

A Novel Dynamic Program and Relaxations: Dual Bounds for the Capacitated Vehicle Routing Problem

Anthony Karahalios

Naval Postgraduate School
anthony.karahalios@nps.edu

Abstract. State-of-the-art exact methods for solving vehicle routing problems rely on a dynamic program that enumerates the set of feasible routes and their distances. Using relaxations of the dynamic program improves these exact methods. This work is motivated by two observations. First, the dynamic program enumerates some routes with a suboptimal permutation of the locations. Second, the relaxations only relax the feasibility of routes and maintain their exact distances. We propose a novel dynamic program that enumerates the feasible sets of locations that can be visited on a route and associates with each set the distance of a route with an optimal permutation of the locations. We develop relaxations of our novel dynamic program that relax both the feasibility and distances of the routes. We embed the dynamic program in an arc-flow formulation and use relaxations to compute dual bounds for the capacitated vehicle routing problem. We give experimental results for instances with up to 10,000 locations and are the first to compute dual bounds for the XL set of benchmark instances.

Keywords: Dynamic Programming · Vehicle Routing · Arc-Flow Formulations

1 Introduction

Vehicle routing problems are ubiquitous in the transportation sector. There are many variants of vehicle routing problems that involve different constraints and objectives [8]. These problems are challenging to solve, and many benchmarks contain instances with at most 100 locations [13]. The capacitated vehicle routing problem is a variant that involves assigning routes to a fleet of homogeneous vehicles with fixed capacity to cover demands at a set of locations using minimum total distance [14]. The fleet begins at a depot with inventory to cover the demands and must return to the depot.

State-of-the-art solvers for the capacitated vehicle routing problem are able to solve instances with up to about 250 locations to optimality [15]. Several benchmark sets of instances are used to measure the performance of a solver. One benchmark set of instances has between 10 and 100 locations and are all solved [1]. A second benchmark set of ‘X’ instances has between 100 and 1,000 locations with many open instances [15]. Only one solver has computed dual bounds for these instances, many of which require several hours to produce an initial bound [15]. A third benchmark set of ‘XL’ instances has between 1,000 and 10,000 locations for which no solver has reported dual bounds [10].

Dynamic programming is an important part of the exact method used by state-of-the-art solvers. These solvers implement branch-and-cut-and-price, which is an integer

programming method that is based on column generation and uses dynamic programming for solving the pricing problem [9,16]. In particular, the pricing problem is solved as a resource-constrained shortest path problem via dynamic programming-based labeling algorithms [5]. The dynamic program encodes each feasible route as a *sequence* of locations that begins and ends at the depot. These routes include those that visit a subset of locations using a suboptimal permutation with respect to the distance traveled. Dominance rules are used to improve computational efficiency by reasoning about routes with suboptimal permutations [12].

Relaxations of dynamic programs are used within exact methods to improve computational efficiency [5]. In the vehicle routing context, a relaxation of a dynamic program updates the states and transitions to allow infeasible routes. The most commonly used relaxations are state-space relaxations that do not maintain the exact set of visited locations and have routes that visit a location more than once [4,2]. These relaxations can be iteratively strengthened using ideas like decremental state-space relaxations and column elimination [12,7]. The distances of the routes remain exact.

In this work, we develop a novel dynamic program for the capacitated vehicle routing problem that enumerates each feasible route as a *set* of locations and the distance of a route using an optimal permutation of the locations. We formulate relaxations of this dynamic program that also relax the routes' distances. We embed the relaxations in an integer programming formulation and extend an exact method to solve the formulation. We give experimental results that provide the first dual bounds for the 'XL' instances and in particular compute strong dual bounds for several 'XL' instances for which a state-of-the-art solver does not produce a dual bound within a 3,600 second time limit.

2 The Capacitated Vehicle Routing Problem

This section introduces definitions and notations for the capacitated vehicle routing problem. The problem is to assign routes to a fleet of vehicles to visit locations with the least total travel distance. Each route can be viewed as a set of locations to visit and a permutation of those locations [11].

We formally define the *capacitated vehicle routing problem* as follows [14]. Let $G = (V, A)$ be a complete directed graph with vertices $V = \{0, 1, \dots, n\}$ and a set of arcs $A = \{(i, j) \mid i, j \in V, i \neq j\}$. Vertex 0 represents the depot and vertices $\{1, \dots, n\}$ represent the locations to be visited. Each vertex $i \in V$ has a demand $q_i \geq 0$, and each arc $a \in A$ has a length $l_a \geq 0$. We consider the version of the problem that permits a sufficiently large fleet of homogeneous vehicles, each with capacity Q . A *route* is a sequence of vertices $[v_1, v_2, \dots, v_k = 0]$ starting with the first visited location and ending at the depot with a total demand at most Q . Let R be the set of feasible routes. The *distance* of a route is the sum of its arc lengths, i.e. $l_{0,v_1} + \sum_{i=1}^{k-1} l_{(v_i, v_{i+1})}$. Let $d : R \rightarrow \mathbb{R}$ give the distance of a route. The capacitated vehicle routing problem is to find a set of routes such that each vertex except for the depot belongs to exactly one route and the sum of the route distances is minimized.

A route consists of a set of locations and a permutation of those locations. Denote $\mathcal{L}$ as the set of feasible sets of locations that can be visited by a single vehicle. We define an *optimal route* as a route that has an optimal permutation of its set of locations

with respect to the distance traveled. Denote $R_{opt} \subseteq R$ as the set of optimal routes. The problem of finding an optimal permutation of a set of locations is equivalent to the well-known *traveling salesperson problem*. Denote TSP_L as the minimum distance of a route that visits a set of locations L .

3 Dynamic Programs for Route-Enumeration

In this section, we consider dynamic programs for enumerating a set of routes and their distances. We introduce a general definition and notation for a dynamic program. Then, we describe two dynamic programs for the capacitated vehicle routing problem. The first is the commonly used dynamic program that enumerates all feasible routes. The second is a novel dynamic program that enumerates all optimal routes.

A dynamic program can be characterized by a set of states, a set of actions, a transition function, and a cost function. Each state contains one or more pieces of information. The transition function takes as input a state with an action and outputs another state. The cost function maps each transition to a real number. The set of states contains an initial state and a terminal state. A dynamic program can be viewed as enumerating sequences of actions and accumulated costs that begin at the initial state and end at the terminal state. A solution is a sequence of actions that transition from the initial state to the terminal state.

Formally, we define a *dynamic program* by a set of states S , an initial state $r \in S$, a terminal state $t \in S$, a set of actions U , a state transition function $h : (S \times U) \rightarrow S$, a cost function $c : (S \times U) \rightarrow \mathbb{R}$. We denote $DP = (S, U, h, c)$ as a dynamic program. We assume an initial state r and terminal state t exist in S . We define a *solution* as $[u_1, \dots, u_k]$ for $k \in \mathbb{Z}$ such that $h(h(\dots h(h(r, u_1), u_2) \dots, u_{k-1}), u_k) = t$. Let $[s_1 = r, s_2, \dots, s_{k-1}, s_k = t]$ be the corresponding sequence of states that are the intermediate evaluations of the transition function for a solution. We define the *cost* of a solution as the sum of the costs of the transitions, $\sum_{i=1}^k c(s_i, u_i)$.

The set of solutions and their costs are useful to characterize a dynamic program. Denote R_{DP} as the set of solutions to DP , and denote $f_{DP} : R_{DP} \rightarrow \mathbb{R}$ as the function that maps each solution to its cost. In the following sections, a solution corresponds to a route and the cost of a solution corresponds to the distance of the route.

3.1 Sequence-Based Dynamic Program

The dynamic program that is most commonly used in previous works on the capacitated vehicle routing problem has a set of solutions that corresponds to the set of feasible routes. Each action of the dynamic program is a location and each transition is equivalent to a vehicle moving from one location to the next. We call this the *sequence-based* dynamic program because each route is described as a sequence of actions (i.e. locations). We denote the sequence-based dynamic program as $DP_{seq} = (S_{seq}, V, h_{seq}, c_{seq})$.

The sequence-based dynamic program has states that represent the location of a vehicle, the load on the vehicle, and a set of visited locations. The initial state represents an empty vehicle at the depot and transitions represent moving to the next location. The

cost of a transition is the distance between the input state's location and the location represented by the transition.

Formally, we define the sequence-based dynamic program as follows. Let each state in S_{seq} be a tuple (v, w, L) , where v is the current location, w is the current load, and L is the set of visited locations. The initial state $s_{seq}^{initial}$ is $(0, 0, \{0\})$ and the terminal state $s_{seq}^{terminal}$ is a tuple of sentinel values. Define the transition function and the cost function for two different cases. First, we define the functions for the case of including a new location on the route. For each state $s = (v, w, L)$ and action $j \in V \setminus \{0\}$ such that $j \notin L$ and $w + q_j \leq Q$, define $h_{seq}(s, j) = (j, w + q_j, L \cup \{j\})$ and $c_{seq}(s, j) = l_{(v,j)}$. Second, we define the functions for the case of completing a route by returning to the depot. For each state $s = (v, w, L)$, define $h_{seq}(s, 0) = s_{seq}^{terminal}$ and $c_{seq}(s, 0) = l_{(v,0)}$. All other transitions and costs have a sentinel value to represent infeasibility.

The set of solutions to the sequence-based dynamic program and costs are equivalent to the set of feasible routes and distances. Proposition 1 states this well-known result [3].

Proposition 1. $R_{DP_{seq}} = R$, and $f_{DP_{seq}}(r) = d(r) \quad \forall r \in R$.

3.2 Set-Based Dynamic Program

We propose a novel *set-based dynamic program* to describe the set of optimal routes and distances for the capacitated vehicle routing problem. The set-based dynamic program enumerates optimal routes instead of feasible routes by using different states and transitions compared to the sequence-based dynamic program. The term set-based dynamic program is used because the sequence of actions in a solution only implies the set of visited locations instead of the sequence of visited locations. We denote $DP_{set} = (S_{set}, V, h_{set}, c_{set})$ as the set-based dynamic program.

The set-based dynamic program has states that represent the load on the vehicle and the set of locations visited. The transition function decides whether a location is included in a route or not but does not indicate the permutation of the locations. The cost function is the difference in the distances of optimal routes for a set of locations and a set of locations including one additional location.

The transition function uses an *ordering* of the locations. Assume that the ordering is given by the indices of the vertices in V . An ordering ensures that no two solutions in the dynamic program use the same set of locations because each solution is a sequence of locations with increasing indices, until the depot location. The choice of ordering changes the number of states and transitions in a set-based dynamic program.

Formally, we define the set-based dynamic program as follows. Let each state in S_{set} be a tuple (w, L) , w is the current load and L is a set of visited locations. The initial state $s_{set}^{initial}$ is $(0, \{0\})$ and the terminal state $s_{set}^{terminal}$ is a tuple of sentinel values. Define the transition function and the cost function for two different cases. First, we define the functions for the case of including a new location on the route. For each state $s = (w, L)$ and element $j \in V \setminus \{0\}$ such that $j > \max_{v \in L}(v)$, and $w + q_j \leq Q$, define $h_{set}(s, j) = (w + q_j, L \cup \{j\})$ and $c_{set}(s, j) = \text{TSP}_{L \cup \{j\}} - \text{TSP}_L$. Second, we define the functions for the case of completing a route. For each state $s \in S_{set}$, define $h_{set}(s, 0) = s_{set}^{terminal}$ and cost $c_{set}(s, 0) = 0$. All other transitions and costs have a sentinel value to represent infeasibility.

The set of solutions to the set-based dynamic program is equivalent to the set of feasible sets of locations that can be visited by a single vehicle. The cost of the each solution is equal to the distance of an optimal route for visiting the set of locations in the solution. To show this, we slightly abuse notation and let each solution $r \in R_{DP_{set}}$ be synonymous with a set of locations $L \in \mathcal{L}$. Proposition 2 states the result.

Proposition 2. $R_{DP_{set}} = \mathcal{L}$ and $f_{DP_{set}}(L) = TSP_L \quad \forall L \in \mathcal{L}$.

The cost function requires solutions to the traveling salesperson problem which makes this dynamic program appear impractical. In the next section, we relax the cost function so that the dynamic program can be used within an exact method. The idea is that the distance of an optimal route may be well-approximated by the distance of an optimal route for visiting a subset of the locations, and solutions to the traveling salesperson problem can be efficiently computed for a small number of locations.

4 Relaxations of Dynamic Programs for Route-Enumeration

This section considers relaxations of the dynamic programs. In general, a relaxation of a dynamic program enumerates a superset of the routes and/or underapproximates the distances of the routes. The goal of such a relaxation is to compile a dynamic program that contains fewer states and feasible transitions while approximating the distances of the routes. The relaxations differ for the sequence-based dynamic program and the set-based dynamic program based on their properties.

We use the term *relaxed dynamic program* as a definition because it is based on the set of solutions and their costs, whereas a similar commonly used term state-space relaxation involves a mapping of the states. Considering the objective of the capacitated vehicle routing problem is to minimize distance, we consider a lower cost as a better one. Formally, let DP and $DP_{relaxed}$ be two dynamic programs.

Definition 1. $DP_{relaxed}$ is a relaxed dynamic program w.r.t. DP if two properties hold:

1. $R_{DP_{relaxed}} \supseteq R_{DP}$
2. $f_{DP_{relaxed}}(r) = f_{DP}(r) \quad \forall r \in R_{DP}$

4.1 Relaxations of the Sequence-Based Dynamic Program

The sequence-based dynamic program is commonly relaxed by updating the transition function to only remember a subset of the visited locations. Such a relaxation enumerates a superset of the feasible set of routes by allowing some routes to visit a location multiple times. The benefit is that the relaxed dynamic program contains fewer states and transitions.

The commonly used relaxations are state-space relaxations known as the *q-route relaxation* [5] and the *ng-route relaxation* [2]. A *q-route* relaxation alters the transition function to only remember the last q locations that have been visited. An *ng-route* relaxation alters the transition function to only keep locations in the visited set of locations if they are nearby the next location being visited.

Formally, define the *ng*-route relaxation w.r.t. the sequence-based dynamic program, denoted $DP_{seq}^{ng} = (S_{seq}^{ng}, V, h_{seq}^{ng}, c_{seq}^{ng})$, as follows. The definition of a state, the set of actions, and the cost function are equal to those in the sequence-based dynamic program. The transition function is updated for a state $s = (v, w, L)$ and $j \in V \setminus \{0\}$ as follows: $h_{seq}^{ng}(s, j) = (w + q_j, (L \cup \{j\}) \cap NG_j)$ where NG_j is a *no-good* set of locations to remember in the state for each location $j \in V$. The set of solutions becomes a superset of the feasible routes. Proposition 3 states that the relaxation is valid [2].

Proposition 3. DP_{seq}^{ng} is a relaxed dynamic program w.r.t. DP_{seq} .

4.2 Relaxations of the Set-Based Dynamic Program

We develop relaxations of the set-based dynamic program that enumerate a superset of the feasible set of routes and relax the costs of the routes. The structure of the set-based dynamic program enables the construction of different relaxations compared to the sequence-based dynamic program. The goal is to produce relaxations for large-scale instances that achieve a good tradeoff between the number of states and transitions and the approximation of the set of feasible routes and their costs.

We construct a relaxation that updates the transition function to only maintain the last included location, similar to the *q*-route relaxation with $q = 1$. This update to the transition function implies that the cost function must also be updated, which is not the case for the sequence-based dynamic program. The reason the cost function must be updated is that the set-based dynamic program has a cost function that uses the full set of visited locations L while the sequence-based dynamic program has a cost function that only relies on the last visited location. To help strengthen the relaxed cost function, we use exact transitions and costs from the initial state, exact costs for these states with one location, and relax everything else.

Formally, define the relaxed dynamic program w.r.t. the set-based dynamic program, denoted $DP_{set}^1 = (S_{set}^1, V, h_{set}^1, c_{set}^1)$, as follows. The states and set of actions are defined in the same manner as for the set-based dynamic program. The transition function is updated as follows: for each state $s = (w, L)$, $s \neq s_{set}^{initial}$ and location $j \in V \setminus \{0\}$ such that $j > \max_{v \in L}(v)$, and $w + q_j \leq Q$, define $h_{set}^1(s, j) = (w + q_j, \{j\})$. Let S_1 be the set of the initial state and all states after one transition from the initial state. The cost function is updated as follows: for any state $s \notin S_1$ and $\forall j \in V$, let $c_{set}^1(s, j) = 0$. The cost function with an input state from S_1 only requires solving the traveling salesperson problem with one or two locations which have closed-form solutions.

The relaxed dynamic program has the same set of solutions as the set-based dynamic program, but the costs of the solutions are relaxed. The set of solutions remains the same because the condition that requires a transition to be to a location larger than any location in the visited set remains the same. Proposition 4 states that the relaxation is valid.

Proposition 4. DP_{set}^1 is a relaxed dynamic program w.r.t. DP_{set} .

We further relax the dynamic program to only maintain a discretized value of the last location visited. Let $d \in \mathbb{Z}$ be the discretization value. Then, update the transition function for each state $s = (w, L)$, $s \notin S_1$ and element $j \in V \setminus \{0\}$ such that $j >$

$\max_{v \in L}(v)$, and $w + q_j \leq Q$, define $h_{set}^1(s, j) = (w + q_j, \{d \lfloor \frac{j}{d} \rfloor\})$. This type of relaxation is less amenable in the sequence-based dynamic program, because there is no condition that transitions need to be greater than the largest visited value.

For large-scale instances, the number of transitions may greatly impact the use of the dynamic program in an exact method. Let A^1 be the number of transitions for DP_{set}^1 and let A^{ng} be the number of transitions for DP_{set}^{ng} . Proposition 5 states that the relaxed dynamic program for the set-based dynamic program has a smaller upper bound on the number of transitions.

Proposition 5. A^1 is at most $\frac{|Q||V|^2}{2d} + \frac{|Q||V|}{2}$ and A^{ng} is at most $|Q||V|^2$.

Consider the following example instance of the capacitated vehicle routing problem. Each vehicle has a capacity of three. There are four locations each with demand of one at these coordinates: (13,0),(12,0),(11,0),(10,0). The depot is at location (0,0). Figure 1 shows a network representation of each type of relaxed dynamic program. The networks differ in the number of states, transitions, costs, and the encoded set of routes.

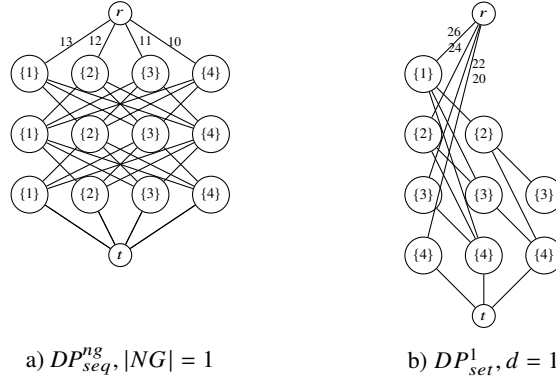


Fig. 1. The state-transition graphs for the relaxed dynamic programs. Each state shows the visited set of locations L but not the load w . To improve readability, only the costs of transitions from the initial state are written and transitions to the terminal are not included.

5 Methodology

In this section, we embed the set-based dynamic program into an integer programming formulation of the capacitated vehicle routing problem. Using a relaxed dynamic program in the formulation creates a relaxation of the integer program. Starting from a relaxed dynamic program, we show how to extend an exact method called column elimination to solve the exact formulation. This extension includes updating an algorithm within column elimination to handle the relaxed costs and to more efficiently solve the traveling salesperson problem for many different sets of locations.

5.1 Arc-Flow Formulation

An arc-flow formulation is an integer program defined over the state-transition graph of a dynamic program. The state-transition graph is a network representation of the dynamic program that has a node for each state and an arc for each transition. The integer program has variables that represent flow through the state-transition graph and has solutions that are multiple paths (i.e. routes) through the state-transition graph. Linear constraints are used to coordinate the paths. In the capacitated vehicle routing problem, they ensure the selected routes cover the locations.

We define the *state-transition graph* of a dynamic program as follows. Let $DP = (S, V, h, c)$ be a dynamic program. Denote its state-transition graph as $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ with node set $\mathcal{N}$ and arc set $\mathcal{A}$, defined as follows. For each state $s \in S$ we introduce a node in $\mathcal{N}$. For each transition $h(s_i, v) = s_j$, we define an arc in $\mathcal{A}$ from the node for s_i to the node for s_j .

The arc-flow formulation for the capacitated vehicle routing problem is given as follows [7]. There is a variable $y_a \geq 0$ for each $a \in \mathcal{A}$. The set of incoming arcs of a node u is denoted by $\delta^-(u)$. Likewise $\delta^+(u)$ denotes the set of outgoing arcs of u . We denote $\mathcal{A}^i$ as the set of arcs in $\mathcal{A}$ that transition to location i .

$$F(\mathcal{D}) : \min \sum_{a \in \mathcal{A}} c_a y_a \quad (1)$$

$$\text{s.t.} \quad \sum_{a \in \delta^-(u)} y_a - \sum_{a \in \delta^+(u)} y_a = 0 \quad \forall u \in \mathcal{N} \setminus \{r, t\} \quad (2)$$

$$\sum_{a \in \mathcal{A}^i} y_a = 1 \quad \forall i \in V \setminus \{0\} \quad (3)$$

$$y_a \in \mathbb{Z}^+ \quad \forall a \in \mathcal{A}. \quad (4)$$

The objective function (1) minimizes the sum of all arc costs. The flow conservation constraints (2) ensure that the solution is a collection of r - t paths. Constraints (3) ensure that all locations are visited once. The non-negative integrality constraints (4) complete the formulation. In the example from Figure 1, for both networks the optimal solution to the linear programming relaxation of the formulation is the instance's optimal solution.

5.2 Column Elimination

We develop a column elimination method for solving the arc-flow formulation. Column elimination was introduced using the *ng*-route relaxation of the sequence-based dynamic program. We extend column elimination to work for the set-based dynamic program, which requires handling relaxations that have a relaxed cost function.

The existing column elimination method works as follows [7]. The method begins by defining a relaxed dynamic program which translates to a relaxation of the arc-flow formulation. Then, it iterates between solving the linear programming relaxation of the arc flow formulation, decomposing the solution into a set of routes, and updating the dynamic program by removing any infeasible routes. After the set of routes is feasible, the arc-flow formulation is iteratively solved and refined as an integer program.

The refinement algorithm must be updated from the existing column elimination method to be able to handle using the set-based dynamic program. There are two key updates. First, we update the refinement algorithm in column elimination to take as input a route with relaxed cost instead of only an infeasible route. Second, we devise a method to more efficiently compute optimal solutions to the traveling salesperson problem for many similar sets of locations.

5.3 Refinement Algorithm for Relaxed Costs

We extend the refinement algorithm from the existing column elimination to work for conflicts that arise from the relaxed dynamic program's inexact cost function. The refinement algorithm in the previous work on column elimination only considers removing infeasible routes from the dynamic program relaxation. This work considers a refinement algorithm that also updates inexact costs of routes.

The existing refinement algorithm includes the creation of new states, transitions, and costs. While the algorithm is described in terms of the dynamic programs, in practice the algorithm is performed on the state-transition graph. The algorithm checks a route from the relaxed dynamic program for feasibility by considering each transition starting from the root node and checking the visited states against the states in the exact dynamic program. When a state along the route is not the exact state, the state is created, and transitions are copied from an existing state and checked for feasibility.

We update the refinement algorithm as follows. The first update is to detect solutions as either infeasible or as having inexact cost. When inspecting a solution both the exact state and the exact cost must be compared for each transition. The second update is when altering the relaxed dynamic program, the cost function is changed to the exact cost. We update the cost function to be exact for the last transition in a detected solution. We compute a simple lower bound on the exact distance for other transitions that are copied.

5.4 Solving the Traveling Salesperson Problem for Similar Sets of Locations

The Traveling Salesperson Problem must be repeatedly solved for similar sets of locations during column elimination for the set-based dynamic program. In particular, to obtain an exact transition cost, the refinement algorithm requires solving $TSP_{L \cup \{k\}} - TSP_L$ for a set of locations L and additional location $k \in V$. We devise a method to more efficiently solve an instance by reusing computation from previous instances.

Consider the problem of solving a sequence of instances of the traveling salesperson problem for T sets of locations, $\{L_0, L_1, \dots, L_T\}$. During column elimination for solving the capacitated vehicle routing problem, it is likely that these sets of locations are similar, often differing by only one location. This similarity implies that a dynamic program to enumerate the routes for each set of locations is also similar. Our idea is to also use column elimination to solve the traveling salesperson problem and to store the resulting relaxed dynamic program to be used by future iterations.

We introduce our method as follows. The method begins by compiling a relaxed dynamic program for the first set of locations L_0 and using column elimination to solve the instance. The output of this first iteration is an optimal solution to the traveling salesperson problem for L_0 and a relaxed dynamic program. The method continues

iteratively. At iteration k , it compiles a relaxed dynamic program for the set of locations L_k by building on the relaxed dynamic program that is output of column elimination for L_{k-1} . Then, the instance is solved with column elimination and the resulting dynamic program relaxation is stored for the next iteration.

The compilation of the relaxed dynamic program at iteration k works as follows. For each state s in the existing relaxed dynamic program with last visited set L_s , if $L_s \subseteq L_k$, then the state is added to a queue. Next, a relaxed set of routes is compiled using this queue of states to build routes based on the transition function and cost function of a relaxed dynamic program for L_k . There are two advantages to reusing computation in this way. First, column elimination for previous iterations might have strengthened the initial relaxed dynamic program already. Second, the compilation is more efficient because many states and transitions already exist and do not need to be recomputed.

6 Experimental Results

In this section, we give experimental results for using relaxations of the set-based dynamic program to model and solve the capacitated vehicle routing problem. We start by comparing the size and strength of the arc-flow formulation created from different initial relaxations. Then, we compare with the performance of column elimination using the set-based dynamic program, column elimination using the sequence-based dynamic program and a state-of-the-art branch-and-price solver.

For the set-based dynamic program, we order the locations from largest to smallest distance from the depot. When solving the traveling salesperson problem, we use a q -route relaxation for the sequence-based dynamic program with $q = 1$. For the sequence-based dynamic program, we always use ng -route relaxation with $|NG| = 2$. We consider all instances from the ‘X’ benchmark set [15] and the ‘XL’ benchmark set [10]. All experiments are given a timeout of 3,600 seconds and the best-known solution as input.

Each solver is run for each instance using an AMD EPYC 7662 64-Core Processor with 20GB memory as a single thread. The linear programming solver is Gurobi version 13 [6]. The software repository on Github will be made available upon publication.

6.1 Size of Arc-Flow Formulations

We evaluate the number of variables in the arc-flow formulations compiled from different relaxed dynamic programs. The number of variables affects how quickly each iteration of column elimination is solved. In particular, when the number of variables is small, the linear programming relaxation of the arc-flow formulation may be solved with a linear programming solver; if not, a specialized Lagrangian method must be used [7].

We consider the relaxed dynamic program DP_{set}^1 with different discretization values. Let d be the discretization value. Figure 2a shows a plot of the number of variables in an arc-flow formulation versus the number of locations for each instance. The values considered for d are 1, 50, 100, 500, 1000. A mark is not shown if compilation is not completed in the time limit.

The plot reveals that the number of variables greatly changes with the different discretization values. The larger values of d create fewer variables in the arc-flow

formulation. Also, compared to DP_{seq}^{ng} , the set-based relaxations are much smaller. The majority of large instances cannot be compiled with DP_{seq}^{ng} or DP_{set}^1 with $d = 1$.

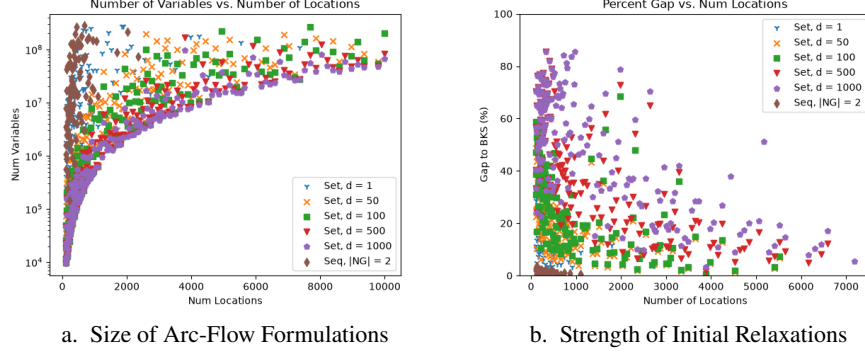


Fig. 2. a) The number of variables in the arc-flow formulation versus the number of locations for different relaxed dynamic programs. b) The percent gap to the best-known solution versus the number of locations for the linear programming relaxation of different relaxed dynamic programs.

6.2 Strength of Initial Relaxations

We analyze the strength of the dual bound produced by the *linear programming relaxation* of arc-flow formulations compiled from different relaxed dynamic programs. The goal is to create small arc-flow formulations that produce strong dual bounds, although there is usually a tradeoff between the model size and its strength. We compare this dual bound for relaxed dynamic programs of the set-based dynamic program and the sequence-based dynamic program.

We consider DP_{set}^1 with the set of values d as previously used. Define the percent gap to the best-known solution as $100(\frac{BKS-lb}{BKS})$ where BKS is the best-known solution and lb is the dual bound of the linear programming relaxation of the associated arc-flow formulation. Figure 2b shows a plot of the percent gap versus the number of locations for each instance. A mark is not shown if compilation or an initial solution are not completed in the time limit.

The plot reveals that for several ‘XL’ instances the gap is strong for DP_{set}^1 and $d = 50$ or $d = 100$. As expected, when the relaxed dynamic program is compiled within the time limit, the gap is stronger for smaller values of d . DP_{seq}^{ng} has small gaps for many of the ‘X’ instances, but does not compile for many ‘XL’ instances.

6.3 Comparison with State-of-the-Art

We compare the performance of column elimination using the set-based dynamic program against two state-of-the-art solvers: RouteOpt [16] (RO) and column elimination

using the sequence-based dynamic program [7] (SEQ+C). Both solvers use their default parameters. The column elimination solver that uses the sequence-based dynamic program uses an advanced version of column elimination involving subgradient descent that more effectively handles large-scale instances. RouteOpt only produces a bound after the termination of column generation; it is possible to compute a bound before termination, but RouteOpt does not compute it, and we do not consider it. In this section our novel method (SET+C) initializes column elimination with DP_{set}^1 and $d = 50$.

Table 1 shows the number of instances for which each of the three solvers achieve different percent gaps, partitioned by the number of locations in the instances. For instances with under 500 locations and up to 1000 locations, RouteOpt performs the best. For large-scale instances, column elimination with the set-based dynamic program outperforms both other solvers. Table 2 shows a sample of instances for which the set-based dynamic program performs well and the benefit of using column elimination. The performance is often strong for instances with short feasible routes.

Table 1. Comparing the number of instances for which each method computes a percent gap.

Gap	0 < V < 500			500 ≤ V < 1000			1000 ≤ V < 5000			5000 ≤ V < 10001		
	RO	SEQ+C	SET+C	RO	SEQ+C	SET+C	RO	SEQ+C	SET+C	RO	SEQ+C	SET+C
< 1%	58	5	5	16	4	1	1	1	2	0	0	0
< 5%	64	21	16	17	6	8	1	1	11	0	0	1
< 10%	64	21	25	17	6	17	1	1	22	0	0	2
< 20%	64	21	46	17	6	29	1	1	32	0	0	2

Table 2. A sample of instances showing the dual bounds achieved for each method. SET gives the dual bound before column elimination. ‘-’ means no value was reported within the time limit. ‘NumIt’ gives the number of iterations of column elimination for SET+C.

Instance	BKS	RO	SEQ+C	SET	SET+C	NumIt
X-n148-k46	43,448	43,448	42,937	36,296	43,105	514
X-n502-k39	69,227	69,106	68,240	61,913	64,318	118
X-n876-k59	99,300	-	-	88,843	89,485	3
XL-n2354-k631	940,635	-	-	921,168	930,059	7
XL-n5406-k783	1,040,140	-	-	1,011,000	1,011,000	1

7 Conclusion

In conclusion, we develop a novel set-based dynamic program to enumerate routes and their distances for the capacitated vehicle routing problem. We introduce relaxations

and embed them in an arc-flow formulation. We extend column elimination to handle the relaxations. We give experimental results showing strong bounds for several ‘XL’ instances. An extended version of this paper can explore using the dynamic program within column generation and other vehicle routing problems.

References

1. Augerat, P., Naddef, D., Belenguer, J.M., Benavent, E., Corberan, A., Rinaldi, G.: Computational results with a branch and cut code for the capacitated vehicle routing problem (1995)
2. Baldacci, R., Mingozzi, A., Roberti, R.: New route relaxation and pricing strategies for the vehicle routing problem. *Operations research* **59**(5), 1269–1283 (2011)
3. Christofides, N., Mingozzi, A., Toth, P.: Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming* **20**(1), 255–282 (1981)
4. Christofides, N., Mingozzi, A., Toth, P.: State-space relaxation procedures for the computation of bounds to routing problems. *Networks* **11**(2), 145–164 (1981)
5. Fukasawa, R., Longo, H., Lysgaard, J., Aragão, M.P.d., Reis, M., Uchoa, E., Werneck, R.F.: Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming* **106**(3), 491–511 (2006)
6. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2026), <https://www.gurobi.com>
7. Karahalios, A., van Hoeve, W.J.: Column elimination for capacitated vehicle routing problems. In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. pp. 35–51. Springer (2023)
8. Mor, A., Speranza, M.G.: Vehicle routing problems over time: a survey. *Annals of Operations Research* **314**(1), 255–275 (2022)
9. Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F.: A generic exact solver for vehicle routing and related problems. *Mathematical Programming* **183**(1), 483–523 (2020)
10. Queiroga, E., Martinelli, R., Subramanian, A., Uchoa, E., Vidal, T.: The xl instances for the capacitated vehicle routing problem. *arXiv preprint arXiv:2601.11467* (2026)
11. Ralphs, T.K., Kopman, L., Pulleyblank, W.R., Trotter, L.E.: On the capacitated vehicle routing problem. *Mathematical programming* **94**(2), 343–359 (2003)
12. Righini, G., Salani, M.: New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal* **51**(3), 155–170 (2008)
13. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* **35**(2), 254–265 (1987)
14. Toth, P., Vigo, D.: *Vehicle Routing: Problems, Methods, and Applications*. SIAM, second edn. (2014)
15. Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., Subramanian, A.: New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research* **257**(3), 845–858 (2017)
16. You, Z., Yang, Y.: Routeopt: An open-source modular exact solver for vehicle routing problems. *INFORMS Journal on Computing* (2026)

Appendices

A Proofs of Propositions

Proof (Proposition 1). Let $r \in R_{DP_{seq}}$ be a solution in the dynamic program. So, r is a sequence of transitions starting from the initial state and ending at the terminal state. Each transition updates the load by addition, and a transition is only feasible if adding the location's demand does not exceed the capacity. Also, the visited set ensures a location is not visited twice. Thus, the locations corresponding to the transitions in r create a feasible route in R . Therefore, $R_{DP_{seq}} \subseteq R$. The converse is also true by considering a route $r \in R$ and mapping it to transitions in $R_{DP_{seq}}$. The cost of a solution is equal to the sum of the costs of the transitions. Each transition cost is equal to the distance between the routes, so the cost of each solution is equivalent to the distance of each associated route.

Proof (Proposition 2). Let $r \in R_{DP_{seq}}$ be a solution in the dynamic program. So, r is a sequence of transitions starting from the initial state and ending at the terminal state. Each transition updates the load by addition, and a transition is only feasible if adding the location's demand does not exceed the capacity. Also, the visited set ensures a location is not visited twice. Thus, the locations corresponding to the transitions in r create a feasible set of locations in $\mathcal{L}$. Therefore, $R_{DP_{seq}} \subseteq \mathcal{L}$. The converse is also true by considering a set of locations $L \in \mathcal{L}$ and mapping it to transitions in $R_{DP_{seq}}$. The cost of a solution is equal to the sum of the costs of the transitions. Each transition cost is equal to the additional distance added to the solution of TSP for the locations, so the cost of each solution is equivalent to the distance of each associated route.

Proof (Proposition 3). The proof of $R_{DP_{seq}^{ng}} \supseteq R$ is the same as in the proof of Proposition 1. However, some routes $r \in R_{DP_{seq}^{ng}}$ visit a location multiple times because the visited set is relaxed in the transition function, so the other direction is not true. The costs of the routes $r \in R$ still holds, and the costs of the infeasible routes do not affect the correctness of the statement.

Proof (Proposition 4). The proof of $R_{DP_{set}^1} \supseteq R$ is the same as in the proof of Proposition 2. The converse is also the same, because the transition function uses an ordering to ensure that locations are not repeated in the relaxed dynamic program. The costs of the routes $r \in R_{DP_{set}^1}$ are relaxed because many transitions have a cost of 0, which is a relaxation given the assumption that the triangle inequality holds which implies that the distance of an optimal route is monotonically increasing as locations are added to a route.

Proof (Proposition 5). The number of possible states for $R_{DP_{seq}^{ng}}$ is $|V||Q|$ and each state can have at most $|V|$ transitions. So the maximum number of possible transitions is $|V|^2|Q|$. Similarly, the number of possible states for $R_{DP_{set}^1}$ is $\frac{|V||Q|}{d}$ and each state can only transition to locations with larger index in the ordering. So the maximum number of possible transitions is $|Q|(|V| + (|V| - d) + \dots + 2d + d + 0) = |Q|\frac{|V|}{2}(\frac{|V|}{d} + 1) = \frac{|Q||V|^2}{2d} + \frac{|Q||V|}{2}$.

B Column Elimination Refinement Algorithm

We give pseudocode for the updated refinement algorithm as Algorithm 1. The input to the algorithm is a dynamic program, a relaxed dynamic program, and a solution in the relaxed dynamic program. The algorithm starts with the relaxed dynamic program and makes updates to it. Line 2 begins with setting the current state to the initial state. Line 3 begins a for-loop over the transitions in the solution. Line 4 gets the next state in the relaxed dynamic program by using the solution's transition. Line 5 gets the transition cost of the relevant transition. Line 7 gets the state from the exact dynamic program, and Line 8 compares this exact state with the state in the relaxed dynamic program. Line 9 creates the exact state if it does not already exist. Line 10 copies the transitions from the state in the relaxed dynamic program, and Line 11 filters infeasible transitions and updates their costs.

Algorithm 1 Conflict Refinement Algorithm

Input: A dynamic program $P = (S, V, h, c)$, a relaxed dynamic program $P_i = (S_i, V, h_i, c_i)$ with initial state r , and a solution $x = [u_1, \dots, u_k] \in R_i$.

Output: P_{i+1}

```

1:  $P_{i+1} := (S_{i+1}, V, h_{i+1}, c_{i+1}) = (S_i, V, h_i, c_i)$ 
2: currState =  $r$ 
3: for  $j = 1, \dots, k$  do
4:   nextState =  $h_{i+1}(\text{currState}, u_j)$ 
5:   transitionCost =  $c_{i+1}(\text{currState}, u_j)$ 
6:
7:   exactNextState =  $h(\text{currState}, u_j)$ 
8:   if nextState  $\neq$  exactNextState then
9:     createExactState(exactNextState)
10:    copyTransitions(currState, exactNextState)
11:    filterTransitionsAndUpdateCosts()
12:    return
13:   end if
14:
15:   exactTransitionCost =  $c(\text{currState}, u_j)$ 
16:   if transitionCost  $\neq$  exactTransitionCost then
17:     updateCosts()
18:    return
19:   end if
20:
21:   currState = nextState
22: end for

```

C Algorithm to Iteratively Solve Traveling Salesperson Problems

The pseudocode in Algorithm 2 displays the steps for solving the traveling salesperson problem given an initial relaxed dynamic program that has been generated from previous

iterations as input. Line 1 copies the current relaxed dynamic program. Line 2 creates a queue of nodes. Line 3 begins a for-loop over the states in the relaxed dynamic program. Line 4 and Line 5 insert the states that have a set of visited locations that is a subset of the set of locations L . Line 9 begins compiling the updated relaxed dynamic program by working from the node queue. Line 10 pops a node from the queue. Line 11-14 loops over the locations, uses the relaxed transition function to produce new nodes, and inserts them into the queue if they are new. Line 19 runs column elimination to solve the traveling salesperson problem for L using the compiled relaxed dynamic program and the exact one as reference for the refinement algorithm.

Algorithm 2 Iterative-TSP-Algorithm

Input: A dynamic program $P = (S, V, h, c)$, a relaxed dynamic program $P_{relaxed} = (S_{relaxed}, V, h_{relaxed}, c_{relaxed})$, and a set of locations L .

Output: $TSP(L)$

```

1:  $P_L := (S_L, V, h_L, c_L) = (S_{relaxed}, V, h_{relaxed}, c_{relaxed})$ 
2: nodeQueue := {r}
3: for  $s = (\_, \_, L_s) \in S_{relaxed}$  do
4:   if  $L_s \subseteq L$  then
5:     nodeQueue.insert( $s$ )
6:   end if
7: end for
8:
9: while nodeQueue  $\neq \emptyset$  do
10:  node = nodeQueue.pop()
11:  for  $v \in L \cup \{0\}$  do
12:    (nextNode, isNodeNew) = transitionRelaxed( $P_{relaxed}$ , node,  $v$ )
13:    if isNodeNew then
14:      nodeQueue.insert(nextNode)
15:    end if
16:  end for
17: end while
18:
19: optValue = columnElimination( $P_L, P$ )
20: return optValue

```
