Restricted Dynamic Programming: Evaluating an Advanced Construction Algorithm in Solving Vehicle Routing Problems

Bachelor Thesis Econometrie en Operationele Research
Erasmus University Rotterdam

Casper de Winter, 385910
Supervisor: R.B.O. Kerkkamp
Second reader: prof. dr. D. Huisman

4 July 2016

Abstract

The purpose of this thesis is to evaluate a new and advanced approach for solving Vehicle Routing Problems (VRPs). This approach is based on an exact solution method, dynamic programming, and by restricting its state space, it becomes a heuristic method which could be used in practice. We compared this restricted dynamic programming (RDP) algorithm on both classical and more realistic VRPs with a simple construction algorithm, the savings heuristic. We showed that, while the computation times of the RDP algorithm were exceptionally higher, it did not always gave better solutions than fast construction heuristics and that the claim that the RDP algorithm competes with state of the art local search solutions when more realistic constraints are considered should be rejected. By using two k-opt local search methods to improve the construction solutions, we also discovered that a better construction solution leads to better solutions after the improvement phase. However, because the RDP algorithm could not provide us with high quality construction solutions within reasonable computing time, we can conclude that this new and advanced solution method cannot compete with other known solution methods for the VRP.

Contents

1 Introduction

The Vehicle Routing Problem (VRP) is a well-known combinatorial optimization problem which has been introduced over 50 years ago by Dantzig & Ramser (1959). In the VRP, a set of routes has to be determined from several depots to a number of addresses while minimizing the total costs or travel distance. There is a wide variety of different VRPs and a broad literature on this class of problems. This is due to the fact that the VRP is widely used in practice and even the basic VRP is an NP-hard problem. This means that many practical problems cannot be solved within reasonable computing time using exact solution methods. As a consequence, many different heuristics and meta-heuristics are being used to reach high-quality solutions within limited computing times.

Most heuristic methods are divided in two parts: a route construction phase and a route improvement phase. The construction phase is often assumed to be simple and fast (Pisinger & Ropke, 2007; Tan et al., 2001) and its sole purpose is to construct a reasonably good feasible set of routes which will be used as initial solution for the improvement phase. During this improvement phase, routes can be modified or recreated to improve the initial solution.

In the standard vehicle routing problem, there is only one depot, one vehicle and no extra constraints. This problem is called the Travelling Salesman Problem (TSP). In practice, many different extensions of this TSP exist. For example, one can think of capacity or time window constraints for the vehicles and the orders, multiple depots, multiple routes, pickup and delivery within a route, different characteristics for the vehicles and the orders and of course combinations of these extensions. Considering this extensive set of possible VRPs and the fact that every heuristic can perform differently on different types of VRPs, the number of different solution methods is very large, as can be seen in the surveys by Parragh et al. (2007, 2008).

1.1 Restricted Dynamic Programming

The basis of this research is the paper by Gromicho et al. (2012) in which they introduce and improve a restricted dynamic programming (RDP) algorithm for solving realistic VRPs. We will fully explain this algorithm in Section 3.1 by using the PHD Thesis of Kok (2010) as a guideline. Of all known exact algorithms to solve the TSP, dynamic programming has the best complexity (Woeginger, 2003). For larger, more realistic problems, the running time of all exact algorithms are still not fast enough. By restricting the number of expansions in every dynamic programming stage, Gromicho et al. (2012) have found a reasonably good performing framework which can be applied to all kinds of VRPs.

This RDP algorithm can be seen as an advanced construction method which allows for a trade-off between computation time and quality of the solution. Of course, the less strict the restrictions in RDP, the longer it takes to find a solution, but the better the solution will be. The given results in the paper for three different types of VRPs show us that this framework performs quite well and is faster when the routing problem becomes more extensive and thus more realistic. In comparison with results of some well-known construction heuristics on the same instances, we can already see that the RDP construction algorithm performs slightly better on average (Solomon, 1987). However, these simpler heuristics are exceptionally faster. Furthermore, the RDP solutions are, based on the total distance, on average still 10% larger than the best found solutions. So there is still some room for improvement.

1.2 Research Objective

Gromicho et al. (2012) already suggested that the solutions found may certainly be improved by local search or other improvement algorithms. In practice, some fast steps of local search heuristics will always be performed while finding a solution. We want to check whether it is worth to wait significantly longer for the construction phase to finish before starting to improve the initial solution. So, we want to check if the RDP construction method is useful to implement and outperforms other construction methods. In some specific cases it is mentioned that an improved initial solution does have a positive effect for the final solution (van Breedam, 1996), but this has never been thoroughly investigated. It is remarkable that most research of the last years on VRPs is about developing improvement heuristics and there is almost no research on improving the construction heuristics. We suspect that the latest heuristics are very robust and have such a good performance that it does not matter how good the initial solution is.

If the RDP method does prove to be useful for solving realistic VRPs, as Gromicho et al. (2012) suggest, it will be relevant to implement this or similar improved construction methods in new solution methods and to do more research on this advanced construction method. On the other hand, when it does not matter how good the initial solution is, simple construction heuristics will be preferred over this extensive framework.

In order to evaluate the RDP framework, we first compare its solution with solutions of a simple construction algorithm. For this purpose, we make use of some well-known benchmark instances for the VRP with Time Windows, the Solomon instances. Its best-known solutions will be used for comparison of both initial construction solutions. We also compare the solutions after the improvement phase for both initial solutions. Finally, we compare the solutions of the RDP method and the simple construction algorithm on some more realistic instances, the Goel instances.

The remainder of this thesis is structured as follows. In Section 2, the VRP, the VRPTW and the Solomon and the Goel instances are formally introduced. Afterwards, the different solution methods that we use are introduced in Section 3. In Section 4, the computational results to these methods are presented. Lastly, we present our discussion and the final conclusions of our results in Section 5 and 6.

2 Vehicle Routing Problem

In this section, we formally introduce the VRPTW and its notation which will be used throughout this thesis. Besides that, the VRPTW Solomon benchmark instances and the VRPTW-EC Goel benchmark instances are also defined.

The Vehicle Routing Problem with Time Windows can be defined as follows. We are given a set of vehicles $M = \{1, 2, \ldots, m\}$ and n addresses or nodes in the set $V = \{0, 1, \ldots, n-1\}$, in which 0 is the depot address. Any address $i > 0$ represents a customer with its own demand d_i, service time q_i and time window $[e_i, l_i]$. e_i represents the earliest time on which the demand can be delivered and e_i for the latest possible time. When a vehicle gets to an address before the earliest time e_i, the vehicle has to wait until it is e_i. The depot address also has its own time window $[e_0, l_0]$ for when it is opened. The travel distance between each pair of nodes $i, j \in V$ is defined by c_{ij}, $c_{ij} \geq 0$. The travel time between two nodes is t_{ij}, $i, j \in V$, $t_{ij} \geq 0$. The goal of the problem is to find a set of routes for the vehicles against minimal number of vehicles and minimum distance, each vehicle starting and ending at the depot, such that the total demand in each route does not exceed the capacity K of a vehicle and that each service at a customer starts in the given time window.

2.1 Solomon Instances

The Solomon instances (Solomon, 1987) are the standard reference in the VRP literature. This has the consequence that the best known results for these instances are very valuable in comparing with our test results. These benchmark instances consist of six problem sets (c1, c2, r1, r2, rc1 and rc2) and in total 56 instances. In the sets c1 and c2 the customers are clustered, in r1 and r2 these are randomly scattered and in the rc1 and rc2 instances they are semi-clustered. The 2-instances have relatively larger vehicle capacities and time horizon which allows vehicles to visit more customers in one route.

All instances contain 100 addresses and 1 depot which are all located in a 100 by 100 grid. The location of address i is given by its x-coordinate x_i and its y-coordinate y_i. The travel distance between two addresses is defined by the Euclidean distance. $c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. In the Solomon instances, the travel time t_{ij} is equal to the travel distance. The primary objective for all instances is to minimize the number of vehicles and the secondary objective is to minimize the total travel distance and the total waiting time.

2.2 Goel Instances

Since the Solomon instances only contain a few realistic constraints and Gromicho et al. claim that the RDP method is a promising approach for more practical applications, we also test our solution framework with the Goel instances. These instances are benchmark instances for the VRPTW with the European Community social legislation on driving and working hours (VRPTW-EC) which were proposed by Goel (2009). The EC social legislation poses restrictions on the amount of driving and working times before a mandatory break or rest is taken. For example, after each accumulated driving period of 4.5 hours, the driver needs to have a break of at least 45 minutes. Also, the maximum daily and weekly driving and working time is taken into account. For a full description of the legislation rules, we refer to Kok et al. (2010). We also implement the basic break scheduling method which Kok et al. proposed to schedule breaks and rests in our construction methods.

The Goel instances are modifications of the Solomon instances. The depot opening time is considered as a period of 144 hours, which corresponds to a weekly working period with Sunday as a rest day. The time windows of the other addresses are scaled accordingly. Besides this, all service times are set to 1 hour and the driving speed is 5 distance units per hour. However, due to these adjustments, it may be impossible to reach a certain customer i before its latest possible time l_i or to return in time to the depot from e_i. These time windows need to be adapted such that every address can be reached from and to the depot.

3 Solution Methods

First, the Restricted Dynamic Programming (RDP) algorithm will be explained in Section 3.1 based on the PHD Thesis of Kok (2010). After that a simple and fast construction algorithm, the saving algorithm, is defined in Section 3.2 and finally two special k-opt improvement algorithms will be explained in Section 3.3.

3.1 Restricted Dynamic Programming

3.1.1 Dynamic programming for the TSP

The RDP algorithm is based on the exact dynamic programming algorithm for the TSP of Bellman (1962) and Held & Karp (1962). A state $(S, j), j \in S, S \subseteq V$ in dynamic programming represents a path starting at address 0, visiting all addresses in S once and ending in city j. $C(S, j)$ is the cost of the smallest of all possible paths (S, j). The idea behind dynamic programming is that the final problem of all V addresses is broken down into smaller and simpler subproblems. In order to end with an optimal solution that includes all n addresses, ending in address j, which is state (V, j), one should minimize the following: $\min\{C(V \setminus \{j\}, k) + c_{kj} : k \in V, k \neq j\}$. We are not able to solve this yet, because we first need to find all optimal $C(V \setminus \{j\}, k)$. This continues until $C(\{h\}, h)$ is reached, which is the initialization of the DP algorithm and can easily be worked out: $C(\{h\}, h) = c_{0h}$.

So, in the first stage, the costs of the states $C(\{h\}, h) = c_{0h}$ are set. Next, in each stage the costs of the states are calculated with the recurrence relation $C(S, j) = \min_{i \in S \setminus \{j\}}\{C(S \setminus \{j\}, i) + c_{ij}\} \; \forall S \subseteq V \setminus 0, j \in S$. Finally, when all addresses are visited, we should return to the depot address 0. The length of the optimal TSP route is given by $\min_{i \in V \setminus \{0\}}\{C(V \setminus \{0\}, i) + c_{i0}\}$.

3.1.2 Giant-tour representation

The dynamic programming algorithm only constructs one full route, so this method for the TSP cannot immediately be applied to the VRP. To make sure that this algorithm can also be used for the VRP and thus for multiple routes, the giant-tour representation (GTR) of vehicle routing solution introduced by Funke et al. (2005) is used. In the GTR, we connect all distinct routes into one cycle where every node is visited only once. For each route, we need to add one origin node and one destination node to the set of addresses, which could represent the same location. If the number of routes is equal to m, $2m$ addresses need to be added to the set V. In the GTR of a VRP, each destination node d^v is connected to the next origin node o^{v+1} and the cycle is closed by connecting d^m with o^1. The costs of connecting an origin and a destination node is always zero.

To use this representation in dynamic programming, the number of total addresses and thus the available vehicles must be known beforehand. If this is not given, we can use an upper bound on the required number of vehicles. When a vehicle is not used in the final solution, its origin node and destination node are directly connected. In Figure 1, an example of the GTR of a VRP solution is presented.

3.1.3 Dynamic progrmming for the VRP

The GTR can now be used to transform the VRP in a sequencing problem where we need to find one fully connected route instead of multiple separate routes. We are now able to use the DP formulation of Section 3.1.1 to solve the VRP. In order to ensure that the final DP solution is a feasible VRP solution, we check the feasibility of a partial solution while expanding a state. For example when there is a state (S, d^v), its only feasible expansion is o^{v+1}.

We also need to perform several other feasibility checks to ensure that the time window and capacity constraints are not violated. For the time and capacity constraints, extra state dimensions are added to the algorithm. While expanding a state, we check if this expansion is feasible according to the capacity of the

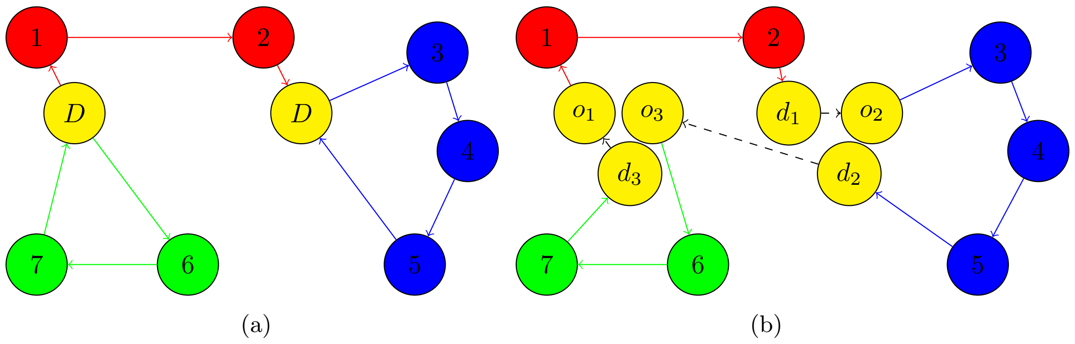

Figure 1: An example of a VRP solution (a) and its giant-tour representation (b).

vehicle and the time windows of the customer and the vehicle. To ensure that we do not lose the optimality guarantee of the unrestricted DP algorithm, it is possible to make multiple copies of the state (S, j). For example, consider two states $(S, j)_1$ and $(S, j)_2$. $(S, j)_1$ can be better than $(S, j)_2$ based on the total distance, whereas $(S, j)_2$ has more slack in time. Both states can possibly result in the optimal feasible solution and we cannot claim that one state dominates the other.

3.1.4 Restricting the state space

As mentioned earlier, DP is not fast enough to solve VRPs in general. Therefore, Gromicho et al. (2012) defined two parameters, H and E, on which the DP algorithm can be bounded. They proposed a restriction on the number of states in each stage, based on Malandraki & Dial (1996). Due to this restrictions, we lose the claim of finding the optimal solution with this algorithm. In each stage, only the best H states will be expanded in the next stage. States with a lower distance are more likely to end in a better final solution than states with a high distance. By increasing the value of H, the solutions will usually improve, but it will also result in higher computation times. To determine the most promising H states, the following hierarchical criteria are used: the minimum number of vehicles used, the minimum total distance traveled, the minimum current time of the current vehicle and the maximal remaining capacity of the current vehicle.

The state space is even further restricted by expanding each state only to the nearest unvisited and feasible nodes, until a maximum number of expansions E is reached. For the final solution, it is usually better when an edge is between two nodes who are very close to each other. By only expanding to the nearest E feasible nodes, we believe to exclude many states which will never result in a good solution. Setting H or E equal to 1 results in the nearest neighbor heuristic and setting both H and E to infinity results in the original unrestricted DP algorithm.

Besides H and E, we also bound the states in our RDP algorithm on the following. A vehicle is not allowed to return empty if there are customers left and returning low filled vehicles is discouraged. A vehicle can only return to a depot if the vehicle is not empty when there is some demand left and if the percentage of demand delivered so far by all vehicles is higher than the percentages of vehicles used so far (Kok, 2010). In contradiction to the restrictions H and E, this does not exclude the optimal solution, it only reduces the number of states. This is because we can always arrange the order of the vehicles such that the empty vehicles are at the end of the giant tour and the fullest vehicles are in front.

3.2 Saving Algorithm

The saving algorithm is one of the most well-known and easiest construction heuristics for the VRP. It was originally developed for the classical VRP by Clarke & Wright (1964) and implemented for the VRPTW by Solomon (1987). It begins with a solution of n distinct routes in which every route supplies one customer and returns to the depot. From this point, two routes are combined in every iteration according to the cost saving of combining those routes. Consider two customers i and j which are both in distinct routes and at the start or at the end in their current route. When these routes are connected, the arcs (i,0) and (0,j) are replaced by arc (i,j). This results in a cost saving S_{ij}, which is defined as follows: $S_{ij} = c_{i0} + c_{j0} - c_{ij}$. Connecting the two routes is illustrated in Figure 2.

In more detail, the saving algorithm works as follows. After the initialization which constructed n distinct routes, a list of cost savings is constructed. The savings list contains every pair of customers i and j, $i \neq j$ and

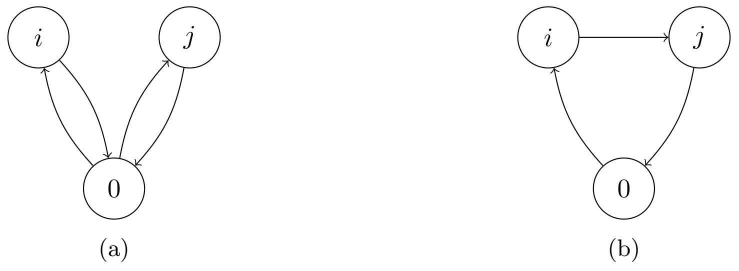

Figure 2: The saving algorithm. First, the customers i and j are in separate routes (a). These routes are connected by adding edge (i, j) (b).

its corresponding cost savings S_{ij} and the list is sorted such that the best saving is on top. In every iteration, the next best candidate S_{ij} is evaluated by doing some feasibility checks. First, capacity constraints could be violated by joining the two routes. The same applies to time window constraints. In order to connect the two routes with edge (i, j), it is possible that one or both of the routes should be reversed which could make the newly formed route infeasible. Because of these time window constraints, it is also possible that customer j can never be placed after customer i in a route which also makes the new route infeasible. For the Goel instances, it is also checked if the new route is feasible with respect to the rest and break periods which should be added. When it is feasible for the two routes to be connected, the partial routes can now be combined into one route.

The main disadvantage of the saving algorithm when working with time windows is that it easily connects two nodes which are very close to each other while the time windows are not adjacent. This results in a vehicle waiting a long time between serving both nodes and that the algorithm is not able to join many more customers into this route. This makes sure that the final outcome will probably not be optimal, because a larger number of vehicles is being used.

3.3 Improvement Algorithms

To improve VRP solutions, the k-opt heuristic which is a simple local search algorithm can be used. In this algorithm, k edges are deleted from the route and subsequently the full tour is reconnected in all possible ways to find a better route. In practice, only $k = 2$ and $k = 3$ are used due to the enormous run times of $k > 3$. We only look at restricted versions of 2-opt and 3-opt specialized for multiple routes and time windows. Here, not every case of 2 or 3 edges is being considered. These special forms are called 2-opt* and Or-opt.

3.3.1 2-opt* Heuristic

The 2-opt* heuristic was introduced by Potvin & Rousseau (1995). They adapt the classical 2-opt method, because this method is not well suited for problems with multiple routes and time windows. The 2-opt method does normally reverse the direction of a part of the routes. For example, in Figure 3, when the edges $(i, i + 1)$ en $(j, j + 1)$ are deleted and replaced by edges (i, j) en $(i + 1, j + 1)$, a part of the newly formed route is reversed. When the time windows are quite strict, it is very unlikely for the new route to be feasible due to the reversion. Besides that, when i and j are in different routes, it is even more unlikely that the new route is feasible, because two partial routes need to be reversed.

Especially for multiple routes, we can look at another way of reconnecting the tour which preserves the orientation of the route. When the edges $(i, i + 1)$ en $(j, j + 1)$ are in different routes, one can make sure that the time window constraints are probably not conflicted by replacing those edges with edges $(i, j + 1)$ en $(j, i + 1)$ and by keeping all other routes the same. Basically, the first customers of the first route are linked to the last customers of the second route and vice versa. This is a 2-opt* iteration. An example is shown in Figure 4.

The 2-opt* heuristic combines the 2-opt* method and the 2-opt method. Starting with the current solution, the heuristic evaluates all pairs of links. When two links are in the same route, the classical 2-opt method is performed. If not, the 2-opt* method is used. After forming a new solution or set of routes, it is checked whether the new set of routes is feasible according to the capacity and time window constraints. If this is true, the new solution is compared to the current one according to the number of vehicles and total

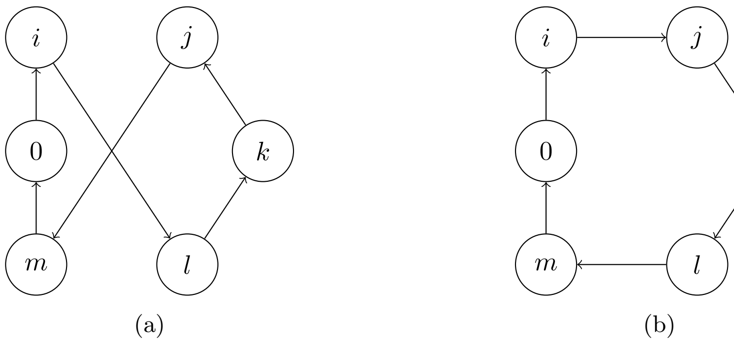

Figure 3: A classical 2-opt iteration within a route by replacing links (i, j) and (l, m) in (a) by links (i, l) and (j, m) in (b). As can be seen, a part of the route is reversed.

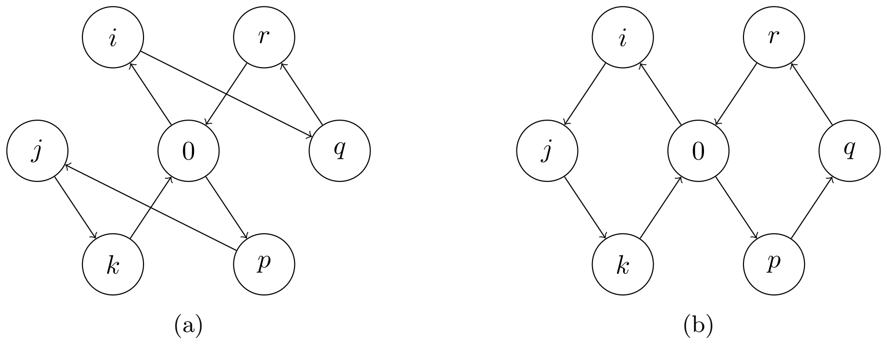

Figure 4: A 2-opt* iteration by replacing links (i, q) and (p, j) in (a) by links (i, j) and (p, q) in (b). As can be seen, the orientation of the route stays the same.

distance of the current solution. When the newly composed solution is better than the current one, this new solution is set as the current solution and the whole algorithm starts again. This procedure is called First Improvement (FI). After the first found improvement on the current solution, the new route is chosen and the heuristic starts again.

Another procedure is the Best Improvement (BI) procedure. All possible pairs of links are first evaluated in BI and the route with the best improvement of all these pairs is picked and set as the current route. In both procedures, when the 2-opt* heuristic is not able to find an improvement after evaluating all pairs of links, the current route is returned. The current route is now 2-opt* optimal. In our research we implement both the BI and the FI procedure and simply pick the best of the resulting two routes as our solution. FI can give a better solution, because BI might get trapped in a local optima sooner.

3.3.2 Or-opt Heuristic

As explained before, 3-opt generally deletes three links and reconnect the routes in every possible way. Or-opt, introduced by Or (1976), reduces the number of sets of links to inspect drastically. Again, this reduced subset of changes always makes sure that the orientation of the routes is preserved.

The main idea behind Or-opt is to move a small sequence of adjacent customers to another place. The number of moved customers in this small sequence is equal to m, with $m \in \{1, 2, 3\}$. The link before and after the small sequence and the link on the place of insertion are deleted and replaced by three new links. This is illustrated in Figure 5 with an example for $m = 2$. In contradiction to 2-opt*, Or-opt can be used for both inter-route and intra-route improvements. The sequence of m customer could also be placed somewhere else in the same route.

In the Or-opt algorithm, $m = 3$ is first evaluated, thereafter $m = 2$ and finally $m = 1$, which only moves one customer. Again, every move needs to be checked according to the capacity and time window constraints. We can also apply FI and BI to Or-opt to find a better solution, which is now Or-opt optimal. It is also possible to use both the 2-opt* heuristic and the Or-opt heuristic to improve a solution. A 2-opt* optimal tour could still be improved by the Or-opt heuristic and vice versa. Both heuristics are executed one after

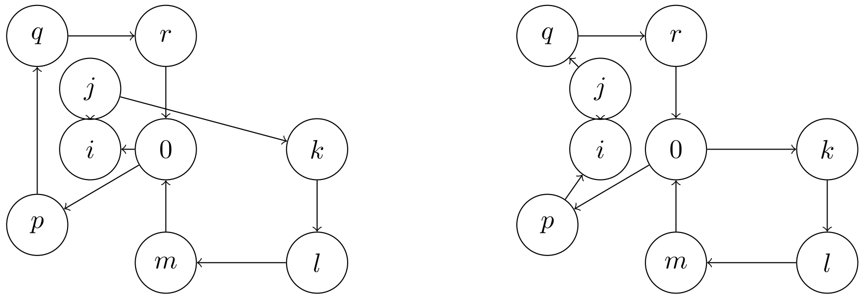

Figure 5: An Or-opt iteration. Here, $m = 2$ and customers i and j are placed into another route.

another until the tour is both 2-opt* optimal and Or-opt optimal. Since the order of the heuristics could be important, we execute the heuristic twice, once starting with the 2-opt* heuristic and once starting with the Or-opt heuristic. Again, the best final solution will be picked. This last Or-opt/2-opt* heuristic will be used to improve our initial solutions.

4 Computational Results

4.1 VRPTW

4.1.1 Construction Heuristic Results

In this section, we compare the solutions obtained via the saving heuristic and the RDP algorithm. For the RDP algorithm, we set $H = 10.000$ and $E = 20$. In Table 1, the results for both construction heuristics are presented. The results are averaged for all instances in each set of instances. We implemented all the algorithms in Java 8 and ran our experiments on a PC with a Core i7-3612QM, 2.10 GHz CPU and 8 GB of RAM.

Table 1: Construction heuristic solutions and its computation time for the RDP and the saving algorithm.

Set	RDP			Saving			Best known	Best known
	Distance	# Veh.	cpu (s)	Distance	# Veh.	cpu (s)	distance	# veh.
c1	881.31	10.33	296.00	911.77	11.44	0.0482	828.38	10.00
c2	656.94	3.13	369.72	704.39	4.88	0.0320	589.86	3.00
r1	1396.30	15.58	260.63	1371.91	18.00	0.0685	1210.34	11.92
r2	1168.83	5.45	320.02	1055.94	11.09	0.0474	951.03	2.73
rc1	1620.54	15.38	252.87	1583.62	17.38	0.0721	1384.16	11.50
rc2	1380.14	6.13	423.15	1279.23	11.88	0.0658	1119.24	3.25

As can be seen, the RDP algorithm with these parameters obtains better solutions based on the number of vehicles and the distance than the saving method for the c1 and c2 instances. For the other four sets, the distance obtained with the saving algorithm is better than the RDP algorithm, but the number of vehicles used is larger. The large number of used vehicles for the saving algorithm is mainly due to its disadvantage which we explained in Section 3.2: the algorithm can easily connect two addresses very close in distance but far apart in time.

It is still quite remarkable that, based on distance, only 21 out of the 56 problem instances got a better solution for the RDP algorithm than the saving algorithm. This could be due to the low values of H and E. However, Gromicho et al. (2012) also did not obtain better solutions than our distance found with the saving solutions for the r2 and rc2 sets with H set to 100.000 and E set to n. The computation times for $H = 10.000$ and $E = 20$ in our implementation were approximately 6000 times as large than the computation time of the saving algorithm. So, similar solutions with slightly more vehicles can be constructed within 0.1 seconds with the most simple construction algorithm for the VRPTW. And as Solomon (1987) already showed, the saving algorithm is not even the best performing construction algorithm. They showed that other fast construction algorithms could obtain solutions on which both the distance and especially the number of vehicles was better than the saving algorithm. For example, the Push Forward Insertion Heuristic (PFIH or I1) obtained better solutions based on the number of vehicles than the RDP method for all instance sets in only 10 times the computation time of the saving algorithm.

Gromicho et al. stated that their RDP algorithm could not compete with state of the art methods for the VRPTW. We can even conclude on top of this that this RDP algorithm can also not compete with simple and fast construction methods for the VRP, especially given the enormous computation times.

4.1.2 Improvement Heuristic Results

We use the solutions obtained in the previous section, to improve them in this section with the improvement heuristics as explained in Section 3.3. These initial solutions will be improved with the 2-opt* algorithm and the Or-opt/2-opt* algorithm. The solutions averaged per set of instances can be seen in Table 2 for the 2-opt* algorithm and in Table 3 for the Or-opt/2-opt* algorithm.

Table 2: Construction heuristic solutions and the 2-opt* improvement for the RDP and the saving algorithm.

Set	RDP				Saving				Best known solutions	
	Constr.	#Veh.	2-opt*	#Veh.	Constr.	#Veh.	2-opt*	#Veh.	Distance	#Veh.
c1	881.31	10.33	835.64	10.00	911.77	11.44	905.43	11.44	828.38	10.00
c2	656.94	3.13	616.35	3.13	704.39	4.88	671.91	4.50	589.86	3.00
r1	1396.30	15.58	1299.20	15.25	1371.91	18.00	1352.84	17.83	1210.34	11.92
r2	1168.83	5.45	1054.36	4.91	1055.94	11.09	1033.48	10.73	951.03	2.73
rc1	1620.54	15.38	1497.39	14.75	1583.62	17.38	1572.26	17.25	1384.16	11.50
rc2	1380.14	6.13	1202.11	5.50	1279.23	11.88	1267.94	11.75	1119.24	3.25

Table 3: Construction heuristic solutions and the Or-opt/2-opt* improvement for the RDP and the saving algorithm.

Set	RDP				Saving				Best known solutions	
	Constr.	#Veh.	Or/2*	#Veh.	Constr.	#Veh.	Or/2*	#Veh.	Distance	#Veh.
c1	881.31	10.33	829.68	10.00	911.77	11.44	856.45	10.67	828.38	10.00
c2	656.94	3.13	596.72	3.00	704.39	4.88	603.36	3.25	589.86	3.00
r1	1396.30	15.58	1248.65	14.50	1371.91	18.00	1275.24	15.58	1210.34	11.92
r2	1168.83	5.45	956.87	4.73	1055.94	11.09	968.74	8.27	951.03	2.73
rc1	1620.54	15.38	1446.58	14.00	1583.62	17.38	1459.26	15.13	1384.16	11.50
rc2	1380.14	6.13	1096.94	5.38	1279.23	11.88	1130.79	9.13	1119.24	3.25

After executing both improvement algorithms, the improved solutions are in almost each set on average better for the RDP construction solution than the saving construction solution. On 35 of the 56 instances with 2-opt* and on 40 instances with Or-opt/2-opt*, the final solution obtained with the RDP construction solution outperforms the solution with the saving construction solution on both the number of vehicles and the total distance. We can conclude that these local search k-opt heuristics perform on average better with an initial solution which has a fewer number of vehicles. So, a construction heuristic which can obtain a solution with a low number of vehicles, such as the fast PFIH algorithm, will be preferred for the VRPTW.

4.2 VRPTW-EC

As expected, the RDP framework is not preferred as a construction heuristic for the common VRPTW. Many good performing heuristics are especially created for the VRPTW and the Solomon instances. However, due to the flexibility of the RDP method, the RDP method is very easy to use when extra constraints such as the EC social legislations comes in. The same applies for the saving algorithm. The solutions obtained for the VRPTW-EC with both construction heuristics, can be seen in Table 4. We compare our solutions with the solutions obtained by Prescott-Gagnon et al. (2010) in which they use a state of the art local search method and an average computation time of approximately 90 minutes. Again, we set $H = 10.000$ and $E = 20$ for the RDP method.

Table 4: Construction heuristic solutions for the VRPTW-EC.

Set	RDP			Saving			Prescott-Gagnon solutions	
	Distance	#Veh.	cpu (s)	Distance	#Veh.	cpu (s)	Distance	#Veh.
c1	950.07	10.67	161.92	934.45	11.44	0.06	847.61	10.00
c2	893.88	7.00	270.39	912.11	7.75	0.05	692.47	4.63
r1	1214.38	11.08	307.11	1178.82	13.58	0.09	948.61	8.08
r2	1145.10	8.82	278.46	1103.20	12.36	0.07	933.93	5.45
rc1	1384.23	11.38	189.89	1296.92	13.00	0.05	1107.33	9.00
rc2	1322.12	9.38	178.30	1393.88	13.88	0.05	1090.80	6.13

Just as for the VRPTW, the RDP algorithm outperforms the saving algorithm based on the number of vehicles and, most of the times, not on the total distance. When we consider the Prescott-Gagnon solutions to be the best known solutions, we can compare these results to the construction heuristic results for the VRPTW. The solutions obtained with the RDP algorithm for the VRPTW-EC are not closer to the best known results than they were for the VRPTW. For the Goel instances, the RDP algorithm with our parameter settings used 35% more vehicles and had 23% more distance whereas for the Solomon instances, these values were respectively 33% and 17%. The claim by Gromicho et al. (2012) that the framework competes with state of the art local search solutions when more realistic constraints are considered can therefore be rejected.

5 Discussion

This section compares our results of the RDP algorithm with the results of Gromicho et al. (2012). Next, some possible suggestions for further research are given.

Although we use the same RDP algorithm as Gromicho et al. (2012), the results differ a little. The main difference can be found in the computation times. Their implementation of the RDP algorithm was much faster in comparison with ours. This could be due to the way states were saved and compared in our implementation. However, as we could have seen, their implementation was still not fast enough to get quite good results within reasonable computing time. Also, in an attempt to reduce the number of states, we discouraged the return of low-filled trucks and . In further research, other methods to reduce the number of states without worsening the results can be examined.

We only considered the simple saving algorithm which could not outperform the RDP method based on its solutions. However, we showed that the PFIH construction method could potentially outperform the RDP algorithm. This could be formally investigated in further research. Next to this, we only implemented two simple local search methods which quickly get trapped in local optima. We found out a better initial solution is important for these simple local search methods. RDP could in the long run produce better initial solutions, but you should wonder if it is worth it. For other improvement algorithms, for example more advanced metaheuristics like tabu search, the initial construction solution would probably matter less. This could also be examined in further research.

Finally, it is clear that restricted dynamic programming is not the best solving method for VRPs. Restricted Dynamic Programming can however be useful for other applications as it can be applied to most problems in operations research.

6 Conclusions

In this thesis, we evaluated the restricted dynamic programming framework for VRPs, which was described by Gromicho et al. (2012). We determined that the RDP algorithm, despite its exceptionally longer computation times, does not always outperform simple construction algorithms as the saving algorithm or the PFIH. Next to this, the solutions obtained with the RDP algorithm for the more realistic VRPTW-EC are not closer to the best known results than they were for the VRPTW. The claim by Gromicho et al. that the framework competes with state of the art local search solutions when more realistic constraints are considered can therefore be rejected.

By using two k-opt local search methods to improve the construction solutions, we also discovered that a better construction solution leads to better solutions after the improvement phase. For more advanced metaheuristics, this effect will probably be much smaller. However, the RDP algorithm could not provide us with high quality construction solutions within reasonable computing time. Therefore we can conclude that this new and advanced solution method cannot compete with other known solution methods for the VRP.

References

Bellman, R. (1962), ‘Dynamic programming treatment of the travelling salesman problem’, *Journal of the ACM (JACM)* **9**(1), 61–63.

Clarke, G. & Wright, J. W. (1964), ‘Scheduling of vehicles from a central depot to a number of delivery points’, *Operations research* **12**(4), 568–581.

Dantzig, G. B. & Ramser, J. H. (1959), ‘The truck dispatching problem’, *Management science* **6**(1), 80–91.

Funke, B., Grünert, T. & Irnich, S. (2005), ‘Local search for vehicle routing and scheduling problems: Review and conceptual integration’, *Journal of heuristics* **11**(4), 267–306.

Goel, A. (2009), ‘Vehicle scheduling and routing with drivers’ working hours’, *Transportation Science* **43**(1), 17–26.

Gromicho, J., van Hoorn, J. J., Kok, A. L. & Schutten, J. (2012), ‘Restricted dynamic programming: a flexible framework for solving realistic VRPs’, *Computers & operations research* **39**(5), 902–909.

Held, M. & Karp, R. M. (1962), ‘A dynamic programming approach to sequencing problems’, *Journal of the Society for Industrial and Applied Mathematics* **10**(1), 196–210.

Kok, A. L. (2010), Congestion avoidance and break scheduling within vehicle routing, PhD thesis, University of Twente.

Kok, A. L., Meyer, C. M., Kopfer, H. & Schutten, J. M. J. (2010), ‘A dynamic programming heuristic for the vehicle routing problem with time windows and european community social legislation’, *Transportation Science* **44**(4), 442–454.

Malandraki, C. & Dial, R. B. (1996), ‘A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem’, *European Journal of Operational Research* **90**(1), 45–55.

Or, I. (1976), Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking, PhD thesis, Xerox University Microfilms.

Parragh, S. N., Doerner, K. F. & Hartl, R. F. (2007), ‘A survey on pickup and delivery problems part I’.

Parragh, S. N., Doerner, K. F. & Hartl, R. F. (2008), ‘A survey on pickup and delivery problems part II’.

Pisinger, D. & Ropke, S. (2007), ‘A general heuristic for vehicle routing problems’, *Computers & operations research* **34**(8), 2403–2435.

Potvin, J.-Y. & Rousseau, J.-M. (1995), ‘An exchange heuristic for routeing problems with time windows’, *Journal of the Operational Research Society* **46**(12), 1433–1446.

Prescott-Gagnon, E., Desaulniers, G., Drexl, M. & Rousseau, L.-M. (2010), ‘European driver rules in vehicle routing with time windows’, *Transportation Science* **44**(4), 455–473.

Solomon, M. M. (1987), ‘Algorithms for the vehicle routing and scheduling problems with time window constraints’, *Operations research* **35**(2), 254–265.

Tan, K. C., Lee, L. H., Zhu, Q. & Ou, K. (2001), ‘Heuristic methods for vehicle routing problem with time windows’, *Artificial intelligence in Engineering* **15**(3), 281–295.

van Breedam, A. (1996), An analysis of the effect of local improvement operators in genetic algorithms and simulated annealing for the vehicle routing problem, PhD thesis, RUCA.

Woeginger, G. J. (2003), Exact algorithms for np-hard problems: A survey, *in* ‘Combinatorial Optimization—Eureka, You Shrink!’, Springer, pp. 185–207.

Appendix

Table 5: All solutions for the Solomon instances with the saving method for the construction phase and after the 2-opt* and Or-opt/2-opt* methods.

Instance	Constr.	#Veh.	2-Opt*	#Veh.	Or/2*	#Veh.	Best known solutions	
c101	860.58	11	828.94	10	828.94	10	828.94	10
c102	861.78	10	835.60	10	828.94	10	828.94	10
c103	1004.83	10	866.16	10	828.06	10	828.06	10
c104	1028.30	11	845.38	10	836.46	10	824.78	10
c105	860.58	11	828.94	10	828.94	10	828.94	10
c106	828.94	10	828.94	10	828.94	10	828.94	10
c107	828.94	10	828.94	10	828.94	10	828.94	10
c108	828.94	10	828.94	10	828.94	10	828.94	10
c109	828.94	10	828.94	10	828.94	10	828.94	10
c201	591.56	3	591.56	3	591.56	3	591.56	3
c202	629.26	3	624.41	3	591.56	3	591.56	3
c203	629.42	3	629.42	3	622.27	3	591.17	3
c204	862.72	4	731.26	4	614.38	3	590.60	3
c205	627.21	3	588.88	3	588.88	3	588.88	3
c206	637.19	3	588.49	3	588.49	3	588.49	3
c207	637.39	3	588.29	3	588.29	3	588.29	3
c208	640.80	3	588.49	3	588.32	3	588.32	3
r101	1742.55	21	1680.89	21	1669.36	21	1650.80	19
r102	1666.88	19	1547.08	18	1479.02	18	1486.12	17
r103	1521.39	19	1359.65	17	1292.81	16	1292.68	13
r104	1225.15	14	1127.62	14	1088.46	13	1007.31	9
r105	1539.02	17	1444.42	17	1443.37	17	1377.11	14
r106	1510.31	17	1415.55	17	1312.16	14	1252.03	12
r107	1361.29	14	1254.40	13	1159.07	12	1104.66	10
r108	1189.41	12	1076.65	12	1001.62	11	960.88	9
r109	1385.45	16	1295.81	16	1237.45	15	1194.73	11
r110	1256.78	13	1176.34	13	1163.80	13	1118.84	10
r111	1262.67	14	1178.54	14	1118.22	13	1096.72	10
r112	1094.70	11	1033.41	11	1018.43	11	982.14	9
r201	1519.82	7	1249.99	6	1216.44	6	1252.37	4
r202	1276.30	7	1162.37	7	1088.38	7	1191.70	3
r203	1208.14	6	1094.60	6	930.68	5	939.50	3
r204	985.49	4	939.58	4	813.50	4	825.52	2
r205	1253.25	6	1173.77	5	1084.50	5	994.42	3
r206	1189.18	6	1077.72	5	989.98	5	906.14	3
r207	1057.13	5	983.92	4	848.98	4	890.61	2
r208	894.77	3	796.63	3	746.18	3	726.82	2
r209	1194.43	5	1056.89	5	967.70	4	909.16	3
r210	1243.22	6	1122.51	5	969.49	5	939.37	3
r211	1035.40	5	939.98	4	869.75	4	885.71	2
rc101	1902.29	19	1739.52	18	1688.16	17	1696.94	14
rc102	1754.53	17	1597.06	16	1549.85	16	1554.75	12
rc103	1621.96	15	1479.76	15	1450.88	14	1261.67	11
rc104	1351.30	12	1296.15	12	1225.50	11	1135.48	10
rc105	1889.98	18	1699.92	18	1589.88	16	1629.44	13
rc106	1601.30	15	1471.69	14	1470.43	14	1424.73	11
rc107	1466.96	14	1372.70	13	1362.54	13	1230.48	11
rc108	1375.97	13	1322.30	12	1235.41	11	1139.82	10
rc201	1669.82	8	1450.15	8	1418.23	7	1406.94	4
rc202	1547.11	6	1346.88	6	1140.55	6	1365.65	3
rc203	1398.89	7	1158.79	5	1040.80	5	1049.62	3
rc204	1039.61	5	959.42	4	806.30	4	798.46	3
rc205	1556.85	8	1389.71	7	1283.30	7	1297.65	4
rc206	1390.42	5	1261.67	5	1191.45	5	1146.32	3
rc207	1308.38	6	1133.96	5	1068.83	5	1061.14	3
rc208	1130.04	4	916.30	4	826.09	4	828.14	3

Table 6: All solutions for the Solomon instances with the RDP method for the construction phase and after the 2-opt* and Or-opt/2-opt* methods.

Instance	Constr.	#Veh.	2-Opt*	#Veh.	Or/2*	#Veh.	Best known solutions	
c101	930.12	12	930.12	12	866.00	11	828.94	10
c102	931.05	12	931.05	12	866.00	11	828.94	10
c103	903.35	11	863.29	11	828.07	10	828.06	10
c104	872.88	10	867.97	10	839.48	10	824.78	10
c105	930.12	12	930.12	12	866.00	11	828.94	10
c106	930.12	12	930.12	12	866.00	11	828.94	10
c107	930.12	12	930.12	12	866.00	11	828.94	10
c108	893.97	11	886.77	11	861.24	11	828.94	10
c109	884.24	11	879.33	11	849.25	10	828.94	10
c201	751.82	6	714.77	5	591.56	3	591.56	3
c202	789.55	7	781.50	6	620.29	3	591.56	3
c203	742.94	5	713.88	5	620.30	4	591.17	3
c204	701.01	4	647.27	4	620.30	4	590.6	3
c205	651.85	4	627.19	4	609.36	3	588.88	3
c206	644.66	4	620.65	4	588.49	3	588.49	3
c207	672.56	4	627.45	4	588.29	3	588.29	3
c208	680.75	5	642.53	4	588.32	3	588.32	3
r101	2002.40	31	1929.86	30	1740.81	23	1650.8	19
r102	1748.55	25	1727.37	25	1571.62	21	1486.12	17
r103	1438.82	20	1432.44	20	1322.47	16	1292.68	13
r104	1121.01	14	1119.84	14	1023.98	12	1007.31	9
r105	1654.54	23	1616.53	23	1543.48	21	1377.11	14
r106	1458.44	19	1410.82	18	1363.17	17	1252.03	12
r107	1246.48	16	1235.65	16	1166.10	13	1104.66	10
r108	1082.62	12	1068.07	12	1013.46	11	960.88	9
r109	1315.13	17	1308.95	17	1254.31	16	1194.73	11
r110	1179.79	14	1173.93	14	1165.28	14	1118.84	10
r111	1180.15	14	1178.78	14	1111.35	12	1096.72	10
r112	1035.00	11	1031.78	11	1026.91	11	982.14	9
r201	1460.52	19	1410.97	18	1213.21	11	1252.37	4
r202	1256.64	14	1234.75	13	1132.80	9	1191.7	3
r203	1061.09	12	1038.61	12	935.21	7	939.5	3
r204	821.21	7	816.81	7	800.66	5	825.52	2
r205	1175.89	13	1138.70	13	1071.63	11	994.42	3
r206	1124.79	13	1080.48	12	1062.37	11	906.14	3
r207	963.94	9	927.41	8	900.88	7	890.61	2
r208	766.78	4	757.96	4	748.12	4	726.82	2
r209	1092.56	13	1090.33	13	949.97	9	909.16	3
r210	1013.65	10	1010.65	10	992.96	9	939.37	3
r211	878.33	8	861.63	8	848.29	8	885.71	2
rc101	2081.59	25	2067.51	25	1722.28	18	1696.94	14
rc102	1784.05	20	1770.82	20	1562.25	16	1554.75	12
rc103	1546.56	16	1523.11	16	1408.29	14	1261.67	11
rc104	1226.21	12	1225.77	12	1213.37	12	1135.48	10
rc105	1853.77	22	1841.23	22	1646.99	18	1629.44	13
rc106	1592.14	17	1586.47	17	1573.17	17	1424.73	11
rc107	1383.80	15	1369.77	14	1367.27	14	1230.48	11
rc108	1200.82	12	1193.42	12	1180.44	12	1139.82	10
rc201	1763.65	18	1762.11	18	1378.88	10	1406.94	4
rc202	1440.52	15	1436.88	15	1210.19	10	1365.65	3
rc203	1134.29	10	1123.39	10	1033.00	9	1049.62	3
rc204	931.49	8	930.57	8	901.01	7	798.46	3
rc205	1524.31	15	1522.04	15	1361.42	12	1297.65	4
rc206	1424.88	13	1419.05	13	1278.51	11	1146.32	3
rc207	1185.68	11	1142.21	10	1091.44	9	1061.14	3
rc208	828.99	5	807.28	5	791.87	5	828.14	3

Table 7: All solutions for the Goel instances with the RDP method and the saving method for the construction phase.

Instance	RDP	#Veh.	Saving	#Veh.	Prescott-Gagnon solutions	
ECc101	986.85	12	1083.35	13	931.37	10
ECc102	1025.56	11	1104.97	13	904.25	10
ECc103	1121.00	12	998.86	12	833.19	10
ECc104	1005.98	10	834.31	10	819.81	10
ECc105	828.94	10	895.89	11	828.94	10
ECc106	970.05	11	930.12	12	828.94	10
ECc107	828.94	10	865.99	11	828.94	10
ECc108	877.53	10	860.71	11	827.38	10
ECc109	905.76	10	835.88	10	825.65	10
ECc201	1061.20	8	1252.28	10	810.05	5
ECc202	961.15	7	1140.36	10	695.75	5
ECc203	902.21	7	936.89	8	661.84	4
ECc204	928.46	7	804.92	6	649.70	4
ECc205	798.75	7	768.51	7	678.70	5
ECc206	830.75	7	765.68	7	676.57	5
ECc207	893.53	7	884.33	8	674.67	5
ECc208	775.03	6	743.91	6	672.30	4
ECr101	1592.58	17	1700.81	23	1319.88	9
ECr102	1640.19	17	1519.41	20	1177.31	8
ECr103	1271.41	12	1275.58	17	967.96	8
ECr104	1090.42	9	983.42	10	855.72	8
ECr105	1342.96	12	1351.50	16	1090.69	8
ECr106	1233.53	11	1283.45	16	998.35	8
ECr107	1108.33	10	1132.02	13	892.90	8
ECr108	1056.17	9	920.70	8	840.95	8
ECr109	1103.40	10	1062.89	11	923.28	8
ECr110	1085.12	9	1026.01	11	880.19	8
ECr111	1080.59	9	983.58	10	881.27	8
ECr112	967.91	8	906.48	8	831.13	8
ECr201	1421.47	12	1475.88	19	1220.86	7
ECr202	1320.73	11	1351.56	17	1093.46	6
ECr203	1119.79	9	1102.26	13	957.70	5
ECr204	1014.41	8	875.18	8	770.21	5
ECr205	1162.05	9	1224.09	14	1000.40	6
ECr206	1174.81	9	1156.12	13	958.17	5
ECr207	1067.16	8	970.53	9	840.61	5
ECr208	968.25	7	821.66	7	754.21	5
ECr209	1113.03	8	1139.40	14	950.53	5
ECr210	1189.77	9	1087.82	12	938.69	6
ECr211	1044.68	7	930.70	10	788.35	5
ECrc101	1551.48	13	1810.57	19	1293.82	9
ECrc102	1524.33	13	1485.26	16	1177.51	9
ECrc103	1368.30	11	1153.40	11	1085.66	9
ECrc104	1263.51	10	1058.49	10	993.66	9
ECrc105	1575.17	13	1458.56	15	1203.34	9
ECrc106	1342.60	11	1226.22	12	1093.96	9
ECrc107	1273.73	10	1108.18	11	1028.11	9
ECrc108	1174.77	10	1074.66	10	982.59	9
ECrc201	1555.01	11	1876.61	21	1395.15	7
ECrc202	1445.17	10	1619.04	17	1153.45	7
ECrc203	1296.03	9	1188.30	10	1016.92	6
ECrc204	1003.66	8	976.40	9	863.22	5
ECrc205	1470.66	10	1672.62	18	1270.78	7
ECrc206	1316.89	10	1545.78	16	1129.39	6
ECrc207	1388.38	10	1338.67	13	1046.83	6
ECrc208	1101.17	7	933.66	7	850.63	5