Sergejs Pugacs

Master Thesis

A Clustering Approach for Vehicle Routing Problems with Hard Time Windows

Dissertação para obtenção do Grau de Mestre em
Logica Computicional

Orientador : Professor Pedro Barahona, CENTRIA, Universidade Nova de Lisboa

Júri:

Presidente:

Arguente:

Vogal:

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

June, 2014

A Clustering Approach for Vehicle Routing Problems with Hard Time Windows

Acknowledgements

First of all, I have to express great gratitude to my supervisor professor Pedro Barahona, who guided me in this long journey and helped me at every step of the way. It would be impossible to finish this work without his vision and support. Second, I want to thank Dr. Philip Kilby who gave the initial idea to investigate this problem. Dr Kilby would always answer my questions and give valuable suggestions. I want to state gratitude to my dear friend Sofia Gomez for giving me suggestions on writing and structuring my work. And lastly I want to express gratitude to my mother, who was patient and provided her love and support throughout my long studies.

Abstract

The Vehicle Routing Problem (VRP) is a well known combinatorial optimization problem and many studies have been dedicated to it over the years since solving the VRP optimally or near-optimally for very large size problems has many practical applications (e.g. in various logistics systems). Vehicle Routing Problem with hard Time Windows (VRPTW) is probably the most studied variant of the VRP problem and the presence of time windows requires complex techniques to handle it. In fact, finding a feasible solution to the VRPTW when the number of vehicles is fixed is an NP-complete problem. However, VRPTW is well studied and many different approaches to solve it have been developed over the years.

Due to the inherent complexity of the underlying problem VRPTW is NP-Hard. Therefore, optimally solving problems with no more than one hundred requests is considered intractably hard. For this reason the literature is full with inexact methods that use metaheuristics, local search and hybrid approaches which are capable of producing high quality solutions within practical time limits.

In this work we are interested in applying clustering techniques to VRPTW problem. The idea of clustering has been successfully applied to the basic VRP problem. However very little work has yet been done in using clustering in the VRPTW variant. We present a novel approach based on clustering, that any VRPTW solver can adapt, by running a preprocessing stage before attempting to solve the problem.

Our proposed method, tested with a state of the art solver (Indigo), enables the solver to find solutions much faster (up to an order of magnitude speed-up). In general this comes with at slightly reduced solution quality, but in somes types of problems, Indigo is able to obtain better solutions than those obtained with no clustering.

Resumo

O Problema de Roteamento de Veículos (VRP) é um problema de otimização combinatória bem conhecido e objecto de muitos estudos ao longo dos anos, desde a resolução óptima de VRPs ou à resolução aproximada para grandes problemas de tamanho tem muitas aplicações práticas (por exemplo, em vários sistemas de logística). Problemas de roteamento de veículos com janelas temporais (VRPTW) é provavelmente a variante mais estudada do problema VRP em que a presença de janelas de tempo requer técnicas complexas para a sua resolução. De facto, encontrar uma solução viável para o VRTPTW quando o número de veículos está fixado é um problema NP-completo. No entanto,o VRTPTW está bem estudado e muitas abordagens diferentes para resolvê-lo foram desenvolvidas ao longo dos anos.

Devido à sua complexidade inerente o problema VRPTW é NP-Difícil. Portanto, resolver de forma otimizada problemas com mais de cerca de uma centena de pedidos é considerada intratável difícil. Por isso, a literatura está plena de métodos que usam metaheurísticas inexactas, pesquisa local e abordagens híbridas, capazes de produzir soluções de alta qualidade em tempos razoáveis.

Neste trabalho estamos interessados em aplicar técnicas de agrupamento para o problema VRPTW. A idéia de agrupamento tem sido aplicado com sucesso para o problema básico VRP. No entanto, muito pouco tem sido feito ainda no uso de agrupamento na variante VRPTW. Nesta dissertação, é apresentada uma nova abordagem baseada em agrupamento, que qualquer resolvedor VRPTW pode adaptar, executando uma etapa de pré-processamento antes de se tentar resolver o problema.

O método proposto, testado com um resolvedor estado-da-arte (Indigo), permite ao resolvedor encontrar soluções muito mais rapidamente (com ganho de velocidades de até cerca de uma ordem de magnitude). Em geral, esta melhoria vem associada a uma pequena redução na qualidade da solução, mas em certo tipo de problemas, o Indigo é capaz de obter melhores soluções que as obtidos sem agrupamento.

Contents

List of Figures

1

Introduction

1.1 Introduction

Logistics is the management of the flow of goods between the point of origin and the point of consumption in order to meet some requirements set by the customers or companies. Any institution that performs any kind of large scale deliveries faces the problems of deciding how, what and when to deliver to minimize their costs. Even though the problem can be traced back to ancient civilizations, only with the appearance of digital computers, this problem could be solved automatically.

Modern day logistics systems have to handle problems of servicing thousands or even tens of thousands requests. And the problem becomes even harder when the logistics systems have to take into account external constraints that have to be respected in order to fulfill the request.

The Vehicle Routing Problem (VRP) is the core problem in these systems. It is a well known combinatorial optimization problem and many studies have been dedicated to it over the years. The problem was first formalized by [7]. Therefore solving the VRP optimally or near-optimally for very large size problems has many practical applications.

Due to the inherent complexity of the underlying problem VRP is NP-Hard. Therefore, optimally solving problems with no more than one hundred requests is considered intractably hard [22, 1], due to the inherent complexity of the problem. For this reason the literature is full with inexact methods that use metaheuristics, local search and hybrid approaches which are capable of producing high quality solutions within practical time limits.

The more general version of VRP is the VRP with Time Windows (VRPTW) introduced by [28]. VRPTW is a generalization of the Vehicle Routing Problem where every request is required to be serviced in a particular time period. VRPTW is probably the most studied variant of the VRP problem and it can be considered the primary ‘rich‘ variant because the presence of time windows require complicated techniques for constant time feasibility checks. [27] has shown that, even, finding a feasible solution to the VRPTW when the number of vehicles is fixed is a NP-complete problem. However, VRPTW is well studied and many different approaches to solve it have been developed over the years. Recent advances have been presented in [13].

In this work we are interested in applying clustering techniques to VRPTW problem. The idea of clustering has been successfully applied to the basic VRP problem [14]. However very little work has yet been done in using clustering in the VRPTW variant.

Our proposed algorithm uses the clustering idea to identify a set of nearby requests that can be replaced by a single macro request. This process is repeated and the set of requests is eventually replaced by a set of macro node requests. The new problem, consisting of macro nodes, is much smaller and a solver can find a solution for this problem much faster. The cost of this speed-up is the error that is introduced by treating clusters as single node requests. The hypothesis that we want to verify, is whether the error introduced by clustering is small enough and is worth the benefits in the speedup.

To test this hypothesis, we implement our proposed clustering algorithm and numerically evaluate it on the benchmark set [12]. In addition, we use the benchmark set to construct new larger problems, to see if the clustering works with very large problems.

This thesis is organized as follows. Chapter 1. consists of the introduction. Chapter 2 states the formal definition of the VRPTW and presents the state of the art review of the current approaches to solving VRPTW, followed by a review of the most recent applications of clustering to VRP and its variants. Chapter 2 finishes by giving an overview of the solver we use to conduct our numerical experiments. Chapter 3 is the core of this work. It shows how the idea of clustering can be used to identify sequences of requests that can be replaced by a single macro node request. Chapter 4 gives a detailed description of the implementation details one has to consider to implement the previously presented macro node clustering approach. The chapter then presents a number of heuristic functions that could be used to drive the clustering process. Chapter 5 starts by describing the benchmark data set used for evaluation of the proposed clustering method. The chapter presents numerical results obtained by running our implementation over the benchmark data set. Chapter 6 finishes our work by giving a summary of the clustering approach and the empirical results obtained in our implementation as well as providing some directions for future work.

2

State of Art

2.1 Preliminaries

The goal of this section is to introduce a unified notation, that can be used unambiguously throughout all of the work.

2.1.1 Formal Problem Definition

The VRPTW can be formally stated as the problem of designing *least cost routes* from a depot D to a set of requests R using a homogeneous fleet of vehicles V. Each request $r \in R$ is associated with a demand $q_r \geq 0$ and a service time $s_r \geq 0$. The travel cost between request r_i and r_j is denoted by t_{ij}.

In addition each request r is associated with a interval $[e_r, l_r]$ that represents its service time window. The request r must be serviced within this time interval. Specifically e_r represents the earliest service time for request r and l_r represents the latest service time for request r. A vehicle may arrive earlier than e_r but it has to wait until time e_r to be begin servicing the request.

Finally, all feasible vehicle routes start and end at the depot, while the quantities to deliver up to any request of a route must not exceed the vehicle capacity Q. The vehicles leave the depot at time e_o, at the earliest and must return to the depot by time l_o, at the latest.

2.1.2 Objective Function

The VRPTW usually consists of a hierarchical multi objective optimization. The goal is to first determine the minimum number of vehicles and then the minimal total travel duration, such that all requests are served and all constraints imposed by vehicle capacity, service times and time windows are satisfied.

The presented formulation of VRPTW is an established model and many different approaches have been proposed and successfully employed to solve it. Broadly these approaches can be divided into two separate groups

1. Exact methods
2. Heuristic methods

2.2 Exact Methods

Exact algorithms for VRPTW are based on Integer Linear Programming (ILP). Integer Linear Programming is a Linear Programming (LP) technique were the domains of all of the variables are restricted to be integers. Linear Programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints. Its feasible region is a convex polyhedron, which is a set defined as the intersection of finitely many half spaces, each of which is defined by a linear inequality. Its objective function is a real-valued affine function defined on this polyhedron. A linear programming algorithm finds a point in the polyhedron where this function has the smallest (or largest) value if such a point exists.

The general form of a Linear Programming formulation looks like

$$\begin{aligned} &\text{maximize } c^T x \\ &\text{subject to } Ax \leq b \\ &\text{and } x \geq 0 \end{aligned}$$

2.2.1 Integer Programming formulation of the VRPTW

Let $N = \{1, \dots, n\}$ be the set of requests. In addition 0 and $n+1$ represent the starting and ending node for all paths, which in our case is a single depo location. Let K, indexed by k, be the set of available vehicles to be routed and scheduled. Consider the graph $G = (V, A)$, where the set of nodes is equal to $V = N \cup \{0, n+1\}$, and the set A contains all feasible arcs, that is $A \subseteq V \times V$. For each request $i \in N$, there is a known quantity q_i, a time window $[e_i, l_i]$ and a service time s_i. We assume that all vehicles are empty in the depot therefore $q_{0(k)} = 0$ for each vehicle $k \in K$. Every vehicle k has a working time interval $[E^k, L^k]$, that we express by associating a time window with nodes 0 and $n+1$, $[e_{0(k)}, l_{0(k)}] = [e_{n+1(k)}, l_{n+1(k)}] = [E^k, L^k]$.

For each arc $(i, j) \in A$, there is a cost c_{ij} and a travel time t_{ij}. We assume that $c_{ij} = t_{ij}$. All the requests must be assigned to at most v vehicles, $v \leq |K|$, such that the capacity Q^k of each vehicle is not exceeded.

We present a Integer Programming formulation involving three types of variables: flow variables $X_{ij}^k, (i,j) \in A, k \in K$, equal to 1 if arc (i,j) is used by vehicle k and 0 otherwise; time variables $T_i^k, i \in V, k \in K$ specifying the start of service at node i by vehicle k; and load variables $L_i^k, i \in V, k \in K$ specifying the load of the vehicle k after servicing node i.

$$\text{(VRPTW)} \min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} X_{ij}^k \tag{2.1}$$

subject to

$$\sum_{k \in K} \sum_{j \in N \cup \{n+1\}} X_{ij}^k = 1 \qquad \forall i \in N \tag{2.2}$$

$$\sum_{k \in K} \sum_{j \in N} X_{0,j}^k \leq v \tag{2.3}$$

$$\sum_{j \in N \cup \{n+1\}} X_{0j}^k = 1 \qquad \forall k \in K \tag{2.4}$$

$$\sum_{i \in N \cup \{0\}} X_{ij}^k - \sum_{i \in N \cup \{n+1\}} X_{ji}^k = 0 \qquad \forall k \in K, \forall j \in N \tag{2.5}$$

$$\sum_{i \in N \cup \{0\}} X_{i,n+1}^k = 1 \qquad \forall k \in K \tag{2.6}$$

$$X_{ij}^k (T_i^k + s_i + t_{ij}^k - T_j^k) \leq 0 \qquad \forall k \in K, (i,j) \in A \tag{2.7}$$

$$e_i \leq T_i^k \leq l_i \qquad \forall k \in K, \forall i \in V \tag{2.8}$$

$$X_{ij}^k (L_i^k + q_i - L_j^k) = 0 \qquad \forall k \in K, (i,j) \in A \tag{2.9}$$

$$L_i^k \leq Q^k \qquad \forall k \in K, \forall i \in N \cup \{n+1\} \tag{2.10}$$

$$L_0^k = 0 \qquad \forall k \in K \tag{2.11}$$

$$X_{ij}^k \geq 0 \qquad \forall k \in K, (i,j) \in A \tag{2.12}$$

$$X_{ij}^k \in \{0,1\} \qquad \forall k \in K, (i,j) \in A \tag{2.13}$$

The objective function (2.1) of this nonlinear formulation expresses the total cost. It is possible to add a fixed cost c of using a vehicle by adding $c_{0,j}, j \in N$, then to minimize the number of vehicles c has to be selected large enough to be greater than any route total distance. Given that $N = V \setminus \{0, n+1\}$ represents the set of customers, constraint 2.2 restricts the assignment of each customer to exactly one vehicle route. Constraints 2.4 - 2.6 characterize the flow on the path to be followed by vehicle k. Constraints 2.7 - 2.8 guarantee schedule feasibility with respect to time considerations and 2.9 - 2.11 ensure feasibility of capacity aspects. Constraint 2.13 makes the condition variables binary.

To actually solve the proposed model, one can use several approaches to solving ILPs. In particular we will look at two approaches, the Column Generation and its specialization, and the Lagrangian relaxation.

2.2.2 Column Generation

The main idea in Column Generation is that many linear programs are too large to consider all the variables explicitly. Since most of the variables will be non-basic and assume a value of zero in the optimal solution, only a subset of variables need to be considered in theory when solving the problem. Column generation leverages this idea to generate only the variables which have the potential to improve the objective function.

The problem being solved is split into two problems: the master problem and the subproblem. The master problem is the original problem with only a subset of variables being considered. The subproblem is a new problem created to identify a new variable. The objective function of the subproblem is the reduced cost of the new variable with respect to the current dual variables, and the constraints require that the variable obey the naturally occurring constraints.

2.2.3 Dantzig Method

The Dantzig-Wolfe decomposition is a special case of the Delayed Column Generation process and works by identifying which of the constraints are connected. Constraints are connected if the same variable appears in both of them. The method separates the global list of constraints into buckets such that only connected constraints appears in each bucket. In addition the objective function is simplified for each bucket to contain only those variables that are present in the constraints.

Each bucket then forms a subproblem. The solution for each subproblem gives rise to a set of points that represent the feasible region of this subproblem. The solution to the master problem must lie somewhere on the intersection of all this feasible regions. Therefore the solution to the master problem, cab be expressed as a convex combination of the points forming the feasible regions of the subproblems.

The essence of the Dantzig-Wolfe decomposition is to construct the master problem that tries to minimize the objective function by finding a convex combination of the points from the polyhedron formed by the intersection of all the polyhedra formed by the individual subproblems.

More formally, given that each subproblem has a feasible region that forms a convex polyhedron represented by a set of points $X_i =< x_1, \dots, x_n >$, the polyhedron formed by the intersection of all polyhedra is equal to $\hat{X} = \bigcap_{i=1}^{m} X_i$. And the set of points that represent this polyhedron is equal to $\hat{X} =< y_1, \dots, y_k >$. We can formulate the objective function of the master problem using this set. The objective function becomes a linear combination Λ of the points $\hat{X}$.

$$\text{maximize } c^T \hat{X} \Lambda$$

subject to

$$
\begin{aligned}
& Ax \leq b \\
& x \geq 0 \\
& \Lambda \geq 0 \\
& \sum_{i=1} \lambda_i = 1 \qquad\qquad \lambda_i \in \Lambda
\end{aligned}
$$

2.2.3.1 Dantzig-Wolfe Method with Integer Linear Programming

The Dantzig-Wolfe method was originally designed to work with Linear Programming and not Integer Linear Programming. However under certain assumptions we can still apply it to ILP. The conditions for application must be that the solution of the linear relaxation (LP) of the Integer Programming model must be equal to the original Integer Programming model. This property is called the integrality property.

Since we can not solve any ILP models directly we first linearize them and solve the linear versions using standard Simplex algorithms for solving LP. But we must have the guarantee that the linear solutions are transferable to the integer models.

2.2.3.2 Dantzig-Wolfe decomposition of the VRPTW

Here we give an example one can use to define the subproblem. We can decompose the original problem based on the observation that it consists of $|K|$ disjoint subproblems, one for each vehicle. Then each subproblem becomes that of finding an elementary shortest path with time and capacity constraints. Consequently, the flow variables X_{ij}^k can be expressed as a nonnegative convex combination of the paths generated from the subproblems.

In the case of VRPTW, the linearized subproblem does not posses the integrality property. But it can still be solved as a nonlinear integer program. The elementary shortest path problem with capacity and time constraints is known to be NP-hard. However if we allow nonelementary path solutions, i.e. solutions where a path may contain cycles of finite duration, pseudo-polynomial algorithms exist.

2.2.4 Lagrangian Relaxation Method

Lagrangian relaxation approximates a difficult problem of constrained optimization by a simpler problem. A solution to the relaxed problem is an approximate solution to the original problem, and provides useful information.

2.2.4.1 Mathematical Description

Given a linear programming model s.t. $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m,n}$.

$$\begin{aligned}
&\text{maximize } c^T x\\
&\text{subject to}\\
&\qquad Ax \leq b
\end{aligned}$$

We can separate the constraints in A into two parts $A_1 \in \mathbb{R}^{m1,n}$ and $A_2 \in \mathbb{R}^{m2,n}$. We can then write the original problem as.

$$\begin{aligned}
&\text{maximize } c^T x\\
&\text{subject to}\\
&\qquad A_1 x \leq b\\
&\qquad A_2 x \leq b
\end{aligned}$$

Then we can apply the main idea of the Lagrangian method of introducing some of the constraints inside the objective value. And assigning nonnegative weights $\lambda = (\lambda_1, \ldots, \lambda_{m2})$ to penalize the objective value for violating the constraints.

$$\begin{aligned}
&\text{maximize } c^T x + \lambda(b_2 - A_2 x)\\
&\text{subject to}\\
&\qquad A_1 x \leq b\\
&\qquad \lambda \geq 0
\end{aligned}$$

Therefore it is always the case that the optimal result to the Lagrangian Relaxation problem will be no smaller than the optimal result to the original problem.

2.2.4.2 Lagrangian relaxation for VRPTW

There are several ways how one can apply the Lagrangian relaxation to the VRPTW problem. One way is to relax the difficult constraints - time window and capacity, so that the resulting Lagrangian subproblem is easy to solve. Another way is to relax the network flow constraints, retaining the complicated constraints in the Lagrangian subproblem.

2.2.5 Numerical Results

For the single depot VRPTW with a homogeneous fleet the Dantzig-Wolfe decomposition found optimal solutions to a number of 100-customer problems [8]. Later it was improved by [20]. This work added 2-path inequalities to linear relaxation of the subproblem formulation.

In [19] presented a Branch and Bound (BB) algorithm were the relaxation was computed using the Lagrangian Relaxation. These methods were based on 2-cycle elimination algorithms. Later [16] described a BB algorithm where the pricing subproblem is solved with a k-cycle elimination procedure.

The results show that the Dantzig-Wolfe decomposition performs much better then the Lagrangian relaxation methods. Not only did the Dantzig-Wolfe method solve more problems, it did so in with less computational time used.

2.3 Heuristics

Given the inherent computational difficulty of the VRPTW, a variety of heuristics have been reported in the literature that achieve sub-optimal solutions.

We divide heuristics into the following categories:

1. Insertion Heuristics
2. Improvment Heuristics
3. Metaheuristics

Heuristics are often used because they provide a tractable way of tackling NP-Hard problems. Even though they do not provide a guarantee of finding the best solution, often their solutions are good enough. Additionally, heuristics are often simple to describe and implement, which leads to their easy adaptability to many VRP variants.

2.3.1 Insertion Heuristics

Insertion heuristics build a feasible solution by inserting at every iteration an unrouted customer into a partial route. This process is performed either sequentially, one route at a time, or in parallel, where several routes are considered simultaneously. The process has to decide two things, which unrouted customer to insert and where to insert it. Usually the algorithms use metrics based on distance, waiting time and savings to answer these questions.

2.3.1.1 Clarke and Wright savings algorithm

The Clarke and Wright savings algorithm is the first and the most well known heuristic for the VRP problem. It was first presented in [5] and it has since been successfully applied to the case where one tries to minimize the number of vehicles used.

It is based on the notion of *savings*. Initially, the solution consists of n back and forth routes between the depo and each customer. Then during every iteration, two routes $(d,\ldots,c_i,d)$ and $(d,c_j,\ldots,d)$ are merged into a single route $(d,\ldots,c_i,c_j,\ldots,d)$ whenever that is feasible. This merge then generates a saving of $s_{ij} = c_{id} + c_{dj} - c_{ij}$. See Figure 2.1 for an example.

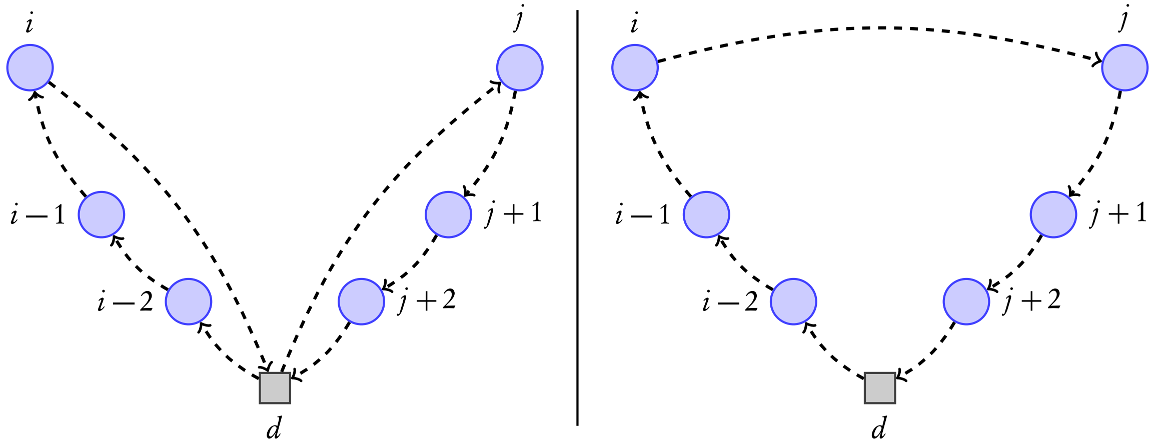

Figure 2.1: Example of savings heuristic. Two routes $\langle i-2, i-1, i, \rangle$ and $\langle j, j+1, j+2, \rangle$ merged into a single route $\langle i-2, i-1, i, j, j+1, j+2 \rangle$.

Solomon in [29] reports a variant of the CW algorithm in which savings are adapted to handle time windows, but results are disappointing.

2.3.1.2 Solomon

Sequential insertion heuristics for the VRP with time windows were first proposed by [29]. It proposed a two-criteria insertion algorithms. The first criteria assigns all unrouted customers their best feasible insertion position based on distance and waiting time. And the second criteria selects the best candidate based on the savings concept.

Formally, let $c_1(i,u,j)$ and $c_2(i,u,j)$ be the first and the second criteria to insert customer u between two adjacent customers i and j. Then for each unrouted customer u we compute its best feasible insertion cost as

$$c_1(i(u),u,j(u)) = \min_{\rho=1,\ldots,m} c_1(i_{\rho-1},u,i_\rho)$$

After selection the insertion position, then the customer to be selected is chosen by the second criteria as

$$c_2(i(u^*),u^*,j(u^*)) = \max_u \{c_2(i(u),u,j(u)) | u \text{ is feasible}\}$$

2.3.2 Improvement Heuristics

Route improvement heuristics start with a feasible solution and then in each iteration try to improve it. This is done by exploring a neighborhood of possible candidate solutions. Generally, a neighborhood is the set of solutions that can be reached from the present one by swapping a subset of r arcs between solutions. An arc exchange is applied only if it improves the objective function. Arc improvement heuristics are classified by the number or arcs they interchange - 2-Opt, 3-Opt, and in general k-Opt. Or by the type of operation that the exchange performs - Or-Opt. In the case of Or-Opt, there is not need to reverse any route segments.

2.3.2.1 2-Opt

First presented in [6] for solving the traveling salesman problem.

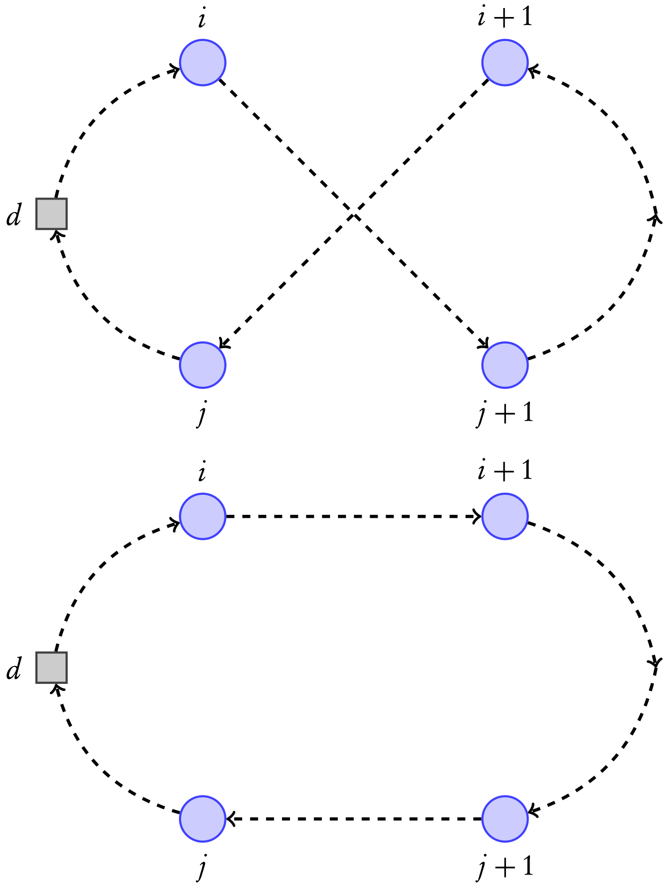

Figure 2.2: Example of a 2-opt move

The 2-opt algorithm removes two edges from a route, and reconnects the two new sub-tours created (see example at Figure 2.2).

2.3.2.2 Or-Opt

There is a problem with the 2-opt heuristic, in that it requires one to reverse the customer order. An alternative approach proposed by [23] consists of generating only those moves that do not require customer reordering. The idea is to relocate a chain of consecutive customers. This is achieved by replacing three edges in the original tour by three new ones without modifying the orientation of the route as shown in the Figure 2.3

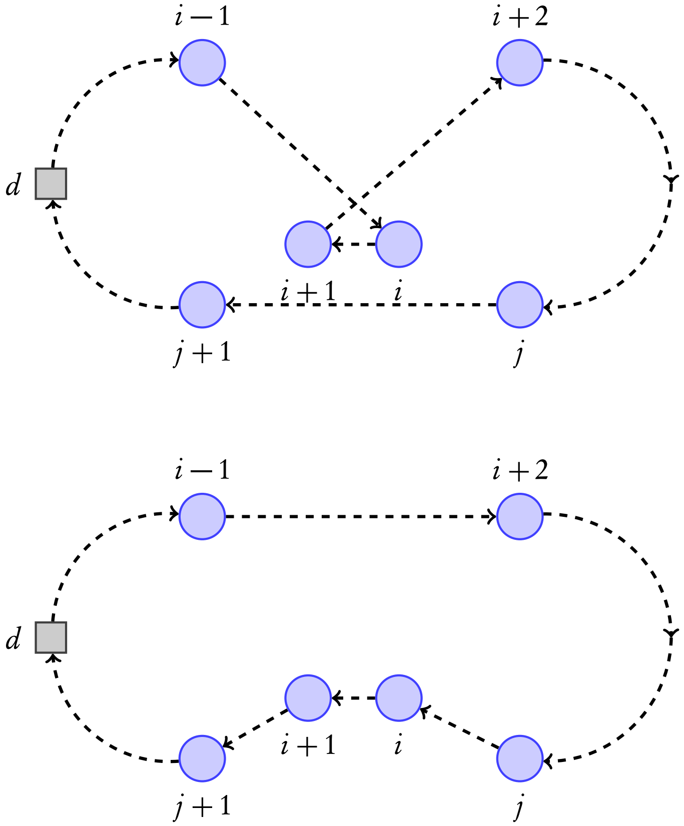

Figure 2.3: Example of a Or-opt move.

2.3.3 Metaheuristics

A metaheuristic is an iterative generation process which guides and subordinates heuristics by combining intelligently different concepts for exploring and exploiting the search space, while learning strategies are used to structure information in order to find efficient near-optimal solutions [24].

Metaheuristics are the core of recent work on approximation methods for the VRPTW, and they mainly include Simulated Annealing (SA) and Tabu Search (TS).

Unlike local search heuristics that terminate once a local optimum has been reached, these methods explore a larger subset of the solution space in the hope of finding a near-optimal solution.

2.3.3.1 Tabu Search

Tabu search was first proposed in [15]. The main idea is to avoid looping in cycles by forbidding or penalizing moves which return states that were already previously visited. Thus a heuristic that is driven by TS may end up accepting a solution of lower quality and making a degrading move, in order to escape visiting states that were previously explored. This insures that new regions of problem solution space will be explored in the goal of avoiding local minima.

[11] were the first to adapt the TS metaheuristic to the VRPTW problem. Their approach consisted of using the Solomon's insertion heuristic to produce an inital solution and then post-optimize it with both 2-opt and Or-opt.

2.3.3.2 Simulated Annealing

Simulated Annealing (SA) is a probabilistic metaheuristic first described by [18] where a modification to the current solution that leads to an increase in solution cost can be accepted with some probability. This mechanism allows the method to escape from bad local optima.

At each step, the SA heuristic considers some neighbouring state s' of the current state s, and probabilistically decides between moving the system to state s' or staying in state s. The probability of making this transition is specified by an *acceptance function* that depends on the objective values of states s and s' being $E(s)$ and $E(s')$ respectively, and on a global time-varying parameter T called the *temperature*. As the temperature decreases the probability that the acceptance function will make a degrading move must decrease as well. Therefore it is usual to model the acceptance function as

$$p(s,s',T) = e^{-\frac{(E(s)-E(s'))}{T}}$$

SA has been successfully applied to the VRPTW by [4].

2.4 Clustering and Decomposition Approaches in VRPTW

In the previous sections we have described the general and historically well studied methods for solving VRP with time windows. In this part we present those methods that demonstrate the idea of clustering to help solve VRPTWs.

2.4.1 Cluster-First and Route-Second

Cluster-first route second are methods that perform partitioning of the customer set and then determine a vehicle route for each cluster. This allows to reduce the complexity of the problem by solving individual smaller instances. Known approaches of cluster-first and route second are [14, 10, 26, 30].

2.4.2 Gillett 1974

[14] are the first authors adopting the *cluster-first and route-second* method to solve VRPs. They developed a sweep-based heuristic in which customers are partitioned into different groups according to their polar coordinates as well as the vehicle capacity. A TSP is then solved in each group.

However, in the case for the VRPTW, it usually fails to construct a feasible route in the second stage due to the required service time window. [29] extended this VRP algorithm to VRPTW by repeating the *cluster-first and route-second* procedure with all unscheduled customers until the problem is solved.

2.4.3 Fisher 1981

[10] proposed a two-phase algorithm for vehicle routing: in the first phase, it finds an assignment of customers to vehicle routes, and then continued by a route improvement procedure. A number of seed customers are selected by some criteria, and then the cost from each non-seed customer to each seed customer is calculated by the additional distance when the non-seed customer is inserted between the seed customer and the depot.

2.4.4 Dondo 2007

[9] proposed a three stage hybrid approach. During stage 1 requests are combined into clusters. Then, during stage 2, a new problem is solved based on the combined clusters and the remaining unclustered requests, using an exact MILP solver. After, as stage 3, for each route a separate TSP optimization is run to find the best order of visits within this route.

Work proposed by Dondo is very similar to our approach, it also tries to construct clusters in order to reduce the size of the problem, however it then uses an exact solver to route the clusters into complete routes.

The difference between our work and Dondo is in the way Dondo creates clusters. First, Dondos cluster construction is parameterized by the two parameters

(a) the maximum distance between two nodes inside the cluster

(b) the maximum waiting time between two nodes inside the cluster.

In this way the process implicitly controls the reduction size, by only allowing clusters with the given properties. These numbers have to be manually selected depending on the structure of the problem. Instead, our approach is explicitly controlled by setting the desired reduction size.

The second difference, is that Dondo only allows to insert an unrouted request into the current cluster, however we are more flexible and allow for merges between any two clusters.

The third difference is in the order Dondo decides to insert requests into the current cluster. Dondo always tries to augment the cluster by the first request with the earliest arrival time, that fulfills the parameters a) and b). We, on the other, propose a heuristic function based on distance, waiting time and time window flexibility.

Since Dondo uses an exact solver, it is only able to solve problems up to 100 requests. Therefore we can not directly compare the results of Dondo with our approach.

2.4.5 Qi 2012

[25] proposed a method to partition the customers into clusters that jointly considers the spatial and temporal information. It represents time and space in the same coordination system, and develops a method to measure the space-temporal distance between two customers. However the method considers the soft version of the time window problem, where a violation of time windows does not invalidate a solution, but just penalize the objective function value.

2.4.6 Savelsbergh 1985

[27] was the first to suggest that information about a route can be summarized by a so called *macro node* in a few parameters, that allowed to decide if a new request can be inserted into the route in constant time instead of recomputing time feasibility constraints.

2.4.7 Bent 2010

Another way to speed-up neighborhood search algorithms is to perform decoupling in an effort to define sub-problems that can be optimized independently and then reinserted into an existing solution.

[2] proposed a so-called randomized adaptive spatial decoupling (RAND). RAND considers spatial decoupling and produces independent feasible sub-problems. A further improvement was later proposed in [3]. Where decoupling of the subproblem was based on spacial, temporal and space-temporal properties of customer requests. These approaches have been successfully applied on large-scale VRPTW instances.

Similarly, spatial decoupling acceleration techniques have been also utilized by [21].

2.5 Indigo

Although many techniques are presented in the literature, there are no publicly available implementations that one can use to compare performances of these techniques, since most of Operations Research work is done by either private companies or private research laboratories. However the authors of this work had access to the VRPTW solver **Indigo** developed at NICTA[1] .

We briefly present the Indigo solver and show that this solver is comparable with other state of the art solutions. [17] presents the architecture as well as the benchmark results of the Indigo solver.

[1] http://www.nicta.com.au/

2.5.1 Solver Architecture

Indigo builds the solution in two phases. During phase one, a feasible solution is constructed using the Clarke's savings method and during phase two a local search algorithm is used to improve on the solution.

The improvement phase uses the Large Neighborhood Search (LNS) metaheuristic. Indigo internally uses a Constraint Programming (CP) framework to model the problem as a variable assignment problem. Combining this CP representation allows Indigo to effectively prune large sections of the unfeasible search space.

Using CP as the underlying model allows Indigo to be very flexible in modeling the side constraints of the classical VRP. One can easily extend indigo to handle much more expressive VRP variants.

2.5.2 Benchmarks

According to [17] in 2011, Indigo was able to improve the results of the benchmark data set in [12], namely, to find new best results for 83 out of 300 benchmarks.

Bellow in Table 2.1, we show performance as a ratio between Indigo's result and the best know results reported in the literature. *Size* gives the number of customers in the problem; *Best* gives the number of problems where a new best was found; *Mean* is the mean increase; *80%* gives the 80th percentile of increase; and *Max* gives the maximum increase over best-known solution. For example, 1.02 means that Indigo results were 2% worse than the best-known solution.

Size	Best	Mean	80%	Max
200	11	1.01	1.02	1.05
400	13	1.01	1.03	1.06
600	19	1.02	1.04	1.10
800	18	1.02	1.05	1.11
1000	22	1.03	1.06	1.14

Table 2.1: Indigo's results on [12] promblem set

3

Macro Nodes

The final goal of this work is to speed up VRPTW solvers by encoding a problem instance into one that is smaller in size and, therefore, much faster to solve. The new instance is constructed in such a way, that the solution of the new instance can be translated back to the solution of the original instance.

To perform this reduction in size, we identify sets of requests that can be replaced by a single request. Since the new instance has several requests replaced by a single request it is smaller in size. This new instance is then solved with the state of the art solver Indigo. The solution of the new instance that is obtained with Indigo can then be translated back to the solution of the original larger instance.

The task then is to find sets of requests that can be replaced by a single request. The new request which would replace this set acts like a *macro node*. This new request must characterize servicing of the whole set. Nevertheless the macro node must be represented as a single VRPTW request.

In this chapter we show how starting with individual requests one can combine them into larger macro nodes and how to represent such macro nodes by an abstraction of single requests.

3.1 Macro Nodes

The design of this algorithm is constrained by the use of the Indigo solver (Section 2.5). Since the goal is not to design a new solution method for VRPTW, but instead to introduce a pre-processing step that can be executed before the real solver attempts to solve the problem. Therefore, the final output of our macro node construction algorithm must produce some problem representation, that can be passed to Indigo as a proper VRPTW problem.

A VRPTW problem description consists of the following data:

- A set of requests
- Description of the vehicle fleet

Since we do not do anything that affects the vehicle fleet, this part of the problem specification is just passed over to Indigo as is. However, we change the request set by reducing its size and introducing new requests.

We repeat what exactly constitutes a request. A request r is defined by the following properties:

- Its geographic location p_r where the service is to take place.
- Its demand quantity denoted by q_r.
- Its servicing time window interval denoted by $[e_r, l_r]$.
- Its servicing time duration s_r.

More formally a request can be defined as a tuple with 5 components.

Definition 1. *A request r is defined by a tuple $\langle e_r, l_r, s_r, q_r, p_r \rangle$, where e_r and l_r are the earliest and latest arrival times, s_r is the servicing time, p_r is the geographic location and q_r is the demand quantity of the request r.*

Therefore, any macro node object that is constructed must be replaceable by a single request with the listed properties. This fact puts a limitation on what kind of macro node objects we are be able to construct.

3.1.1 Structure of the macro nodes

As already stated, the goal of the macro node object is to replace several requests by a single request. This single request must have a fixed location, a fixed servicing time window interval, a fixed demand quantity and a fixed servicing time duration.

In order to simplify the mapping of a macro node to a single request, the proposed algorithm assumes that the order in which we visit requests in the macro node is fixed at macro node construction time. This assumption makes the structure of a macro node object not a set like object, but instead a sequence like object, where the sequence determines the visit order of the requests inside the macro node. Committing to a fixed visit order inside the macro node gives us an easier way to replace a set of requests by a single request. Therefore servicing a macro nodes automatically determines the order of servicing all the requests inside the macro node.

Now that we gave the intuition of what a macro node object represents, we can give a formal definition.

Definition 2. *A **macro node** m is a sequence of requests $\langle r_1, \ldots, r_n \rangle$.*

3.2 Macro Node Request

Given that a macro node is a sequence of requests that have to be serviced and given that the servicing of this macro node must be mapped to servicing of a single new request, we introduce the concept of a *macro node request*. The properties of the macro node request are computed in such a way, that if a feasible solution of a VRPTW problem contains a macro node request then replacing this macro node request with the actual sequence of requests of the macro node would not make this solution unfeasible.

Definition 3. *If $m_k = \langle r_1, \ldots, r_n \rangle$ is a macro node then this macro node can be represented by a* ***macro node request*** *r_{m_k}. Where r_{m_k} is a tuple*

$$r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$$

Arrival at the request r_1 within the time windows $[E_k, L_k]$ will make it feasible to service all requests in the macro node m_k, the servicing time S_k should be equivalent in duration to servicing all the requests in m_k and traveling between the requests and, also, waiting if necessary, and the demand quantity Q_k has to represent the total demand for the whole macro node. The locations p_{first_k} and p_{last_k} are the locations of the first request of the macro node and the location of the last request of the macro node.

Therefore the **macro node request** r_{m_k} is a single request with a feasible arrival interval $[E_k, L_k]$ a servicing time S_k and demand quantity Q_k. But instead of a single location as in the case of a request, a macro node request has two locations - a starting location p_{first_k} and a finishing location p_{last_k}. The above definition does not specify how to compute the values E_k, L_k, S_k and Q_k given a macro node m_k. The remaining part of this section is dedicated to the problem of computing these values.

3.3 Macro Node Visualization

Before continuing to explain additional properties of macro nodes, we introduce our request visualization technique. The purpose of this visualization technique is to show the proximity of two requests in time and space simultaneously.

The visualization uses a two dimensional Cartesian coordinate system. The x-axis represents time, and the y-axis represents distance in space. To display a request we draw an interval along the x-axis (time axis). This interval depicts the servicing time window of this request. To show movement from one request to another, we draw a vector at a 45°. The 45° represent movement in both space and time.

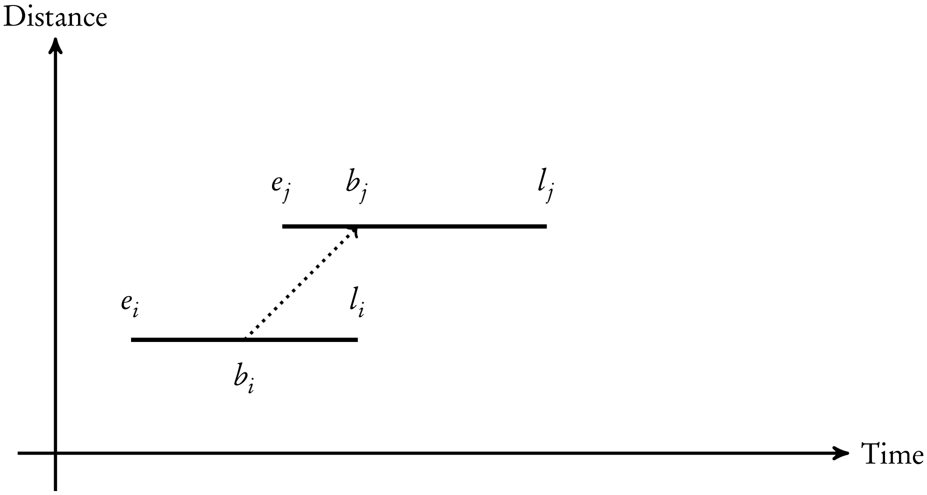

Figure 3.1: Request distance visualization in both space and time

Figure 3.1 shows two requests i and j, with their corresponding time window intervals $[e_i, l_i]$ and $[e_j, l_j]$ respectively. Also, it shows a departure from request i at time b_i and an arrival at request j at time b_j.

3.4 Servicing time of a Macro Node Request

Servicing a macro node entails servicing all the requests in this macro node. We can decompose this servicing time into three components.

- Time spent traveling between all requests of the macro node.
- Minimal time spent waiting for the feasible interval to become active.
- Time spent servicing individual requests.

Definition 4. *If $m_k = \langle r_1, \ldots, r_n \rangle$ is a macro node and $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ is its macro node request then the value S_k is equal to*

$$S_k = \sum_{i=1}^{n} s_i + \sum_{i=1}^{n-1} t_{i,i+1} + \sum_{i=2}^{n} w_i$$

where w_i is the minimal waiting time incured before request r_i.

There is something to be said about the minimal waiting time between two requests. Waiting time between any two requests r_i and r_j is influenced by the departure time from the request r_i. The minimal waiting time is incured if we depart from the request r_i as late as possible.

Figure 3.2 illustrates the different waiting times between two requests.

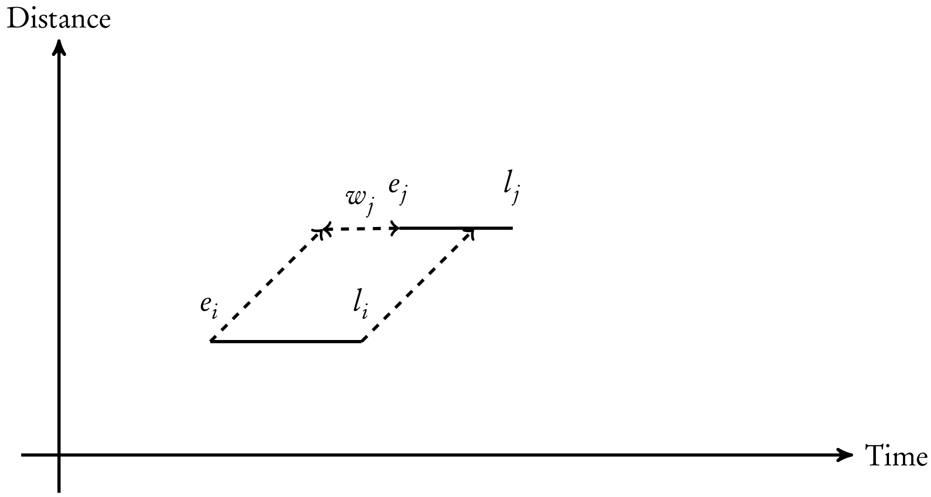

Figure 3.2: Waiting time dependence on the departure time

In Figure 3.2 departing from the request r_i at time $e_i + s_i$ would lead to some waiting time w_j, but departing at time $l_i + s_i$ would lead to no waiting time. Therefore the total servicing time of both requests depends on what time we start servicing the first request, since we might or might not incur some waiting time in between the requests. This means the total servicing time is a function on the start time of the two requests.

This leads to a problem, since if we want to represent a macro node as a single request it must have a fixed total servicing time. To overcome this dependency on the starting time, the resulting macro node requests, that represent servicing all of their requests, should have a time window that always ensures a fixed minimal total servicing time.

As shown in Figure 3.3, for the example that we have just given, if we restrict the time window of the resulting macro node request to the interval $[b_i, l_i]$ then the overall servicing time of this macro node is fixed and the overall waiting time is minimal.

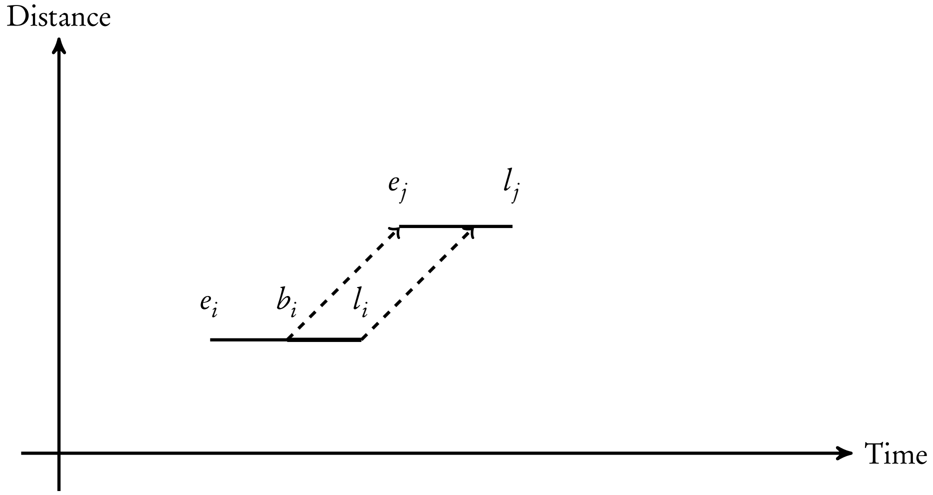

Figure 3.3: Constant servicing time within interval $[b_i, l_i]$

3.5 Demand quantity of a Macro Node Request

Computing demand quantity of a macro node request is simple to compute, since for a fixed macro node the total demand of this macro node is always the sum of the demands of individual requests.

Definition 5. *If $m_k = \langle r_1, \ldots, r_n \rangle$ is a macro node then the demand quantity Q_k of the macro node request $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ of m_k is equal to*

$$Q_k = \sum_{i=1}^{n} q_i$$

3.6 Distance between Macro Node Requests

We already mentioned that each request r is associated with a geographic location p_r. And for every pair of requests r_i and r_j the distance between them is defined as:

$$t_{p_i, p_j}$$

Since Euclidean distance is a symmetric property it is equal in both ways

$$t_{p_i, p_j} = t_{p_j, p_i}$$

Nevertheless, real distances may not be symmetrical (they might use different edges in the digraph). Indigo accepts non-symmetrical distance matrices. And we take advantage of this as follows.

To encode a macro node as a single request one has to find a way to encode the geographical distance between macro nodes. Since a macro node is just a sequence of requests one can represent the distance to a macro node as the distance to the first request of this macro node. In the same way, the distance from a macro node to other objects can be measured as the distance from the last request of this macro node. Formally we define it as:

Definition 6. *If* $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ *and* $r_{m_l} = \langle E_l, L_l, S_l, Q_l, p_{first_l}, p_{last_l} \rangle$ *are two macro node requests, then the distance between these two requests is computed as :*

$$t'_{k,l} = t_{p_{last_k}, p_{first_l}}$$
$$t'_{l,k} = t_{p_{last_l}, p_{first_k}}$$

Now the distance between two macro node requests is the distance between the location of the last request in the macro node m_k and the location of the first request in the macro node m_l and vice versa.

Also, for all macro node requests r_{m_k} the distance from the macro node request to itself is 0.

$$t'_{k,k} = 0$$

3.7 Constructing Macro Nodes

We start the macro node creation process by creating a macro node m_k of size 1 for every request $k \in R$. The properties of the macro node request r_{m_k} of this macro node m_k are equal to the original request k.

$$E_k = e_k$$
$$L_k = l_k$$
$$S_k = s_k$$
$$Q_k = q_k$$
$$p_{first_k} = p_k$$
$$p_{last_K} = p_k$$

After having constructed the set of all macro node requests, we subsequently combine two macro node requests into a single new macro node request. This new request then represents servicing all the requests in both macro nodes. Next we show how to compute the properties of the new macro node request after combining two macro nodes.

3.7.1 Joining two Macro Nodes

In future visualizations instead of displaying all the request sequence of a macro node, we depict the macro node by drawing its macro node request. This simplification does not remove any important details since the time window of the macro node request adequately characterizes it, and more so allows to simplify the visualizations.

There are three distinct cases of joining two macro nodes. We examine each case separately and show how to decide if the two macro nodes can be combined and, whenever possible, what is the resulting macro node request that characterizes servicing both macro nodes.

The three cases are based on the temporal relationship between the two macro nodes we are considering to combine together. To avoid repetition, we analyze all cases by considering two macro node requests $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ and $r_{m_l} = \langle E_l, L_l, S_l, Q_l, p_{first_l}, p_{last_l} \rangle$ to form a combined macro node request $r_{m_{k'}} = \langle E_{k'}, L_{k'}, S_{k'}, Q_{k'}, p_{first_{k'}}, p_{last_{k'}} \rangle$

3.7.2 Case one - unfeasible clustering

As the first case we consider the situation where the earliest departure time $E_k + S_k$ of the first macro node request $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ plus the travel time $t'_{k,l}$ to reach the macro node request $r_{m_l} = \langle E_l, L_l, S_l, Q_l, p_{first_l}, p_{last_l} \rangle$ exceeds the latest arrival L_l. Formally we can state this condition as:

$$E_k + S_k + t'_{k,l} > L_l$$

Figure 3.4 demonstrates this case.

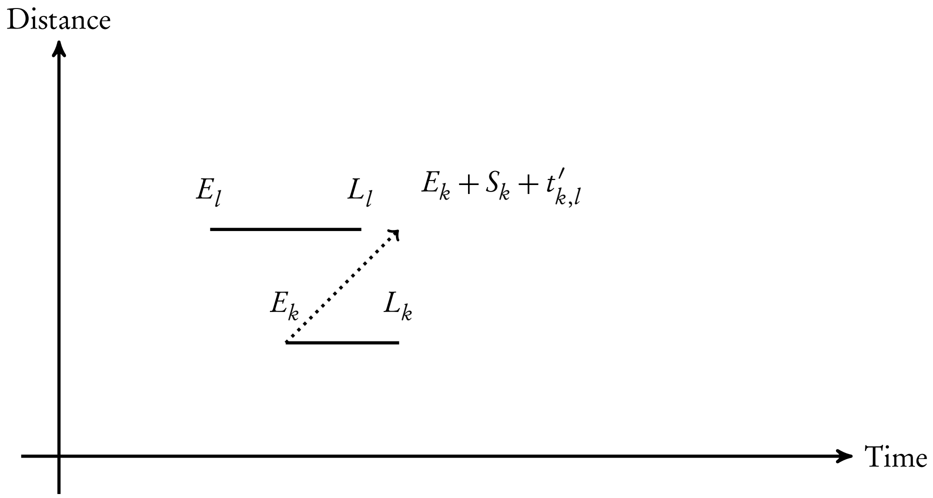

Figure 3.4: Second macro node is before the first macro node

Whenever two macro nodes fulfill this condition, combining them is impossible.

3.7.3 Case two - clustering with extra waiting time

In our second case, we consider the temporal situation when the earliest arrival of the second macro node request E_l is later then the latest departure $L_k + S_k$ plus travel time t'_{kl} between the macro node requests. Formally we write this condition as:

$$L_k + S_k + t'_{k,l} < E_l$$

Figure 3.5 illustrates case two.

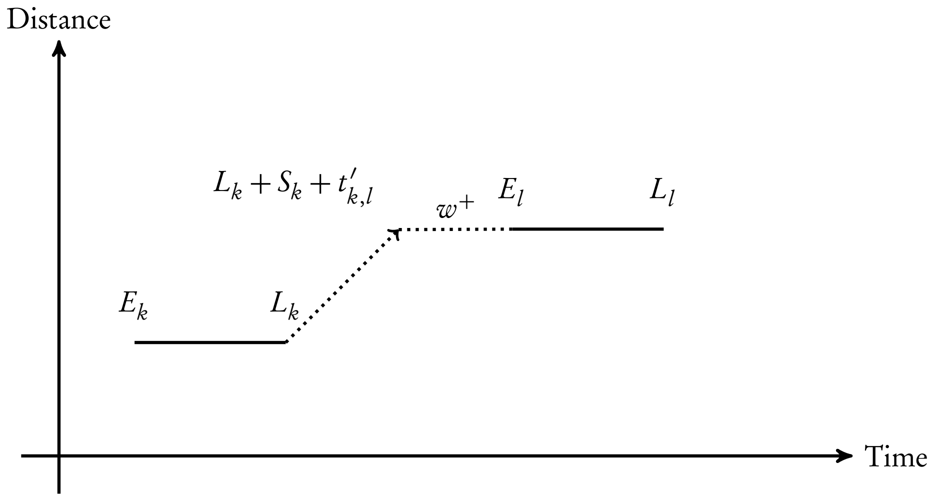

Figure 3.5: Second macro node is after the first macro node

In this case, even if we service the first macro node as late as possible at time L_k then we will still incur a waiting time cost w^+ between macro nodes m_k and m_l.

To guarantee that the waiting time between the macro node requests is minimal, the first macro node request must be started as late as possible. Hence, the result of combining $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ and $r_{m_l} = \langle E_l, L_l, S_l, Q_l, p_{first_l}, p_{last_l} \rangle$ is equal to the macro node $r_{m_{k'}} = \langle E_{k'}, L_{k'}, S_{k'}, Q_{k'}, p_{first_{k'}}, p_{last_{k'}} \rangle$, where the properties of $r_{m'_k}$ are defined as follows:

$$\begin{aligned} E_{k'} &= L_k \\ L_{k'} &= L_k \\ S_{k'} &= S_k + S_l + w^+ + t'_{k,l} \\ Q_{k'} &= Q_k + Q_l \\ p_{first_{k'}} &= p_{fist_k} \\ p_{last_{k'}} &= p_{last_l} \end{aligned}$$

In fact, to guarantee minimal waiting time, the time window of the new macro node request collapsed to the interval $[L_k, L_k]$, that is a single point interval. The motivation for collapsing the time window of the combined request is as following. The vehicle always has to wait the waiting time w^+. However, starting to service the macro node m_k at some time earlier than L_k would only cause additional waiting time to be added to the unavoidable waiting time w^+. Collapsing the time window leads to an overall smallest total waiting time spent inside the macro node.

3.7.4 Case three - clustering with overlapping time windows

Case one analyzed a second macro node before the first one. Case two analyzed a second macro node well after the first one. Case three is the remaining case, when there is some overlap between the macro node request time windows. Therefore the condition of case three can be defined as the negation of the condition of cases one and two. Negation of the condition of case one is that the latest arrival time of the second macro node request must be greater or equal to the earliest departure time plus the travel time of the first macro node request. Formally it can be written as:

$$L_l \geq E_k + S_k + t'_{k,l}$$

The negation of the condition of case two would be that the earliest arrival of the second macro node request is earlier then the latest departure plus travel time of the first macro node request. Formally we write that as:

$$E_l \leq L_k + S_k + t'_{k,l}$$

To show the result of combining two macro nodes, we further distinguish four different cases. Two cases of temporal relationship between the earliest arrival times of both macro nodes and two cases of temporal relationship between the latest arrival times of both macro nodes.

3.7.4.1 New earliest arrival time

Let us consider case 3.1 where the earliest arrival time of the second macro node request E_l is later than the earliest departure time of the first macro node $E_k + S_k$ plus the travel time between the nodes $t'_{k,l}$. Formally this condition can we written as:

$$E_l > E_k + S_k + t'_{k,l}$$

Figure 3.6 demonstrates the relationship between the early arrival times of both macro nodes.

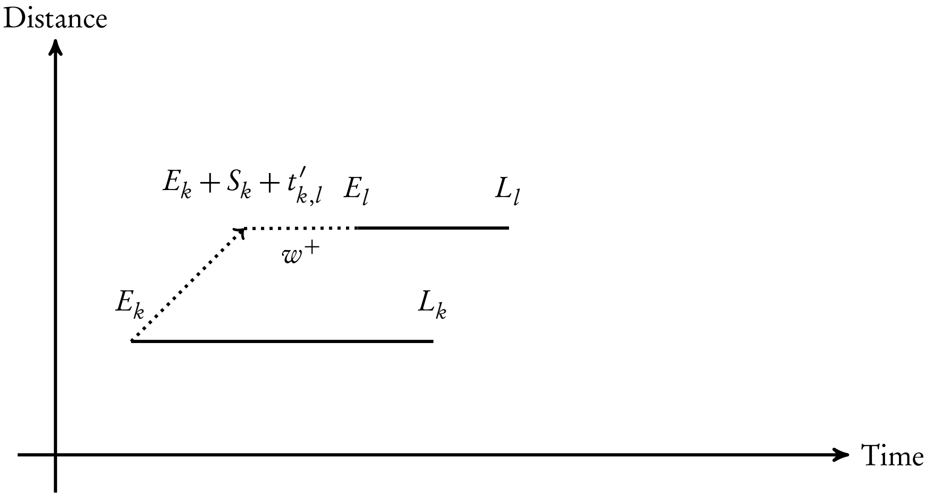

Figure 3.6: Earliest arrival of the second macro node is after the earliest departure of the first macro node

In order to avoid introducing extra waiting time w^+, which will result in a longer overall servicing time of the combined macro node, we will push forward the earliest arrival time E_k to avoid introducing the waiting time w^+.

To avoid the waiting time between the two macro nodes the earliest arrival time of the combined macro node request is equal to

$$E_{k'} = E_l - (S_k + t'_{k,l})$$

Figure 3.7 shows the previous example with the narrowed earliest arrival time of the new combined macro node.

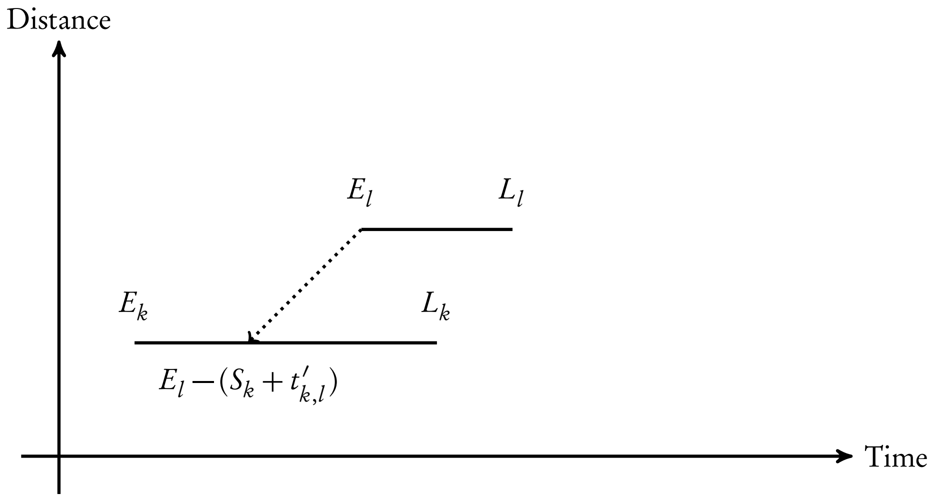

Figure 3.7: Earliest arrival $E_{k'}$ of the new combined macro node

The other case (case 3.2) of the relationship of earliest arrival time between the two macro nodes is when the earliest arrival time of the second macro node is earlier than the earliest departure time of the first macro node plus the travel time between them. Formally we write it as:

$$E_l \leq E_k + S_k + t'_{k,l}$$

See Figure 3.8 for an illustration of the time relationship in case 3.2.

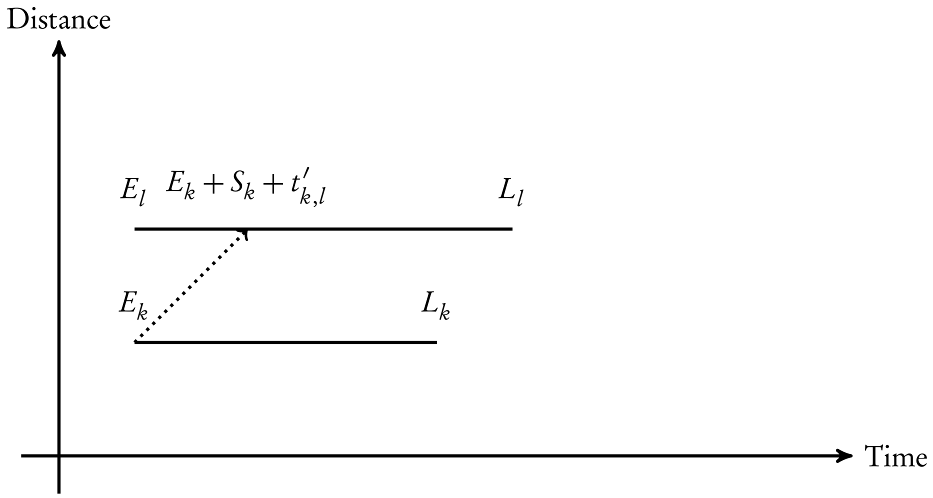

Figure 3.8: Early arrival of second macro node before the early departure of first macro node

Since $E_l \leq E_k + S_k + t'_{k,l}$ we do not introduce any waiting time if we service the first macro node request at time E_k. Therefore the earliest arrival time of the combined macro node does not need to be narrowed, and the earliest arrival time of the combined macro node stays the same as the earliest arrival time of the first macro node.

$$E_{k'} = E_k$$

Cases 3.1 and 3.2 can be handled by a single formulation

$$E_{k'} = max(E_k,\, E_l - (S_k + t_{k,l}))$$

3.7.4.2 New latest arrival time

Similarly, we also have two cases for the relationship between the latest arrival time of the two macro node requests.

In case 3.3 the latest arrival time of the second macro node request L_l is earlier than the latest departure time of the first macro node request $L_k + S_k$ plus the travel time between $t'_{k,l}$. Formally it can be written as:

$$L_l < L_k + S_k + t'_{k,l}$$

We illustrate this case in Figure 3.9

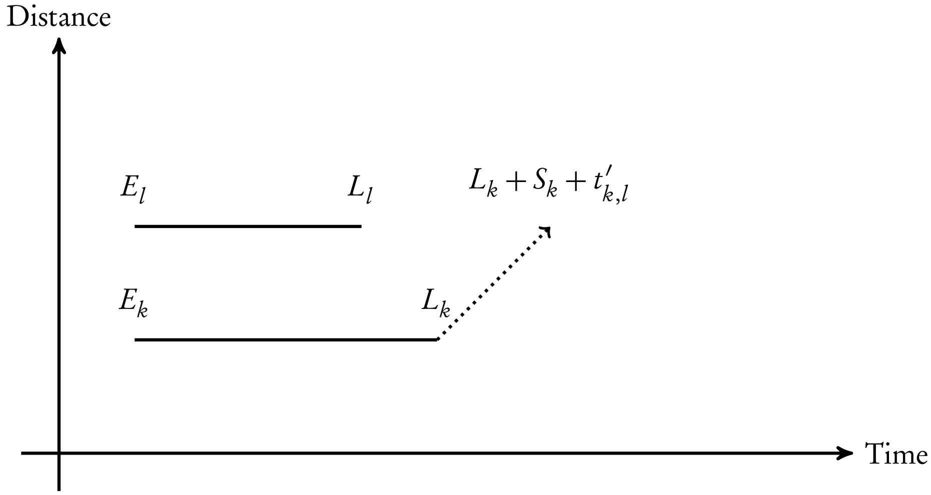

Figure 3.9: Latest arrival of second macro node is before the latest departure of the first macro node

Servicing the first macro node request at its latest arrival time L_k would make it impossible to service the second macro node within its time window $[E_l, L_l]$. Therefore the combined macro node request has to narrow its latest arrival time such that both macro node requests can be feasible. The value of the latest arrival time of the combined macro node request can be calculated as

$$L_{k'} = L_l - (S_k + t'_{k,l})$$

Figure 3.10 illustrates the new narrowed latest arrival time of the combined macro node request.

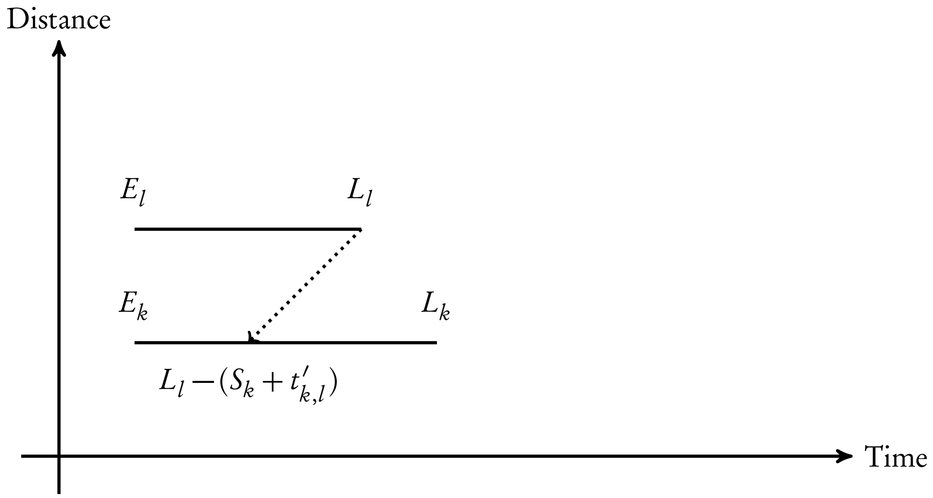

Figure 3.10: Latest arrival $L_{k'}$ of the new combined macro node

As case 3.2 is the opposite of case 3.1, the next case 3.4 is the opposite of case 3.3. In case 3.4 we have that the latest arrival time of the second macro node request L_l to occur later then the latest departure of the first macro node request $L_k + S_k$ plus travel time $t'_{k,l}$. Formally we can write it as :

$$L_l \geq L_k + S_k + t'_{k,l}$$

This condition is illustrated in Figure 3.11

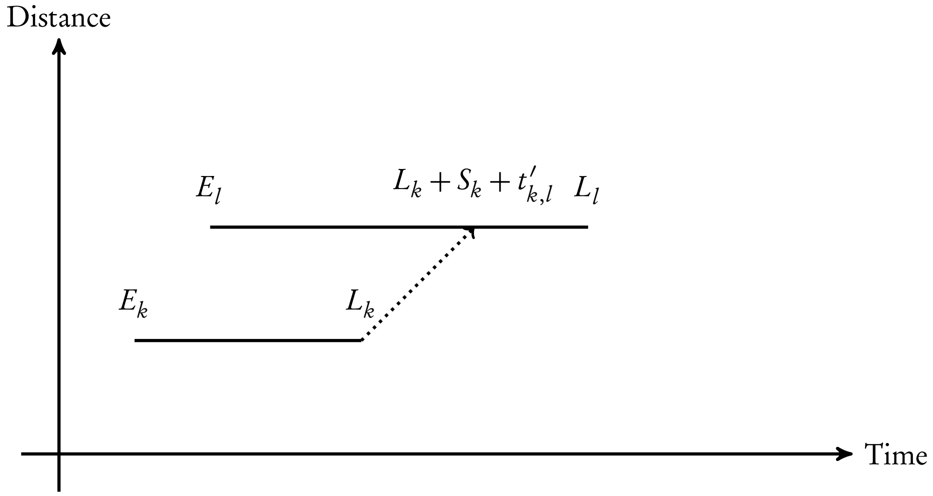

Figure 3.11: Latest arrival of the second macro node request is after the latest departure of the first

The latest arrival time of the second macro node request does not narrow the latest arrival time of the combined macro node request. Therefore we can set the latest arrival time of the combined macro node request as the latest arrival of the first macro node request.

$$L_{k'} = L_k$$

Similar to the single formulation to handle cases 3.1 and 3.2, we introduce a single formulation to handle cases 3.3 and 3.4

$$L_{k'} = min(L_k, L_l - (S_k + t'_{k,l}))$$

3.7.5 Combined macro node request properties

To summarize the full analysis that we performed for all three cases. The resulting properties of the combined macro node request can be computed as follows.

Case one is the only unfeasible case. Therefore the condition to decide if two macro nodes **can be combined** is the negation of the condition in case one. Therefore two macro nodes can be feasibly combined, if they obey the following condition:

$$L_l \geq E_k + S_k + t'_{k,l}$$

If the two macro nodes obey the stated feasibility condition, then the properties of the resulting macro node request can be computed as

$$\begin{aligned}
E_{k'} &= min(L_k, max(E_k, E_l - (S_k + t'_{k,l}))) \\
L_{k'} &= min(L_k, L_l - (S_k + t'_{k,l})) \\
S_{k'} &= S_k + S_l + t'_{k,l} + w^+ \\
&\quad \text{where } w^+ = max(0, E_l - (L_k + S_k + t'_{k,l})) \\
Q_{k'} &= Q_k + Q_l \\
p_{first_{k'}} &= p_{first_k} \\
p_{last_{k'}} &= p_{last_l}
\end{aligned}$$

3.7.6 Vehicle fleet constraints

Until now we have ignored that the VRPTW problem description also defines constraints on the available vehicle fleet, and our macro nodes must respect the constraints imposed by the vehicle fleet. Since we are assuming a homogeneous vehicle fleet, the fleet can be described by:

- Maximal capacity of the vehicles Q_{max}

- Vehicle working time interval $[E_v, L_v]$

To respect the capacity constraint we simply check that we never combine two macro nodes whose combined demand quantity exceeds the vehicle capacity.

$$Q_k + Q_l < Q_{max}$$

To respect the time window constraint we have to check two things.

1. That the latest arrival time at a macro node request is not narrowed beyond the time a vehicle can't arrive to it.
2. That the latest departure time from a macro node request is not later then the working time window of a vehicle.

Hence, when combining two macro node requests $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ and $r_{m_l} = \langle E_l, L_l, S_l, Q_l, p_{first_l}, p_{last_l} \rangle$ into macro node request $r_{m_{k'}} = \langle E_{k'}, L_{k'}, S_{k'}, Q_{k'}, p_{first_{k'}}, p_{last_{k'}} \rangle$. This macro node request must satisfy the following two conditions.

$$E_v + t_{depot, p_{first_{k'}}} \leq L_{k'}$$

and

$$L_{k'} + S_{k'} + t_{p_{last_{k'}}, depot} \leq L_v$$

The first condition ensures that if a vehicles departs from the depot as early as it can (i.e. at time E_v) and travels the distance until the macro node in time $t_{depot, p_{first_{k'}}}$ it will still be able to service the macro node, because it arrives before the latest arrival time $L_{k'}$.

The second condition ensures that departing at the latest feasible time $L_{k'} + S_{k'}$ and taking the traveling time needed to go back to the depot. A vehicle still arrives back to the depot before its finishing time L_v.

3.8 Summary

To summarize, in this chapter we motivated treating macro nodes as sequences, and how such sequences can be represented by a single request. Then we explained how to decide if two macro nodes can be connected, and what is the resulting request that represents both macro nodes.

4

Cluster Construction

The previous chapter introduced the theoretical foundations for combining requests into macro nodes and encoding these macro nodes as individual requests. This chapter discuses the practical aspects of implementing the before-mentioned clustering algorithm. Also, it will introduce and provide motivation for several heuristic metrics which allow to compare the adequacy of combining two macro nodes together.

4.1 Clustering Algorithm

Our approach for macro node construction is based on the idea that one macro node is appended to the end of another macro node. This approach is similar to the way insertion heuristics construct routes. We adopted the idea used by insertion heuristics but instead of building routes we modify it to construct macro nodes.

Analysis from the previous chapter allows to decide when two macro nodes can be feasibly combined. This guarantees that any macro node that we construct is always serviceable by a vehicle.

4.1.1 Pseudo Algorithm

Algorithm 1. presents the idea behind the insertion heuristics. We initialize a partial route by selecting a random seed request. Then, during every iteration, we examine the cost of all unrouted requests and extend the current partial route with the request that minimizes the cost function. Once no more requests can be added to the current partial route, this partial route is finalized into a route and a new partial route is started by selecting a random seed. The algorithm terminates once every request is assigned to a route.

Algorithm 1 Pseudo-Algorithm of the insertion heuristic

```
$C \leftarrow$ Request Set
$R \leftarrow \emptyset$                                    ▷ $R$ is the set of routes
while $|C| \geq 0$ do
    $\langle r \rangle \leftarrow$ Initial seed node from $C$
    $C \leftarrow C \setminus \{r\}$
    do
        $c' \leftarrow \arg\min_{c \in C}$ COST$(r, c)$ if FEASIBLE$(r, c)$
        $C \leftarrow C \setminus \{c'\}$
        $r \leftarrow r + \langle c' \rangle$
    while $c' \neq \emptyset$
    $R \leftarrow R \cup \{r\}$
end while
```

There are two parameters that drive the insertion heuristic. The first parameter is the seed request selection criteria and the second parameter is the cost function of appending a customer c at the end of a partial route r. Next we show how to modify these two parameters to adapt the insertion heuristic to construct macro nodes.

Algorithm 2 Pseudo-Algorithm for constructing macro nodes based on the insertion heuristic

```
function CLUSTER(Customer set, target size)
    $S \leftarrow \emptyset$
    for all $c \in$ Customer set do
        $S \leftarrow S \cup$ MACRONODEREQUEST$(c)$
    end for
    while $|S| \geq$ target size do
        $k, l \leftarrow \min_{k \in S,\, l \in S}$ HEURISTIC$(k, l)$ if FEASIBILE$(k, l)$
        if $k \neq \emptyset$ & $l \neq \emptyset$ then
            $k' \leftarrow$ MERGE$(k,l)$
            $S \leftarrow S \setminus \{k, l\}$
            $S \leftarrow S \cup \{k'\}$
        else
            return $S$
        end if
    end while
    return $S$
end function
```

Algorithm 2. presents the general principal behind our clustering algorithm. Algorithm 2 can be described as follows. We first create a macro node request for each request (the set of all the macro node requests is denoted by S). Then during each iteration we pairwise compare all macro nodes in the set S and compute the heuristic function of combining these macro nodes. The pair with the lowest value is then merged into a single macro node. We repeat this process until either, the macro node request count is reduced to the target size, or there are no more merges possible. It is very important to stress that the clustering process may not reduce the instance size to the desired target size.

The difference between our clustering algorithm and the insertion heuristic presented in Algorithm 1 is that while the insertion heuristic works by constructing one route at a time (when a route can not be extended, the algorithm starts a new route), our algorithm initially creates many macro nodes and tries to extend them all simultaneously. Our algorithm stops combining macro nodes when the amount of macro node requests has decreased to some intended target size. Also, the heuristic function that evaluates appending a request onto a route is different from the cost function of appending one macro node onto another.

4.1.2 Neighborhood Candidate Lists

There is a simple optimization approach one can use to speed up the clustering algorithm. Since in every iteration we scan through all pairs of macro node requests to compute their merge cost, this leads to a quadratic complexity for every iteration. In fact only macro node requests which are nearby, should be considered for merging, but instead we are repeatedly examining merging options that would produce bad results. Therefore we can perform a single prepossessing step where for each request we compute its neighborhood set (i.e. the set of closest requests). Then instead of computing merging cost of a macro node with every other macro node, we will only compute the merge cost with macro nodes included in the neighborhood set.

Computing the neighborhood set is a costly operation, therefore we want to do it only once. However after merging macro node requests, some of the requests that we have marked as in the neighborhood set will become internal nodes of some macro node. Therefore when we are examining the neighborhood set to compute the merge cost, we have to examine the neighborhood set of the last request of the macro node. And in the neighborhood set of the last request we have to ignore requests that are not the first request of some macro node, because we can only connect two macro nodes by their tail and head.

We can motivate this approach, by assuming that if we can not find a good merge within the top-k closest neighbors then this macro node request should not be clustered further. See Figure 4.1 for an example where the top-6 neighbor set does not have a feasible merge option for request i.

Given this new idea, we can rewrite our clustering algorithm as shown in the Algorithm 3, we have precomputed the neighbor list for each request.

Algorithm 3 Final Clustering Algorithm

```
function CLUSTER(Customer set, target size, neighborhood)
    $S \leftarrow \emptyset$
    for all $c \in$ Customer set do
        $S \leftarrow S \cup \text{MACRONODEREQUEST}(c)$
    end for
    $ClusterHeadSet \leftarrow$ Customer set
    while $|S| >$ target size do
        $k\_best \leftarrow \emptyset$
        $l\_best \leftarrow \emptyset$
        $cost\_best = \infty$
        for all $k \in S$ do
            for all $l \in neighborhood(k)$ do
                if $l \in ClusterHeadSet$ then
                    if FEASIBILE(k, l) then
                        $cost \leftarrow \text{HEURISTIC}(k, l)$
                        if $cost < cost\_best$ then
                            $cost\_best \leftarrow cost$
                            $k\_best \leftarrow k$
                            $l\_best \leftarrow l$
                        end if
                    end if
                end if
            end for
        end for
        if $k\_best \neq \emptyset$ & $l\_best \neq \emptyset$ then
            $k' \leftarrow \text{MERGE}(k\_best, l\_best)$
            $S \leftarrow S \setminus \{k\_best, l\_best\}$
            $S \leftarrow S \cup \{k'\}$
            $ClusterHeadSet \leftarrow ClusterHeadSet \setminus \{l\_best\}$
        else
            return $S$
        end if
    end while
    return $S$
end function
```

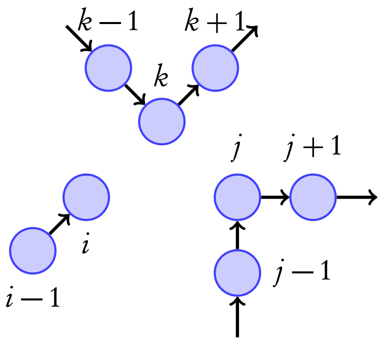

Figure 4.1: Example of neighborhood without any merge options

4.1.3 Computational Complexity

In algorithm 3 every iteration of the while loop reduces the size of our macro node request set by one, because we merge two macro node requests into one. This means that if our initial request count was N and our reduced problem size should be m then this loop will execute $N-m$ iterations.

The body of the while-loop consists of a double for-cycle. The outer for-cycle iterates over all macro node requests in the set S. The inner for-cycle iterates over all the k-neighbors of the tail customer of the current macro node request. Therefore the two for-cycles have a $O(|S|*k)$ complexity, since FEASIBILE is a constant time operation $O(1)$.

The resulting complexity of the whole algorithm is then $O((N-m)*(|S|*k))$. The cardinality of the set S changes every iteration. More specifically, it decreases by one, starting from N and finishing at m.

Since during every iteration we have to execute $|S|*k$ many operations and we have $N-m$ many iterations, we can compute the total number of operations executed as the sum

$$N*k+(N-1)*k+(N-2)*k+\ldots+m*k$$

We notice that every term in the sum is multiplied by k. Therefore we can rewrite this sum as

$$k*(N+(N-1)+(N-2)+\ldots+m)$$

This sum can be rewritten as the mean value of the first and last element multiplied by the number of elements in the sum.

$$k*\frac{N+m}{2}*[N-m]$$

After simplifying, this expression is equal to

$$k * (\frac{N^2 - m^2}{2})$$

Since $m \sim N$ and we are interested in the big O notation complexity, the term $O(N^2 - m^2) = O(N^2)$. Then the final complexity of the clustering algorithm is equal to

$$O(k * N^2)$$

4.2 Clustering Heuristics

In this section, we discuss how to compute the HEURISTIC function that evaluates the profitability of merging two macro node requests together. The particular cost heuristics we have chosen to use have historically showed good results for VRPTWs and are easy to implement and extend for the macro node case.

4.2.1 Distance Heuristic

The distance heuristic between two macro node requests is simply the distance (travel time) between the last request of the first macro node and the first request of the second macro node.

Definition 7. *Let* $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ *and* $r_{m_l} = \langle E_l, L_l, S_l, Q_l, p_{first_l}, p_{last_l} \rangle$ *be two macro node requests. Then the distance heuristic between them is equal to*

$$heuristic_{dist} = t'_{k,l}$$

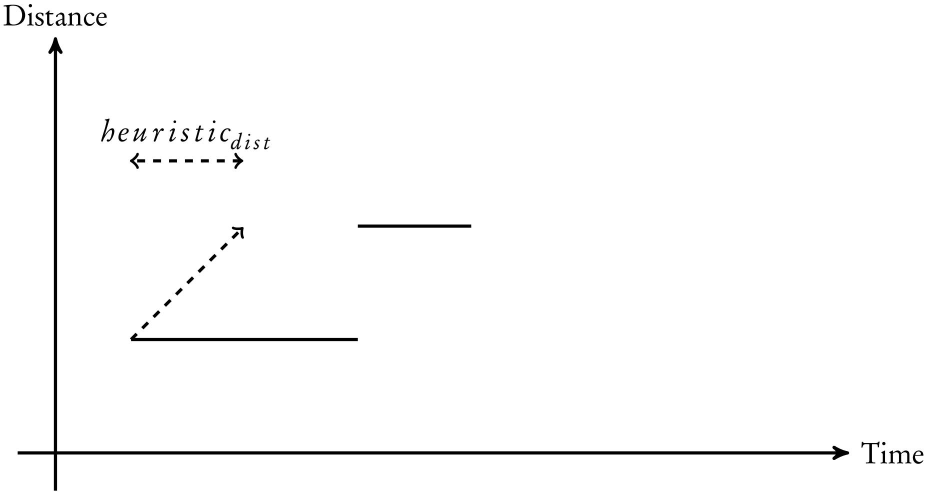

Figure 4.2: Ilustration of the distance heuristic between two macro node requests

4.2.2 Waiting Time Heuristic

The waiting time heuristic between two macro nodes is the waiting time the vehicle must incur between the last request of the first macro node and the first request of the second macro node.

Definition 8. *Let* $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ *and* $r_{m_l} = \langle E_l, L_l, S_l, Q_l, p_{first_l}, p_{last_l} \rangle$ *be two macro node requests. Then the waiting time heuristic between them is equal to*

$$heuristic_{wait} = max(E_l - (L_k + S_k + t'_{k,l}), 0)$$

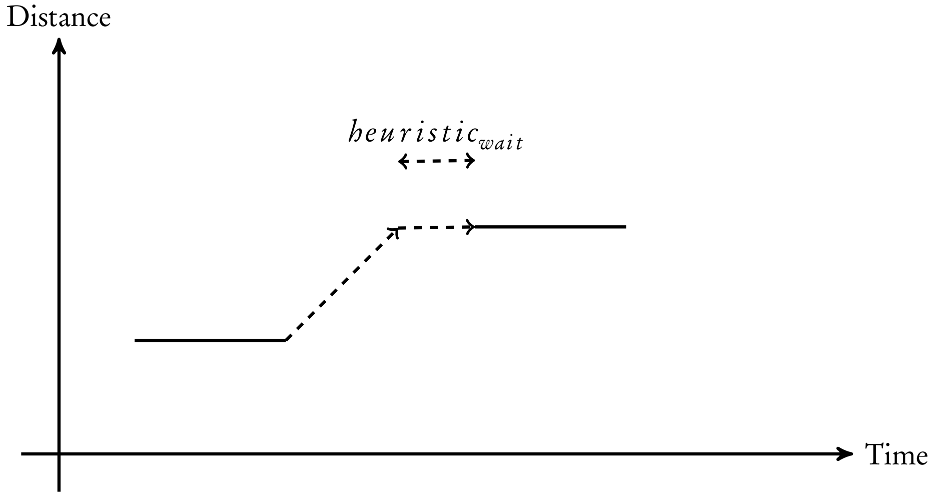

Figure 4.3: Ilustration of the waiting time heuristic between two macro node requests

4.2.3 Flexibility Heuristic

The flexibility heuristic represents the amount of narrowing of the time window of the combined macro node after we have merged two macro nodes together.

Definition 9. *Let* $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ *and* $r_{m_l} = \langle E_l, L_l, S_l, Q_l, p_{first_l}, p_{last_l} \rangle$ *be two macro nodes. Then the flexibility heuristic between them is equal to*

$$heuristic_{flex} = \delta_E + \delta_L$$

Where

$$\delta_E = max((E_l - S_k - t'_{k,l}) - E_k, 0)$$

and

$$\delta_L = max(L_k - (L_l - S_k - t'_{k,l}), 0)$$

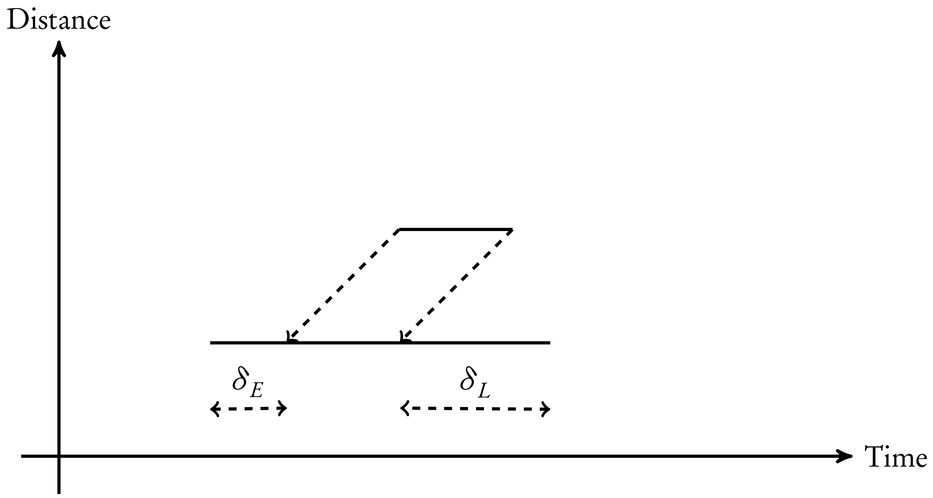

Figure 4.4: Ilustration of the flexibility cost between two macro node requests

4.2.4 General Heuristic

The resulting heuristic we use in our application is a linear combination of the above mentioned heuristics.

Definition 10. *If* $r_{m_k} = \langle E_k, L_k, S_k, Q_k, p_{first_k}, p_{last_k} \rangle$ *and* $r_{m_l} = \langle E_l, L_l, S_l, Q_l, p_{first_l}, p_{last_l} \rangle$ *are two macro node requests then the generalized heuristic is equal to*

$$heuristic_G(r_{m_k}, r_{m_l}) = \gamma heuristic_{dist}(r_{m_k}, r_{m_l}) + \beta heuristic_{wait}(r_{m_k}, r_{m_l}) + \alpha heuristic_{flex}(r_{m_k}, r_{m_l})$$

where $\alpha, \beta, \gamma \geq 0$

4.3 Summary

In this chapter we gave detailed pseudo-code of our approach of combining two macro nodes. We gave a formal complexity analysis of our clustering algorithm. We introduced the concept of a request neighborhood set, an optimization technique that allowed us to avoid unnecessary computations. In addition, we introduced a heuristic function that allows us to evaluate the goodness of combining two macro nodes. This heuristic is based on distance, waiting time and time window width.

5

Experimental Results

5.1 Benchmark Data Set

Empirical analysis is the most common way how Operations Research (OR) and Artificial Intelligence (AI) communities evaluate heuristic quality. Empirical analysis uses an extensive set of problem instances, benchmark instances, which should have the following properties:

- Must represent a wide range of possible case.
- Must represent common patterns of practical problems

[12] presents a widely adopted benchmark set. This set contains instances of 200, 400, 600, 800 and 1000 requests. Every instance has a central depot, vehicle capacity constraints, time windows on the delivery and total route duration restrictions.

Each group of instances of a fixed size consists of six different classes of problems, namely R1, C1, RC1, R2, C2 and RC2. Each class contains 10 problems over a service area defined on a 500x500 grid.

Request coordinates are generated as follows. For the R1 and R2 classes data is selected from a uniform distribution, for the C1 and C2 classes coordinates are generated to produce clusters of nearby customers. Classes RC1 and RC2 contain a mix of clustered and random coordinates.

All problems of a particular class have the same customer locations and the same vehicle capacities. The only difference is the percentage of customers with time window constraints (i.e. 25%, 50%, 75% and 100% time window density).

Classes R1, C1 and RC1, have a short scheduling horizon, narrow time windows and small vehicle capacities, which allows only a few requests to be serviced by the same vehicle. On the contrary, classes R2, C2 and RC2, have a long scheduling horizon enabled by long route duration together with large vehicle capacities, thus allowing individual vehicles to service many customers.

5.1.1 Very large instances

Since the literature does not contain any instances larger than the [12], and since our work is designed to handle cases of very large instances, we had to create very large instances of size 10'000 request on our own.

To achieve this we combined ten of the 1'000 request problems from [12] into a single 10'000 problem. We did this for every one of the 6 classes (R1, C1, ...), obtaining 6 problems of size ten thousand.

5.2 Numerical Results

In the previous chapter we discussed the global heuristic function, which measures the goodness of combining two macro nodes. The global heuristic function is a linear combination of three different heuristic functions - the distance, the waiting and the flexibility. Parameters α, β and γ, determine the weight of these different functions in the global heuristic function.

In order to find the best values for α, β and γ, we evaluated the performance of clustering over the whole benchmark set with different combinations of values of alpha, beta and gamma (parameter scanning). Bellow is the table that lists the combinations that we evaluated

α	β	γ
0.0	0.0	1.0
0.0	1.0	1.0
0.1	1.0	1.0
1.0	0.0	1.0
1.0	1.0	1.0
1.0	0.1	1.0

Values for the combinations were selected as follows. The weight of the distance γ is always set to 1.0 and the weight of the waiting β and flexibility α where evaluated at values 0.0, 0.1 and 1.0.

The motivation for always setting the weight of the distance heuristic to 1.0 is that combining two macro nodes that are far apart is usually a bad idea, hence this parameter favors joining macro nodes that are nearby.

In contrast, the degree of importance of minimizing waiting time or keeping macro node request time windows wide is not obvious. In order to test the importance of these factors, we set their weight to either 0.0 which turns it off, or to 0.1 which enables a small influence, or to 1.0 which turns the heuristic fully on.

It is important to point out, that individual heuristics inside the global heuristic are not normalized. First of all, they all have the same dimension - time units. Secondly, their effect on the degradation of the objective function is comparable, as their values have largely similar orders of magnitude (e.g. making a request one hour less flexible, or introducing an extra one hour waiting time, or even spending one more hour traveling)

5.2.1 Evaluation metrics

For each combination we present a number of characteristic properties, that describe its performance. We are interested in the following performance metrics

- Number of vehicles needed to route the problem
- Sum of duration of all routes
- Objective function
- Time needed to solve the problem

The objective function is computed according to the following formula

$$\text{Number of vehicle} * 1000000 + \text{total route duration}$$

The value 1000000 was selected to be large enough to force the solver to try first to optimize the vehicle count and then optimize the total route duration.

We note that we don't display the absolute value of these metrics, but instead the relative value of the metric w.r.t. the value of an unclustered instance as obtained from the Indigo solver. For example, a value of 1.1 as the vehicle number metric indicates that, comparing to the unclustered solution, the clustered solution needed 10% more vehicles.

5.2.2 Reduction limit

Before running the numerical experiments, we need to decide how much to reduce the original instance, since any instance is only reducible up to some value, and we need to decide what is the smallest reduction size that we will attempt to perform while clustering.

One way to find such a limit is to look at how many vehicles Indigo needed to solve this problem without doing any clustering. If we could reduce the problem to a number of macro nodes that is less than the number of vehicles required by Indigo we immediately get a solution that is better than Indigo's since a macro node can always be serviced by a single vehicle. Since it is unlikely that the greedy macro node construction approach can produce a solution that is better than Indigo's, the number of vehicles needed by Indigo is the reduction limit that we assume.

Figure 5.1 shows the mean value of the number of vehicles needed for the 1000 request instances to solve a problem for each problem class.

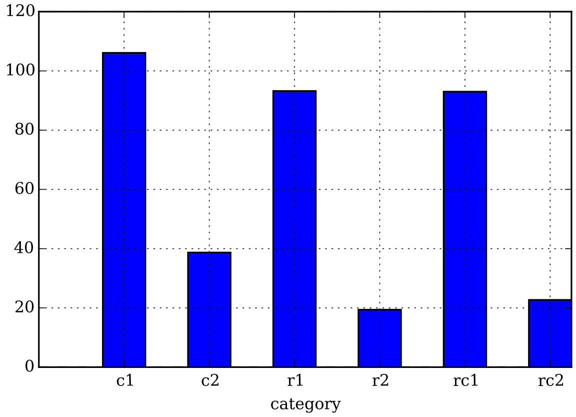

Figure 5.1: Mean number of vehicles needed by class

Figure 5.1 shows that Indigo needs on average 100 vehicles for class C1, 40 vehicles for class C2, 90 vehicles for class R1, 20 vehicles for class R2, 90 vehicles for class RC1, and 20 vehicles for class RC2. We take the upper limit of these numbers, and we do not attempt to reduce any instance more than 10x times, that is from 1000 requests to 100 macro node requests.

5.2.3 Reduction range

To better understand the effects of clustering on the performance of Indigo, for each 1000 request instance, we tried several reduction sizes. First we tried to reduce the problem from 1000 requests to 800 macro nodes, then to 500 macro nodes, then to 200 macro nodes and then to 100 macro nodes. For each reduction we record the four characteristic metrics introduced earlier (number of vehicles, total route duration, objective function, execution time).

In this way, we can see how the reduction amount influences the performance of Indigo relative to not doing any reduction.

5.2.4 Reduction rate

Not every instance in the benchmark set can be reduced to the target size of 800, 500, 200, 100 macro nodes. To measure the amount of successfully reduced instances we introduce the metric - **reduction rate**. Reduction rate indicates how many problems of a given class were successfully reduced to the desired size. A reduction rate of 0.8 for class C2 at reduction size 500, would indicate that only 80% of instances in class C2 could be reduced to the size 500.

In all the future plots, whenever we show a mean value of the four characteristic metrics, we show the mean value of those instances that were succesfully reduced to the reduction size. So the mean value of 1.1 at reduction size 200 means that out of all the instances that were succesfully reduced to 200, the mean value is 1.1. Also, we show a plot indicating the reduction rate for every class at every reduction size.

5.2.5 Performance of all metrics over all heuristics

In Figure 5.2 we display how each heuristic, averaged over all the instances in all the classes, performed relative to other heuristics.

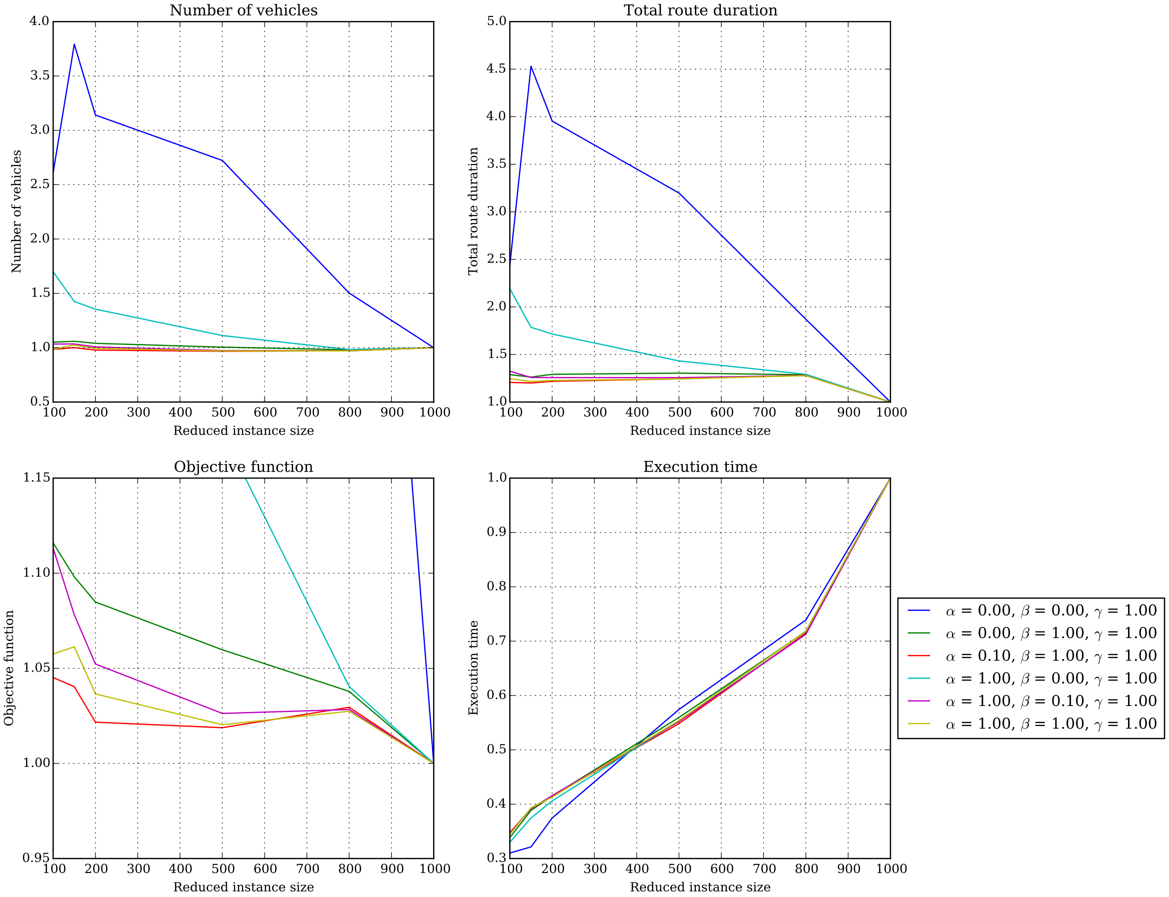

Figure 5.2: Relative performance of different combinations

Before analyzing data provided by Figure 5.2 we introduce a shorthand notation for displaying the weights of a heuristic. We abbreviate a heuristic $\alpha = 0.1, \beta = 0.1, \gamma = 0.1$ as vector $(0.1, 0.1, 0.1)$ s.t. the first argument is the weight of α, the second argument is the weight of β and the third argument is the weight of γ. This notation allows us to avoid explicitly writing the names - alpha, beta, gamma, and instead use the position in the vector as the identifier.

We first look at the vehicle number subplot. This subplot shows that the two heuristics (0.0, 0.0, 1.0) and (1.0, 0.0, 1.0) introduce a large degradation for the required vehicle count (increased to 380% and 170% respectively). Since the objective function is determined by the vehicle count, a similar degradation happens with the objective function.

These are the only heuristics with waiting weight 0. We can conclude that, turning off the waiting factor causes large degradation of the objective function. Intuitively, this means that clustering requests, with significant waiting times between them, does not lead to good overall solutions.

If we examine the objective function subplot, we can see that the overall average best heuristic is (0.1, 1.0, 1.0). This heuristic produces a 5% degradation, if the reduction size is set to 100 macro node requests, and only a 2% degradation, if the reduction size is set to 200 macro node requests.

If we examine the execution time subplot, we can see the same speed up for all heuristics. The reduction to 100 macro node requests, results in a 3x time speed (33% of the original execution time).

All the subsequent plots do not display information regarding the worst performing heuristics (1.0, 0.0, 1.0) and (0.0, 0.0, 1.0). We ignore them since they clearly are bad candidates and their presence makes plots harder to read.

5.2.6 Objective function broken by class

Figure 5.3 shows the relative performance of all heuristics broken down into 6 subplots, one for each class (C1, C2, R1, R2, RC1, RC2).

Out of the remaining four heuristics, the worst performing heuristic is (0.0, 1.0, 1.0). This heuristic has the flexibility weight set to 0. In classes C1, C2 and RC2 this heuristic experiences the largest degradation relative to the other heuristics (it is roughly 10% worse). In the remaining classes this heuristic performs only slightly worse then others (3% worse).

The heuristic (0.1, 1.0, 1.0) performs best in classes C1, C2, RC1, RC2. If we change the flexibility weight of this heuristic from 0.1 to 1.0 then the resulting heuristic gives the best results in the remaining classes R1 and R2 (marginally better by 1%).

We can conclude that the overall best objective function is achieved with heuristic (0.1, 1.0, 1.0) and for the random classes R1 and R2 we have a small improvement of 1% if we increase flexibility weight to 1.0.

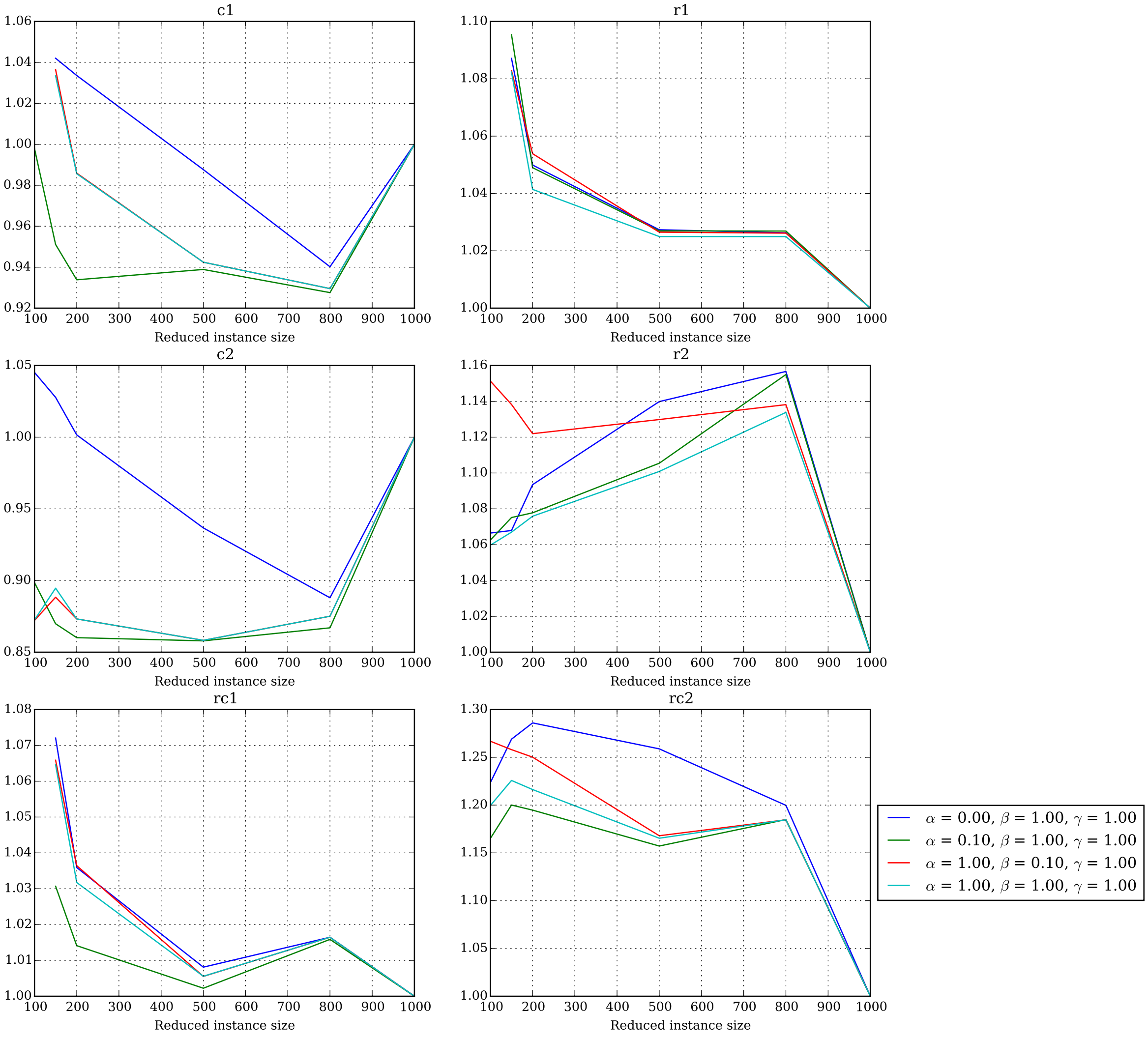

Figure 5.3: Objective function broken down by class

5.2.7 Route duration broken by class

Similarly, how Figure 5.3 shows the break down by class for the objective function, Figure 5.4 shows the break down by class for the total route duration.

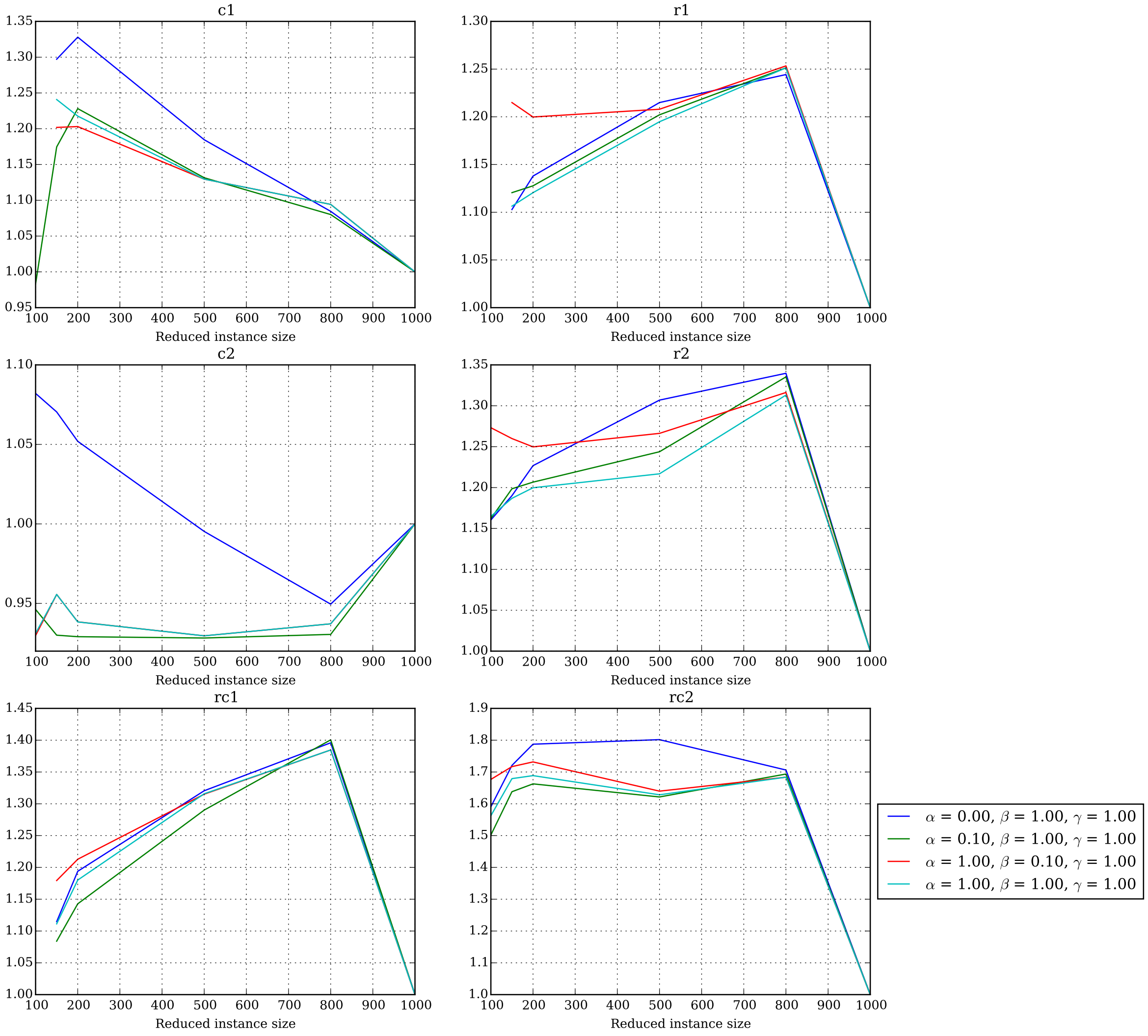

Figure 5.4: Route duration broken down by class

The conclusions we reached, when analyzing different heuristics for the objective function degradation carry over for the route duration degradation.

Namely, the heuristic (0.1, 1.0, 1.0) gives the shortest routes in classes C1, C2, RC1, RC2, while the heuristic (1.0, 1.0, 1.0) gives marginally better results for classes R1 and R2 (1% better).

5.2.8 Total execution time broken by class

Figure 5.5 shows the relative speed up Indigo achieved with the help of clustering.

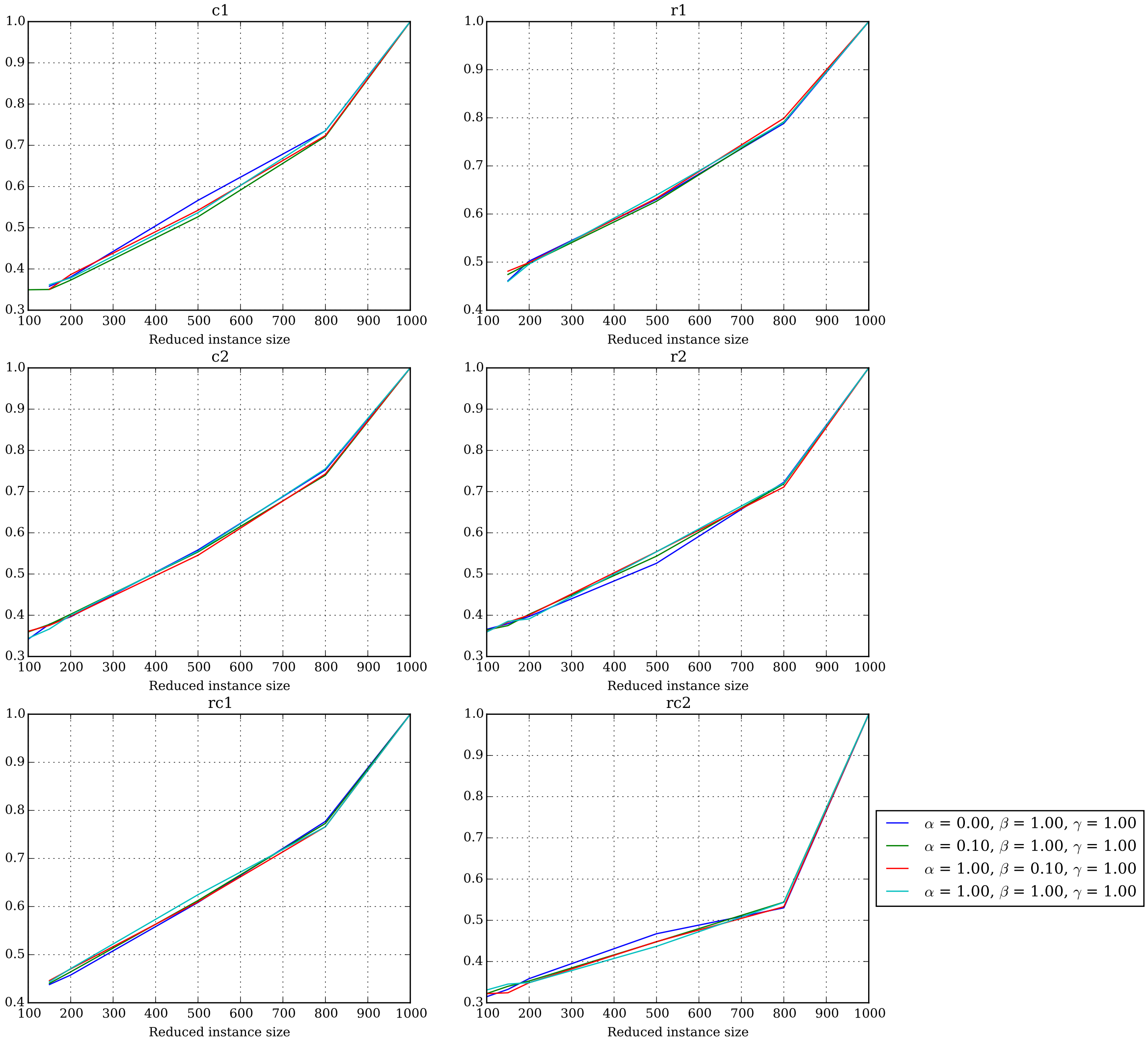

Figure 5.5: Execution time broken down by class

We already saw in Figure 5.2 that overall all heuristics gave a similar speed up of 3x. However, if we break down these results by class, we can observe, that classes RC1 and R1 get a speed up of only 2.5x, while the class RC2 gets a speed up of 3.3x.

5.2.9 Reduction rate broken by class

Figure 5.6 show the reduction rate of each combination broken down by classes.

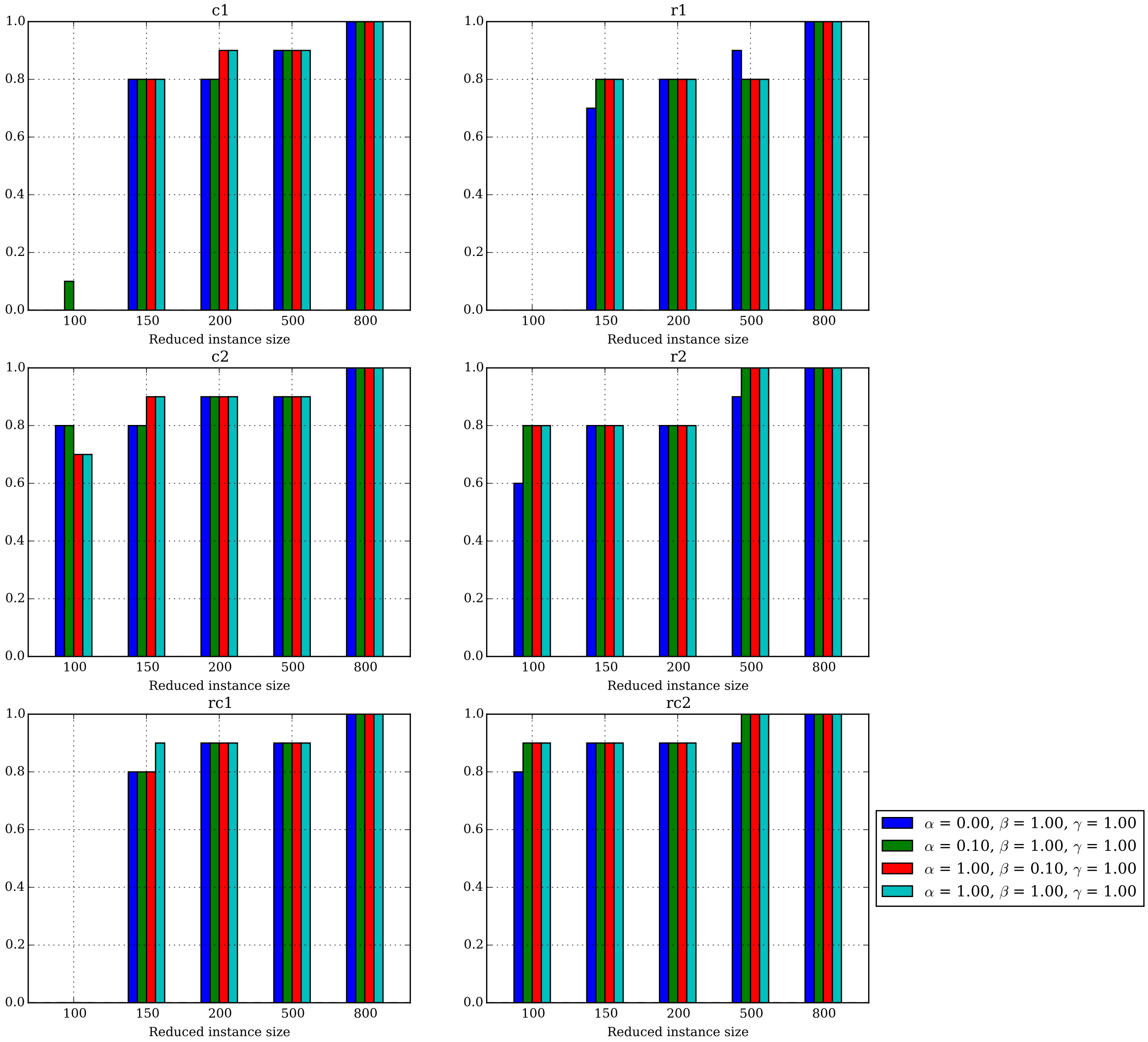

Figure 5.6: Reduction rate broken down by class

We can see that the reduction rate is mostly identical for all heuristics and is equal to 0.9 for most of the reduction values, except at 100 for classes C1, R1 and RC1 the reduction rate becomes 0 for all heuristics.

We remind that according to Figure 5.1 classes C1, R1 and RC1 need on average 100 vehicles to solve the instance. Since our greedy clustering approach is inferior to Indigo's final results, it is likely that clustering can not find a reduction of size 100.

5.2.10 Detailed Figures for the best performing heuristic

In Figures 5.7 we present a detailed break down of performance metrics, for the best performing heuristic (0.1, 1.0, 1.0), broken down over each of the problem classes (C1, C2, R1, R2, RC1, RC2) and the overall average of all classes.

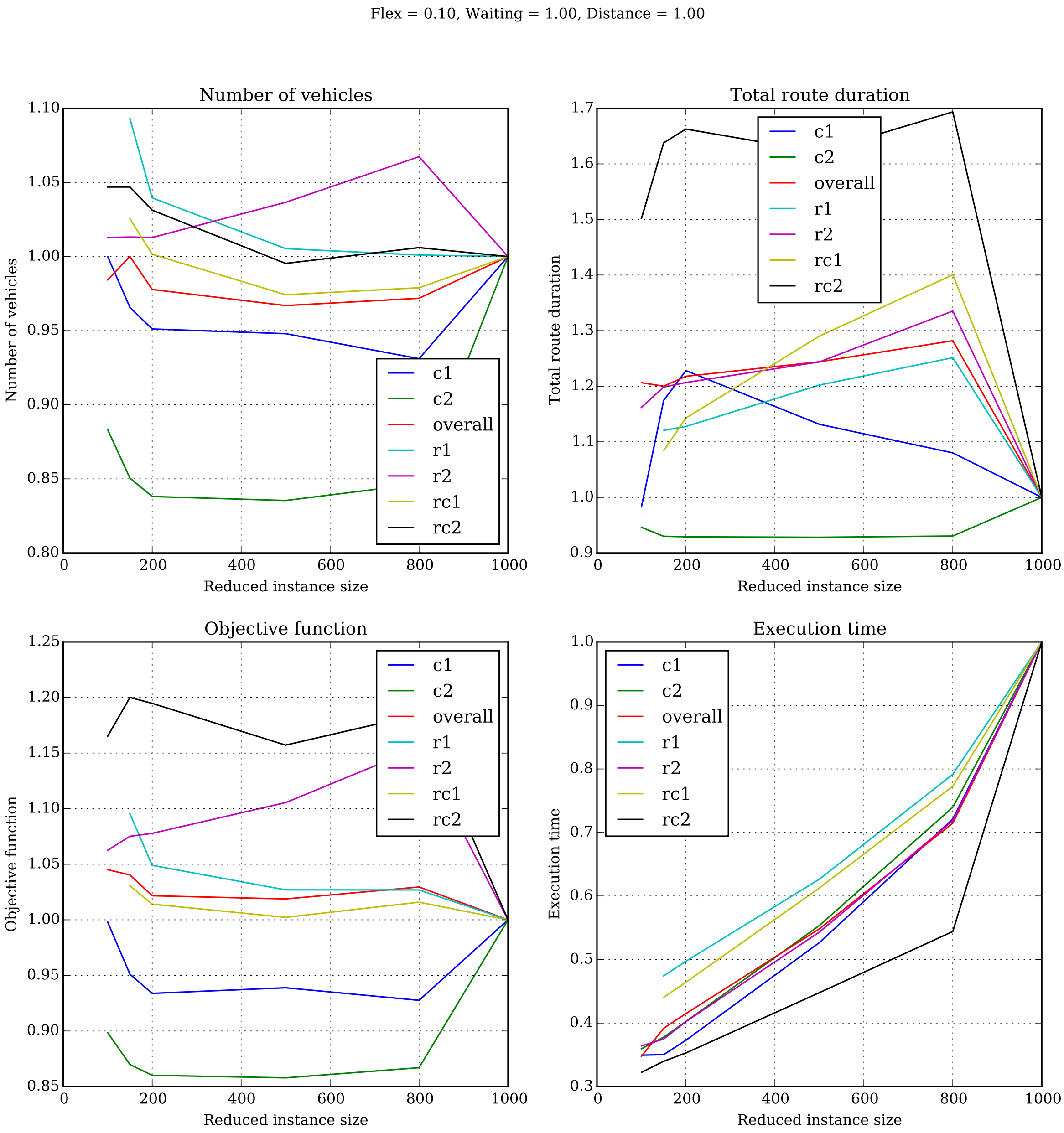

Figure 5.7: Characteristic metrics for $\alpha = 0.1, \beta = 1.0, \gamma = 1.0$

Figure 5.7 shows that for some classes our reduction was able to help Indigo produce better solutions. Namely, for class C1 Indigo was able to find solutions that were 10% more efficient than if no clustering was performed. For class C2 even better improvements of 15% to objective value was reached after clustering. Class RC1 did not receive any degradation to the objective function during clustering. Classes R1 and R2 received a 8% degradation to the objective value after clustering. Class

RC2 suffered the worst degradation, with a 20% degradation to the objective function.

Figure 5.8 shows the reduction rate for each clustering target, broken down by each problem class and the average over all classes.

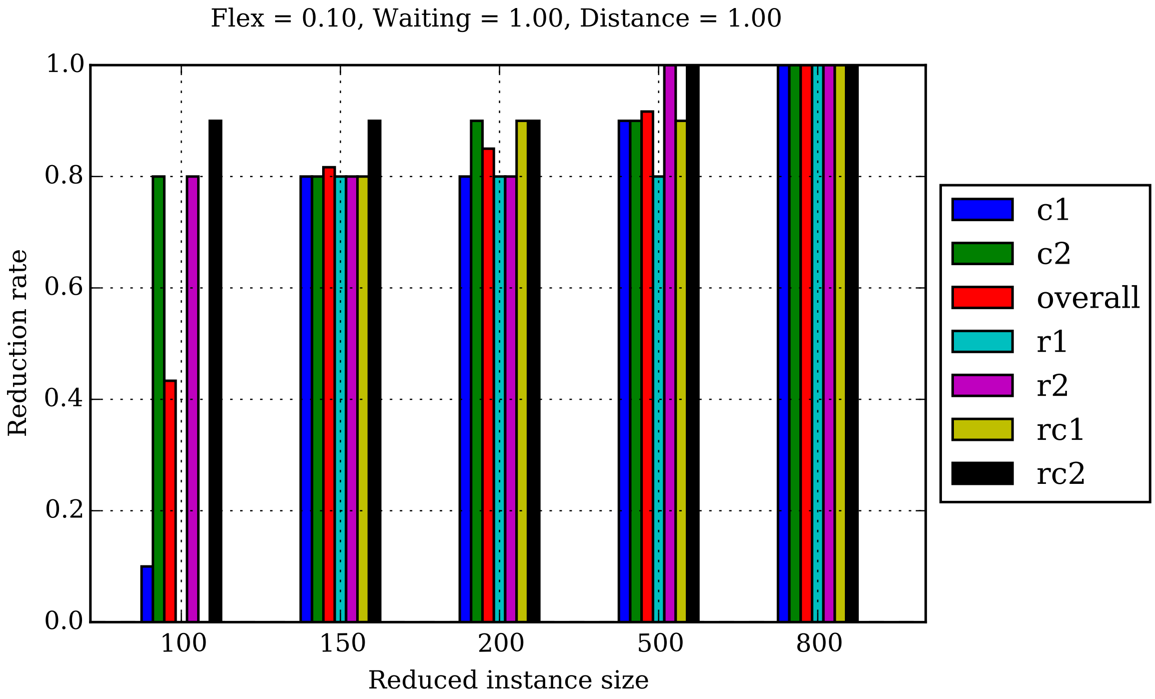

Figure 5.8: Reduction rate for $\alpha = 0.1, \beta = 1.0, \gamma = 1.0$

We can see that the average reduction rate for this heuristic is around 0.9, with a sharp reduction rate fall at 100 macro nodes for classes C1, R1 and RC1.

We can conclude that we found a heuristic, that showed a 5% degradation over all classes, and for the clustered classes showed solution improvements. Independent of the effects to the objective value, this heuristic allowed Indigo to find solutions 3x faster than without clustering.

5.2.11 Very large instances

In Figure 5.9, we show the numerical results of clustering the ten thousand request problems. We had six instances, one instance for each class (C1, R1, C2, R2, RC1, RC2). The reduction sizes during clustering were 8000, 5000, 2000, 1000. Due to the very large execution time it takes to solve a ten thousand request problem (100x longer than a 1k instance, 11 hours comparing to 7 minutes), we only evaluated the heuristic that showed best results for the 1k instances, namely $(0.1, 1.0, 1.0)$.

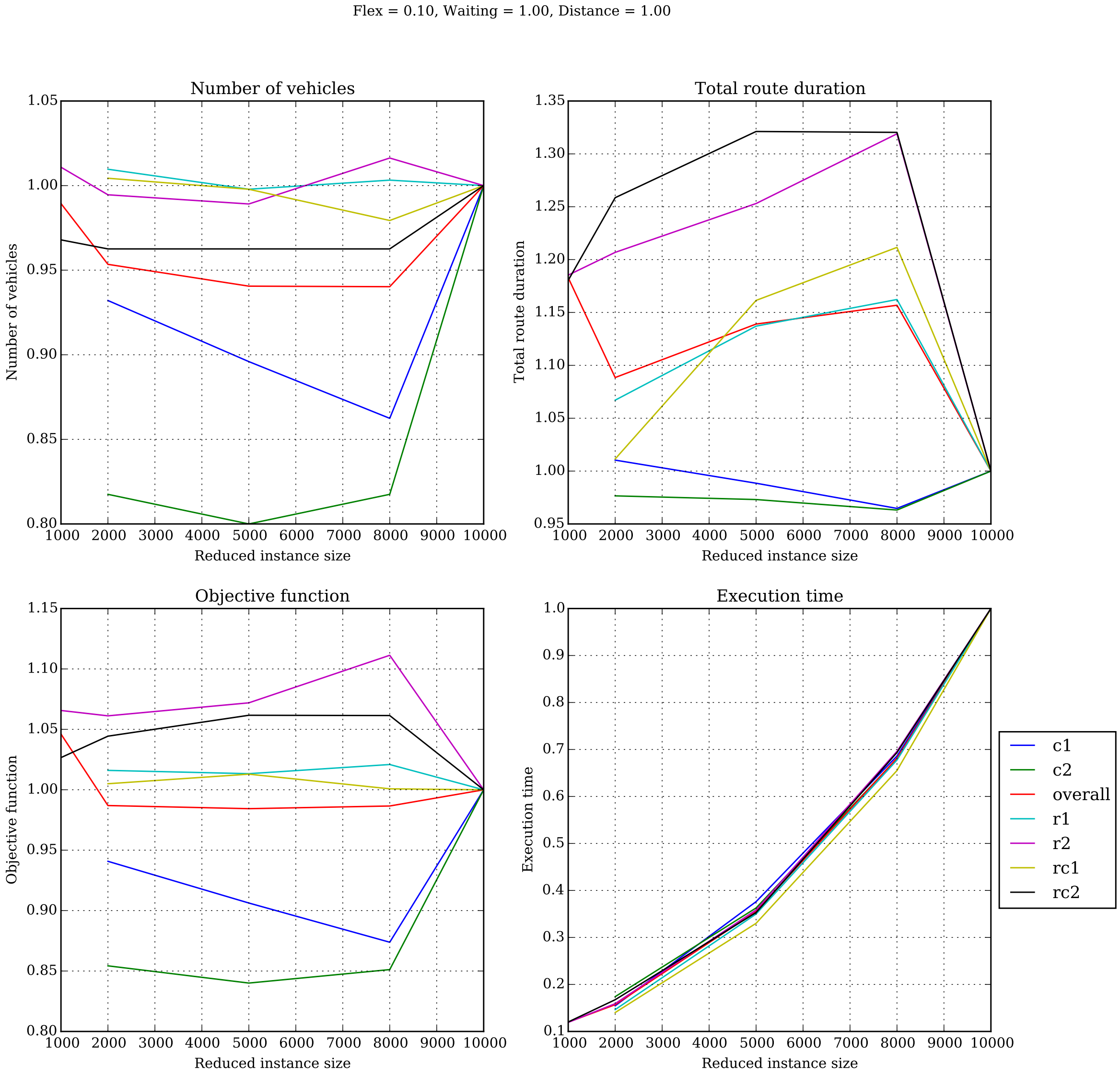

Figure 5.9: Characteristic metrics for $\alpha = 0.1, \beta = 1.0, \gamma = 1.0$ for 10k instances

The results show a similar picture to the results of the 1k instances. The R1, R2, RC1 and RC2 classes suffered a slight degradation. Problems R1 and RC1 received a 1% degradation and problems R2 and RC2 received a 5% degradation. However, problems C1 and C2 received an average of 10% improvement.

We have to be more critical, when interpreting these results, since the structure of the instances may lead to a more favorable results, because the problems were constructed by combining together 10 independent instances and overlaying all the requests together. This could lead to more cluster formation due to the increased request density and therefore problems that are more suitable for clustering.

On the other hand, the execution time improvement are not connected to the structure of the problem, but just its size. In the case of the 10k problems, we see a clear linear reduction in execution time. A 5x reduction of size resulted in a 5x speed up.

Figure 5.10 shows the reduction rate for the ten thousand problems.

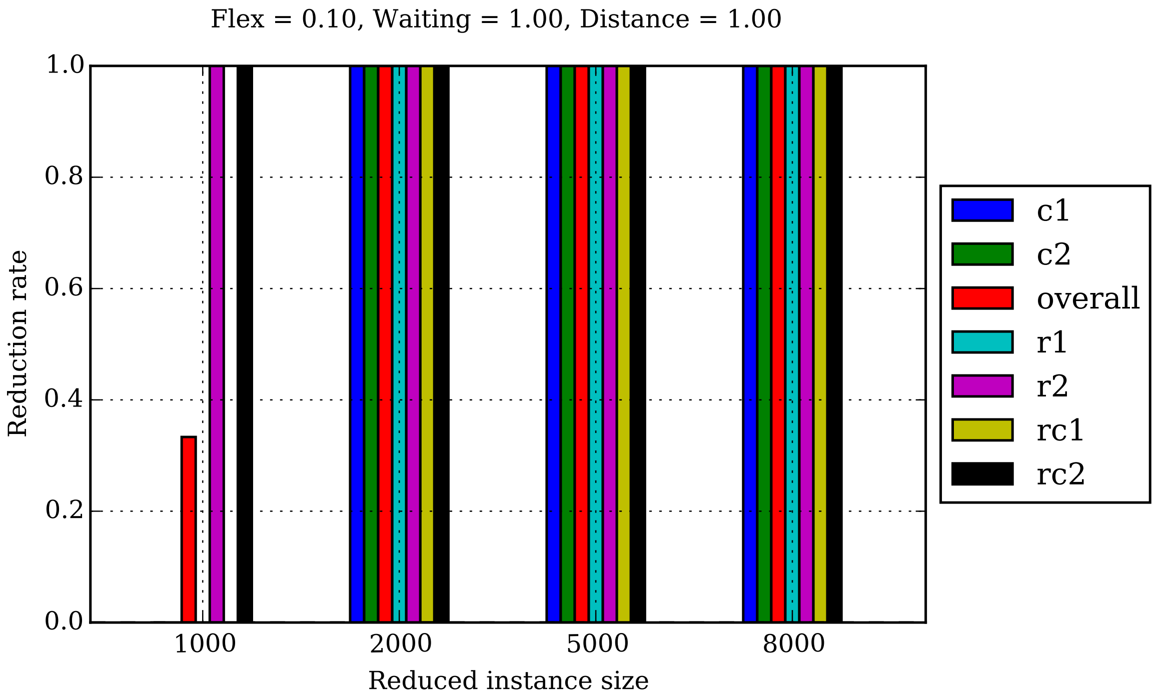

Figure 5.10: Reduction rate for $\alpha = 0.1, \beta = 1.0, \gamma = 1.0$ 10k instances

The reduction rate is, also, similar to the reduction rates with the 1k instances. Namely, all classes were successfully reduced to 5x of the original size. However classes C1, R1, R2 and RC1 could not be reduced to 10x.

6

Conclusions and Future Work

The Vehicle Routing Problem is a well studied combinatorial optimization problem with many real practical applications. The variant of VRP that we considered is the VRPs with hard time windows. Literature is full with different methods for solving this type of VRPs. Exact methods for solving VRPs with optimalitty are based on Mixed Integer Programming (MIP) and are only capable of solving instances with no more than 100 requests. Approximation methods do not provide any guaranties on the solution quality, but are more efficient computationally. Empirical results show that approximation methods are capable of achieving solutions with quality within 1% of optimalitty. However, even approximation methods are computationally expensive due to the large search neighborhoods.

The idea of clustering as a mean to help reduce the size of the search neighborhood was successfully applied to VRPs without time windows, however very little work has been done to apply clustering to VRPs with time windows. Introducing time windows makes it harder to reason if a cluster of requests is serviceable by a vehicle, since requests inside the cluster might be nearby in space and time, but due to the relationship of time windows no possible route is possible that accommodates all the requests of this cluster. In fact, introducing time windows to VRP makes it NP-Complete to decide if a solution for a problem exists at all.

We acknowledge the problem of treating clusters as sets of requests and the uncertainty if such a set can be routed at all. To address the uncertainty of deciding if a set of requests has a route that services all the requests, instead we try to identify sequences of requests (macro nodes). Since a sequence imposes a fixed servicing order on the requests, it is easy to decide, if a macro node can be serviced or not.

Having identified requests that are nearby in space and time we can connect them in a sequence. We can then use a VRP solver to decide how to combine sequences into routes. Since the order of visits is fixed inside the sequence, the solver only needs to decide how to combine the sequences together to form the final solution. Two things become obvious when we try to do this. First, since the order inside the sequences is fixed and the solver can not change it while finding the solution, the quality of the final solution might degrade as the result of constructing bad sequences. And second, since the solver now needs to consider much less objects when routing, finding a solution becomes faster. The degree of quality reduction and the degree of speed up then shows the usefulness of the proposed clustering method.

In order to use a solver to route request sequences (macro nodes) we need to be able to encode a macro node as a single request. Then we can encode all the macro nodes into individual requests, thus constructing a new instance that is smaller and that can be given to any solver as input. The final solution is the solution obtained from the solver where we expand the macro nodes into the sequence of visits that they represent.

To construct the macro nodes we propose a greedy strategy that starts by assuming every request forms a sequence of size one. At every iteration we select two macro nodes and connect them together. In this way, we can control how big the reduced problem will be, since every iteration decreases the size by one. Also, we propose heuristics that guide the macro node combination process. These heuristics consider distance, waiting time and time window width of the macro nodes.

To evaluate the amount of speed up and the amount of solution quality degradation we implemented our algorithm and used a state of the art solver Indigo to route the macro nodes. We executed it over a well recognized benchmark set. This benchmark set consists of three classes of problems: a class of clustered problems whose requests form geographical clusters; a class of random problems, whose requests are randomly scattered; a class that combines both clustered and random locations. All classes have two versions of the problems, one with large capacity and another with small capacity vehicles. Using the problems from the benchmark set, we combined several instances to create new instances of 10x size, and confirmed that the results obtained with 1000 request instances carry over to 10000 request instances.

Numerical results showed, that for 1000 request instances clustering resulted in an average of 5% degradation of the objective function and a speed up of 3x, if compared to running the solver without any clustering. However, the degradation rate of the objective function was different for different classes of problems. The worst degradation happened with the semi random semi clustered class with large capacity vehicles. Degradation in this class was 20%. The degradation of random class was equal to 8%. The degradation of the semi random semi clustered class with small capacity vehicles was around 2%. However for the class of problems, where the requests form geographical clusters, not only did we experience no degradation, but instead clustering enabled the solver to find better solutions. The small capacity clustered class received a 5% improvement and the large capacity clustered class received a 15% improvement.

Similarly, numerical results obtained for the 10000 request instances showed an average degradation of 5% over all classes with an average degradation of 6% in the random and semi random classes and an average improvement of 10% in the clustered classes. The execution speed up was, also, linear, however the slope of the speed up was better, since a 5x reduction in size resulted in a 5x speed up (versus 3x speed up in 1000 request case).

The degradation rate depends on the amount of requests that we joined together in macro nodes. The more requests we join together the less freedom the solver has to find the optimal solution and therefore the more degradation we introduce. However, the more requests we join together the better execution time improvement we receive. This trade off between the speed up and the objective function is not linear. Our results showed that if we perform only a 5x size reduction of the original problem (on average each macro node represents servicing 5 requests), the degradation rate of the objective function is not as bad and on average over all classes is only 2%. However if we perform a 7.5x or a 10x reduction in size there is a sharp increase in the degradation rate, that moves the average to 5%.

We can see several improvements for our approach, that one can explore in the future. First of all, the heuristic function that we proposed to combine macro nodes together is very simple and can be extended. The Operations Research and Artificial Intelligence communities have developed a large set of heuristics one can use to evaluate the goodness of putting a request in a route.

And second, our strategy of combining two requests is greedy - we are always combining the best pair. One can expect better results if we introduced some stochastic element, and sometimes allow a second best combination. Since once a pair of requests is connected, this decision is never revised, and what seemed like the best option locally can lead to worse results globally. Introducing randomness can help search escape local minima.

Bibliography

[1] Roberto Baldacci, E Hadjiconstantinou, and Aristide Mingozzi. “An Exact Algorithm for the Capacitated Vehicle Routing Problem Based on a Two-Commodity Network Flow Formulation”. In: *Operations Research* 52.5 (2004), pp. 723–738. ISSN: 0030364X.

[2] Russell Bent and Pascal Van Hentenryck. “Randomized Adaptive Spatial Decoupling for Large-Scale Vehicle Routing with Time Windows.” In: *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*. Vol. 22. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. 2007, p. 173.

[3] Russell Bent and Pascal Van Hentenryck. “Spatial, temporal, and hybrid decompositions for large-scale vehicle routing with time windows”. In: *Principles and Practice of Constraint Programming–CP 2010*. Springer, 2010, pp. 99–113.

[4] Wen-Chyuan Chiang and Robert A Russell. “Simulated annealing metaheuristics for the vehicle routing problem with time windows”. In: *Annals of Operations Research* 63.1 (1996), pp. 3–27.

[5] G Clarke and John W Wright. “Scheduling of vehicles from a central depot to a number of delivery points”. In: *Operations research* 12.4 (1964), pp. 568–581.

[6] GA Croes. “A method for solving traveling-salesman problems”. In: *Operations Research* 6.6 (1958), pp. 791–812.

[7] George B Dantzig and John H Ramser. “The truck dispatching problem”. In: *Management science* 6.1 (1959), pp. 80–91.

[8] Martin Desrochers and Gilbert Laporte. “Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints”. In: *Operations Research Letters* 10.1 (1991), pp. 27–36.

[9] Rodolfo Dondo and Jaime Cerdá. “A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows”. In: *European Journal of Operational Research* 176.3 (Feb. 2007), pp. 1478–1507.

[10] Marshall L Fisher and Ramchandran Jaikumar. “A generalized assignment heuristic for vehicle routing”. In: *Networks* 11.2 (1981), pp. 109–124.

[11] Bruno-Laurent Garcia, Jean-Yves Potvin, and Jean-Marc Rousseau. “A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints”. In: *Computers & Operations Research* 21.9 (1994), pp. 1025–1033.

[12] Hermann Gehring and Jörg Homberger. “A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows”. In: *Proceedings of EUROGEN99*. Vol. 2. 1999, pp. 57–64.

[13] Michel Gendreau and Christos D Tarantilis. *Solving large-scale vehicle routing problems with time windows: The state-of-the-art*. CIRRELT, 2010.

[14] Billy E Gillett and Leland R Miller. “A heuristic algorithm for the vehicle-dispatch problem”. In: *Operations research* 22.2 (1974), pp. 340–349. ISSN: 0030364X.

[15] Fred Glover. “Future paths for integer programming and links to artificial intelligence”. In: *Computers & Operations Research* 13.5 (1986), pp. 533–549.

[16] Stefan Irnich and Daniel Villeneuve. “The shortest-path problem with resource constraints and k-cycle elimination for $k > 3$”. In: *INFORMS Journal on Computing* 18.3 (2006), pp. 391–406.

[17] Philip Kilby and Andrew Verden. “Flexible routing combing constraint programming, large neighbourhood search, and feature-based insertion”. In: *Proceedings 2nd Workshop on Artificial Intelligence and Logistics*. 2011, pp. 43–49.

[18] Scott Kirkpatrick. “Optimization by simulated annealing: Quantitative studies”. In: *Journal of statistical physics* 34.5-6 (1984), pp. 975–986.

[19] Niklas Kohl and Oli BG Madsen. “An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation”. In: *Operations Research* 45.3 (1997), pp. 395–406.

[20] Niklas Kohl et al. “2-path cuts for the vehicle routing problem with time windows”. In: *Transportation Science* 33.1 (1999), pp. 101–116.

[21] David Mester, Olli Braysy, and Wout Dullaert. “A multi-parametric evolution strategies algorithm for vehicle routing problems”. In: *Expert Systems with Applications* 32.2 (Feb. 2007), pp. 508–517. ISSN: 09574174.

[22] Denis Naddef and Giovanni Rinaldi. “Branch-and-cut algorithms for the capacitated VRP”. In: *The vehicle routing problem* 9 (2002), pp. 53–81.

[23] Ilhan Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*. Xerox University Microfilms, 1976.

[24] Ibrahim H Osman and Gilbert Laporte. “Metaheuristics: A bibliography”. In: *Annals of Operations Research* 63.5 (1996), pp. 511–623.

[25] Mingyao Qi et al. "A spatiotemporal partitioning approach for large-scale vehicle routing problems with time windows". In: *Transportation Research Part E: Logistics and Transportation Review* 48.1 (2012), pp. 248–257.

[26] David M Ryan, Curt Hjorring, and Fred Glover. "Extensions of the Petal Method for Vehicle Routing". In: *Journal of the Operational Research Society* (1993), pp. 289–296.

[27] Martin WP Savelsbergh. "Local search in routing problems with time windows". In: *Annals of Operations Research* 4.1 (1985), pp. 285–305.

[28] Linus Schrage. "Formulation and structure of more complex/realistic routing and scheduling problems". In: *Networks* 11.2 (1981), pp. 229–232. ISSN: 1097-0037.

[29] Marius M Solomon. "Algorithms for the vehicle routing and scheduling problems with time window constraints". In: *Operations research* 35.2 (1987), pp. 254–265.

[30] Éric Taillard. "Parallel iterative search methods for vehicle routing problems". In: *Networks* 23.8 (1993), pp. 661–673.