Restricted dynamic programming: A flexible framework for solving realistic VRPs

J. Gromicho, J.J. van Hoorn, A.L. Kok, J.M.J. Schutten

PII: S0305-0548(11)00195-X
DOI: doi:10.1016/j.cor.2011.07.002
Reference: CAOR 2860

To appear in: *Computers & Operations Research*

Received date: 18 January 2010
Revised date: 30 June 2011
Accepted date: 1 July 2011

Cite this article as: J. Gromicho, J.J. van Hoorn, A.L. Kok and J.M.J. Schutten, Restricted dynamic programming: A flexible framework for solving realistic VRPs, *Computers & Operations Research*, doi:10.1016/j.cor.2011.07.002

Restricted dynamic programming: a flexible framework for solving realistic VRPs

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

[a] *Vrije Universiteit, Amsterdam, The Netherlands*

[b] *ORTEC, Gouda, The Netherlands*

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

Abstract

Most successful solution methods for solving large vehicle routing and scheduling problems are based on local search. These approaches are designed and optimized for specific types of vehicle routing problems (VRPs). VRPs appearing in practice typically accommodate restrictions that are not accommodated in classical VRP models, such as time-dependent travel times and driving hours regulations. We present a new construction framework for solving VRPs that can handle a wide range of different types of VRPs. In addition, this framework accommodates various restrictions that are not considered in classical vehicle routing models, but that regularly appear in practice. Within this framework, restricted dynamic programming is applied to the VRP through the giant-tour representation. This algorithm is a construction heuristic which for many types of restrictions and objective functions leads to an optimal algorithm when applied in an unrestricted way. We demonstrate the flexibility of the framework for various restrictions appearing in practice. The computational experiments demonstrate that the framework competes with state of the art local search methods when more realistic constraints are considered than in classical VRPs. Therefore, this new framework for solving

*Corresponding author. Tel.: +31 53 489 4676; fax: +31 53 489 2159

E-mail adresses: joaquim.gromicho@ortec.nl, JHoorn@feweb.vu.nl, leendert.kok@ortec.nl, j.m.j.schutten@utwente.nl

VRPs is a promising approach for practical applications.

Keywords: Restricted DP; Giant-Tour Representation; VRP; Real-life Restrictions

1 Introduction

Meta heuristics have proved to be very successful in solving large vehicle routing and scheduling problems. The survey by Gendreau et al. [1] lists six types for the Capacitated VRP alone, and from the surveys of Parragh et al. [2, 3] we can only estimate the number to be much larger when allowing different types of VRP. While meta heuristics rely on the careful definition (and redefinition) of neighborhoods for each type of VRP, Pisinger and Ropke [4] propose a framework for construction, destruction and amendment of solutions, in which they assume construction to be simple and focus on identifying parts of the solution suspected of compromising quality and amending them. Alongside with the extensive work on meta heuristics for the Vehicle Routing Problem and the work of Pisinger and Ropke, we propose a general construction only framework for very general Vehicle Routing Problems based on dynamic programming.

We believe that there is still room for a general construction method that is able to deal with realistic and general types of Vehicle Routing Problems. Solutions found may certainly be improved by either local search or other amendment algorithms, but our approach already allows for a trade-off between performance and quality. Moreover, the computational experiments indicate that our approach is generic enough to allow finding reasonable solutions for many types of vehicle routing problems without substantial redesign and coding effort.

The approach we propose is a restricted dynamic programming heuristic for very general vehicle Routing Problems, along the same lines as followed by Malandraki and Dial [5] for the time-dependent Traveling Salesman Problem. Furthermore, our parameter H has the same interpretation as theirs homonym parameter: $H = 1$ renders our approach to a nearest neighbor approach, while higher values of H generalize this simple approach by allowing at most H solutions to be expanded further in each stage of the state space. By also including a new restriction, now on the number E of expansions per state, we reason as Toth and Vigo [6] in their Granular Tabu Search

approach: long arcs are not likely to be part of optimal solutions. Therefore, ignoring those long arcs may substantially reduce the required computation times, without significant quality loss. Our approach can also be seen as a beam search approach, following the terminology introduced by Raj Reddy on his Artificial Intelligence courses at the Carnegie Mellon University in 1976. The first published use of the term seems to be Lowerre [7]. Novel from our perspective is the concept being used on the state space of a dynamic equation and not on the solution space, as traditional enumerations algorithms do.

In practice, extensions of various 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 [8] give an extensive overview of different types of VRPs and proposed solution methods. In addition, 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 most companies also have some unique restrictions. As a consequence, each company requires a unique solution method to solve their routing problems.

The framework we propose covers a wide range of different types of VRPs. Moreover, it accommodates various restrictions that appear in practice, but that have been generally ignored in VRP literature, such as time-dependent travel times and driving hours regulations. We apply restricted dynamic programming to the VRP through the giant-tour representation, which was introduced by Funke et al. [9]. The giant-tour representation allows us to handle single tour and multiple tour problems in a similar way.

The contributions of this paper are the following. First, we propose an exact algorithm for the VRP. In the framework of this algorithm, various real life restrictions and VRP variants can be accommodated. Second, we demonstrate this flexibility by showing how the algorithm can be applied to known VRP variants, and by showing how various additional real life restrictions can be accommodated. Third, in addition to the way Malandraki and Dial [5] propose to restrict the state space to reduce computation times, we propose an additional way to restrict the state space which reduces computation times even further while maintaining or even improving solution quality. As a result of restricting the state space, the algorithm runs in practical (polynomial) computation times and produces high quality solutions for realistic types of VRPs. Fourth, we show the quality of our framework by solving the classical Solomon [10] benchmark instances for the VRPTW. Moreover, we

apply it to benchmarks of more restrictive vehicle routing problems: the PDP with time windows (PDPTW) and the VRPTW with the European Community (EC) social legislation on driving and working hours (VRPTW-EC). The results illustrate the power of this construction framework when applied to more realistic vehicle routing problems: when more restrictions are added to the problem, the performance of the framework improves, making it even competitive with state of the art improvement methods for the VRPTW-EC.

Our paper is organized as follows. Section 2 first describes dynamic programming for the TSP, then describes the giant-tour representation of VRP solutions, and finally describes our framework for solving VRPs. Section 3 describes a way to reduce the state space of this dynamic programming formulation to obtain solutions within practical computation times. Section 4 demonstrates the flexibility of our framework by showing how known types of VRPs can be solved within this framework, and how various additional real life restrictions can be accommodated. Section 5 presents the results of computational experiments. Section 6 summarizes the main findings in 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] We apply the (DP) heuristic to the VRP through the giant-tour representation (GTR) of vehicle routing solutions introduced by Funke et al. [9]. The term giant tour was also introduced by Beasley [11]. However, this giant tour is actually a TSP solution with only one single copy of the depot. Therefore, it is not a representation of a feasible VRP solution, but it has to be turned into a feasible VRP solution by somehow cutting the giant tour into feasible VRP routes. 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 [12] and Bellman [13]. 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 travel distance. The travel distance between each pair of cities $i, j \in V$ is given by c_{ij}.

A state $(S, j), j \in S, S \subseteq V\backslash 0$ in the DP algorithm 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 $|S| \leq n-1$ possible end nodes, the total number of states is $\mathcal{O}(n2^n)$. Next each state is calculated by comparing at most $n-2$ additions, resulting in an algorithm with a running time 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. We propose to apply it to the VRP through the GTR of vehicle routing solutions.

2.2 Giant-tour representation

Funke et al. [9] 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 $D \subset V$, and the arc set A represents feasible travels between these nodes. 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.

To use this representation, the number of available vehicles must be known beforehand. If the number of available vehicles is not given, we can use an upperbound on the required number of vehicles, e.g., by setting it equal to the number of customers. If it turns out that some of the vehicles are not required (typically when one of the objectives is to minimize the number of vehicles), this is represented by directly connecting their origin and destination nodes.

In Figure 1, we present an example of a vehicle routing solution with three vehicles, two depots (A and B) and nine customers. Vehicle 1 starts at depot A and ends at depot B, vehicle 2 starts and ends at depot B, and vehicle 3 starts and ends at depot A. Figure 2 presents the same solution with its corresponding GTR.

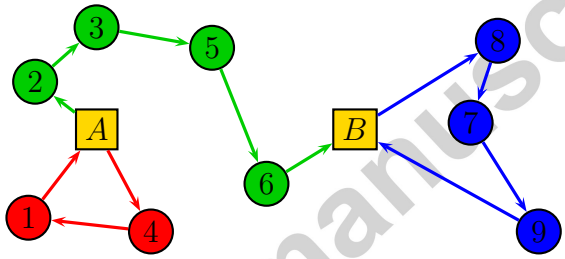

Figure 1: Example of a solution to a VRP with three vehicles

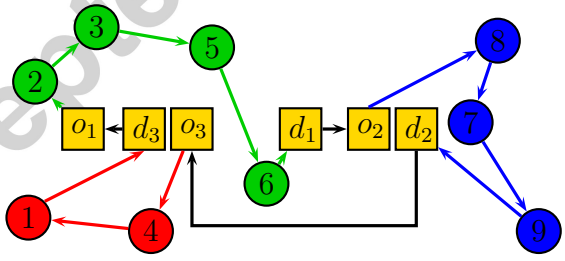

Figure 2: The giant-tour representation of a solution to the VRP of Fig. 1

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. A 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}. Furthermore, 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 derive the running time complexity of the DP algorithm for the VRP, observe that we have to add $2m$ nodes to the $|R|$ nodes for the customers. This would lead to a total of $2m + |R|$ nodes. However, each end node of a vehicle is attached to the start node of the following vehicle leaving only m extra nodes in consideration $(i_1, i_2, \ldots, i_m)$, 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 $i_1 \rightarrow i_2 \rightarrow \ldots \rightarrow i_m$. For every state (S, j) for which holds that $i_a \notin S$, $i_b \in S$ and $i_a \rightarrow i_b$, there exists no feasible partial solution, because the precedence relation is not satisfied. Therefore, for every (S, j) that does have a feasible partial solution, there exists a $k, 0 \leq k \leq m$, such that $i_a \in S$ if $a \leq k$ and $i_a \notin S$ if $a > k$. We first derive the fraction of the total number of subsets that have this property, since this fraction equals the fraction of states that contain feasible partial solutions.

Suppose $|S| = l$ and let $i_1, ..., i_k \subseteq S$. Then it must hold that $k \leq l$ and $l - k \leq n - m$, with $l - k$ request nodes in S and $n - m$ request nodes in total. The total number of such subsets S equals $\binom{n-m}{l-k}$. If we sum this over all k and l, we get $\sum_{k=0}^{m} \sum_{l=k}^{n-m+k} \binom{n-m}{l-k} = (m+1)2^{n-m}$ subsets. If we divide this by the total number of subsets $S \subseteq V$, 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 the precedence relations $i_1 \rightarrow i_2 \rightarrow \ldots \rightarrow i_m$. This implies that each independent precedence relation $i \rightarrow j$ reduces the state space by $\frac{1}{4}$ (in this case $m = 2$ such that the fraction of states that contain feasible partial solutions is $\frac{3}{4}$).

Finally, observe that any given state can possibly end in $|R| + m = n$ nodes, but can only be expanded to $|R| + 1$ nodes (because of the serial precedence relation). Therefore, the running time complexity of the DP algorithm 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 complexity of all known exact algorithms for the TSP (see Woeginger [14]), it is 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 use the H states with the smallest costs to expand in 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 states with smallest costs 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, i.e., $(S \cup k, k)$), then, according to the original recurrence relation, only the one with lowest costs is maintained.

Malandraki and Dial [5] 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 also in substantial 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 by expanding each state (S, j) to its nearest unvisited nodes until we have found at most E feasible expansions of state (S, j). Note that $E = 1$ also results in the nearest neighbor heuristic, regardless of the value of H. This restriction is similar to beam search (Bisiani

[15]). However, beam search is applied to the *solution space*, whereas we use a restriction on the search through the *state space*. The restriction on the number of expansions of each state is reasonable, because edges in the optimal solution will most likely be between two nodes that are near neighbors of each other, as observed by Rego and Glover [16] and Toth and Vigo [6].

The same principle can be applied to the DP algorithm for the VRP. Since state expansions may be infeasible, we expand to the nearest unvisited nodes until we have found at most E feasible expansions. We respect the precedence relations on the route start and end nodes when expanding the states. We observe that our beam through the state space will require a polynomial effort, i.e., for each fixed H and E we expand to $\mathcal{O}(nHE)$ states. The expansion of a state can be done in $\mathcal{O}(1)$. However, since we need to select the H best states we get a running time complexity of $\mathcal{O}(n^2 HE \log(H))$. We give a numerical analysis of the effects of E in Section 5.1, which shows that a small value of E has a positive effect on computation time, and sometimes even also on the solution quality.

4 The flexibility of our solution approach

To demonstrate the flexibility of the DP algorithm, we show two techniques to accommodate various problem extensions. Then, we show how various realistic constraints, both ones that are considered in classical vehicle routing models and ones that have never or hardly been considered in VRP literature, can be accommodated in the DP framework. Note that any conceivable combination of these constraints is equally suited. Therefore, our generic approach is unique in comparison with the large variety of approaches found in literature, see Parragh et al. [2, 3], all of which are diverse and specific for some variants of the VRP. Also the application of the DP framework to VRP variants with complex timing restrictions as time-dependent travel times and driving hours regulations by Kok [17] emphasizes the flexibility of the DP framework.

4.1 Extra state dimensions

For the CVRP, as well as the VRPTW, we add state dimensions on capacity or time. When expanding a state, we perform a feasibility check to ensure that there is enough slack in capacity or time. However, we have to be careful

not to lose the optimality guarantee of the (unrestricted) DP algorithm for the VRP. 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 expansion results in a partial solution with lower costs than the second expansion, but with less slack in capacity or time. According to the original recurrence relation, the first expansion will be selected. However, it may prove impossible to complete the first expansion to a feasible complete solution, while the second expansion can be completed to a feasible solution. This is resolved by making two copies of this state such that one represents the partial solution with lower costs, while the other represents the partial solution with more slack. This is formalized in dominance rules as in Dumas et al. [18], which can result in several copies of a single state. However, in practice costs and slack are correlated, such that for all states with the same visited node set S and end node $j \in S$ it is unlikely that no state is dominated by another one.

4.2 Input characteristics and precedence relations

In certain cases it may be infeasible to visit a certain customer with a specific vehicle. For example, frozen goods need to be transported in a refrigerated truck. This feasibility and others, even depending on the actual visit sequence, can be checked when expanding a state. To apply our model to VRPs with sequencing restrictions (e.g., the PDP), we add precedence constraints to the nodes involved (e.g., for each pickup-delivery pair, we add a precedence constraint), thereby reducing the state space. Furthermore, we need to add a feasibility check when a vehicle returns to a depot to ensure that the vehicle only returns if it has visited all the deliveries corresponding to the pickups that it has visited. These precedence relations also have a big impact on the running time, because every independent relation of a pickup and delivery reduces the state space by $\frac{1}{4}$ (see Section 2.3). When the state space is restricted, the effect on the running time is in general negligible. However, the reduction of the state space generally has a positive impact on the solution quality: the (restricted) part of the state space that is explored is now a larger portion of the entire state space.

4.3 Realistic constraints

Several other realistic constraints fit within our algorithmic framework. State dimensions and input characteristics allow for various extensions of the VRP. Section 4.3.1 presents an overview of classical VRP extensions and how they fit within our algorithmic framework. Section 4.3.2 presents an overview of practical VRP extensions that have never or hardly been considered in literature, but can be accommodated in the DP framework using the techniques described above. Moreover, any conceivable combination of these extensions can be accommodated by combining the single extensions in the DP framework. This list of extensions is motivated by practice [19], but is certainly not exhaustive.

4.3.1 Overview of classical VRP extensions that fit within the DP framework

Capacities (CVRP) For the CVRP, we add a state dimension for the remaining capacity of the current vehicle. A state dominates another state if it has no more costs than that other state and if it has at least the same amount of slack with respect to the remaining capacity. Moreover, we add a feasibility check to test whether the remaining capacity is sufficient to be able to visit a customer.

Time windows (VRPTW) We add a state dimension for the departure time at the last visited node. Furthermore, we add a feasibility check to each expansion to see whether the time window at the customer can be met. The dominance criteria are similar as with the CVRP with time dimensions added.

Pickup and deliveries (PDP) We add precedence relations for the pickup and delivery pairs in the PDP. We add feasibility checks preventing expansions to deliveries where the predecessor (corresponding pickup) has not been visited yet. Furthermore, we add a feasibility check when returning to the depot to ensure that all corresponding deliveries to the visited pickups have also been visited.

Open VRP[1] The open VRP can be controlled by the input, setting all distances from customers back to the depot to 0.

[1] In the open VRP, vehicles do not have to return to the depot.

Multiple depots Different start and finish locations of the vehicles can be regulated by the input. The start and finish location of each vehicle are characteristics of the nodes representing the origin and destination of that vehicle.

Heterogeneous fleet Different vehicle capacities and availability periods can be controlled by setting the state dimensions at the departure from the depot to different values corresponding to the vehicle characteristics of the current vehicle.

Multiple routes for a single vehicle Each route of a single vehicle can be modeled by a separate (successive) vehicle route. The dependency of a vehicle route B on its preceding route A can be controlled by a flexible time window (or location) on the departure node of route B, depending on the return time and location of route A. The routes of the same vehicle should appear ordered in the GTR to be able to provide the start time and location of successive routes.

4.3.2 Overview of VRP extensions from practice that fit within the DP framework

Time-dependent travel times As with the VRPTW, a state dimension is added on the departure time at the last visited node to determine the travel time. Moreover, the applicability of restricted DP to the VRP with time-dependent travel times is illustrated in Malandraki and Dial [5], Kok et al. [20], and Kok et al. [21]. Furthermore, the assumption is normally made that leaving at a later time cannot result in an earlier arrival (the non-passing property). If the non-passing property does not hold, a function that returns the earliest arrival according to the current possible departure time, possibly waiting before departure, can be implemented to still provide this property.

Driving hours regulations To account for driving hours regulations, we add state dimensions to keep track of the remaining driving and working times until a break is required. When a break occurs during a travel between two customers, this travel is extended by the duration of the break (see Kok et al. [22]).

Rolling time horizon We add state dimensions according to the initial state of the problem. For example, at the start of the planning horizon,

we set the capacities of the vehicles to the remaining capacity, given the stops that are already planned for these vehicles (see also the scenario with multiple routes for a single vehicle).

Multiple compartments We add state dimensions to keep track of the remaining capacity of each compartment of the current vehicle. We expand a state not only to each node, but for each node we consider placing the load into each compartment. Feasibility checks are added to test whether a load fits in a compartment. Note that the specifics of these feasibility checks depend on the problem at hand. For example, indivisible loads require to be placed entirely within the same compartment, whereas formless load can be divided over different compartments, but each compartment may then contain only one type of product.

Vehicle characteristics We add feasibility checks to ensure that customers are only visited by vehicles that are allowed to visit them (e.g., large trailer combinations may not be allowed to serve customers located in city centers).

Customer combinations We add feasibility checks to ensure that all customers visited in the same route can actually be combined in the same vehicle (e.g., loads from different customers may contaminate, such that they should not be visited in the same route).

Drivers exchanging trucks at country borders This can be modeled by different routes for each country and for each route to and from the country border. Flexible time windows can be used to ensure the route in the next country can only depart when the previous route arrived at the border (see also multiple routes for a single vehicle). Furthermore, state dimensions are added on time and, e.g., driving hours regulations, to ensure that each route starting at the country border is initialized correctly. As with multiple routes per vehicle, a correct order in the GTR should be preserved (i.e., the two routes *to* the border should precede the two routes *from* the border).

4.4 A case study

To demonstrate how a set of realistic constraints can be incorporated into our DP framework, we consider the rich VRP variant described in Oppen and Løkketangen [23] and show how this problem can be incorporated in our DP framework. Oppen and Løkketangen describe a VRP variant based on livestock collection in Norway. We briefly describe the livestock collection problem and show how to incorporate this problem into our DP framework.

The livestock collection problem considers the collection of two types of livestock to a slaughterhouse: bovine and pigs. This collection is done with a heterogeneous vehicle fleet, where each vehicle can make a maximum of four routes per day. The planning period is several consecutive days, and each driver starts at home each day.

Each vehicle contains three sections that need to be (un)loaded in a specific order. Each section can be divided into an upper and a lower compartment, where the upper compartment must be loaded first. Bovine can never be loaded into an upper compartment. Moreover, when bovine is loaded into the lower compartment, then it depends on the specific vehicle whether the upper compartment is available for pigs.

Due to divisions, certain vehicles may not collect at certain customers. To protect animal welfare, a constraint is added indicating that no livestock may be transported for more than eight hours continuously, effectively restricting the time from the first collection to the unloading at the slaughterhouse to eight hours. Furthermore, some customers have to be visited as the first stop or as the last stop in a route, due to protection of the livestock used for breeding purposes (first) or to avoid spreading diseases (last). Finally, inventory demands and constraints at the slaughterhouse give a minimum and maximum of both livestock that has to be delivered at the slaughterhouse, where the minimum and maximum depends on the amount of livestock delivered in the previous days.

We model the livestock collection problem to incorporate it into our DP framework as follows. For each vehicle, we create four routes for each day (possibly empty), where each route starts and ends at the slaughterhouse, except for the first route for each vehicle, which starts at the driver's home. Since the routes for a single vehicle depend on each other in time, we order them by time in the GTR. Furthermore, to incorporate the inventory constraints (which depend on what has been delivered in the previous days), we also order the routes by day in the GTR.

The inventory constraints are modeled by feasibility checks preventing to add pickups that will exceed the limit for the current day and prevent the return to the slaughterhouse of the last route of the day when the minimum inventory is not met. Since all routes of the preceding days are finished before any route of a day starts, the minimum and maximum inventory requirements for that day will be known for the feasibility checks. The amounts of livestock delivered of both types are added as state dimensions to preserve optimality.

Feasibility checks are used to ensure certain farms are visited as first or as last in a route, and feasibility checks are also used for the constraints regarding vehicle customer combinations due to the divisions. As the compartments have a specific loading order, a state dimension is added for the number of available compartments, so both used and blocked compartments can be accounted for (since in each state the loading history is precisely known, we always know whether there is a compartment available for the livestock at a particular customer). At a pickup, we do not need an extra choice regarding the compartment, as the livestock can be loaded into the first available compartment for the type of livestock. This ensures the most slack on the capacity as loading in any other available compartment will block at least all compartments now blocked or loaded. Finally, time is added as a state dimension to be able to keep track of the remaining time since the first pickup to ensure that livestock is transported continuously for at most eight hours.

5 Computational experiments

We test our framework on known benchmark instances. Since most benchmark instances in literature are proposed for classical VRPs in which only few realistic constraints are considered, they are not well suited for this study. The recently developed benchmark instances for the VRPTW with the EC social legislation on driving and working hours (VRPTW-EC) proposed by Goel [24] form an exception to these classical benchmark instances. Therefore, we selected these instances to test our solution framework. To illustrate the difference in performance of our framework on classical VRP variants instead of more realistic VRPs, we first solve the well-known Solomon [10] instances with our solution framework. Moreover, the Solomon instances are standard reference in VRP literature, whereas Goel's instances are modifications of the Solomon instances. In addition, we test our framework on the benchmarks for the PDPTW by Li and Lim [25], to illustrate the impact

of precedence relations on the performance of our solution framework. We implemented the DP framework in Delphi 7 and ran our experiments on a PC with a Core I7 Q 840, 1.87 GHz CPU and 8 GB of RAM.

5.1 VRPTW

The Solomon instances consist of 6 problem sets: in the c1 and c2 instances customer locations are clustered, in the r1 and r2 instances they are random, and in the rc1 and rc2 instances they are semi-clustered; the 2-instances have a relatively longer time horizon and larger vehicle capacities than the 1-instances allowing for larger vehicle routes in terms of number of customers. The primary objective is to minimize the number of vehicles used, the secondary objective is to minimize the total travel distance.

In order to satisfy the time window and capacity constraints, we add state dimensions t and q, indicating the departure time from the last visited customer and the remaining vehicle capacity, respectively. If states A and B visit the same customer set S and end in the same node $j \in S$, then state A dominates state B if its costs are not larger than the costs of state B, its departure time is not larger than the departure time in 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 most promising H states are maintained. To determine the most promising states, we use the following hierarchical criteria: number of vehicles used, total distance traveled, departure time from the last visited node, remaining vehicle capacity of the active vehicle. The first two criteria correspond to the objective function; the last two criteria select the most flexible solutions with respect to adding more customers (smaller departure times imply more time for visiting additional customers; larger remaining capacities imply more capacity available for visiting additional customers). We set $H = 100{,}000$ and we first set E to its maximum value n.

Table 1 presents the results for the six Solomon problem sets. The first three columns present the average number of vehicles used, the average travel distance, and the cpu time in seconds, respectively. The last two columns present the objective values for the best known solutions identified by heuristics.

On average, the DP algorithm results in 1.27 more vehicle routes than state of the art local search methods for the VRPTW. The total travel dis-

Problem	DP Algorithm			Best Known Solutions	
Set	# Veh.	Dist.	cpu (s)	# Veh.	Dist.
c1	10.00	852.71	228	10.00	828.38
c2	3.00	608.72	252	3.00	589.86
r1	14.08	1333.52	235	11.92	1205.39
r2	4.18	1111.24	274	2.73	951.91
rc1	14.13	1510.09	266	11.50	1384.16
rc2	4.25	1336.96	275	3.25	1119.35

Table 1: results for the VRPTW

tance is 11.0% larger, on average. The average computation time per problem instance is 254 seconds. Increasing H results in substantial reductions of the gaps with the best known solutions. We run our experiments also for H = 10,000 and H = 1,000,000, and it turns out that increasing H reduces the gaps in number of vehicles used and total travel distance considerably: increasing H from 10,000 to 100,000 reduces these gaps by 14.5% and 18.3%, respectively, and increasing H from 100,000 to 1,000,000 reduces these gaps by 12.7% and 20.5%, respectively. In these experiments, computation times increase (approximately) by a factor 11 when H increases by a factor 10.

Table 2 presents the impact of different E values on the performance of the algorithm. We trie to expand each state to its nearest unvisited nodes, until we have E feasible expansions, or we have tried all expansions. Since the problem includes time windows, we sort the unvisited nodes to increasing arrival times when traveling from the last visited node. In case two or more expansions result in the same arrival time, we sort them to increasing travel distance from the last visited node. We set the values of E to n, 20, 10, and 5 and present average gaps and average computation times over all problem instances.

The results indicate that decreasing E results in smaller computation times, while the solution quality is maintained or even improved. In particular, smaller values of E result in less vehicles used, but with a slightly bigger total travel distance. This is due to the primary sorting criterion (arrival time), allowing for more slack in adding stops to the same vehicle, but also allowing slightly more travel distance.

Since E = 5 results in the smallest computation times and the best objective values, we intensified the search for E = 5 with H = 1,000,000. These results are presented in Table 3. The overall gaps with the best known

E	# vehicles[a]	travel distance[b]	cpu (s)
n	1.27	11.11%	254
20	1.27	10.27%	229
10	1.20	11.11%	214
5	1.11	11.86%	197

[a]absolute average gap
[b]relative average gap

Table 2: Impact of different E values

solutions are 0.84 vehicles and $10,88\%$ travel distance, with an average computation time of 1994 seconds.

Problem	DP Algorithm			Best Known Solutions	
Set	# Veh.	Dist.	cpu (s)	# Veh.	Dist.
c1	10.00	865.12	2000	10.00	828.38
c2	3.13	615.79	1969	3.00	589.86
r1	13.58	1322.70	1986	11.92	1205.39
r2	3.64	1129.96	1971	2.73	951.91
rc1	13.00	1510.46	2021	11.50	1384.16
rc2	3.75	1289.45	2028	3.25	1119.35

Table 3: results for the VRPTW with $H = 1{,}000{,}000$ and $E = 5$

Note that, in addition to CPU time, also memory requirements could become a limit when intensifying the search with larger H values. In our implementation, we only keep two stages in memory: the current stage to be expanded and the new stage (we write the required information for backtracking the best found solution to disk). This leads to at most $2H$ stages in memory. Each state contains information on, a.o., the visited node set, the end-node, the index of the current vehicle, costs, and time. If we assume that each node, as well as the index of the current vehicle, is represented by a 4-byte integer, and the costs and time by 8-byte floats, this leads to 424 bytes per state. It requires a bit more than 20 million 424-byte states to fill 8GB of RAM. This implies that, due to memory requirements, we can increase H to approximately = 10,000,000. The estimated CPU time per problem instance, based on the results with $H = 100{,}000$ and $H = 1{,}000{,}000$, will be about 20,000 seconds for this value of H, which is not acceptable for practice.

Therefore, the computation time is the real limit in these experiments, and having a larger memory would not be benificial.

The DP algorithm cannot compete with state of the art methods for the VRPTW. For example, Pisinger and Ropke [4] attain the minimum number of vehicles for all problem instances, and the total travel distance is within 1% of the best known solutions. However, the DP framework is designed for solving realistic VRPs instead of classical VRPs. In Section 5.2, we show that the performance of the DP framework substantially improves when the problem is more restrictive and in Section 5.3, we demonstrate that the DP framework even competes with state of the art improvement methods when they are applied to more realistic VRPs.

5.2 PDPTW

To apply the DP framework to the PDPTW, we ensure that delivery nodes can only be selected when the corresponding pickup node has already been selected. Next, we only allow a vehicle to return to the depot when it is empty, i.e., when all deliveries corresponding to its pickups have been carried out. Since selecting too many pickups may result in a state with no feasible expansions (all unvisited nodes are infeasible, but returning to the depot is also not possible, since there are still deliveries to be carried out), we check for each state expansion whether each delivery that still has to be done is feasible in the next state expansion. We test the DP on the set of modified Solomon instances proposed by Li and Lim [25]. The primary objective is to minimize the number of vehicles used and the secondary objective is to minimize the total travel distance. We again select $H = 1,000,000$ and $E = 5$ for these experiments, and use the same criteria for selecting the H best states and the E nearest unvisited nodes as with the VRPTW.

Table 4 presents our results and the best known solutions for these problem instances. On average, the DP framework requires 0.32 more vehicles than the best known solutions and the travel distance is on average 6.30% larger. Therefore, the performance on these benchmarks is considerably better than on the VRPTW (where these gaps were 0.8 and 10.88%, respectively). Moreover, these results are obtained with an average computation time of 1263 seconds, against 1994 seconds for the VRPTW. The reason for this performance improvement is that the precedence relations in the PDPTW substantially reduce the state space. This indicates that the DP framework performs better on more restrictive VRP variants, making

it almost competitive with state of the art improvement methods for the PDPTW.

Problem Set	DP Algorithm # Veh.	Dist.	cpu (s)	Best Known Solutions # Veh.	Dist.
lc1	9.89	848.80	1233	9.67	847.12
lc2	3.00	589.86	1259	3.00	589.16
lr1	12.42	1255.10	1144	11.92	1219.62
lr2	3.18	1170.79	1431	2.80	994.44
lrc1	11.88	1420.21	1115	11.50	1386.74
lrc2	3.50	1265.38	1396	3.25	1133.12

Table 4: results for the PDPTW

5.3 VRPTW with EC social legislation

We further illustrate the better performance of the DP framework when more realistic restrictions are accommodated by applying the algorithm to a set of benchmark instances for a more restrictive VRP. The Solomon benchmarks have recently been adapted for the Vehicle Routing Problem with Time Windows and the European Community social legislation (VRPTW-EC). Due to the generality of this legislation (it is valid for all member countries of the European Union, and the EC social legislation is more restrictive than the US Hours-Of-Service Regulations [26]), we selected this set of problem instances as benchmarks for more realistic VRPs.

The EC social legislation poses restrictions on the amount of driving and working times before a mandatory rest is taken. The legislation considers different time horizons for scheduling breaks and rest periods: single driving periods, daily driving and working times, and weekly driving and working times. After a restricted amount of accumulated driving and working time, a break or rest of certain minimum duration must be scheduled. For a detailed description of the rules in the legislation, we refer to Kok et al. [22]. Kok et al. [22] also propose a break scheduling heuristic that schedules breaks and rest periods such that the vehicle routes satisfy the requirements posed by the EC social legislation on driving and working hours. They embed this break scheduling method in the DP framework. Goel [24] proposes the following modification of the Solomon instances for the VRPTW-EC.

Goel proposes to consider the depot opening time as a period of 144 hours, corresponding to a weekly working period, and to scale the customer time windows accordingly. Next, Goel suggests a driving speed of 5 distance units per hour, and to set all service times to 1 hour. Due to the required breaks and rest periods, it may turn out to be impossible to reach certain customers before their due dates, or the vehicle may not be able to return in time to the depot after serving a customer at his ready time. Therefore, Goel suggests to adjust such time windows in such a way that the ready time equals the earliest time the vehicle can reach the customer, and the due date is such that starting service at this due date and directly returning to the depot results in a return time at the depot's due date, respectively. Table 5 presents the results found by the DP algorithm, including the break scheduling heuristic of Kok et al. [22], for $H = 10{,}000$ and $E = n$ compared with Goel's solutions for this problem, which were obtained by a state of the art large neighborhood search algorithm.

Problem	DP Algorithm			Goel's Solutions	
Set	# Veh.	Dist.	cpu (s)	# Veh.	Dist.
c1	10.33	965.44	55	11.11	1054.45
c2	5.00	770.42	72	8.38	954.64
r1	9.67	1152.39	63	10.92	1144.23
r2	7.55	1100.83	66	10.27	1107.14
rc1	10.25	1300.60	71	11.13	1347.75
rc2	8.13	1266.64	69	10.00	1347.26

Table 5: results for the VRPTW-EC

In contrary to the classical Solomon instances, the DP algorithm significantly improves the VRPTW solutions when the EC social legislation is accommodated in the problem[2]. The number of vehicles used reduces by 1.8, on average, and the travel distance by 5.4%. The results were obtained with an average computation time of 65 seconds. A reason for this improvement is that the evaluation of neighborhood solutions with Goel's approach is much more expensive when driving hours regulations are considered. As a consequence, much fewer neighborhood solutions can be evaluated in the same

[2]Goel's results have also recently been improved by Prescott-Gagnon et al. [27], who also obtained better results than the DP algorithm (about 17% less vehicles and 13% less travel distance), but they also allowed more computation time (about 50 times). If we allow similar computation times, these gaps become 16% and 9%, respectively.

amount of computation time. In contrary, embedding the break scheduling heuristic of Kok et al. [22] in the DP framework does not increase the running time complexity of the DP algorithm, such that the DP algorithm can evaluate a similar amount of states in the same computation time when considering driving hours regulations. This illustrates the opportunities of applying the DP framework in practice: when real life restrictions are added to the problem, the performance of the DP framework improves with respect to state of the art local search methods, making it even competitive with state of the art heuristic for the VRPTW-EC.

6 Conclusions

This paper introduces a highly flexible framework based on dynamic programming for Vehicle Routing Problems that is able to model and solve different types of problems previously tackled by tailored algorithms. Moreover, this framework accommodates various real life restrictions that have been generally ignored in classical vehicle routing models, such as time-dependent travel times and driving hours regulations. We demonstrate this flexibility by a case study, in which we applied the framework to a rich VRP variant based on livestock collection. 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 tailored approach to handle it. Therefore, a general framework that is able to accommodate different types of problems offers the added benefit to handle several variants of problems which are hybrid variants 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 the state space by restricting the number of expansions of a state leading to practical and efficient polynomial construction heuristics. The quality of these heuristics for solving realistic Vehicle Routing Problems is demonstrated by significantly improving the best known solutions for the VRPTW with the EC social legislation on driving and working hours, which were obtained by a state of the art local search method. 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. We thank the anonymous referees for their helpful comments to improve this paper.

References

[1] M. Gendreau, G. Laporte, and J.-Y. Potvin. *Metaheuristics for the capacitated VRP*, pages 129–54. The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications. SIAM Publishing: Philadelphia, PA, 2002.

[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] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007. 0305-0548.

[5] C. Malandraki and R.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] Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15:333–346, December 2003.

[7] B. T. Lowerre. *The Harpy Speech Recognition System*. PhD thesis, Carnegie Mellon University, 1976.

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

[9] 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.

[10] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[11] J.E. Beasley. Route first–cluster second methods for vehicle routing. *Omega*, 11(4):403 – 408, 1983.

[12] 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.

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

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

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

[16] 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.

[17] A.L. Kok. *Congestion Avoidance and Break Scheduling within Vehicle Routing*. PhD thesis, University of Twente, School of Management and Governance, 2010.

[18] 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.

[19] ORTEC Algorithmic Research and Development Team. Flexible vehicle routing and scheduling. *ORTEC White Paper Series*, 2011.

[20] A. L. Kok, E. W. Hans, and J. M. J. Schutten. Vehicle routing under time-dependent travel times: the impact of congestion avoidance. *Beta working paper series*, 267, 2009. `http://beta.ieis.tue.nl/node/1441`.

[21] A. L. Kok, E. W. Hans, J. M. J. Schutten, and W. H. M. Zijm. A dynamic programming heuristic for vehicle routing with time-dependent travel times and required breaks. *Flexible Services and Manufacturing Journal*, 22:83–108, 2010.

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

[23] Johan Oppen and Arne Løkketangen. A tabu search approach for the livestock collection problem. *Computers & Operations Research*, 35:3213–3229, 2008.

[24] A. Goel. Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, 43(1):17–26, 2009.

[25] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. *Tools with Artificial Intelligence, Proceedings of the 13th International Conference on*, pages 160–167, 2001.

[26] Hours-Of-Service Regulations. *Federal Motor Carrier Safety Administration*, 2005. `http://www.fmcsa.dot.gov/rules-regulations/topics/hos/hos-2005.htm`.

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

- new construction framework for solving VRPs (based on dynamic programming) that can handle a wide range of different types of VRPs
- this framework accommodates various restrictions that are not considered in classical vehicle routing models, but that regularly appear in practice.
- framework competes with state of the art local search methods when more realistic constraints are considered than in classical VRPs.