Restricted dynamic programming: a flexible framework for solving realistic VRPs

J. Gromicho[a], J.J. van Hoorn[a], A.L. Kok[b,*], J.M.J. Schutten[b]

[a] *Vrije Universiteit, Amsterdam & ORTEC, Gouda, The Netherlands*

[b] *University of Twente, Enschede, The Netherlands*

Abstract

Most solution methods for solving large vehicle routing and scheduling problems are based on local search. A drawback of these approaches is that they are designed and optimized for specific types of vehicle routing problems (VRPs). As a consequence, it is hard to adapt these solution methods to handle new restrictions, without losing solution quality. We present a new framework for solving VRPs that can handle a wide range of different types of VRPs. Within this framework, restricted dynamic programming is applied to the VRP through the giant-tour representation. This algorithm is a construction heuristic which finds provably optimal solutions when unrestricted. We demonstrate the flexibility of the framework for a wide variety of different types of VRPs. The quality of solutions found by the framework is demonstrated by solving a set of benchmark instances for the capacitated VRP. The computational experiments show that restricted dynamic programming, which is a construction heuristic, develops routes of high quality. Therefore, this new framework for solving VRPs is highly valuable in practice.

Keywords: Restricted DP; Giant-Tour Representation; VRP;

*Corresponding author. Tel.: +31 53 489 4164; fax: +31 53 489 2159

E-mail adresses: JGromicho@ortec.nl, JHoorn@feweb.vu.nl, a.l.kok@utwente.nl, j.m.j.schutten@utwente.nl

1 Introduction

Local search methods have proved to be very successful in solving large vehicle routing and scheduling problems. Funke et al. [1] present an extensive overview of local search methods for vehicle routing and scheduling problems. These methods have been designed for a wide range of different types of VRPs and they develop high quality solutions for these problems.

A drawback of local search methods, however, is that it is difficult to generalize a local search method to many different types of VRPs. The main difficulty is to adapt local search methods when new restrictions are introduced to the problem. If such a restriction is added, then the neighborhood structure and the search strategy need to be carefully redesigned in order to obtain high quality solutions. Therefore, adding a single restriction requires the development of a new algorithm as the extensive lists in Parragh et al. [2, 3] show.

In practice, many different types of VRPs need to be solved, such as the capacitated VRP (CVRP), the VRP with time windows (VRPTW), and the pickup and delivery problem (PDP). Toth and Vigo [4] give an extensive overview of different types of VRPs and proposed solution methods. Companies such as logistic service providers and distribution firms have their own set of restrictions of which a certain part may be general to all companies, but often there is also a unique part for each company. As a consequence, each company requires a unique solution method to solve their routing problems. Therefore, solution strategies for the VRP that are flexible with respect to the addition of new restrictions and still produce high quality solutions are extremely valuable in practice. In general, solution methods based on local search lack this property.

We propose a highly flexible framework for solving large vehicle routing and scheduling problems. This framework covers a wide range of different types of VRPs. The solution approach within this framework is a construction heuristic, as opposed to the majority of the VRP-literature in which the focus is on route improvement heuristics. The solution approach is a generalization of the restricted dynamic programming heuristic for the TSP of Malandraki and Dial [5]. This can be seen as a form of beam search, see Bisiani [6]. We apply restricted dynamic programming to the VRP through the giant-tour representation, which was introduced by Funke et al. [1]. The giant-tour representation allows us to handle single tour and multiple tour problems in a similar way.

By solving a set of benchmark instances for the CVRP, we demonstrate that the quality of the obtained routes approaches the optimal solutions (average gap of 3%) within practical computation times. We select the CVRP since this VRP variant has a large solution space. Therefore, this new framework is a very promising approach for solving VRPs in practice.

Our paper is organized as follows. Section 2 describes our framework for solving VRPs by first describing dynamic programming for the TSP, then describing the giant-tour representation of VRP solutions, and finally proposing our framework for solving VRPs. Section 3 describes a heuristic using this dynamic programming formulation to obtain solutions within practical computation times. Section 4 demonstrates the flexibility of our framework by showing how a wide range of different types of VRPs can be solved within this framework. Section 5 presents the results of the computational experiments, which consist of solving a set of benchmark problems for the CVRP. Section 6 concludes this paper.

2 DP applied to the VRP

Our framework for solving VRPs is based on the the restricted dynamic programming (DP) heuristic for the TSP proposed by Malandraki and Dial [5], which is applied to the VRP through the giant-tour representation (GTR) introduced by Funke et al. [1]. Therefore, we first describe the DP for the TSP and the GTR of vehicle routing solutions.

2.1 Dynamic programming for the TSP

The restricted dynamic programming heuristic for the TSP is based on the exact dynamic programming algorithm for the TSP of Held and Karp [7] and Bellman [8]. The exact dynamic programming algorithm for the TSP can be described as follows.

The TSP considers the problem of visiting a set $V = \{0, 1, ..., n-1\}$ of n cities exactly once, starting and ending at city 0, and minimizing the total distance traveled. The distances between each pair of cities $i, j \in V$ are given by c_{ij}. A state $(S, j), j \in S, S \subseteq V \backslash 0$ represents a path starting at city 0, visiting all cities in S exactly once, and ending in city j. The cost $C(S, j)$ of a state is given by the length of the smallest of such paths. In the first stage the costs of the states are determined by $C(\{j\}, j) = c_{0j}, \forall j \in V \backslash 0$.

Next, in each successive stage the costs of the states are calculated with the recurrence relation $C(S,j) = \min_{i \in S \backslash j} \{C(S \backslash j, i) + c_{ij}\}$. Finally, the length of the optimal TSP tour is given by $\min_{j \in V \backslash 0} \{C(V \backslash 0, j) + c_{j0}\}$.

Since there are $\sum_{|S|=1}^{n-1} \binom{n-1}{|S|} \approx 2^n$ subsets S and each subset S contains $|S| \leq n$ possible end nodes, the total number of states is bounded by $\mathcal{O}(n * 2^n)$. Next each state is calculated by comparing at most $n-1$ additions, resulting in an algorithm with total complexity of $\mathcal{O}(n^2 * 2^n)$. The optimal TSP-tour can be backtracked by saving for each state (S,j) the city $i \in S \backslash j$ that minimizes $C(S \backslash j, i) + c_{ij}$.

Since this approach constructs only one route, it cannot be applied directly to the VRP. Therefore, we propose to apply it to the VRP through the GTR of vehicle routing solutions.

2.2 Giant-tour representation

Funke et al. [1] introduce the GTR of vehicle routing solutions, because it allows to handle single and multiple route problems in a similar way. Besides, it is a 'natural' representation of vehicle routing solutions. We use the GTR for the development of our general framework for solving VRPs. The GTR can be described as follows.

The basis of any routing problem is a directed graph $G = (V, A)$, in which the node set V consists of request nodes $R \subset V$, origin nodes $O \subset V$, and destination nodes $D \subset V$. For a VRP, the request nodes R correspond to all customer requests. Furthermore, for each vehicle there is one origin and one destination node, which all may represent the same location. Therefore, if m is the number of vehicles available, we get $|O| = |D| = m$. If we order the vehicle routes $r^v, v = 1, ..., m$ in a routing solution, then the GTR of this solution is a cycle in the graph G in which each route-end node d^v is connected to the route-start node of the next vehicle route o^{v+1}. Finally, the cycle is closed by connecting d^m with o^1. In Figure 1, we present an example of a vehicle routing solution with three vehicles, two depots (A and B) and nine customers. One vehicle starts at depot A and ends at depot B, while the other two vehicles start and end at depots A and B, respectively. Figure 2 presents the same solution with its corresponding GTR.

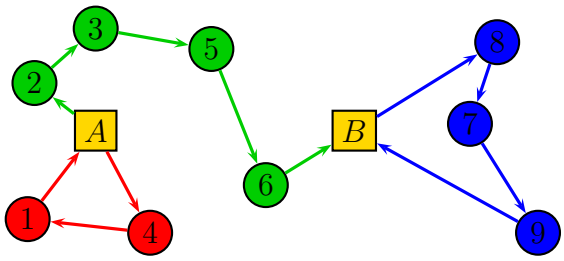

Figure 1: Example of a solution to a VRP with three vehicles

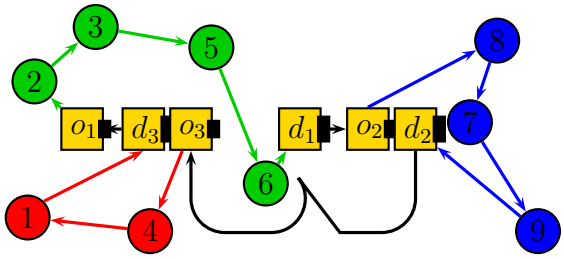

Figure 2: The Giant-tour representation of a solution to the VRP of Fig. 1

2.3 Dynamic programming for the VRP

By using the GTR we transform the VRP into a sequencing problem and we can use the DP-formulation for the TSP to solve it. However, we need to ensure that the DP-solution is the GTR of a feasible VRP solution. There are several ways to ensure this, for example by making the graph G incomplete. However, a more general way to do this is by checking the feasibility of a partial solution while expanding a state. We only call an expansion feasible if it represents the giant tour of a feasible partial VRP-solution. So for a state (S, d^v) where the partial solution ends with node d^v the only feasible expansion is o^{v+1}. On the other hand, o^{v+1} can only be an expansion of a state with end node d^v. Therefore, unlike the TSP, not every expansion is feasible for the VRP and we must perform a feasibility check when expanding a state. This seems a downside, but it actually gives us the power to use this framework for almost every type of VRP.

To apply the DP to the VRP, we add $2m$ nodes to the $|R|$ nodes for the customers, which would lead to $2m + |R|$ nodes for the DP. However, each end node of a vehicle is attached to the start node of the following vehicle

leaving only m extra nodes in consideration, i.e., $n = m + |R|$. Furthermore, these nodes have a fixed order in the GTR which reduces the state space considerably.

The ordering of the vehicles imposes a serial precedence relation of m nodes $i_1 < i_2 < \ldots < i_m$. For every state (S, j) where $i_a \notin S$ and $i_b \in S$ with $i_a < i_b$ there exists no feasible partial solution because the precedence relation is not satisfied. So for every (S, j) that has a feasible partial solution there exists a $k, 1 \leq k \leq m$ such that $i_a \in S$ if $a \leq k$ and $i_a \notin S$ if $a > k$. The number of states that exists for a certain k and $|S| = l$ are $\binom{n-m}{l-k}$, with $k \leq l$ and $l - k \leq n - m$. If we sum this over all k and l we get, $\sum_{k=0}^{m} \sum_{l=k}^{n-m+k} \binom{n-m}{l-k} = (m+1)2^{n-m}$ (which includes $S = \emptyset$.) If we divide this by all possible $S \subseteq V$ we get $\frac{(m+1)2^{n-m}}{2^n} = \frac{m+1}{2^m}$, which is the fraction of states that can contain feasible partial solutions given precedence relation $i_1 < i_2 < \ldots < i_m$.

Any given state can possibly end in $(|R| + m) = n$ nodes but can only be expanded to $(|R| + 1)$ nodes due the the precedence relation of the depot nodes. So the complexity of the DP for the VRP is

$$\mathcal{O}\left((|R| + m)(|R| + 1)\frac{m+1}{2^m}2^{|R|+m}\right) = \mathcal{O}\left(n^2 m 2^{n-m}\right).$$

3 Restricting the state space

Although dynamic programming has the best running time of all exact algorithms for the TSP proposed so far, see Woeginger [9], it is still not fast enough to solve problems of realistic sizes in practical computation times. Therefore, Malandraki and Dial [5] propose a restricted version of this algorithm, in which the number of states in each stage is bounded by a parameter H. This bounding procedure works as follows.

In each stage, we only take the H states with the smallest cost to expand to the next stage. Since each state reflects a partial tour and in each stage all partial tours visit the same number of customers, a state with low costs is more likely to yield the first part of a good TSP solution than a state with high costs. Therefore, continuing the algorithm with only the H smallest cost states will, although it does not guarantee to find the optimal TSP-tour, probably still result in a good TSP solution. Next, each of these H states is expanded in all possible ways with one extra city. If certain states

are reached multiple times (e.g., expanding (S, j) with customer k results in the same state as expanding (S, i) with customer k, namely, $(S \cup k, k)$), then, according to the original recurrence relation, only the one with lowest costs is maintained.

Malandraki and Daskin show that restricted dynamic programming is a flexible approach for solving TSPs by applying it to the TSP with time-dependent travel times. They also show that increasing the value of H results in better solutions, but with the cost of higher computation times. Note that setting $H = 1$ results in the nearest neighbor heuristic and setting $H = \infty$ results in the exact dynamic programming algorithm for the TSP.

We restrict the state-space even further, using a form of beam search Bisiani [6], by expanding each state (S, j) only to the E nearest i nodes with $i \notin S$. This is reasonable because we expect that all edges in the optimal solution will most likely be between two nodes that are near-neighbors of each other, as observed on page 347 of Rego and Glover [10].

The same principle can be applied to the DP algorithm for the VRP. However we may choose to expand each state (S, j) to at most E feasible states, starting with the nearest nodes, to prevent from only expanding to infeasible states. We observe that our beam through the state space will require a polynomial effort for each fixed H and E amounting to expanding to $\mathcal{O}(nHE)$ states, leading to a complexity of $\mathcal{O}(n^2HE)$.

4 The flexibility of our general framework

We demonstrate the flexibility of our framework, by showing how it can solve the capacitated vehicle routing problem and the vehicle routing problem with time windows by adding more dimensions, and the pickup and delivery problem by adding precedence relations. Furthermore, we show how other realistic constraints such as multiple depots, a heterogeneous vehicle fleet, the open vehicle routing problem and multiple routes per vehicle can be incorporated within the framework. Note that any conceivable combination of these problems is equally suited. If we compare this uniformity with the large variety of approaches found in the literature, see Parragh et al. [2, 3], then we may conclude that it is quite remarkable to obtain a general algorithmic framework for these problems which are in general considered very diverse and specific.

4.1 Adding more dimensions

For the capacitated vehicle routing problem (CVRP), as well as the vehicle routing problem with time windows (VRPTW), we add feasibility checks on capacity or time. However, the optimality guarantee of (unrestricted) DP for the VRP is lost. We demonstrate this by the following example. Suppose two states (S, j) and (S, i) can be feasibly expanded with the same node k such that the first extension results in a partial solution with lower cost than the second extension, but with less slack in capacity or time. According to the original recurrence relation the first extension will be selected. However, it may prove impossible to complete the first extension to a feasible complete solution, while the second extension can be completed to a feasible solution. This is resolved by only leaving a partial solution out of consideration when there is another partial solution of the same state with lower cost and more slack. This is formalized in dominance rules as in Dumas et al. [11], which can result in several partial solutions that need to be considered for some states. In the worst case this leads to a brute force algorithm. However, in practice the cost and the slack are correlated, so for all partial solutions of the same state it is unlikely that no solution is dominated by an other solution. Besides, in the restricted DP we only expand at most H states.

4.2 Adding precedence relations

To apply our model to the pickup and delivery problem (PDP), we add precedence constraints to the pickup-delivery pairs, thereby reducing the state space. Furthermore, we need to add a feasibility check to ensure that before returning to a depot all deliveries corresponding to the pickups visited by the returning vehicle are visited, otherwise the return to the depot of that vehicle is an infeasible expansion. These precedence relations have also a big effect on the complexity because every independent relation of a pickup and delivery reduces the state-space by $\frac{3}{4}$, see Section 2.3. So the complexity for the PDP is $\mathcal{O}\left(\left(\frac{3}{4}\right)^{\frac{n-m}{2}} n^2 m 2^{n-m}\right)$ since there are $\frac{n-m}{2}$ pickup-delivery pairs.

4.3 More realistic constraints

The characteristics of each vehicle can vary greatly in our model, only the feasibility checks have to be changed according to the current vehicle. This gives us the power to implement a heterogeneous vehicle fleet with different

time windows or capacity per vehicle as well as implementing multiple depots by changing the depot for certain vehicles. To implement the open vehicle routing problem (OVRP), where the vehicles do not return to the depot, we only need to set all travel distances to the route end nodes to 0. If we want to allow a vehicle to return to the depot in a route to reload, we need to implement the two routes as different vehicles. However, we need to make sure the second route does not start before the first route ends. This can be done by using a flexible time window on the route start of the second vehicle depending on the route end time of the first plus using this order in the GTR.

5 Computational experiments

We test the performance of the DP-framework on a set of benchmark instances for the CVRP developed by Augerat [12]. We select the CVRP, since it is the least restricted variant of the VRP and, therefore, has the largest solutions space. In general additional restrictions reduce the state space. Note that when we have a homogeneous fleet and impose no restrictions at all, the solution to a general VRP will be equal to a TSP because returning to the depot and start with a new vehicle makes no sense. Furthermore, even if it is required to use all vehicles it can be reduced to a TSP, this is a well known reduction from the m-TSP to the TSP, which amounts to replicating the depot with as many copies as the number of salesman and forbidding direct transitions from copy to copy by setting the inter-depot costs to infinite. We demonstrate that, despite the fact that restricted dynamic programming is a construction heuristic, it produces good results for the CVRP benchmark instances.

Within the dynamic programming framework, the cost of each state $(S, j), j \in S, S \subseteq V \backslash 0$ is defined by the distance of a shortest path visiting the nodes in S and ending in node $j \in S$. For the CVRP, this cost definition may cause some problems when we restrict the state space. For example, in the second stage the state which represents the path of visiting first the node nearest to the depot and then directly returning to the depot is likely to have a quite small cost compared to the other states in the same stage (this cost being the smallest if the distance between the depot and its nearest node is less than half the distance between the depot and its second nearest node). However, since the number of vehicles is limited in the CVRP,

returning low-filled trucks to the depot may lead to infeasible solutions. We avoid this problem by only allowing a vehicle to return to the depot if the percentage of demand served so far is not smaller than the percentage of vehicles used so far. This extra restriction does not exclude the optimal solution, since for each solution there always exists an ordering of the vehicles, such that the average demand served by the first k vehicles is not smaller than k times the average demand over all vehicles.

If states A and B visit the same customer set S and end in the same node $j \in S$, then state A only dominates state B if its cost is not larger than the cost of state B and its remaining capacity is not smaller than the remaining capacity in state B. This ensures that we do not exclude the optimal solution. If in a certain stage the number of states exceeds its maximum H, then only the H states with the lowest cost are maintained. Possible ties are broken by selecting the state with the highest remaining capacity. Finally, we first set E to its maximal value n.

We implemented the DP framework in Delphi 7 and ran our experiments on a Pentium 4, 3.40GHz CPU and 1.00 GB of RAM. Table 1 presents the average gaps of the solutions found by the DP framework with the optimal solutions (all problem instances and optimal solutions are available at [13]). The Augerat instances consists of three problem sets: A, B and P. Customer locations and demands are random in the A instances, the B instances are clustered and the P instances are modified instances from the literature. There are 27 A instances, 23 B instances, and 24 P instances. The number of customers in the A, B, and P instances varies from 32 to 80, 31 to 78, and 16 to 101, respectively. In the column 'All', we also report, between brackets, the maximum gaps over all problem instances. The last column presents the average computation times over all problem instances in seconds. The results show that if we set $H = 1,000$, we get results within 10% of the optimum on average. For this value of H computation times are still very small (1 second on average). If we increase H to 100,000, then the average gaps drop below 5 percent with computation times around 2.5 minutes.

The maximum gaps are rather large even for large values of H. However, the number of instances with a gap larger than 10% is very small for large values of H. For example, if $H = 1,000,000$, then only 2 instances have a gap larger than 10%. Table 2 presents the number of problem instances with a gap larger than 10% for the different values of H. It also presents the number of problem instances for which the optimal solution is found. As can be seen, the number of instances with a large gap reduces considerably for

H \ Problem Set	A	B	P	All	cpu(s)
10	23.96	20.80	18.78	21.40 (43.53)	0.063
100	14.71	16.73	10.14	13.95 (36.43)	0.171
1,000	10.11	10.56	6.12	9.01 (21.59)	1.29
10,000	6.79	6.99	4.54	6.16 (21.15)	12.9
100,000	4.59	5.68	3.41	4.58 (17.68)	151
1,000,000	3.11	3.77	2.55	3.15 (15.86)	1727

Table 1: Gaps (%) with optimal solutions

increasing values of H, while the number of instances for which the optimal solution is found increases.

	Gap > 10%		Optimal	
H	# Instances	Percentage	# Instances	Percentage
10	63	91.3%	0	0.00%
100	54	73.0%	2	2.70%
1,000	30	40.6%	4	5.41%
10,000	10	13.5%	7	9.46%
100,000	5	6.76%	8	10.81%
1,000,000	2	2.70%	9	12.16%

Table 2: Instances with gap larger than 10% and optimal solutions

Tables 3 and 4 present the impact of different E values on the performance of the algorithm. We set the values of E to n, $0.5n$, $0.25n$, $0.125n$, and $0.05n$ (we round in case of fractional values) and we present average gaps and average computation times over all problem instances. Since $E = 0.05n$ results in an E value of 1 for problem instances with less than 30 customers yielding the nearest neighbor heuristic, we set the absolute minimum of E to 2. The results indicate that it is possible to decrease E such that computation times also decrease, while the solution quality is maintained. However, when E is getting smaller than $0.125n$, then the solution quality starts to reduce significantly, especially for large H. For example, $E = 0.05n$ and $H = 1,000,000$ has a larger gap than $E = 0.25n$ and $H = 100,000$, while computation times are significantly smaller in the latter case (127 seconds versus 533 seconds). Note that for one problem instance no feasible solution could be found with $E = 0.05n$ and all values of H.

$H \setminus E$	n	$0.5n$	$0.25n$	$0.125n$	$0.05n$
10	21.40	21.40	21.48	21.49	21.89
100	13.95	14.11	14.13	13.86	12.86
1,000	9.01	9.22	9.03	9.23	9.47
10,000	6.16	6.21	6.16	6.85	6.96
100,000	4.58	4.68	4.67	5.03	5.80
1,000,000	3.15	3.12	3.19	3.59	4.83

Table 3: Impact of different E values: average gaps (%)

$H \setminus E$	n	$0.5n$	$0.25n$	$0.125n$	$0.05n$
10	0.063	0.063	0.079	0.060	0.066
100	0.171	0.187	0.178	0.133	0.107
1,000	1.29	1.19	1.02	0.874	0.574
10,000	12.9	12.5	11.1	9.08	5.26
100,000	151	145	127	103	56.2
1,000,000	1727	1636	1409	1076	533

Table 4: Impact of different E values: cpu (s)

We can conclude that the dynamic programming framework solves the CVRP benchmark instances to a practically desirable quality within practical computation times. The number of problem instances with a gap larger than 10% reduces considerably when H increases. These results are particularly interesting, since restricted dynamic programming is a construction heuristic, such that no other methods are needed to get some initial solutions. Finally, increasing H will eventually result in an optimal algorithm for the CVRP, implying a guarantee for finding better solutions if H is raised enough.

6 Conclusions

This paper introduces a highly flexible framework based on dynamic programming for Vehicle Routing Problems which is able to model and solve different types of problems previously tackled by tailor-made algorithms. Formerly, if a type of Vehicle Routing Problem would emerge from a known type by just changing or adding some constraints one would often be forced to go back to the drawing board and devise a whole new tailor-made approach to handle it. Therefore, a general framework which is able to accommodate dif-

ferent types of problems offers the added benefit to handle several variants of problems which are hybrid of existing ones. In order to demonstrate that the benefit of our framework extends further than just theoretical possibilities the paper also shows how to control the size of state space, otherwise exponential, by restricting the number of expansions of a state leading to practical and efficient polynomial construction heuristics. The quality of these heuristics is measured by applying them to well-known benchmark instances of the Capacitated Vehicle Routing Problem and producing solutions deviating very little from the optimum. Such achievements are generally only the realm of powerful neighborhood search methods, and hence a remarkable outcome of a generic construction algorithm.

Acknowledgment

This work was financially supported by Stichting Transumo through the project ketensynchronisatie.

References

[1] B. Funke, T. Grünert, and S. Irnich. Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of heuristics*, 11(4):267–306, 2005.

[2] S. N. Parragh, K.F. Doerner, and R.F. Hartl. A survey on pickup and delivery problems. part i: Transportation between customers and depot. *Journal für Betriebswirtschaft*, 58(1):21–51, 2008.

[3] S. N. Parragh, K.F. Doerner, and R.F. Hartl. A survey on pickup and delivery problems. part ii: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

[4] P. Toth and D. Vigo. *The vehicle routing problem*. SIAM Monographs on Discrete Mathematics and Applications. Philadelphia, 2002.

[5] Chryssi Malandraki and Robert B. Dial. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research*, 90(1):45–55, 1996.

[6] R. Bisiani. Beam search. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 56–58. Wiley & Sons, 1987.

[7] M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196, 1962.

[8] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the Association for Computing Machinery*, 9(1):61, 1962.

[9] G.J. Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial OptimizationEureka! You shrink!, Lecture Notes in Computer Science, Vol. 2570*, pages 185–207. Springer, Berlin, 2003.

[10] C. Rego and F. Glover. Local search and metaheuristics. In G. Gutin and A. Punnen, editors, *Traveling Salesman Problem and Its Variations*, pages 309–367. Kluwer Academic Publishers Dordrecht, 2002.

[11] Y. Dumas, J. Desrochiers, E. Gelinas, and Marius M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations research*, 43(2):367–371, 1995. 0030-364X.

[12] P. Augerat. *Approche Polyèdrale du Problème de Tournées de Véhicules.* PhD thesis, Institut National Polytechnique de Grenoble, 1995.

[13] T. Ralphs. Branch cut and price resource web. World Wide Web, 2003. `http://www.brachandcut.org`.