithms proposed in this thesis use classical local search operators such as exchanges and moves, but these operators are nested in each other and used together. Another point is that the improvement part is supported with a restart mechanism called *diversification* in order to escape the local optima and widen the search space. There are two diversification methods proposed in the study.

The proposed hybrid algorithms include the combinations of the initial solution, improvement and diversification methods.

Solomon (1987) instances are used as a benchmark and the results of the hybrid heuristics are compared with some known heuristics and the best known results of the problems in the literature. The results of the proposed heuristics are generally good when compared with the benchmark heuristics, but they are not that competitive with the best published results on the individual problem basis.

As a consequence, using hybrid heuristics that includes various methods are successful at solving VRPTW. The heuristics give better results than some known metaheuristics in the literature.

In this study, Solomon's benchmark problems are used, but it will be interesting as a subject of further research to use real data in order to test the proposed heuristics although benchmark is not possible. As a matter of fact, Solomon's 56 problem instances have been used in almost all papers as a benchmark since 1987. Hence, extending this set of problems to cover a wide variety of problem instance properties probably with more customers is also a potential research area.

From a practical point of view, there are various applications of the VRP. Adding the time window constraints make the framework more general to handle this type of situations. On the other hand, when one tries to solve a specific application, it is very likely that new side constraints will be added making the problem even harder. The basic ideas proposed in this thesis may be a good starting point for such problems.

Another further research area is designing effective restarting methods for escaping local optima, which is the main concern of the VRPTW.

6. REFERENCES

1. Aarts, E., Lenstra, J.K., *Lomi Search in Combinatorial Optimization,* Wiley, New York, 1997.

2. Antes, J., Derigs U., "A new parallel tour construction algorithm for the vehicle routing problem with time windows," Technical Report, Lehrstuhl fur Wirtschaftsininformatik und Operations Research, Universtat zu Köln, 1995.

3. Backer, B., Furnon, V., Kilby P., Prosser P., Shaw, P., "Solving Vehicle Routing Problems using Constraint Programming and Metaheuristics," *Journal of Heuristics,* vol.6, no.4, pp. 501-525, 2000.

4. Badeau, P.,Gendreaou, M., Guertin,F., Potvin J., Taillard E.D. "A parallel tabu search heuristic for the vehicle routing problem with time windows," *Transportation Research,(55),* pp. 109-122, 1997.

5. Berger, J., Sassi, M., Salois, M., "A hybrid genetic algorithm for the vehicle routing problem with time windows and Itinerary Constraints," *GECCO,* pp.44-51, 1999.

6. Blanton, J.L., Wainwright, R.L., "Multiple vehicle routing with time and capacity constraints using genetic algorithms," *ICGA,* pp. 452-459, 1993.

7. Bramle, J., Simchi-Levi, D., The Logic of Logistics: theory, algorithms, and applications for logistics management, Springer, New York, 1997.

8. Braysy, O., "Genetic algorithms for the vehicle routing problem with time windows," Technical Report, Department of Mathematics and Statistics, University of Vaasa, Finland, 1999.

9. Braysy, O., Berger, J., Barkaoi, M., "A new hybrid evolutionary algorithm for the vehicle routing problem with time windows," Presented at the *Route 2000-Workshop,* Skodsborg, Denmark, 2000.

10. Chiang, W., Russell, R. "Simulated annealing metaheuristics for the vehicle routing problem with time windows," *Annals of Operations Research,* (63), pp.3-27,1996.

11. Chiang, W., Russell, R. "A reactive tabu search metaheuristic for the vehicle routing problem with time windows", *INFORMS Journal on Computing,* (9), pp.417-430, 1997.

12. Christofides, N.,Beasley E., "The period routing problem," *Networks,* vol.14, no.2, pp.237-256, 1984.

13. Clarke, G., Wright W., "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research,* (12), pp.568-581, 1964

14. Cordone, R., Calvo, W.R., "Note about time window constraints in routing problems," Internal Report, 96-005, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, 1996.

15. Croes, G.A., "A method for solvmg the traveling salesman problems," *Operations Research,* (6), pp.791-812, 1958.

16. Dantzig, G.,B, Ramser, J.H., "The truck dispatching problem," *Management Science,* vol.6, no.l, pp.58-64,1959.

17. Davis, A., *Handbook of Genetic Algorithms,* Van Strand Reinhold , New York, 1991.

18. Deitel, H.M., Deitel, P.J., C++, *How to Program,* Prentice Hall, New Jersey, 2001.

19. Desrochers, M., Desroiers, J.,Solomon, M.M., "A new optimization algorithm for the vehicle routing problem with time windows," *Operation Research,* (40), pp.342-354, 1992.

20. Fisher,M.L., Jornsten, K.O., Madsen, O.B.G., "Vehicle routing with time windows : two optimization algorithms," *Operations Research,* vol.45,no.3,1997

21. Fogel, M., *How to Solve it: Modern Heuristics,* Springer, New York, 2000.

22. Foisy, C, Potvin J., "Iniplemeting an insertion heuristic for vehicle routing on parallel hardware," *Computers & Operations Research,* vol.20, no.7, pp.737-745, 1993.

23. Garcia, B.D., Potvin J., Rousseau J., "A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints," *Computers & Operations Research,* vol.21, no.9, pp.1025-1033, 1994.

24. Gillet, E., Miller, L.R., "A heuristic algorithm for the vehicle routing dispatch problem," *Operational Research,* 22, pp.340-349,1974.

25. Glover, F., "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research,* 13, pp.533-549, 1986.

26. Glover, F., Laguna, M., "Tabu search," *Modern Heuristic Techniques for Combinatorial Problems,* Blackwell, Oxford, pp.76-150, 1993.

27. Glover, F., McMillan, R. "The general employee scheduling problem: an integration of MS and Al," *Computers & Operations Research,* vol.13, no.5, pp.563-573, 1986.

28. Homberger, J., Gehring, H., "Two evolutionary metaheuristics for the vehicle routing problem with time windows," *INFOR,* (37), pp.297-318, 1999.

29. Jornsten, K.O., Madsen, O.B.G., Sorensen, B., "Exact solution of the vehicle routing and scheduling problem with time windows by variable splitting," Technical Report, Department of Mathematical Modeling, Technical University of Denmark, 1986.

30. Halze, K., "Modeling and solving complex vehicle routing problems," Phd. Thesis, Department of Mathematical Modeling, Technical University of Denmark, 1992.

31. Kilby, P., Prosser,P., Shaw, P., "Guided local search for the vehicle routing problems with time windows," *Metaheuristics: Advances and Trends in Local Search for Optimization,* Kluwer Academic Publishers, pp.473-486, Boston, 1999.

32. Kirkpartick, S., Gellat, CD., Vecchi, P.M., "Optimization by simulated annealing," *Science,* (220), pp.671-680, 1983.

33. Kohl, N., "Exact methods for time constrained routing and scheduling problems," Phd. Thesis, Department of Mathematical Modeling, Technical University of Denmark, 1995.

34. Kohl, N., Madsen O., "An optimization algorithm for the vehicle routing problem with time windows based on Lagrangean Relaxation," *Operations Research,* (45), pp.395-406, 1997.

35. Kolen, A., Rinnooy A., Trienekens, H., "Vehicle routing with time windows," *Operations Research,* (35), pp.266-273, 1987

36. Landeghem, H., "A bi-criteria heuristic for the vehicle routing problem with time windows," *European Journal of Operational Research,* (36), pp.217-226, 1988

37. Larsen, J., "Parallelization of the vehicle routing problem with time Windows," PhD. Thesis, Technical University of Denmark, Lyngby, 1999.

38. Li, H., Lim, A., "Local Search with annealing-like restarts to solve the VRPTW," *European Journal of Operational Research,* Corrected Proof, Article in Press., 2003.

39. Madsen, O.B.G., "Variable splitting and vehicle routing problem with time windows," Technical Report 1A/1988, Department of Mathematical Modeling, Technical University of Denmark, 1988.

40. Metropolis, N., Rosenbluth, A., Rosenbluth M., Teller, A., Teller, E., "Equations of state calculations by fast computing machines," *Journal of Chemical Physics,* (21), pp.1087-1091, 1983.

41. Or, I., "Traveling salesman-type combinatorial optimization problems and their relation to the logistics of blood banking," Phd. Thesis, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL., 1976.

42. Osman, I.H.,Christofides, N., "Capacitated clustering problem by hybrid simulated annealing and tabu search," *International Transaction in Operational Research,* vol.1, no.3, pp.317-336, 1994.

43. PotvinJ., Bengio, S., "The vehicle routing problem with time windows-part II: genetic search," *INFORMS Journal on Computing,* (8), pp. 165-172, 1996.

44. Potvin, J., Kervahut, T., Garcia, B.L., Rousseau, J.M., "The vehicle routing problem with time windows; part I: tabu search," *INFORMS Journal on Computing,* (8), pp.158-164, 1995.

45. Potvin, J., Robillard, C, "Clustering for vehicle routing with a competitive neural network," *Neurocomputing,* (8), pp.125-139, 1999.

46. Potvin, J., Rousseau J.M., "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows," *European Journal of Operational Research,* (66), pp.331, 1993

47. Potvin, J., Rousseau J.M., "An exchange heuristic for routing problems with time windows," *Journal of Operational Research Society,* (46), pp.1433-1446, 1995.

48. Reeves, R.C., Modern Heuristic Techniques for Combinatorial Problems, Mc Graw Hill, New York, 1995.

49. Rochat, Y., Taillard, E.D., "Probabilistic Diversification and intensification in local search for vehicle routing," *Journal of Heuristics,* (1), pp. 147-167, 1995.

50. Russell, R., "Hybrid heuristics for the vehicle routing problem with time windows," *Transportation Science,* (29), pp. 156-166, 1995

51. Savelsbergh, M.W.P., "Local search for routing problems with time windows," Annals *of Operations Research,* (4), pp.285-305, 1985.

52. Savelsbergh, M.W.P., "An efficient implementation of local search algorithms for constrained routing," *European Journal of Operational Research,* (47), pp.75,85, 1990

53. Schulze,J., Fahle, T., "A parallel algorithm for the vehicle routing problem with time window constraints," *Combinatorial Optimization: Recent Advances in Theory and Praxis, Special volume of Annals of Operations Research,* 86, pp.585-607,1999.

54. Smith, R., Osman, I.H., Reeves, C.R., Smith, G.D., *Modern Heuristic Search Methods,* Wiley, New York, 1996.

55. Solomon, M.M., "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations Research,* vol.35, no.2, pp.254-265, 1987

56. Tan, K.C., Lee, L.H., Zhu, Q.L., Ou, K., "Heuristic methods for vehicle routing problem with time windows," *Artificial Intelligence in Engineering,* (15), pp.281-295,2001.

57. Tan, K.C., Lee, L.H., Zhu, Q.L., Ou, K., "Artificial intelligence heuristics in solving vehicle routing problems with time windows," *Engineering Applications of Artificial Intelligence,* (14), pp.825-837, 2001.

58. Thangiah, S., "Vehicle routing with time windows using genetic algorithms," Technical Report, SR4-CPSC-TR-93, 23, Computer Science Department, Slippery Rock University, Slippery Rock, PA, 1993.

59. Thangiah, S., Osman, I.H., Sun, T., "Hybrid genetic algorithm, simulated annealing and tabu search method for vehicle routing problems with time windows," Technical Report, UCK/OR94/4, Institute of Mathematics and Statistics, University of Kent, Canterbuiy, UK, 1994.

60. Toth, P.,Vigo, D., "Models, relaxations and exact approaches for the capacitated vehicle routing problem," *Discrete Applied Mathematics,* vol.123, no.1-3, pp.487-512,2002.

7. APPENDICES

Appendix A: Pseudo-Code for the Parallel Initial Solution Algorithm

1. *[Calculate Lowerbound]*

   Lowerbound = *m* = Total Demand / Capacity of a Vehicle

2. *[Center selection]*

   Select depot as center *0*

   For (n= 1 to *m)*

   {

   For each unassigned customer *i*

   {

   Calculate distance *d*

   d = Sum of the Euclidean distances to the previously selected centers and to the depot

   }

   Select customer with maximum *d* as center

   Necessary updates

   }

3. *[Initialization of the Routes]*

   For (n=1 to *m)*

   {

   Generate Routes (0 - s_n - 0)

   }

4. *[Insertion]*

   Start: For each unassigned customer *i*

{

For each route j

{

If the route is full (if any customer cannot be inserted),

Next route

Check capacity feasibility for insertion of customer i to route j

If not feasible (Current Capacity of Route $j + d_i > Q$),

Next route

For each location k at route j

{

Check time feasibility conditions at location k at route j for customer i

If not feasible ($t_{location[j][k]} + w_{location[j][k]} > l_{location[j][k]}$)

{

Next location at the same route

}

If time and capacity constraints are satisfied,

($t_{location[j][k]} + w_{location[j][k]} \leq l_{location[j][k]}$) & (Current Capacity of Route $j + d_i \leq Q$)

{

calculate cost change Δc_{k-1k}

$\Delta c_{k-1k} = c_{k-1i} + c_{ki} - c_{k-1k}$

}

}

Select the location with minimum cost increase Δc_{k-1k} at each route for customer i

}

Select the overall minimum cost increase Δc_{k-1k} location for customer i

}

Select the customer with minimum cost increase Δc_{k-1k}

Insert selected customer to the selected location for the customer

Necessary updates

Go to *Start*

If there is no feasible location for any customer,

{

Construct a new route [0 - s_{new} - 0] by selecting the customer farthest to the depot go

to *Start*

}

If all customers are assigned, *terminate.*

Appendix B: Pseudo-Code for the Sequential Initial Solution Algorithm

1. [Route Initialization]

For each unassigned customer *i*

{

Calculate distance *d*

d = Sum of the Euclidean distances to the previously assigned customers and to the depot

}

Select customer *i* with maximum *d*

Update the *Current Route* (0 - *i* - 0)

2. [Insertion]

Start: For each unassigned customer *i*

{

For the *Current Route*

{

Check capacity feasibility for insertion of customer *i* to the *Current Route*

If not feasible (Current Capacity of Route *j* + *di* > *Q*),

next customer For each

location *k* at the *Current Route*

{

Check time feasibility at location *k* at the *Current Route* for customer *i If* not feasible

$(t_{location[cuurent][k]} + w_{location[current][k]} > l_{location[j][k]})$,

{

next location at the *Current Route*

}

If time and capacity constraints are satisfied

$(t_{location[cuurent][k]} + w_{location[current][k]} \leq l_{location[j][k]})$ & Current Capacity of Route $j + d_i \leq Q$)

{

Calculate cost change $\Delta c_{k\text{-}1k}$

$\Delta c_{k\text{-}1k} = c_{k\text{-}1i} + c_{ki} - c_{k\text{-}1k}$

}

}

Select the location with minimum cost increase $\Delta c_{k\text{-}1k}$ for customer i

}

}

Select the customer with minimum cost increase $\Delta c_{k\text{-}1k}$

Insert selected customer to the selected location

Necessary updates Go to *Start*

If there are no feasible points for any customer on the *Current Route,*

{

go to Step 1 for route initialization

}

If all customers are assigned, *terminate.*

Appendix C: Pseudo-Code for the Improvement Algorithm 1

Part 1: Inter-Route Exchange and Inter-Route Move Applied Together *e4:*

II The algorithm starts with one tour of sequential inter-route moves. Since the initial solution generates more routes at the end, the latterly created routes are searched first.

For (j = number of routes to 0)

{

For (k = 1 to *size of the route- 1)*

II The depot is not included in the search since all the routes should start and end with the depot

{

For (w = *number of routes* to 0)

{

For (*e* = 1 to *size of the route -1)*

{

Capacity check for the move of the customer at the point *[j] [k]* to the point *[w][e];*

If it's not possible, go to *ell;*

II If the capacity of route *w* is not enough for the insertion of the customer at the location *[j] [k]* the next route is searched.

Time feasibility and cost improvement ($\Delta c_{(e-1)(e)}$) check;

$\Delta c_{(e-1)(e)} = c_{(e-1)(k)} + c_{(e)(k)} - c_{(e-1)(e)} + c_{(k-1)(k+1)} - c_{(k-1)(k)} - c_{(k)(k+1)}$

If it's not possible, go to *e21;*

// If a the location *[w][e],* which means the e^{th} location of route *w,* is not appropriate in terms of time feasibility and cost improvement for the insertion of the customer at location *[j][k],* this customer is then tried to be inserted to location *[w][e+1].*

If possible,
{
insert customer at the point *[j][k]* to the point *[w][e];*
necessary updates;
go *to e31;*
}

// If it is feasible and cost improving to insert customer at the location *[j][k]* to the location *[w]'[e]',* do the insertion and necessary updates of assigned and unassigned customer sets. After a feasible insertion, the search continues with the customer at the location *[j][k+1].* When route j is finished, route *j*-1 is searched until all routes are searched.

e21: }
e11: }
e31: }
}

II At this point inter-route moves of one tour ends (all routes are searched for one time and the algorithm continues with the next step.)

// After one tour of moving customers between routes, inter-route exchange operator is used for one tour

For (j=number of routes to 0)

// Since the initial solution generates more routes at the end, the latterly created routes are searched first.

{

For (k=1 to *size of the route- 1)*

II The depot is not included in the search since all the routes should start and end with the depot

{

For (w=number of routes to 0)

{

For (*e*=1 to *size of the route-1)*

{

Capacity check for the exchange of the customer at the point *[j][k]* with the customer at the point *[w] [e]; If* it's not possible, go to *e22;*

II If the customer at the location *[j][k]* cannot be exchanged (in terms of capacity feasibility) with the customer at the location *[w][e],* then the customer at the location *[w][e+ 1]* is tried.

Time feasibility and cost improvement check;

$\Delta c_{(e-1)(e)} = c_{(e)(k-1)} + c_{(e)(k+1)} - c_{(k)(k-1)} - c_{(k+1)(k)} + c_{(e-1)(k)} + c_{(k)(e+1)} - c_{(e)(e-1)} - c_{(e)(e+1)}$

If it's not possible, go to *e22;*

II If the customer at the location *[j][k]* cannot be exchanged (in terms of time feasibility and cost improvement) with the customer at the location *[w][e],* then the customer at the location *[w][e+1]* is tried.

if possible,

{

replace customers at the point *[j] [k]* and *[w][e];*

necessary updates; go
to *e32;*

}

II After a feasible insertion, the search continues with the customer at the location *[j][k+1].* When route *j* is finished, route *j*-1 is searched until all routes are searched.

e22: }

el2: }

e32: }

}

// At this point exchange of one tour ends (all routes are searched for one time).

If there can be more feasible cost improving assignments, go to *e4;*
else, end part 1;

// If there can be more feasible and cost improving assignments, the search restarts itself from the beginning of part 1. If there are no feasible and cost improving assignments, a local optimum is reached and part 1 terminates.

Part 2: Intra-Route Exchange Applied:

// In the second part, intra-route exchange (exchanges of customers in the same route) is applied. The procedure repeats itself until a local optimum is reached.

e6:

For (*j*=0 to *number of routes)*

{

For (k=1 to *size of the route-1)*

II The depot is not included in the search since all the routes should start and end with the depot. Capacity check is not necessary since there is no move or exchange between routes.

{

For (e=1 to *size of the route*-1*)*

{

Time (Δc_{e-1e}) check;

$$\Delta c_{(e-1)(e)} = c_{(k-1)(e)} + c_{(e)(k+1)} + c_{(k)(e-1)} + c_{(k)(e+1)} - c_{(k)(k-1)} - c_{(k)(k+1)} - c_{(e)(e-1)} - c_{(e)(+1)}$$

// If it is not feasible and cost improving to exchange the customers at the location *[j][k]* and *[j][e],* continue searching with the customer at the location *[j]'[e+1].*

If it's not possible, go to *e13;*

impossible,

{

replace customers at the point *[j][k]* and

[j][e]; necessary updates;

// If it is feasible and cost improving to exchange the customers at the location *[j][k]* and *[j][e],* apply the exchange and make necessary updates. The search continues with the customer *[j][k+1]*

Go to *e33;*

e13: }

e33: }

}

II At this point exchange of one tour ends (all routes are searched for one time).

If there can be more feasible assignments, go to *e6;*

else, end part 2;

// If there can be more feasible and cost improving assignments, the search restarts itself from the beginning of part 2. If there are no feasible and cost improving assignments, a local optimum is reached and part 2 terminates.

Part 3 : Diversification

// Since Part 1 and 2 ended, the local search makes no feasible cost improving moves. Diversification is used in order to destroy the current solution and jump up to a neighbor.

For (*j* = 0 to *number of routes)*
{
For (k=1 to *size of the route-1)*
{
For (w=0 to *number of routes)*
{
For (e=l to *size of the route-1)*
{
Capacity check for the exchange of the customer at the point *[j][k]* with the customer at the point *[w][e];*

II If the customer at the location *[j][k]* cannot be exchanged (in terms of capacity feasibility) with the customer at the location *[w][e],* then the customer at the location *[w][e+1]* is tried.

If it's not possible, go to *e24;*
Time feasibility check
If it's not possible, go to *e24;*

// If the customer at the location *[j][k]* cannot be exchanged (in terms of time feasibility) with the customer at the location *[w][e],* then the customer at the location *[w][e+1]* is tried.

impossible,
{
replace customers at the point *[j][k]* and *[w][e];*
Necessary updates;
Go to *e34;*
}

// If it is feasible to exchange the customers at the location *[j][k]* and *[w][e],* apply the exchange and make necessary updates. Diversification continues with the customer *[j][k+1].* It should be noted that there is no cost improvement check here, since the diversification uses feasible moves regardless of cost improvement.

e24: }
e14: }
e34: }
}

// Diversification is done over one tour. There is no return to the beginning of Part 3.

If (diversification<M)
diversification++;
go to *e4;*
If (diversification=M)
terminate.

II The current solution is diversified and if this procedure is not applied for *M* times, the search restarts itself from the beginning of Part 1. If diversification is applied for *M* times, the algorithm terminates.

Appendix D: Pseudo-Code for the Improvement Algorithm 2

Inter-Route Move, inter-route Exchange and Intra-Route exchange Applied Together

e4:

Inter-Route Move:

II The algorithm starts with one tour of sequential inter-route moves. Since the initial solution generates more routes at the end, the latterly created routes are searched first.

For (j=number of routes to 0)

{

For (k = 1 to *size of the route-1)*

II The depot is not included in the search since all the routes should start and end with the depot

{

For (w = *number of routes* to 0)

{

For (*e*=l to *size of the route-l)*

{

Capacity check for the move of the customer at the point *[j] [k]* to the point *[w] [e]; If* it's not possible, go to *e11;*

II If the capacity of route w is not enough for the insertion of the customer at the location *[j][k],* the next route is searched.

Time feasibility and cost improvement ($\Delta c_{(e-1)(e)}$) check;

$\Delta c_{(e-1)(e)} = c_{(e-1)(k)} + c_{(e)(k)} - c_{(e-1)(e)} + c_{(k-1)(k+1)} - c_{(k-1)(k)} - c_{(k)(k+1)}$

If it's not possible, go to *e21;*

II If a the location *[w][e],* which means the e^{th} location of route *w,* is not appropriate in terms of time feasibility and cost improvement for the insertion of the customer at location *[j][k],* this customer is then tried to be inserted to location *[w][e+1].*

impossible,
{
insert customer at the point *[j][k]* to the point *[w][e];*
necessary updates;
go to
e31; }

II If it is feasible and cost improving to insert customer at the location *[j][k]* to the location *[w][e],* do the insertion and necessary updates of assigned and unassigned customer sets. After a feasible insertion, the search continues with the customer at the location *[j][k+1].* When route j is finished, route *j-1* is searched until all routes are searched.

e21 } *e11:* }
e31: }
}

II At this point move of one tour ends (all routes are searched for one time and the algorithm continues with the next step).

// After one tour of moving customers between routes, inter-route exchange operator is used for one tour

Inter-Route Exchange:

For (j=number of routes to 0)

// Since the initial solution generates more routes at the end, the latterly created routes are searched first.

{

For (k=1 to *size of the route*-1*)*

II The depot is not included in the search since all the routes should start and end with the depot

{ *For (w=number of routes* to 0)

{

For (*e=1* to *size of the route-1)*

{

Capacity check for the exchange of the customer at the point *[j][k]* with the customer at the point *[w][e]; If* it's not possible, go to *e22;*

II If the customer at the location *[j][k]* cannot be exchanged (in terms of capacity feasibility) with the customer at the location *[w][e],* then the customer at the location *[w][e+1]* is tried.

Time feasibility and cost improvement check;

$$\Delta C(e\text{-}1)(e) = C(e)(k\text{-}1) + C(e)(k+1) - C(k)(k\text{-}1) - C(k+1)(k) + C(e\text{-}1)(k) + C(k)(e+1) - C(e)(e\text{-}1) - C(e)(e+1)$$

If it's not possible, go to *e22;*

II If the customer at the location *[j][k]* cannot be exchanged (in terms of time feasibility and cost improvement) with the customer at the location *[w][e],* then the customer at the location *[w][e+1]* is tried.

*If p*ossible,

{

replace customers at the point *[j][k]* and

[w][e]; necessary updates; go to *e32;*

}

// After a feasible insertion, the search continues with the customer at the location *[j][k+1].* When route j is finished, route *j*-1 is searched until all routes are searched.

e22: }

e12: }

e32: }

}

// At this point exchange of one tour ends (all routes are searched for one time and the algorithm continues with the next step.)

// After one tour of exchanging customers between routes, intra-route exchange operator is used for one tour

Intra-Route Exchange:

For (*j*=0 to *number of routes)*

For (k=l to *size of the route-1)*

II The depot is not included in the search since all the routes should start and end with the depot. Capacity check is not necessary since there is no move or exchange between routes.

{

For (e=1 to *size of the route-1)*

{

Time $(\Delta c_{e\text{-}1e})$ check;

$$\Delta c_{(e-1)(e)} = c_{(k-1)(e)} + c_{(e)(k+1)} + c_{(k)(e-1)} + c_{(k)(e+1)} - c_{(k)(k-1)} - c_{(k)(k+1)} - c_{(e)(e1)} - c_{(e)(+1)}$$

II If it is not feasible and cost improving to exchange the customers at the location *[j][k]* and *[j][e],* continue searching with the customer at the location *[j][e+1].*

If it's not possible, go to el3;

If possible,

{

replace customers at the point *[j][k]* and*[j][e];*

necessary updates;

// If it is feasible and cost improving to exchange the customers at the location *[j][k]* and *[j][e],* apply the exchange and make necessary updates. The search continues with the customer *[j][k+1]*

Go to *e33;*

}

e13: }

e33: }

}

II At this point one overall tour ends (all routes are searched for one tour with three operators).

If there can be more feasible assignments, go to *e4;*

else, end part 1;

// If there can be more feasible and cost improving assignments, the search restarts itself from the beginning. If there are no feasible and cost improving assignments, a local optimum is reached and part 2 terminates.

Part 2 : Diversification

// Since Part 1 and 2 ended, the local search makes no feasible cost improving moves. Diversification is used in order to destroy the current solution and jump up to a neighbor.

For (/=0 to *number of routes)*

{

For (k=1 to *size of the route-X)*

{

For (w=0 to *number of routes)*

{

For (e=l to *size of the route*-l *)*

{

Capacity check for the exchange of the customer at the point *[j][k]* with the customer at the point *[w][e];*

II If the customer at the location *[j][k]* cannot be exchanged (in terms of capacity feasibility) with the customer at the location *[w][e],* then the customer at the location *[w][e+1]* is tried.

If it's not possible, go to *e24;*

Time feasibility check

If it's not possible, go to *e24;*

II If the customer at the location *[j][k]* cannot be exchanged (in terms of time feasibility) with the customer at the location *[w][e],* then the customer at the location *[w][e+1]* is tried.

If possible, {

replace customers at the point *[j][k]* and *[w][e];*

necessary updates; go to *e34;*

}

II If it is feasible to exchange the customers at the location *[j][k]* and *[w][e],* apply the exchange and make necessary updates. Diversification continues with the customer *[j][k+1].* It should be noted that there is no cost improvement check here, since the diversification uses feasible moves regardless of cost improvement

e24: *}*

e14: *}*

e34: *} }*

II Diversification is done over one tour. There is no return to the beginning of Part 3.

If (diversification<M)

diversification++;

go to *e4;*

If (diversification=M)

terminate.

II The current solution is diversified and if this procedure is not applied for *M* times, the search restarts itself from the beginning. If diversification is applied for *M* times, the algorithm terminates.

Appendix E: Pseudo-Code for the Diversification with Probability

For (*j*-0 to *number of routes)*

{

For (k=l *to size of the route-1)*

{

For (w=0 to *number of routes)*

{

For (e=l to *size of the route-1)*

{

Capacity check for the exchange of the customer at the point *[j][k]* with the customer at the point *[w][e];*

II If the customer at the location *[j][k]* cannot be exchanged (in terms of capacity feasibility) with the customer at the location *[w][e],* then the customer at the location *[w][e+1]* is tried.

If it's not possible, go to *e24;*

Time feasibility check *If* it's not possible, go to *e24;*

II If the customer at the location *[j][k]* cannot be exchanged (in terms of time feasibility) with the customer at the location *[w][e],* then the customer at the location *[w][e+1]* is tried.

If possible, calculate cost. change for replacing the customers at the points *[j][k]* and [w][e];

If (Cost Change $\leq$ 0)

{

replace customers at the points *[j][k]* and *[w][e];*

necessary updates; go to
e34; }

If(Cost Change > 0)
{
probability =*p;*
generate a random number *r* between 0 and 1;
If (r ≤ p)
{
replace customers at the points *[j][k]* and *[w][e];*
necessary updates;
go to *e34;* }
If (r > p)
{
go to *e34;*
}

II If it is feasible to exchange the customers at the location *[j][k]* and *[w][e],* cost change is calculated. If there is a cost improvement, apply the diversification and make necessary updates. If the cost increases, apply the diversification with probability *p.* Diversification continues with the customer *[j][k+1].* It should be noted that there is no cost improvement check here, since the diversification uses feasible moves regardless of cost improvement.

e24: }
el4: }
e34: }
}

//Diversification is done over one tour. There is no return to the beginning of Part 3.

If (diversification<M)
diversification++;
go to *Start;*
If (diversification=M)
terminate.

II The current solution is diversified and if this procedure is not applied for M times, the search restarts itself from the beginning of the algorithm. If diversification is applied for M times, the algorithm terminates.

Appendix F: Computational Results of the Initial Solution Heuristics*

	Initial-S		Initial-P			Initial-S		Initial-P	
Problem	NV	TC	NV	TC	Problem	NV	TC	NV	TC
R101	54	3629.82	54	3507.89	C106	24	2378.2	24	2398.62
R102	47	3370.61	43	3183.11	C107	21	1982.02	23	2307.41
R103	32	2679.61	29	2426	C108	18	1928.14	19	2095.32
R104	23	2046.61	18	1745.51	C109	13	1647.77	16	1980.17
R105	47	3331.66	41	3141.77	C201	21	2381.35	20	2190.39
R106	43	3159.35	34	2770	C202	16	2135.17	16	2006.1
R107	31	2473.6	25	2120.67	C203	15	1840.1	15	1787.82
R108	24	2087.77	18	1762.12	C204	12	1430.88	9	1316.01
R109	34	2984.79	30	2792.11	C205	14	1841.87	19	2060.69
R110	25	2425.25	26	2301.71	C206	10	1615.92	11	1633.46
R111	29	2659	22	2139.73	C207	12	1574.67	11	1619.78
R112	17	1740.83	17	1715.16	C208	8	1293.14	10	1629.81
R201	17	2063.74	17	2063.74	RC101	54	5000.92	54	4833.52
R202	15	2031.79	15	2031.79	RC102	46	4542.91	46	4267.82
R203	11	1688.82	11	1688.82	RC103	34	3631.59	32	3453.33
R204	9	1313.91	9	1313.91	RC104	22	2679.33	22	2679.33
R205	9	1680.19	9	1680.19	RC105	48	4072.92	48	4072.92
R206	7	1422.85	7	1422.85	RC106	45	4529.5	45	4529.5
R207	7	1371.93	7	1371.93	RC107	34	3641.69	34	3641.69
R208	5	1229.02	5	1229.02	RC108	28	3139.59	28	3139.59
R209	8	1529.05	8	1529.05	RC201	21	3044.09	21	3044.09
R210	9	1627.2	9	1627.2	RC202	17	2580.36	17	2580.36
R211	6	1185.86	6	1185.86	RC203	13	2172.34	13	2122.34
C101	28	2498.43	30	2760.22	RC204	7	1396.04	7	1394.04
C102	23	2225.88	27	2483.81	RC205	15	2617.93	15	2617.93
C103	22	2266.42	20	2095.04	RC206	13	2238.63	13	2238.63
C104	15	1652.86	16	1600.97	RC207	9	2197.01	9	2197.01
C105	26	2286.35	25	2687.35	RC208	6	1617.33	6	1617.33

* Better values of total cost are bold.

Appendix G Comparison of the Proposed Hybrid Heuristics' Results with the Best Known Results in the Literature*

										% Deviation from Best known				
	best of our	hybrid alg.7		hybrid alg.5		hybrid alg.6		Best known		Alg. 7	Alg. 5	Alg. 6	Optimal	
Problem		NV	TC	NV	TC	NV	TC	NV	TC	%	%	%	NV	TC
R101	1686.23	21	1686.23	21	1746.58	21	1735.74	18	1607.7	4.88	8.64	7.96	18	1604.5
R102	1552.52	19	1588.64	19	1575.43	18	1552.52	17	1434	10.78	9.86	8.26	11	904.6
R103	1275.09	14	1292.13	14	1286.24	14	1275.09	13	1207	7.05	6.57	5.64	9	765.4
R104	1032.06	11	1039.81	11	1038	11	1032.06	10	982.01	5.89	5.70	5.10	-	-
R105	1425.52	16	1458.4	16	1471.98	15	1425.52	14	1377.11	5.90	6.89	3.52	-	-
R106	1287.64	13	1293.34	14	1344.28	13	1287.64	12	1252.03	3.30	7.37	2.84	-	-
R107	1116.83	11	1116.83	11	1153.94	11	1140.31	10	1004.7	11.16	14.85	13.50	-	-
R108	987.825	10	994.96	10	997.998	10	987.825	9	960.88	3.55	3.86	2.80	-	-
R109	1213.12	13	1213.12	13	1225.8	13	1233.7	11	1163.95	4.22	5.31	5.99	-	-
R110	1126.77	12	1145.07	12	1157.69	12	1126.77	11	1080.36	5.99	7.16	4.30	-	-
R111	1113.05	11	1136.49	11	1151.75	12	1113.05	10	1086.48	6.01	6.01	2.45	-	-
R112	1002.54	11	1002.54	11	1021.93	11	1020.92	10	953.63	5.13	7.16	7.06	-	-
R201	1292.19	6	1302.89	6	1292.19	6	1311.51	4	1252.37	4.03	3.18	4.72	-	-
R202	1124.12	6	1178.95	6	1124.12	6	1130.42	3	1091.9	4.60	2.95	3.53	-	-
R203	972.667	5	986.057	5	980.509	4	963.886	3	939.54	7.97	4.36	2.59	-	-
R204	818.763	4	843.735	4	818.763	4	819.466	2	779.56	8.23	5.03	5.12	-	-
R205	1022.49	6	1043.88	6	1091.02	4	1022.49	3	994.42	4.97	9.71	2.82	-	-
R206	987.698	5	987.698	5	995.963	5	993.372	3	833	18.57	19.56	19.25	-	-
R207	906.678	3	928.179	3	920.492	3	906.678	3	814.78	13.92	12.97	11.28	-	-
R208	789.132	4	791.84	4	795.225	4	789.132	2	726.75	8.96	9.42	8.58	-	-
R209	947.121	5	991.728	5	947.121	5	953.106	3	855	15.99	10.77	11.47	-	-
R210	1007.76	5	1025.21	5	1024.13	4	1007.76	2	939.34	9.14	9.03	7.28	-	-
R211	835.361	5	884.48	5	854.272	5	835.361	2	795.92	11.33	7.33	4.96	-	-
C101	828.94	10	828.94	11	1014.74	10	848.894	10	827.3	0.20	22.66	2.61	10	827.3
C102	1003.26	12	1003.92	11	1015.82	11	1003.26	10	827.3	21.35	22.79	21.27	10	827.3
C103	949.601	11	1022.51	11	1013.54	10	949.601	10	828.06	23.48	22.40	14.68	10	826.3
C104	961.707	10	1021.53	10	1022.07	10	961.707	10	824.78	23.85	23.92	16.60	10	822.9
C105	886.247	11	993.85	11	915.72	10	886.247	10	828.94	19.89	10.47	6.91	10	827.3
C106	893.799	11	985.656	11	893.799	11	909.804	10	827.3	19.14	8.04	9.97	10	827.3
C107	1026.8	11	1064.41	11	1037.3	10	1026.8	10	827.3	28.66	25.38	24.11	10	827.3
C108	876.713	11	958.148	11	930.1	11	876.713	10	828.94	15.50	12.20	5.76	10	827.3
C109	934.271	11	1000.16	11	969.829	10	934.271	10	828.94	20.66	17.00	12.71	10	827.3
C201	591.557	3	591.557	4	649.216	4	654.925	3	591.56	0.00	9.75	10.71	-	-
C202	627.798	4	660.576	4	642.543	4	627.798	3	591.56	11.67	8.62	6.13	-	-
C203	653.762	4	692.946	4	691.642	4	653.762	3	591.17	17.22	17.00	10.59	-	-
C204	683.009	4	697.958	4	683.009	4	687.986	3	590.6	18.18	15.65	16.49	-	-
C205	588.876	3	588.876	4	617.225	4	623.559	3	588.88	0.00	4.81	5.89	-	-
C206	588.876	4	617.364	4	624.829	3	588.876	3	588.49	4.91	6.17	0.07	-	-

										% Deviation from Best known				
	best of our	hybrid alg.7		hybrid alg.5		hybrid alg.6		Best known		Alg. 7	Alg. 5	Alg. 6	Optimal	
Problem		NV	TC	NV	TC	NV	TC	NV	TC	%	%	%	NV	TC
C207	622.71	4	622.71	4	636.899	4	722.406	3	588.29	5.85	8.26	22.80	-	-
C208	588.49	3	588.49	4	599.484	4	630.142	3	588.32	0.03	1.90	7.11	-	-
RC101	1721.88	17	1769.65	17	1787.95	16	1721.88	14	1635.8	8.18	9.30	5.26	-	-
RC102	1546.77	14	1554.8	14	1546.77	14	1556.49	12	1554.75	0.00	-0.51	0.11	-	-
RC103	1362.91	13	1367.44	13	1394.49	12	1362.91	11	1110	23.1	25.6	22.78	-	-
RC104	1222.05	11	1228.97	11	1229.28	11	1222.05	19	1135.83	8.20	8.23	7.59	-	-
RC105	1594.28	16	1625.06	16	1623.56	16	1594.28	13	1564.4	3.88	3.78	1.91	-	-
RC106	1456.92	13	1496.04	13	1474.8	13	1456.92	11	1407.1	5.01	3.52	2.26	-	-
RC107	1322.21	13	1344.29	13	1336.84	12	1322.21	11	1230.54	9.24	8.64	7.45	-	-
RC108	1191.06	11	1202.93	11	1238.63	11	1191.06	10	1139.82	5.54	8.67	4.50	-	-
RC201	1423	6	1423	6	1435.03	6	1436.06	4	1046.94	35.9	37.0	37.17	-	-
RC202	1228.49	5	1296.5	5	1228.49	5	1246.05	4	1162.9	11.4	5.64	7.15	-	-
RC203	1034.89	4	1156.43	4	1098.86	5	1034.89	3	1049.6	10.18	4.69	-1.40	-	-
RC204	894.478	4	935.766	4	920.66	4	894.478	3	798.41	17.20	15.31	12.03	-	-
RC205	1270.24	4	1343.13	4	1376.76	7	1270.24	4	1297.2	3.54	6.13	-2.08	-	-
RC206	1186.25	5	1236.86	5	1216.52	5	1186.25	3	1146.3	7.90	6.13	3.49	-	-
RC207	1124.92	4	1128.09	4	1150.5	5	1124.92	3	1061.1	6.31	8.43	6.01	-	-
RC208	890.406	4	905.114	4	899.211	4	890.406	3	828.14	9.29	8.58	7.52	-	-

* Bold cells are the best results of the three hybrid algorithms, gray cells shows the results that are better than or equal to the best known results in the literature.

Appendix H: Comparison of the Proposed Hybrid Heuristics' Results with the Benchmark Algorithms in the Literature*

	algorithm/		algorithms		algorithm6		Li & Lim (2002)		Tan *et* a/.(2000)		Potvin & Bengio(1996)		Backer et a/. (2000)	
Problem	NV	TC	NV	TC	NV	TC	NV	TC	NV	TC	NV	TC	NV	TC
R101	21	1686.23	21	1746.58	20	1736	19	1650.8	20	1707.95	19	1784	19	1652.7
R102	19	1588.64	19	1575.43	18	1553	17	1486.41	16	1488.59	17	1777	18	1475.4
R103	14	1292.13	14	1286.24	14	1275	13	1291.68	15	1293.85	14	1433	14	1235
R104	11	1039.81	11	1038	11	1032	9	1007.31	11	1057.02	10	1065	10	992.46
R105	16	1458.4	16	1471.98	16	1426	14	1381.37	16	1431.56	14	1421	14	1431.3
R106	13	1293.34	14	1344.28	13	1288	12	1269.72	14	1331.5	12	1353	12	1258.3
R107	11	1116.83	11	1153.94	11	1140	10	1104.66	12	1174.89	11	1191	11	1094.4
R108	10	994.96	10	997.998	10	988	9	986.25	11	1039.34	10	993	9	971.98
R109	13	1213.12	13	1225.8	13	1234	11	1208.96	14	1256.36	12	1205	12	1164
R110	12	1145.07	12	1157.69	12	1127	10	1159.35	13	1179	11	1136	11	1100.4
R111	11	1136.49	11	1151.75	12	1113	11	1066.32	13	1148	11	1184	11	1086.5
R112	11	1002.54	11	1021.93	11	1021	10	967.88	11	1088.32	10	1020	10	985.91
R201	6	1302.89	6	1292.19	6	1312	4	1252.37	5	1437.49	4	1554	4	1262.8
R202	6	1178.95	6	1124.12	6	1130	4	1084.77	5	1272.6	3	1530	4	1091.9
R203	5	986.057	5	980.509	5	973	3	949.39	4	1081.04	3	1201	3	950.55
R204	4	843.735	4	818.763	4	819	2	849.05	3	895.86	3	904	3	779.56
R205	6	1043.88	6	1091.02	4	1022	3	1032.55	4	1150.34	3	1159	3	1038.3
R206	5	987.698	5	995.963	5	993	3	931.62	4	1103.22	3	1066	3	942.68
R207	3	928.179	3	920.492	3	907	2	905.13	3	1007.3	3	954	3	847.54
R208	4	791.84	4	795.225	4	789	2	732.8	3	806.70	2	759	2	794.28
R209	5	991.728	5	947.121	5	953	3	930.59	4	1110.3	3	1108	3	938.36
R210	5	1025.21	5	1024.13	5	1008	3	1018.95	4	1071.3	3	1146	3	939.37
R211	5	884.48	5	854.272	5	835	3	801.81	3	946.35	3	913	3	795.92
C101	10	828.94	11	1014.74	10	849	10	828.94	10	828.93	10	829	10	828.94
C102	12	1003.92	11	1015.82	11	1003	10	828.94	10	901.52	10	829	10	829.86
C103	11	1022.51	11	1013.54	10	950	10	828.06	10	954.71	10	875	10	831.6
C104	10	1021.53	10	1022.07	10	962	10	824.78	10	895.77	10	865	10	825.54
C105	11	993.85	11	915.72	10	886	10	828.94	10	828.93	10	829	10	829.7
C106	11	985.656	11	893.799	11	910	10	828.94	10	941.15	10	829	10	829.7
C107	11	1064.41	11	1037.3	11	1027	10	828.94	10	828.93	10	829 .	10	830.92
C108	11	958.148	11	930.1	11	877	10	828.94	10	828.93	10	829	10	829.7
C109	11	1000.16	11	969.829	10	934	10	828.94	10	828.93	10	829	10	828.94
C201	3	591.557	4	649.216	4	655	3	591.56	3	591.55	3	592	3	591.56
C202	4	660.576	4	642.543	4	628	3	591.56	4	745.99	3	592	3	591.56
C203	4	692.946	4	691.642	4	654	3	591.17	4	727.221	3	592	3	591.56
C204	4	697.958	4	683.009	4	688	3	590.6	3	590.59	3	592	3	599.29
C205	3	588.876	4	617.225	4	624	3	588.88	3	588.87	3	591	3	588.88
C206	4	617.364	4	624.829	4	589	3	588.49	3	588.49	3	589	3	588.49
C207	4	622.71	4	636.899	4	722	3	588.29	3	600.841	3	589	3	588.29
C208	4	588.49	4	599.484	4	630	3	588.32	3	645.20	3	589	3	588.32
RC101	17	1769.65	17	1787.95	17	1722	15	1658.62	16	1734.17	15	1676	15	1635.8
RC102	14	1554.8	14	1546.77	14	1556	13	1513.6	14	1562.62	13	1671	14	1597.4
RC103	13	1367.44	13	1394.49	12	1363	11	1319.99	13	1377.93	11	1401	11	1297.5
RC104	11	1228.97	11	1229.28	11	1222	10	1141.9	11	1259.28	10	1204	10	1153.1
RC105	16	1625.06	16	1623.56	16	1594	13	1637.62	16	1597.67	14	1670	15	1564.4
RC106	13	1496.04	13	1474.8	13	1457	11	1424.73	14	1476.15	12	1486	13	1407.1
RC107	13	1344.29	13	1336.84	12	1322	11	1240.66	13	1392.97	11	1275	11	1276.1
RC108	11	1202.93	11	1238.63	11	1191	10	1147.42	12	1264.5	11	1187	10	1268.9
RC201	6	1423	6	1435.03	6	1436	4	1425.21	5	1617.5	4	1799	4	1435.5
RC202	5	1296.5	5	1228.49	5	1246	3	1374.27	5	1429.04	4	1603	4	1162.9
RC203	4	1156.43	4	1098.86	4	1035	3	1088.53	4	1179.67	3	1253	3	1093.7
RC204	4	935.766	4	920.66	4	894	3	818.66	4	939.67	3	1002	3	870.48
RC205	4	1343.13	4	1376.76	7	1270	4	1304.64	5	1487.49	4	1693	4	1306.4
RC206	5	1236.86	5	1216.52	6	1186	3	1159.03	4	1357.32	3	1324	3	1267.3
RC207	4	1128.09	4	1150.5	4	1125	3	1107.16	4	1295.9	3	1222	3	1155.2
RC208	4	905.114	4	899.211	4	890	3	862.34	3	1040.47	3	1049	3	905.25

* Bold cells are the best results of the three hvbrid algorithms.