User Manual

Yutai Ke

About This Manual

This user manual accompanies the console application I developed as part of my master's project, *Enhancing Algorithm Efficiency for the Vehicle Routing Problem: A Heuristic Approach Based on Column Generation.* While the original research paper is currently under revision, this manual offers a detailed guide to the problem setup, methodology, and testing results.

Abstract

This paper addresses the computational challenges of solving the Vehicle Routing Problem (VRP) using column generation techniques. I decompose the VRP into a master problem—formulated as a set-covering problem—and a subproblem modeled as a resource-constrained shortest path problem (RCSPP). To enhance algorithm efficiency, I explore several heuristic variations of labeling algorithms for solving the RCSPP, including the elementary labeling algorithm (ELA), state-space relaxed ELA (SSRELA), K-best SSRELA, and two novel "ignoring" algorithms with and without multiplicity considerations.

Through computational experiments on generated graph instances, I analyze the performance of these algorithms in terms of runtime, number of iterations, and columns generated. A key observation is that the proportion of negative-cost edges in the graph significantly impacts the dimensionality of the state space and, consequently, the algorithm's performance. To mitigate this, I introduce a conservative multiplicity approach that stabilizes the number of vertex resources added during each iteration. This method effectively balances the exploration of the solution space while preventing an exponential increase in computational complexity.

My results demonstrate that the conservative multiplicity "ignoring" algorithm outperforms traditional methods by enhancing runtime performance and maintaining a manageable solution space. This approach facilitates the discovery of optimal or near-optimal routes without incurring excessive computational overhead. Future work will focus on analyzing memory usage and improving the quality of the primal bounds obtained.

1 Problem setup and algorithm description

The Vehicle Routing Problem (VRP) is a well-studied topic with numerous formulations. In the context of a column generation algorithm, the problem must be decomposed into a master problem, where the primary objective function is optimized, and subproblems, which generate new columns to be incorporated into the master problem. The algorithm then iterates between solving these two problems until no new variables can be generated that would improve the objective value, at which point the algorithm halts (Desaulniers et al., 2006).

For the deployment of a column generation algorithm to solve the VRP, I decompose it into a master problem, specifically a set-covering formulation of the vehicle routing problem as described in *Vehicle routing: problems, methods, and applications*(Toth and Vigo, 2014), and a subproblem, namely a resource-constrained shortest path problem adapting the setup of *A column generation approach to the heterogeneous fleet vehicle routing problem*(Choi and Tcha, 2007) . Below, I provide their exact formulations and the algorithms employed to solve these problems.

Typically, VRP problems involve a single depot where routes commence and conclude. However, in my approach, I duplicate the depot into a "starting depot" and an "ending depot" for modeling convenience, without altering the underlying setup.

1.1 Master problem: set-covering formulation of a vehicle routing problem

Suppose there are n customers indexed by $\{1, \ldots, n\}$ in a directed graph G that I aim to serve. Additionally, the graph G includes a starting depot with index 0 and an ending depot with index $n+1$.

The problem entails assigning a certain number of vehicles to serve all customers. Each vehicle must depart from the starting depot and arrive at the ending depot. Although there is no strict constraint on the number of vehicles that can be used, a "soft upper bound" is set to n, implying that each customer could potentially be served by a different vehicle. Each customer has a strictly positive service demand, and each vehicle is subject to a capacity constraint regarding this demand. All vehicles possess identical service capabilities, and by default, I set the capacity as:

$$\mathbf{max}\left\{\max(\text{All customers' demand}), \frac{6n}{4}\right\}$$

This capacity setting is an arbitrary choice to establish the problem parameters. Users can override this default by inputting a number between 0 and 1 (excluding the boundaries) to set the capacity as a proportion of the total demand of all customers, or by inputting a number greater than 1 to set a fixed capacity limit. In all cases, if the inputted number results in a capacity smaller than max(All customers' demand), the capacity is automatically scaled up to this maximum value to ensure the feasibility of the problem.

Each edge in the graph has an associated time cost, which is computed based on the Euclidean distance from the generated graph. The objective is to determine a set of routes that minimizes the total time cost while ensuring that the service capacity of each vehicle is not exceeded.

In summary, the problem can be expressed as follows:

Parameters

r: Index for routes. A feasible route must satisfy three conditions:

1. Start from the starting depot and end at the ending depot;
2. Its cost must not exceed the capacity constraint;
3. It is elementary, which means that no vertex is visited twice.

c_r: cost of route r

R: The set of all routes r

V: The set of all customer vertices.$|V| = n$.

$R_i : \{r \in R : \text{r visits vertex } i\}$

$R'_i : \{\text{subsets of } R_i\}$

Formulation

$$\min \sum_{r \in R'} c_r x_r$$

$$\text{s.t.} \quad \sum_{r \in R'_i} x_r \geq 1 \qquad \forall i \in V \quad (1)$$

$$x_r \geq 0 \qquad \forall r \in R \quad (2)$$

Variable Definition

$$x_r = \begin{cases} 1 & \text{if we use route } r \\ 0 & \text{otherwise} \end{cases}$$

The formulation can be interpreted as follows: the objective function minimizes the sum of costs associated with the active routes. There are two sets of constraints. The first set, known as the *covering constraints*, ensures that all customers are visited at least once. The number of these constraints is equal to the number of customers, with a clear one-to-one correspondence between each customer and its associated covering constraint. The second set consists of the standard non-negativity constraints for the variables x_r. It is unnecessary to include the constraint $\sum_{r \in R'_i} x_r = 1$, as visiting a customer more than once would never be optimal.

In my implementation, I utilize Gurobi to solve a continuous relaxed version of this master problem, which simply extends the domain of x_r to the non-negative real line. However, the number of routes (which corresponds to the number of variables) grows exponentially with the number of customers, so I employ a column generation algorithm to include only the relevant routes.

Nonetheless, the algorithm needs to be initialized in a feasible manner to commence its operation. Denote the initial set of routes as R_{start}. I use the following algorithm to initialize our column pool:

Algorithm 1 Initializing routes

Input: Number of customers $n > 0$
Output: A set of routes R_{start}
$R_{\text{start}} \leftarrow \emptyset$
for $i \leftarrow 1$ **to** n **do**
    Create new route r$_i$ by linking starting depot $\rightarrow$ customer i $\rightarrow$ ending depot
    R$_{\text{start}} \leftarrow R_{\text{start}} \cup r_i$
end
return R_{start}

There are at least three compelling reasons for utilizing this algorithm. First, the number of routes in the initial column pool is equal to the number of customers, ensuring a manageable number of routes. Second, the routes are guaranteed to be feasible; if a route r_i for customer i is infeasible, it indicates that there is no possible way to serve this customer without exceeding the capacity constraint, thereby rendering the problem itself infeasible. Third, all customers are guaranteed to be served, ensuring a complete and viable initial solution.

After replacing R' with R_{start}, we can initiate the first iteration of the algorithm. To generate additional routes (columns) that will enhance the solution, we proceed to the subproblem, which is modeled as a resource-constrained shortest path problem.

1.2 Subproblem: resource-constrained shortest path problem

We now have our master problem initialized and solved for the first time (which solution should be $x_r = 1 \forall r \in \{1, .., n\}$ with an "optimal" objective value of $\sum_{r \in \{1,..,n\}} c_r$), we need to find new variables(columns) to further improve our solution. To do this, we will need to start from analysiing the dual problem of the master problem.

Let $\pi_i \forall i \in V$ denote the dual variables associated with the covering constraints. Following the intuition of Lahaie (2008), we derive the dual problem as the following:

Formulation

$$\max \sum_{i \in V} \pi_i$$
$$\text{s.t.} \quad c_r - \sum_{i \in V} \delta_{ir} \pi_i \geq 0 \quad \forall r \in R$$
$$\pi_i \geq 0 \quad \forall i \in V$$

Variable definition

$$\delta_{ir} = \begin{cases} 1 & \text{if route } r \text{ passes customer i} \\ 0 & \text{otherwise} \end{cases}$$

Now we are looking for routes r such that $c_r - \sum_{i \in V} \delta_{ir} \pi_i < 0$ because we know that following the intuition of the Dantzig-Wolfe decomposition (Ralphs and Galati, 2010) these routes can improve the solution.

To find such route, the problem can be remodeled as the following:

Formulation

$$\min_{r \in R} \quad c_r - \sum_{i \in V} \delta_{ir} \pi_i$$
$$\text{s.t.} \quad \delta_{ir} \in \{1, 0\}$$

And to solve this problem, we consider this as a resource-constrained shortest path problem(RCSPP) which goal is to find the shortest path from the starting depot to the ending depot after modifying the original graph with the following algorithm:

Algorithm 2 Modify graph

Input: A directed graph G with edge set E & A set of dual multipliers $\{\pi_i : i \in \{1, ..., n\}\}$
Output: An updated graph $G^{'}$
$G^{'} \leftarrow G$
for $i \leftarrow 1$ **to** n **do**
  $E_i^{'} \leftarrow$ set of arcs in $E^{'}$ that ends at vertex(customer) i
  for $e_i^{'} \in E_i^{'}$ **do**
    $c_i^{'} \leftarrow$ cost of edge $e_i^{'}$
    $c_i^{'} \leftarrow c_i^{'} - \pi_i$
  end
end
return $G^{'}$

After this modification, we can solve the RCSPP and check if the path find has a negative cost. If yes, then we have found a new route that can reduce the total cost, hence we add this route back to the master problem and resolve it.

We will repeat this master problem-subproblem loop until no columns with negative cost can be found in the subproblem, which indicates that we have solved the continuous-relaxed VRP to optimal, then, we can use the columns generated to solve a binary integer programming problem to get a primal bound.

1.3 Algorithms to solve the subproblem

However, this is just the starting point of my analysis. Solving the master problem is not particularly difficult, as it is simply a continuous linear programming problem. Therefore, to improve the efficiency of solving this problem, it is crucial to find an effective method for solving the subproblem and generating new columns.

To solve the RCSPP problem, we use a label-setting algorithm based on dynamic programming, where paths are represented by labels that contain sets of resources. Irnich and Desaulniers (2005) provides an excellent survey on this type of algorithm. For a detailed explanation, refer to their work; here, we will provide a brief summary tailored to our specific case.

In the document of the boost graph library in C++, Drexl (2006) described the algorithm as the following:

> "The (RCSPP)functions are an implementation of a priority-queue-based label-setting algorithm. At each iteration, the algorithm picks a label l from a priority queue (the set of unprocessed labels) and checks it for dominance against the current set of labels at the vertex i where l resides. If l is dominated by some other label residing at i, l is deleted from the set of labels residing at i. If l is undominated, it is extended along all out-edges of i. Each resulting new label is checked for feasibility, and if it is feasible, it is added to the set of unprocessed labels and to the set of labels residing at the respective successor vertex of i. If a new label is not feasible, it is discarded. The algorithm stops when there are no more unprocessed labels. It then checks whether the destination vertex could be reached (which may not be the case even for a strongly connected graph because of the resource constraints), constructs one or all Pareto-optimal (i.e., undominated) s-t-path(s) and returns. "

The concept of dominance mentioned above is a widely used term when describing labeling algorithm Boland et al. (2006), so instead of a rigorous mathematic definition I would like to provide a short but easy-to-understand interpretation: label l_a dominates label l_b if and only if l_a is as good as l_b in all aspects and at least strictly better than l_b in one aspect.

At the starting point of the algorithm, a single label l is placed on the starting depot. The resource vector of said label l is initialized by a vector of 0s, indicating that no resource has been spent.When extending a label from one vertex to another, a cost associated with the arc must be paid. Following the intuition of Beasley and Christofides (1989), where they added an extra resource for each customer, in our version of the label-setting algorithm, we define a vector of resource costs for each arc e_{ij} in the graph G'. This vector has a length of $n + 2$ and is structured as follows:

$$\{d_j, j, 0, 0, 0, ..., 1, ...0, 0\}$$

where:

d_j : service demand of customer$j$

i : this resource is recording the index of the last vertex we visited. Once we extend the label l awayfrom vertex j, this index will replace the original one in label l This way, we can prevent loops with only two vertices

$\{0, 0, 0, .., 1, ..0, 0\}$: This is a binary vector in which only the jth component is 1, other components are all zeros. This is an indicator of "if customer j has already been visited". Consequently, we can only visit j if the vertex j has never been visited before.

As per the above explanation, all non-dominated routes are retained. Therefore, we can consider all non-dominated routes with negative costs at the ending depot, rather than just selecting the single "shortest path".

I will consider the basic elementary labeling algorithm and its four variations, which are described below. The first 2 algorithms are taken directly out of past literature (Beasley and Christofides, 1989), and the last 3 haven been modified in someways. These variations are closely related, and the development of each one is often based on the results observed from the previous version.

Algorithm 3 Elementary labeling algorithm(ELA)

Input: A graph G
Output: A set of columns and related solution
Initialize routes by using Algorithm 1.
while *new columns can still be found* **do**
    Solve the master problem with Gurobi.
    Adjust the edge costs with Algorithm 2.
    Solve the subproblem with the basic elementary labeling algorithm.
    Take all non-dominated paths with negative cost as new columns.
end
With the columns generated, solve an integer version of the problem to get a primal bound.

Algorithm 4 State-Space relaxed ELA(SSRELA)

Input: A graph G
Output: A set of columns and related results
Initialize routes by using Algorithm 1.
while *new columns can still be generated* **do**
  Solve the master problem with Gurobi.
  Adjust the edge cost with Algorithm 2.
  Solve the state-space relaxation RCSPP.
  while *there exists non-elementary paths* **do**
    Add resources for duplicated vertices.
    Resolve the RCSPP.
  end
  Take all paths with negative cost.
end
With the columns generated, solve an integer version of the problem to get a primal bound.

The "state-space relaxed ELA" refering to the idea of Kohl (1995), in which we start from adding no resource for vertices and incrementally add node resources until all paths with negative costs are elementary. The node resources added in one iteration will not be carry over to the next iteration.

Algorithm 5 K-best SSRELA

Input: A graph G
Output: A set of columns and related results
Note: This algorithm is the same as Algorithm 4, but when adding new columns, only the k routes with the least cost are taken.
Initialize routes by using Algorithm 1.
while *new columns can still be generated* **do**
  Solve the master problem with Gurobi.
  Adjust the edge cost with Algorithm 2.
  Solve the state-space relaxation RCSPP.
  while *there exists non-elementary paths* **do**
    Add resources for duplicated vertices.
    Resolve the RCSPP.
  end
  Take the k routes with the least cost as new columns.
end
With the columns generated, solve an integer version of the problem to get a primal bound.

Algorithm 6 "Ignoring" algorithm

Input: A graph G
Output: A set of columns and related results
Initialize routes by using Algorithm 1.
while *true* **do**
  Solve the master problem with Gurobi.
  Adjust the edge cost with Algorithm 2.
  Solve the state-space relaxation RCSPP.
  if *we found elementary routes with negative cost* **then**
    Take these routes as new columns.
    continue
  end
  else
    Add resources for duplicated vertices.
    if *no duplicated vertices found* **then**
      break
    end
    Resolve the RCSPP.
    Go back to the start of the first *if* statement.
  end
end
With the columns generated, solve an integer version of the problem to get a primal bound.

I call algorithm 6 the "ignoring" algorithm because instead of solving the subproblem to optimal at each iteration, we simply "ignore" any non-elementary paths at the ending depot as long as we can find elementary paths with negative cost. The idea is to generate new columns as fast as possible, instead of exploiting the solution space at each iteration. This modification can drastically improve the efficiency.

Algorithm 7 "Ignoring" algorithm with multiplicity

Input: A graph G
Output: A set of columns and related results
Note: This algorithm has the same setup as Algorithm 6, but when adding resources for duplicated vertices, only add the resource for the vertex that has the highest multiplicity.

In Algorithm 7 the concept of multiplicity is inspired by a similar concept in Boland et al. (2006). Here we define the multiplicity as: "the number of times a vertex i appears as a duplicated vertex in all non-dominated routes with negative cost".

2 Algorithm efficiency testing

We have conducted computational experiments on our algorithms to evaluate their performance. The test graph instances are generated using the following method: for each different number of customers, we will use a set of fixed seeds to generate 10 graphs. Our focus is on testing the execution efficiency of the different algorithm variations. For each algorithm, we will report relevant statistics and provide interpretations of these results. The programming language used for the tests is C++. Since we are utilizing GUROBI to solve the master problem, which is a continuous linear programming problem, our main concern is the method used to solve the subproblem. Consequently, the following subsections will be named based on the subproblem-solving method.

2.1 Results for the elementary labeling algorithm(ELA)

This algorithm is the baseline algorithm we are considering. For this algorithm, the statistics we want to report are run time, number of iterations run, and number of columns generated. The reulsts are presented in table 1 and table 2.

From the summary statistics reported in Table 2, we can reach these conclusions: first, the runtime of the algorithm seems to grow exponentially with the number of customer vertices in the graph. This might be related to the fact that increasing the number of customers increases the dimension of the vector of resources considered in the labeling algorithm, as well as the number of potential routes also grows exponentially. Second, the same thing is happening for the number of columns generated, and we can infer that this is mainly due to the fact that having more vertices in the graph increases the complexity of the graph itself since we are generating arcs between all possible pairs of nodes. Third, in all 10 instances and across all numbers of customers reported, the algorithm terminates after two iterations. Our explanation for this phenomenon is that the way we are initializing the column pool is really bad, even a slight change in the routes can result in a better solution, hence our algorithm is exploring the solution space in a maybe unnecessarily excessive way. With the huge number of columns found, the problem easily arrives at the optimal in the second iteration.

2.2 Results for the SRELA

This algorithm is an extension of the baseline algorithm and aims to improve performance by adding node resources only for those customers that are repeatedly visited in negative-cost paths, as described in Kohl (1995). For this algorithm, we will report the previously considered statistics, as well as the average number of node resources added across all iterations and the maximum number of resources added in a single iteration. Since the runtime difference between 7 and 12 customers is quite marginal, we will now only report results for graphs with more than 12 customers. The results are presented in Tables 3 and 4.

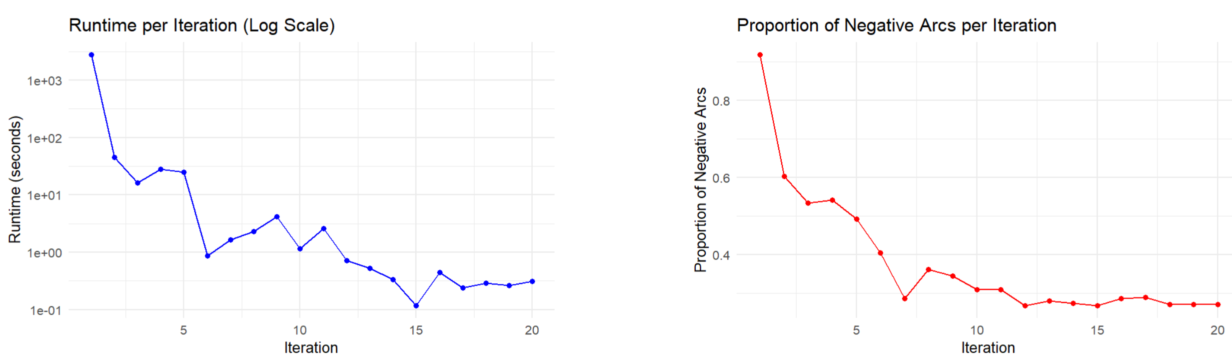
From Table 4, we observe a marginal improvement in the runtime for graph instances with 12 customers. This improvement is due to the fact that the maximum number of resources added to the algorithm is, on average, close to 8, indicating a successful reduction in the size of the state space. However, for instances with 16 customers, the efficiency actually decreases. Although this result might seem counterintuitive, the recorded data provides an explanation: in all 10 instances with 16 customers, node resources were added for all 16 customers during the first iteration. This causes our state-space relaxation of the ELA to converge to the non-relaxed form. Consequently, the actual time used includes both the "time to solve the baseline ELA" and the "time to repeatedly solve the state-space relaxed ELA to add new node resources", which is significantly longer than solving the problem once with the baseline ELA.

2.3 Results for the K-best SSRELA

In the first algorithm, we observe that the number of columns generated, even for relatively small graphs, can be enormous, exploring a vast chunk of the solution space. Although GUROBI's outstanding solving power allows it to handle LPs with tens of thousands of variables effectively, we still need to consider whether such a large number of columns is truly necessary.

The results for the second algorithm also raise some interesting questions. For small graph sizes, the SSRELA appears to effectively reduce the state space. However, as the number of customers increases, the graph's complexity also grows. In such cases, we need to add node resources for all customers to avoid non-elementary paths, causing the algorithm to terminate in just 2 iterations, just like what happened in the baseline ELA. This prevents us from observing the dynamic process of the algorithm.

To address these two problems, we propose a new variation of the algorithm that does the following: while maintaining the same setup as the previous algorithm, it returns at most k "best" routes in each iteration, selecting the elementary routes with the lowest associated costs. We do not expect an improvement in efficiency, as the basic setup remains unchanged. Instead, the purpose of this algorithm is to shed light on the evolution process of the algorithm. To achieve this goal, I will present two plots: one showing the runtime of each iteration on a logarithmic scale, and the other showing the proportion of negative edges in the updated graph G'. The results are displayed in Figure 1.



(a) KBest Runtime

(b) KBest Negative Arcs

Figure 1: Comparison of Runtime(log-scaled) VS Propotion of Negative Arcs

The plots indicate that both statistics decrease exponentially as the iterations progress. This phenomenon suggests a positive correlation between runtime and the proportion of negative-cost edges in the graph.

There are two potential ways to further improve the efficiency of the algorithms. The first approach is to use a different variation of the labeling algorithm to enhance runtime performance given the same number of negative-weight edges in the graph. The second approach is to identify new routes more efficiently by using a heuristic approach since finding the route with the most reduced cost is not required for every iteartionMehrotra and Trick (2007). This way, we can improve the solution even without modifying the labeling algorithm itself. In this paper, we will adopt the second approach.

2.4 Results for the "ignoring" algorithm

As we observed earlier, adding new improving routes (columns) to the master problem can significantly reduce the time spent on solving the subproblem. This is the key reason for using this variation. We call this variation the "ignoring" algorithm because it ignores the existence of non-elementary paths with negative costs as long as there are non-dominated elementary paths with negative costs at the ending depot. Instead of solving the subproblem to optimality at each iteration, we proceed to the next iteration as soon as we find paths that can improve the solution. We will only add node resources when no elementary negative-cost edges can be found with the "ignoring" method to ensure that the master problem is solved to optimal in our algorithm.

This time, because the algorithm performs so well empirically, I will start by reporting the test results for graph instances with 16 customers. The statistics reported for each graph instance include runtime, number of iterations, number of columns generated, and average number of node resources added. The results are shown in Tables 5 and 6.

From these two tables, we can observe the following changes: First, the efficiency in terms of runtime is drastically improved compared to all the algorithms we considered before. Solving a graph instance with 16 customers now takes, on average, less than 2 seconds, compared to the nearly one hundred seconds observed with the ELA. Second, the number of columns generated is also reduced, indicating that we are exploring a smaller portion of the solution space. Third, the algorithm no longer terminates after 2 iterations; it requires more iterations to solve the problem. However, this is not problematic as long as each iteration can be solved in a relatively short time. Nonetheless, when the number of customer vertices increases to 20, we observe that more node resources are added on average in each iteration, which again slows down the algorithm. How can we improve this?

2.5 Result for the "Ignoring" algorithm with multiplicity

As demonstrated previously, when the number of customers increases to 20, the complexity of the graph further increases, and consequently, the average number of resources added also rises.

To further improve our algorithm's efficiency, we will introduce a concept inspired by Boland et al. (2006) called "multiplicity." In their paper, they define this concept as the number of times a node V appears in a certain path p. However, in our context, we define the multiplicity of a node V as the number of times it appears as a repeatedly visited vertex in all non-elementary paths with negative costs at the ending depot. The results are shown in Tables 7 and 8. By adding node resources only for this "most represented customer," we aim to avoid adding too many resources in a single iteration, thereby making the algorithm more stable and efficient. We will compare this algorithm to the previous one to determine if there is any improvement.

First, let us take a look at the graph instances that have 20 customers. We can observe from the result presented that the runtime seems to receive a marginal improvement on average, and the standard deviation is also smaller since we are using this conservative way of adding resources for nodes. In terms of the columns generated, this algorithm is quite similar to the previous one, indicating that they might be exploring a very similar decision space. Another interesting finding is that iterations run are on average similar, but the avereage numebr of nodes added seems to be halved. This suggests that we have successfully avoided adding too much node resources in this case.

Now let us examine the graph instances with 24 customers. The efficiency drastically decreases, with longer runtimes and more iterations required. However, the maximum number of node resources added remains similar, indicating that our method has successfully stabilized the dimension of the state space even when the number of customers in the graph grows. The extra number of iterations may be the "price" we are paying for this stability. This interpretation aligns with the results for the ELA algorithms, where we include node resources for all customers, and the algorithm consistently terminates after the second iteration.

3 Conclusions and Future Work

In this project, I have deployed a series of deterministic heuristic methods based on the column generation algorithm to solve the Vehicle Routing Problem (VRP), with a primary focus on addressing the subproblem, which is modeled as a resource-constrained shortest path problem. A key observation uncovered during this study is that the proportion of negative edges in the graph is a crucial statistic when employing labeling algorithms to solve the shortest path problem, as it is highly related to the dimensionality of the state space. Specifically, a higher proportion of negative edges significantly increases the complexity of the problem by expanding the state space, which in turn adversely affects the performance and scalability of the algorithm.

To mitigate this challenge, I aimed to stabilize the number of vertex resources added in each iteration, thereby limiting the growth of the state space and maintaining computational efficiency. This stabilization was achieved through meticulous management of resource vectors and the implementation of strategic pruning techniques. As a result, the conservative multiplicity approach emerged as the most effective variation tested, effectively balancing the need to explore a sufficient number of routes while preventing an exponential increase in the number of states. This approach not only enhanced the algorithm's runtime performance but also ensured that the solution space remained manageable, facilitating the discovery of optimal or near-optimal routes without incurring excessive computational overhead.

In future work, I intend to address the following issues:

1. **Memory Usage**: While the current focus has been on runtime efficiency, memory usage remains a critical metric in modern computer science. I plan to include an analysis of memory consumption in subsequent studies.

2. **Bound Improvement**: Although the efficiency of the algorithms has been significantly enhanced, there has not been a substantial improvement in the primal bound. This outcome is not surprising, as the reduction in runtime is achieved by exploring a smaller portion of the solution space, which can lead to undiscovered optimal routes. The continuous relaxation of the VRP is guaranteed to be solved optimally; however, different algorithms explore different solution spaces, resulting in the "quality" of the generated columns being somewhat random. I aim to investigate this further to enhance the quality of the algorithms.

Table 1: Test Results for the ELA

Number of Customers	Runtime (seconds)									
	1	2	3	4	5	6	7	8	9	10
7	0.015	0.015	0.019	0.024	0.057	0.018	0.045	0.088	0.011	0.047
12	2.263	2.443	0.695	0.804	0.482	3.465	1.750	1.323	0.330	0.761
16	18.485	39.356	20.635	114.606	754.298	328.890	12.860	558.779	80.296	174.006
Number of Customers	Number of Columns Generated									
	1	2	3	4	5	6	7	8	9	10
7	14	14	26	27	69	25	63	128	12	70
12	2215	2822	1014	1040	694	3534	1953	1379	409	1085
16	9032	13387	10756	21577	34870	26902	8138	33821	17308	25860
Number of Customers	Number of Total Iterations									
	1	2	3	4	5	6	7	8	9	10
7	2	2	2	2	2	2	2	2	2	2
12	2	2	2	2	2	2	2	2	2	2
16	2	2	2	2	2	2	2	2	2	2

Table 2: Summary Statistics for the Test Results of ELA

Number of Customers	Metric	Median	Mean	SD
7	Runtime (seconds)	0.0215	0.0339	0.02487
	Columns Generated	26.5	44.8	37.2821
	Iterations	2	2	0
12	Runtime (seconds)	1.0635	1.4316	1.0254
	Columns Generated	1232	1614.5	996.9258
	Iterations	2	2	0
16	Runtime (seconds)	97.451	210.2211	258.1517
	Columns Generated	19442.5	20165.1	9965.2882
	Iterations	2	2	0

Table 3: Results for the SSRELA

Number of Customers	Runtime (seconds)									
	1	2	3	4	5	6	7	8	9	10
12	0.885	1.663	0.09	0.167	0.118	2.741	1.003	0.373	0.06	0.145
16	37.286	128.194	46.927	272.088	2017.06	808.648	24.695	993.18	185.762	556.049
Number of Customers	Number of Columns Generated									
	1	2	3	4	5	6	7	8	9	10
12	1503	2588	763	712	447	2730	1640	1254	329	744
16	9032	13387	10756	21577	34870	26902	8138	33821	17308	25860
Number of Customers	Number of Iterations									
	1	2	3	4	5	6	7	8	9	10
12	2	2	3	3	4	2	3	3	4	4
16	2	2	2	2	2	2	2	2	2	2
Number of Customers	Average Number of Node Resources Added									
	1	2	3	4	5	6	7	8	9	10
12	4.5	6	10/3	8/3	7/4	5	3	14/3	6/4	7/4
16	9	8	11	19/2	9	17/2	11	17/2	8	19/2
Number of Customers	Max number of node resources									
	1	2	3	4	5	6	7	8	9	10
12	9	10	7	8	7	9	9	9	6	7
16	16	16	16	16	16	16	16	16	16	16

Table 4: Summary Statistics for the results of SSRELA

Number of Customers	Metric	Median	Mean	SD
12	Runtime (seconds)	0.27	0.7245	0.8839
	Columns Generated	1008.5	1271	847.7459
	Iterations	3	3	0.8165
	Avg. Node Resources	3.1667	3.4167	1.5595
	Max Resources	8.5	8.1	1.2867
16	Runtime (seconds)	228.925	506.9889	630.4176
	Columns Generated	19442.5	20165.1	9965.2883
	Iterations	2	2	0
	Avg. Node Resources Added	9	9.2	1.0852
	Max Resources	16	16	0

Table 5: Results for the "Ignoring" Algorithm

Number of Customers	Runtime (seconds)									
	1	2	3	4	5	6	7	8	9	10
16	0.834	0.729	1.577	1.178	4.754	2.948	1.102	2.041	1.179	1.812
20	54.653	84.769	107.098	83.89	9.393	5.277	30.409	102.281	59.457	5.169
Number of Customers	Number of Columns Generated									
	1	2	3	4	5	6	7	8	9	10
16	406	529	438	539	724	644	385	649	556	514
20	1090	1509	1842	1401	1016	895	1335	1156	1143	695
Number of Customers	Number of Iterations									
	1	2	3	4	5	6	7	8	9	10
16	13	11	15	9	18	12	14	14	12	12
20	18	20	22	23	19	12	20	25	17	15
Number of Customers	Average Number of Node Resources Added									
	1	2	3	4	5	6	7	8	9	10
16	5/13	0	1	5/9	5/18	1/12	16/14	2/14	0	6/12
20	32/18	33/20	33/22	1	11/19	0	5/20	19/25	44/17	17/15

Table 6: Summary Statistics for the results of the ”Ignoring” Algorithm

Number of Customers	Metric	Median	Mean	SD
16	Runtime (seconds)	1.378	1.8154	1.2243
	Columns Generated	534	538.4	110.5372
	Iterations	12.5	13	2.4495
	Avg. Node Resources Added	0.3312	0.4087	0.4010
20	Runtime (seconds)	57.055	54.2396	39.8956
	Columns Generated	1149.5	1208.2	327.4123
	Iterations	19.5	19.1	3.8427
	Avg. Node Resources Added	1.0667	1.1238	0.7790

Table 7: Result of the ”Ignoring” algorithm with multiplicity

Number of Customers	Runtime (seconds)									
	1	2	3	4	5	6	7	8	9	10
20	43.661	37.408	70.536	85.922	9.918	5.353	31.253	102.858	77.174	8.351
24	1635.71	63.777	170.607	449.068	346.311	435.662	3858.53	769.326	1068.3	245.196
Number of Customers	Number of Columns Generated									
	1	2	3	4	5	6	7	8	9	10
20	1078	1509	1389	1364	1016	895	1334	1153	1090	693
24	2482	1780	1441	2263	1841	2265	1891	2666	2460	1857
Number of Customers	Number of Iterations									
	1	2	3	4	5	6	7	8	9	10
20	24	20	23	24	19	12	20	29	27	18
24	41	17	39	25	52	38	77	32	63	30
Number of Customers	Average Number of Node Resources Added									
	1	2	3	4	5	6	7	8	9	10
20	20/24	13/20	10/23	7/24	6/19	0	4/20	17/20	40/17	11/18
24	39/41	4/17	22/39	3/25	26/52	18/38	157/77	12/32	124/63	13/30
Number of Customers	Maximum Number of Node Resources Added in One Iteration									
	1	2	3	4	5	6	7	8	9	10
20	6	7	7	2	4	0	3	5	9	8
24	5	4	4	2	5	6	10	4	10	6

Table 8: Summary Statistics for the results of the ”Ignoring” algorithm with multiplicity

Number of Customers	Metric	Median	Mean	SD
20	Runtime (seconds)	40.5345	47.2434	35.0406
	Columns Generated	1121.5	1152.1	250.8287
	Iterations	21.5	21.6	4.881
	Avg. Node Resources Added	0.522947	0.653962	0.657017
	Max Node Resources Added	5.5	5.1	2.846
24	Runtime (seconds)	442.365	904.2487	1140.5243
	Columns Generated	2077.0	2094.6	388.1111
	Iterations	38.5	41.4	18.094
	Avg. Node Resources Added	0.4868	0.7660	0.6879
	Max Node Resources Added	5.0	5.6	2.5905

References

Beasley, J. E., & Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, *19*(4), 379–394.

Boland, N., Dethridge, J., & Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, *34*(1), 58–68.

Choi, E.-S., & Tcha, D. W. (2007). A column generation approach to the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, *34*(7), 2080–2095. https://doi.org/10.1016/j.cor.2005.08.001

Desaulniers, G., Desrosiers, J., & (Eds.), M. M. S. (2006, July). Column generation [Vol. 5].

Drexl, M. (2006). Boost graph library: R_c_shortest_paths [Accessed: 2024-06-05]. *Boost.org*. https://www.boost.org/doc/libs/1_85_0/libs/graph/doc/r_c_shortest_paths.html

Irnich, S., & Desaulniers, G. (2005). Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, & M. Solomon (Eds.), *Column generation* (pp. 33–65). Springer US.

Kohl, N. (1995). *Exact methods for time constrained routing and related scheduling problems* (Publication No. IMM-PHD-1995-16) [Ph.D. dissertation]. Technical University of Denmark.

Lahaie, S. (2008). How to take the dual of a linear program [Lecture notes].

Mehrotra, A., & Trick, M. A. (2007). A branch-and-price approach for graph multi-coloring. In *Extending the horizons: Advances in computing, optimization, and decision technologies* (pp. 15–29). Springer.

Ralphs, T. K., & Galati, M. (2010). Decomposition methods. In *Encyclopedia of operations research and management science*.

Toth, P., & Vigo, D. (2014). Vehicle routing: Problems, methods, and applications [Book].