1 Results

I will conduct computational experiments on the algorithms to evaluate their performance. The test graph instances are generated using the following method: for each different number of customers, I will use a set of fixed seeds to generate 10 graphs. My focus is on testing the execution efficiency of the different algorithm variations. For each algorithm, I will report relevant statistics and provide interpretations of these results. The programming language used for the tests is C++. Since I am utilizing GUROBI to solve the master problem, which is a continuous linear programming problem, my main concern is the method used to solve the subproblem. Consequently, the entire algorithm will be named based on the subproblem-solving method. Most tables and graphs are in the end of the document due to their size.

1.1 Result for the elementary labeling algorithm(ELA)

This algorithm is the baseline algorithm I am considering. For this algorithm, the statistics I 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. 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.

1.2 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.

1.3 Results for the K-best SSRELA

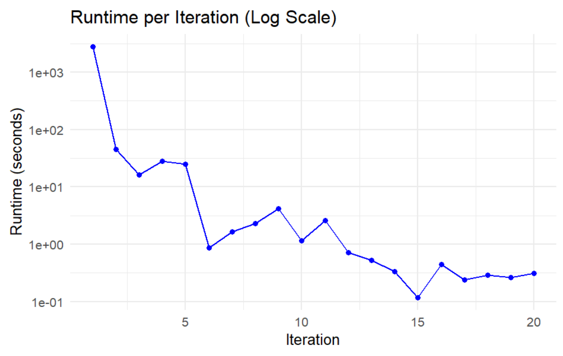
In the first algorithm, we observe that the number of columns generated, even for relatively small graphs, can be enormous, exploring a vast chunk of the solution space. Although GUROBI's outstanding solving power allows it to handle LPs with tens of thousands of variables effectively, we still need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

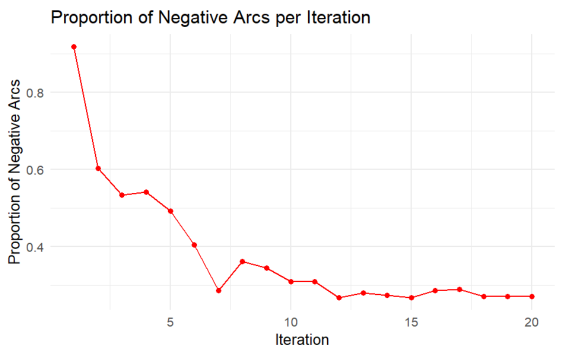
To address these two problems, we propose a new variation of the algorithm that does the following: while maintaining the same setup as the previous algorithm, it returns at most k "best" routes in each iteration, selecting the elementary routes with the lowest associated costs. We do not expect an improvement 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.

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.



(a) KBest Runtime



(b) KBest Negative Arcs

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

1.4 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?

1.5 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.

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

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.

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.

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.