Empirical Study of the Vehicle Routing Problem with Time Windows

Using a Large Neighbourhood Search algorithm for Vehicle Routing Problems with Time Windows to Show the Importance of Empirical Research in the Operations Research Field

Jacco van Wijk, 6599435

Supervision: Tomas Klos

Second Reader: Francisca Pessanha

Bachelor Kunstmatige Intelligentie, UU

15 EC

February 5, 2021

Abstract

In this paper, I try to show the value of using empirical research for problems in the Operations Research field. This is done using the Vehicle Routing Problem with Time Windows, which consists of finding the best set of routes past all customers, with the goal to minimise the cost of all the routes. To find solutions for this problem, the Large Neighbourhood Search algorithm is used. A characteristic of the algorithm is highlighted, after which a hypothesis is created. This hypothesis tried to explain the behaviour of the algorithm in relation to the characteristic. An empirical study is used to refute this hypothesis and the gained understanding of the algorithm is highlighted.

Contents

1 Introduction

The COVID-19 crisis has affected a lot of companies in a lot of different branches worldwide. For instance, 96% of the dutch entrepreneurs in the catering industry have been affected negatively by the COVID-19 crisis (Dutch Ministry of Social Affairs & Employment et al., 2020). However, not all companies are affected negatively. For instance, I have a part-time job as a grocery delivery employee for a company called Picnic, where I noticed a steady increase in customers and orders. Because of this increase, I think it has become even more important to optimise the time I spend delivering the groceries. That way I can deliver more groceries to meet the high demand even better. In this job, I have control over where I park my car, minimising the distance to the door to optimise the walking time to the door. I also have control over the number of products I carry at once to optimise the delivery time without straining my back too much. What I do not control, however, is the delivery time and the route I take to deliver the groceries. The construction of this route with its time windows is done in advance by an algorithm.

This construction of a route is a typical problem for the Operations Research (OR) field, which is closely related to the Artificial Intelligence (AI) research field. Being an AI student myself, and being interested in the optimisation and construction of my routes when driving for my part-time job, I ended up contacting a distribution analyst at Picnic. He explained the way they looked at the problem and the gist of their algorithms for optimising the solutions. This answered my question of how routes at Picnic are created, but it also created new questions like "How do you know a solution is well optimised?" or "Why do some algorithms work better than others?". The latter question was part of the inspiration for picking the routing problem and its algorithms as the subject of my thesis.

To understand how to solve a problem like a routing problem, it is essential to first understand how AI can solve a problem. Therefore, what the meaning of AI entails in research needs to be clear before we can use it to solve a problem. As an AI bachelor student, I have seen multiple approaches to AI pass by. In the book *Artificial Intelligence: A Modern Approach* by Russell and Norvig (2010), some of these approaches are discussed. The four approaches to AI listed are either thinking or acting in either a human way or a rational way. For this thesis, thinking rationally is the most suited approach, because the purpose of the problem is to come close to the ideal solutions, instead of trying to find solutions as a human would.

One of the most famous variants of a routing problem is the Traveling Salesman Problem, which is a problem that tries to find the best single route along all of the customers. For a company like Picnic, the routing problem is slightly more complex as the focus is on finding the best set of routes for delivering to all customers in the promised time windows. So instead of optimising one route, it is also necessary to optimise the number of routes. The general term for this problem is the Vehicle Routing Problem (VRP), with this specific case being the Vehicle Routing Problem with Time Windows (VRPTW). Over the

last few decades, there has been a lot of research within the OR field regarding this problem. Two articles in which this problem has been researched are the articles of Shaw (1998) and Bent and Van Hentenryck (2004), where the latter is a variation of the algorithm of the first paper. Shaw mentioned the necessity of a variation that performs better in certain circumstances and Bent and Van Hentenryck show that it does perform better in these circumstances, but neither go into detail as to why (This will be further explained in section 4.1). This creates a lack of understanding as to *why* the algorithm performs the way it does.

This lack of understanding makes the research fall short on science according to Hooker (1994, 1995). He proposed to use a way of doing research in OR that tries to describe the interaction between observations and the algorithm, which is called empirical research. Even though he proposed this over 25 years ago, the industry-standard within the OR is still biased to experimental data over empirical research.

This paper aims to show the benefit of using an empirical study in the OR field proposed by Hooker. This will be done by trying to gain a better understanding of why the algorithm proposed by Shaw works the way it does. In section 2, the foundation of this study will be set with a theoretical background on the subject. In section 3, the benefits and structure of an empirical study will be further explained. The empirical study will be applied in sections 4 and 5 and discussed in sections 6 and 7 to show its benefits.

2 Theoretical Background

2.1 Vehicle Routing Problem with Time Windows

The VRP is classified as an NP-hard problem (Lenstra and Kan, 1981), which means we expect no efficient algorithms. This also means there is rarely any certainty of knowing the best solution, i.e. the time to compute rises exponentially with the problem size. If we would go over every possible solution of a problem, it would take an unreasonable amount of time. To get to a solution in a reasonable amount of time, it is possible to use certain priority rules to find more optimised solutions in less steps. These priority rules are called heuristics, which are also used for the VRP, and likewise for the VRPTW algorithms.

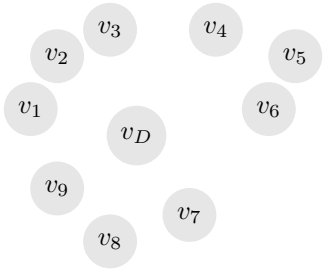

Figure 1: An example of a list of visits and a depot.

Before looking for a heuristic solution, it is essential to first formally define the problem and its variables. The algorithm starts with a problem that consists of a list of N visits $\{v_1, ..., v_N\}$ plus a depot v_D (as visualised in Figure 1). Each visit v_i consists of its coordinates (x_i, y_i), a time window $[e_i, l_i]$ with e_i being the earliest allowed arrival time and l_i the latest, a demand d_i of the capacity of the vehicles, and a service time t_i which describes the time it takes to do the delivery at the visit. The travel cost C_{ij} between v_i and v_j is the straight-line distance, which is equal to $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. The normalized travel cost C'_{ij} is the C_{ij} divided by the largest distance between two visits in the problem.

A solution or routing plan σ consists of a list of routes $\{r_1, ..., r_m\}$. Each route r_k consists of a list of visits that begins and ends with the depot v_D (as visualized in Figure 2, where every list of colored arrows is a route[1]). This is represented by a non-empty list of visits. To check if a route is legal, the sum of the demands $d(r_k) = \sum_{v_i \in r_k} d_i$ has to be lower than the capacity c of each of the problem's vehicles. Furthermore, none of the time windows $[e_i, l_i]$ can be violated. To check if none of the time windows is violated, the departure time δ_i for every visit i in r_k is calculated. Using $v_{i'}$ as the visit before v_i in r_k, δ_i is equal to

$$\begin{cases} \delta_0 = 0 \\ \delta_i = \max(\delta_{i'} + C_{i'i}, e_i) + t_i \end{cases}$$

[1]I would like to apologize to color blind people and people who are reading this in black and white, because in some places color was used to make routes more distinct.

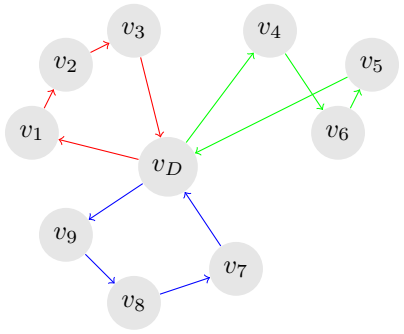

Figure 2: An example of a routing plan.

The earliest service time a_i for v_i is defined as

$$a_i = \max(\delta_{i-} + C_{i'i}, e_i)$$

A routing plan σ is only legal if all of its routes r_k are legal and all visits v_i are in a route, i.e. a routing plan σ is legal if

$$\begin{cases} d(r_k) \leq c & (1 \leq k \leq m) \\ a_i \leq l_i & (v_i \in r_k, 1 \leq k \leq m) \end{cases}$$

Finally, the total cost of a solution C_{total} is described as

$$C_{total} = \sum_{i,j \in routes} C_{ij}$$

2.2 Large Neighbourhood Search

The Large Neighbourhood Search (LNS) algorithm proposed by Shaw (1998) is a variant of a Local Search (LS) algorithm. A LS algorithm operates using a single current solution and it only moves to neighboring solutions (Russell and Norvig, 2010, ch 4.1), i.e. a solution that is a single change away from the current solution. In context of a VRPTW, a LS algorithm looks at a single routing plan at a time, while only going to new routing plans that have a single visit in a different route compared to the current routing plan. The LNS algorithm differs from the LS algorithm in the definition of a neighboring solution, with LNS being able to change the route of multiple visits at a time.

The LNS algorithm, as proposed by Shaw, is an algorithm that relaxes (removes) one or more visits from their current route and reinserts those visits in a route which lowers C_{total}. The relaxation is visualized in Figure 3, where the algorithm selected v_7 to be relaxed. Reinserting v_7 is visualized in Figure 4. The reinsertion is done using a Limited Discrepancy Search (LDS), proposed by Harvey and Ginsberg (1995), which iteratively searches the space within a certain limit on the number of discrepancies or changes allowed. Because of the variability of the discrepancies, the value of this has to be specified as an input for the algorithm.

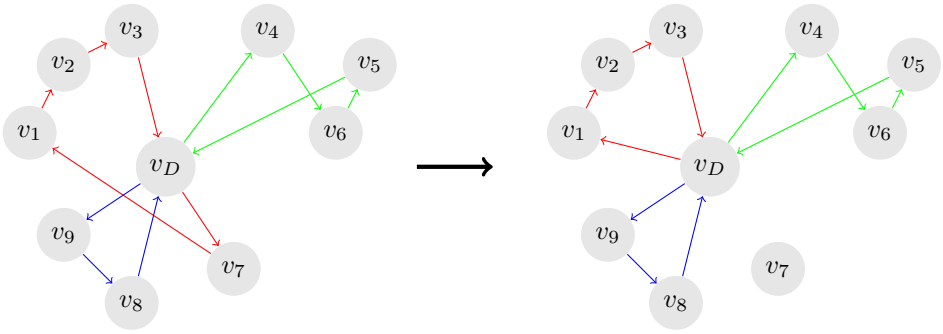

Figure 3: An example of the relaxation of a visit.

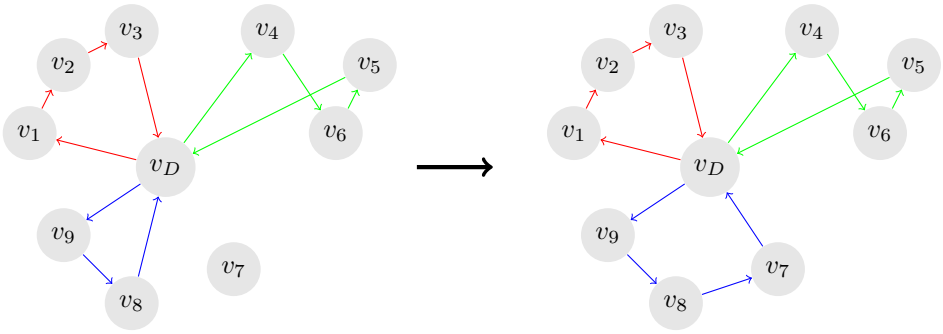

Figure 4: An example of the reinsertion of a visit.

2.2.1 Relaxation

The number of visits that are relaxed by the algorithm are based on the *toRemove* variable, which starts at a value of one, and increases every time the solution is not improved after a certain number of attempts *attempts*. This variable *attempts* is also part of the input of the algorithm. The first visit to be relaxed is always chosen at random. After that, the relaxation is based on relatedness $R(i, j)$ to one of the already relaxed visits, which is defined as

$$R(i,j) = \frac{1}{C'_{ij} + V_{ij}}$$

where V_{ij} is equal to 1 when v_i and v_j are not in the same route, and is equal to 0 when v_i and v_j are in the same route before relaxing the visits. This means that the closer $R(i, j)$ is to 1, the more related the visits v_i and v_j are. This relatedness approaches 1 from above if they were in the same route, and approaches 1 from below if they were not in the same route (as visualised in Figure 5). The importance of the relatedness is dependant on the determinism D, which is also one of the inputs, and a random variable between 0 and 1 *rand*. The next visit to relax is chosen out of a list of all the other visits sorted on

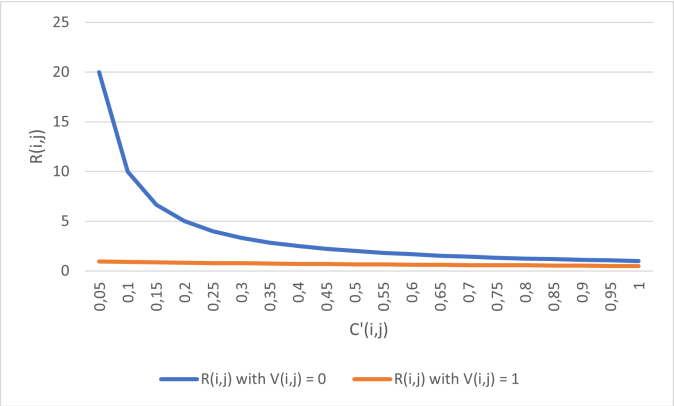

Figure 5: The relatedness of v_i and v_j compared to their normalised cost C'_{ij} if V_{ij} is either 0 or 1.

relatedness to an already relaxed visit. This is done by calculating the location l like $|visits| \cdot rand^D$ rounded down to an integer. The pseudo-code for the relaxation is given in Figure 6.

```
RemoveVisits (RoutingPlan plan)
1  Visit v := GetRandomVisit(plan)
2  VisitSet removed := {v}
3  while |removed| < toRemove
4      v := ChooseRandomVisit(removed)
5      VisitSet ranked := RankUsingRelatedness(v, plan)
6      r := Random.NextDouble(0,1)
7      i = |ranked| * r^D
8      v := ranked[i]
9      removed += v
```

Figure 6: Pseudo-code for the relaxation process.

2.2.2 Reinsertion

The visits are reinserted in the order of distance to the depot v_D, starting with the farthest visit. When a visit is selected to be reinserted, a list of all legal positions in all routes is generated. The list of positions is sorted by the routing plan's total cost C_{total} when a visit is to be placed on that position. Because it is a variant of the LDS algorithm, the sorted list of positions is only evaluated until the limit of discrepancies is reached. If the list of visits to reinsert is empty, the algorithm checks if C_{total} is lowered. The pseudo-code for the reinsertion is

given in Figure 7.

```
REINSERT (RoutingPlan plan, VisitSet toInsert, int discrepancies)
 1  if |toInsert| == 0
 2       double cost = CalculateCost(plan)
 3       if cost < bestCost
 4            bestPlan := plan
 5            bestCost := cost
 6  else
 7       Visit v := ChooseFarthest(toInsert)
 8       toInsert.Remove(v)
 9       int i := 0
10       for p in RankedPositions(v)
11            if i ≤ discrepancies
12                 RoutingPlan copy := Copy(plan)
13                 PlaceVisit(copy, visit, p)
14                 Reinsert(copy, toInsert, discrepancies − i)
15                 i += 1
16            else
17                 break
```

Figure 7: Pseudo-code for the reinsertion process.

3 Methodology

3.1 Empirical study of Algorithms

The empirical research that Hooker proposed to use in the OR field is a kind of research that does not solely look at the performance of an algorithm. Finding an improvement for an algorithm is nice, but understanding why the algorithm behaves the way it does is even better. This understanding is the figurative bridge between the observations about the algorithm and the inner workings of the algorithm itself. This kind of research has three main components which are also explained by Hooker (1994): a few tests to find observations that raise questions (noted as O), an informed hunch or hypothesis that tries to explain the observations (noted as H), and testing the implications that come from the assumption that the hypothesis is true (noted as I).

Setting up a study with this structure has multiple advantages, with Hooker mentioning in his paper *Needed: An Empirical Science of Algorithms* that deductive studies often rely on proving worst and average case theories. These theories are needed to give context to the results of the study or show how significant the results are. To construct these theories, the focus is on very atypical problems that show the worst possible solutions and randomly distributed problems to generate an average solution. An empirical study focuses on more directed hypotheses, which means only problems that test this hypothesis directly are needed. Therefore, an empirical study does not have to rely on the worst and average case theories like the standard deductive study.

To take full advantage of an empirical study, it is best to create a cycle of hypotheses. This is done with either one of these steps:

- If the implications of the previous hypothesis were proven true, try to construct new implications in an effort to disprove the hypothesis. Section 3.2.1 is an example of this setup.
- If the implications of the previous hypothesis were proven false, create a new hypothesis to explain the observation in a different way. Section 3.2.2 is an example of this setup.

The next section describes this cycle process in a few examples from the OR field, but also from outside of the OR field.

3.2 Examples of Empirical Research

3.2.1 Empirical Research in Operations Research

After Hooker proposed the empirical study as a research method in the OR field, he and Vinay used this setup for their paper *Branching Rules for Satisfiability* where they try to understand the performance of the Jeroslow-Wang rule better (Hooker and Vinay, 1995). I will not go into detail on what this rule is and how it works in this paper. In their paper, Hooker and Vinay use an empirical setup to first refute the common rationale for the rule in question. After this they use

an empirical setup to strengthen a different hypothesis on the performance of the rule in question. So, the experimental setup for their research looked like this:

O_1: Observation: Hooker and Vinay noticed that the Jeroslow-Wang rule and its variants performed well in recent work.

H_1: First hypothesis: They capture the common rationale for this behaviour in an empirical hypothesis.

I_1: Implications: They use this hypothesis to create a new rule, which should show a similar performance according to the hypothesis. This does not seem to be the case, which is a reason for them to refute the first hypothesis.

H_2: Second hypothesis: They create a new hypothesis, which they feel fits the performance better.

I_2: Implications: They use this hypothesis to create a new rule again, which should show similar performance according to the hypothesis. This is also what the results show, which is why the hypothesis is not refuted.

This setup gave Hooker and Vinay a way to refute the common rationale, and enough of an understanding to find a better hypothesis that is not refuted as easily. But apart from examples like this, there are a lot of examples within other research fields that show how strong the evidence of an empirical study can be.

3.2.2 Empirical Research in Physics

In the research field of physics there is a very rich history of empirical researches. In 19th and 20th century, physicists were able to refute one of the most famous theories in history: Newton's theory of gravity. Observations like an irregularity in the orbit of Mercury led to the need of a new theory. The physicist Albert Einstein published a theory called General Relativity. This theory has been empirically tested by researchers over the past hundred years, with no evidence to refute the theory as of yet. The structure of this research process looks something like this:

O_1: Observation: The irregularity of Mercury's orbit, together with all other knowledge about gravity on earth.

H_1: Hypothesis: General relativity by Einstein (1915).

$I_{1.1}$: An implication: According to general relativity, we should be able to see stars during a solar eclipse, which are actually directly behind the sun. This was observed during the solar eclipse of 1919 by Dyson et al. (1920), which supports the hypothesis.

$I_{1.2}$: Another implication: Another fundamental implication of the general relativity theory is the existence of black holes, according to Event Horizon Telescope Collaboration and others (2019) with their paper having the first photograph of one ever.

The more implications are tested to be true, the more probable the truth of a hypothesis becomes. For the implications of the General Relativity theory, the example showed the first and latest notable ones.

4 Experimentation

Before the experimentation with LNS, it is important to look at the specifics of the experiment itself, i.e. the specific inputs of the algorithm (as mentioned in section 2.2) and the problems it is tested on.

4.1 Experimental setup

The implementation of the LNS algorithm was done in *C#* and can be seen on github (van Wijk, 2021a). A visual representation was also implemented and can also be seen on github (van Wijk, 2021b). Every outcome is an average over ten separate runs. The inputs for the algorithm Shaw himself proposed and used are discrepancies *discrep* of 5, number of attempts *attempts* of 250, and a determinism D of 10. A sensitivity test was done by changing a single input at a time to see how much the performance of the algorithm changed based on the inputs. Figure 8 shows the difference in average performance with each of the input changes. This shows very minimal difference between each of the input changes, i.e. the algorithm is very insensitive to changes in the inputs. Therefore, the inputs Shaw proposed will be used from here on.

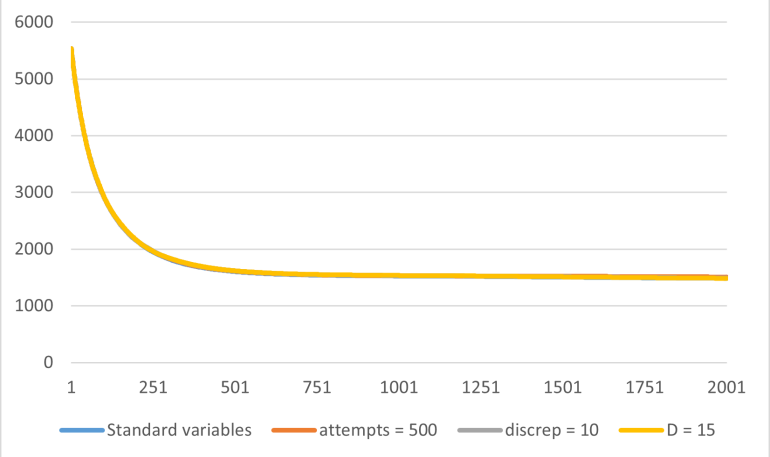

Figure 8: Difference in total cost C_{total} at each iteration with different inputs.

Researchers in the OR tend to use a lot of the same problem sets. Shaw also used one of these sets, specifically the benchmark problems introduced by Solomon (1987). The problems of this benchmark all have 100 visits and are divided into two main classes: S_1 being a class with a smaller capacity and scheduling horizon, and S_2 being a class with a larger capacity and scheduling horizon. For these classes, the average number of routes found by Shaw in the ending solution for S_1 is around 12, and for S_2 around 3. Within each of these classes there are multiple subclasses: the subclass R which has randomly generated coordinates for its visits, the subclass C for which the coordinates are clustered, and the RC subclass which is a combination of the two. Within each of the R and RC subclasses, all problems have their visits on the same coordinates. Within each of the subclasses there are also correlated problems. The two types of correlation that Solomon used are a deletion of time windows and the broadening of time windows. For the deletion correlation, each problem differs in 25 of its visits. For the broadening there is no real systematic change between all the problems using this correlation.

As mentioned in section 1, Shaw stated that his algorithm still needed improvement. This was due to the algorithm being unable to optimise the insertion procedure for a large number of visits, which is needed to find optimised solutions for the S_2 class. Bent and Van Hentenryck improved this using a two-stage hybrid between a simulated annealing algorithm that focuses on lowering the number of routes and the LNS algorithm, but they do not explain *why* this improves the LNS algorithm on S_2 problems and only state it is due to their belief that "LNS is particularly effective in minimising total travel cost when given a solution that minimises the number of routes." (Bent and Van Hentenryck (2004, pg. 2)). The experimentation done in the next section had this belief in mind to try and find out why LNS has a hard time optimising the reinsertion of a large number of visits.

4.2 Empirical Study

The experimentation will use a structure like the examples we saw in section 3.2, which used an observation, a hypothesis which tries to explain the observation, and the implications that the hypothesis has. The observations will be stated in section 4.2.1 and the hypothesis and implications will be stated in section 4.2.2.

4.2.1 Observations

For this experimentation, only the Solomon benchmark problems from the subclasses R and RC with the deletion correlation were used. This is due to the irregularity of differences between the C subclass for its S_1 and S_2 variants and the irregular broadening of the correlation.

Table 1 shows the C_{best} for all used problems as stated by Solomon. The C_{best} of the S_2 problems are lower than their S_1 problem counterparts for all problems. To analyse the performance of the algorithm, the difference between the current total cost C_{total} and the best known cost C_{best} by Solomon himself is

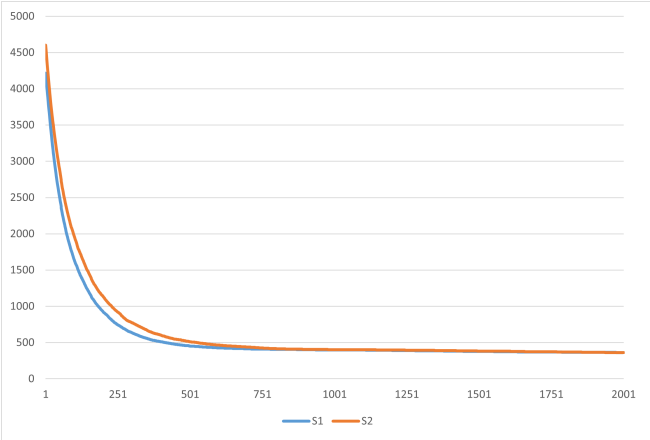

Figure 9: The average difference in cost of each iteration of all used problems with their best known solution.

calculated. The average difference over all used problems can be seen in Figure 9 for S_1 problems and S_2 problems. This figure shows that the difference with their C_{best} is lower for the S_1 problems in the beginning iterations, and they end up being similar from a thousand iterations on out. This tells us that LNS needs more iterations to optimise the S_2 problems compared to the S_1 problems.

Problem	Best Known C_{total}	Problem	Best Known C_{total}
$R101$	1645.79	$R201$	1252.37
$R102$	1486.12	$R202$	1191.70
$R103$	1292.68	$R203$	939.54
$R104$	1007.24	$R204$	825.52
$R105$	1377.11	$R205$	994.42
$R106$	1251.98	$R206$	906.14
$R107$	1104.66	$R207$	893.33
$R108$	960.88	$R208$	726.75
$RC101$	1696.94	$RC201$	1406.91
$RC102$	1554.75	$RC202$	1367.09
$RC103$	1261.67	$RC203$	1049.62
$RC104$	1135.48	$RC204$	798.41

Table 1: Table of the C_{total} for the best known solutions.

When looking into the worse performance of LNS on the S_2 problems, something stood out. The visual representation of the solution produced by the algorithm showed that some visits had the tendency to be split up, even though they would end up in the same route together. After looking into this, it seemed to be happening frequently enough for it to have a possible impact on perfor-

mance of the algorithm.

The splitting of visits that would end up together can be explained by the example route from Figure 2. Figure 10 shows the first iteration, where each of the visits start with their own route. In this example v_4 is relaxed and reinserted. The relatedness $R(4,5)$ is the highest for v_4, because v_5 is the closest visit. This means that the probability of v_4 being placed in the route of v_5 is the highest. Figure 11 shows the second iteration, where v_5 is selected to be relaxed and reinserted. Because v_5 is even closer to v_6, $R(5,6)$ is the better than $R(5,4)$. This means that the probability of v_5 being placed in the route of v_6 is the highest. Looking back at Figure 2, v_4 and v_5 do end up together in a route. Therefore, there seems to be a loss of important information when v_5 is deleted in the second iteration and split up from v_4.

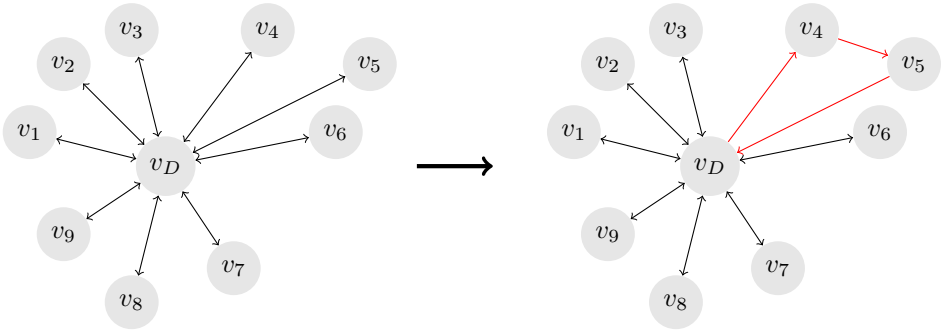

Figure 10: Relaxation and reinsertion of v_4.

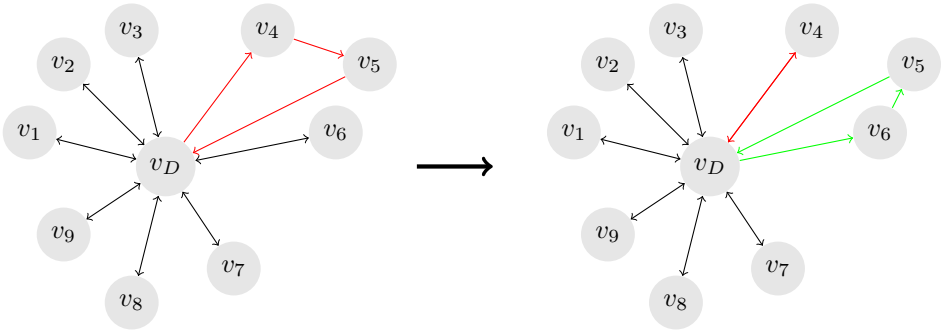

Figure 11: Relaxation and reinsertion of v_5.

To quantify the information loss, the number of times two visits got split up was counted. For example, say there is a route called r_1 with visits $\{v_1, v_5, v_8\}$. If v_5 were to be placed in a different route, the combinations (v_5, v_1) and (v_5, v_8) would be increased by one. When the algorithm is finished, the total number of split-ups for each visit with the visits in its final route is calculated, i.e. the sum of all the split-ups for all visit combinations that ended up in the same route.

The average sum of all visits is displayed in Table 2 for all Solomon problems, which shows that each S_2 problem has a higher average of split-ups for the visits in their ending route in comparison to their S_1 counterparts. This could indicate that there is a negative relationship between the average number of split-ups for each visit and the C_{total} at the end of the algorithm, i.e. a negative relationship between the average number of split-ups and the performance of the algorithm. This would indicate that the higher the average number of split-ups, the worse the performance of LNS.

Problem	Average Split-ups	Problem	Average Split-ups
R101	2.615	R201	5.397
R102	3.317	R202	6.207
R103	4.874	R203	7.598
R104	6.203	R204	9.342
R105	3.774	R205	7.106
R106	4.479	R206	7.488
R107	5.659	R207	7.884
R108	6.836	R208	9.064
RC101	3.853	RC201	5.787
RC102	4.576	RC202	6.755
RC103	5.563	RC203	6.864
RC104	6.350	RC204	8.647

Table 2: Average number of split-ups for all used Solomon benchmark problems.

4.2.2 Hypothesis and implications

The observation of a possible negative relationship between the average number of split-ups and the performance of the LNS algorithm can be captured in a hypothesis to fit an empirical study.

Hypothesis: Lowering the loss of information, which is a result of two visits being split up even though they end up in the same route together, has a positive influence on the performance of the LNS algorithm.

If this hypothesis were to be true, an increase of the average number of split-up visits that end up together would result in a worse performance of the LNS algorithm. This is in line with the higher averages and worse performances of the S_2 problems compared to their S_1 counterparts. If this average were to be lowered, it would result in a better performance of the LNS algorithm. Following this, if a problem is constructed that is very similar to the Solomon benchmark problems, but has a lower average number of split-ups, the performance of the LNS algorithm should improve. Likewise, if a comparable problem is constructed that has a higher average number of split-ups, the performance of the LNS algorithm should worsen.

Implications: When testing the LNS algorithm on a slightly different problem set, if the average number of split-ups for visits that end up in the same route is lowered, the performance of the algorithm should improve. Likewise, if the average number of split-ups for visits which end up in the same route is higher, the performance of the algorithm should worsen.

5 Results

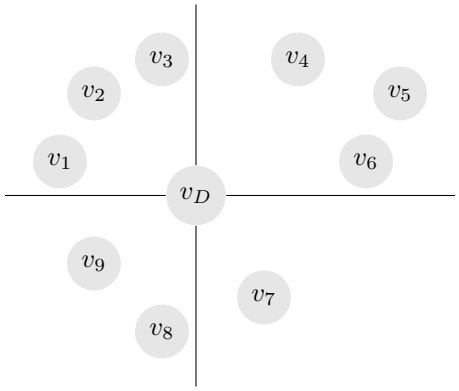

Figure 12: Example of a split problem.

A set of split problems was constructed, where each of the Solomon Benchmark problems was split in four as visualized in Figure 12. All visits had new time windows assigned dependent on the part they were in. This was done by multiplying the total service time by four, adding the old total service time to the time windows of all visits in the top left part, adding two times the old total service time to the time windows of all visits in the bottom right part, and adding three times the old total service time to the top right part. This way there is less incentive for the algorithm to construct routes that pass through more than one part of the problem.

Table 3 shows the average number of split-ups between visits that end up in the same route for the split Solomon Benchmark problems. This shows that the number of split-ups increases for all split S_1 problems, and decreases for all split S_2 problems. According to the implications of the hypothesis, this would indicate that the LNS algorithm would perform less optimised on the split S_1 problems and more optimised on the split S_2 problems compared to the their non-split variants. In a figure that shows the difference between the cost C_{total} of the normal problems and the split problems for each iterations, the expectation is that the S_1 difference is a negative number, and the S_2 difference is a positive number. However, Figure 13 shows a different result, with the difference of S_2 being negative and the difference of S_1 being positive. This means that the

Problem	Average Split-ups	Problem	Average Split-ups
R101Split	4.156	R201Split	5.363
R102Split	4.758	R202Split	5.591
R103Split	5.938	R203Split	7.054
R104Split	6.757	R204Split	8.518
R105Split	4.741	R205Split	6.788
R106Split	5.519	R206Split	7.188
R107Split	6.612	R207Split	7.639
R108Split	7.491	R208Split	8.126
RC101Split	4.642	RC201Split	5.242
RC102Split	5.626	RC202Split	6.395
RC103Split	6.961	RC203Split	6.681
RC104Split	7.186	RC204Split	7.660

Table 3: Percentages of visits split up from other visits in their ending route for split problems.

increase in split-ups for the split S_1 problems was paired with a more optimised performance, and the decrease in split-ups for the split S_2 problems was paired with a less optimised performance. This is a direct contradiction with the implications of the hypothesis, which means the hypothesis has to be rejected, i.e. the loss of information from two visits splitting up before ending up in the same route together does not have a direct negative effect on the performance of the algorithm.

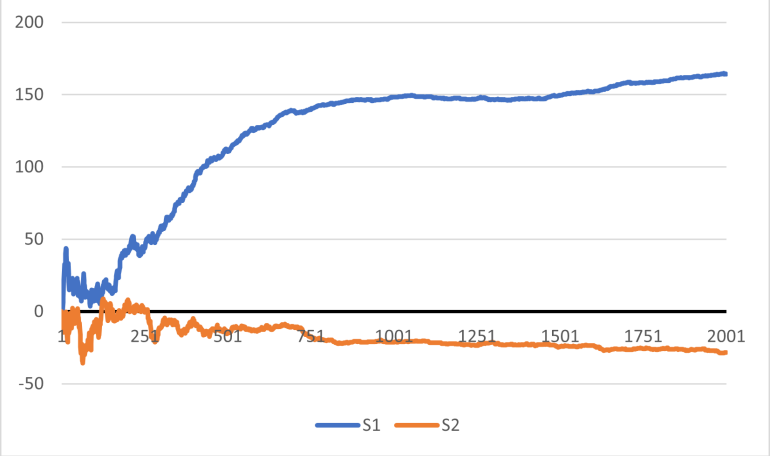

Figure 13: Average difference between each normal and split variant.

6 Discussion

The refuting of the hypothesis in section 5 can be a result of multiple factors. Either the hypothesis is (partly) wrong or the hypothesis is right and the way of testing was (partly) wrong.

For the first factor, I can think of a few reasons as to why the hypothesis could be wrong. Firstly, the information loss could be very minimal. Also, if two routes can be combined, and the *toRemove* is lower than the length of each of the routes, there is no way to combine the routes without splitting up at least two visits that end up in the later route. Finally, it could also mean that the split-ups are defined in a wrong way. A different way to test a similar loss of information is to look at the average relatedness between visits in the ending route, and checking how often visits with a similar relatedness are deleted from each other. If these three points would be researched further, this could strengthen or weaken the hypothesis.

For the second factor, the main reason I can think of is the split Solomon problems being an uncontrolled environment. When testing the influence of the average number of split-ups on the performance of the algorithm, it is best to change as little as possible apart from the average number of split-ups. It is also still possible for routes to go between different parts of the used split problems. Furthermore, relatively related visits that end up in different parts of the split problem could also cause a big reduction in the performance. Creating a problem set which influences the LNS algorithm so it will lower its average number of split ups would be a way to do further research into this. It is also important to note that the algorithm must not be changed, because that makes it even harder to control the environment. Also, researching the LNS algorithm is more accurate when you are using the LNS algorithm instead of a variant of it.

The fact that the hypothesis was refuted in section 5 does not mean it is a bad result. Negative results are often overlooked in the OR field, but can give a better understanding of an algorithm, just like positive results. Because the characteristic of the number of split-ups is very specific, the importance of the empirical study is highlighted. It is harder in a standard deductive science to explain an observation with a single characteristic, as was tried in this paper. But as stated in section 3.1, this process is a never ending cycle and is far from done. The more these cycle tests are done, the stronger the result from an empirical study is. This is what gives an empirical study a noticeable edge over a standard deductive study in the OR field.

A possible next test could be looking deeper into the split Solomon problem set. For this research, only the performance of the LNS algorithm and the average number of split-ups were taken into account. This leaves a lot of variables to be further explored. If the downgrade in performance on the split problems were to be explained by a different factor, there is still a possibility of the hypothesis from this paper holding some truth. This research would first have a hypothesis discrediting the split problems as a problem set which tests the effect of the split-ups. After this, a new problem set needs to be created

which is more fitting to test the effect of the split-ups on the performance of the algorithm.

Using the empirical setup for this research has shown how big of a gap in understanding there is between the paper of Shaw and its improvement by Bent and Van Hentenryck. The change made to the Solomon problem set to make the split variants was quite a minimal change, but the difference in performance was very noticeable. Together with the refuting of the hypothesis, this shows that just stating the improvement without knowing why it improved, lacks a lot of the understanding as to why the LNS algorithm even performs the way it does.

7 Conclusion

The goal of this paper was to show the value of doing empirical research in the Operations Research field. This was done using the Large Neighbourhood Search algorithm proposed by Shaw. Experimental results show that the LNS algorithm performs better in optimising the C_{total} for Solomon benchmark problems with smaller capacities (S_1 problems) compared to problems with bigger capacities (S_2 problems). Because of the higher number of split-ups between visits that end up together in a route for the S_2 problems, the split-ups were identified as a possible factor in the difference in performance between the S_1 and S_2 problems. However, the research of this paper indicates that the split-ups are not a direct factor. This would mean that the relationship between the performance of the algorithm and the average number of split-ups is more complex than a simple negative relationship. This needs to be tested further for a more concrete relationship to be defined, according to the empirical cycle that was highlighted in this paper.

8 Acknowledegments

I would like to thanks my thesis supervisor Tomas Klos for all his effort in helping me. He has helped me give shape to this thesis in a way I could not have done myself. The video calls always were a good time and useful at the same time. Furthermore, I would like to thank my second reader Francisca Pessanha.

I would also like to thank Peter Bijl from Picnic for taking time to explain the basics of vehicle routing problems to me. He had no need to help me that much, but in doing so he enlarged my interest in the topic which was a big help.

Finally, I would like to thank Bruce Harrems, Sarah Angenent, Minke van Wijk and Iris Reitsma for proofreading the thesis, helping me find spelling mistakes and unclear writing.

References

Bent, R. and Van Hentenryck, P. (2004). A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530.

Dutch Ministry of Social Affairs & Employment, Dutch Ministry of Economic Affairs & Climate Policy, and Ministry of Finance (2020). Ondernemers en corona: Resultaten per branche en grootteklasse.

Dyson, F. W., Eddington, A. S., and Davidson, C. (1920). Ix. a determination of the deflection of light by the sun's gravitational field, from observations made at the total eclipse of may 29, 1919. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 220(571-581):291–333.

Einstein, A. (1915). Erklarung der perihelionbewegung der merkur aus der allgemeinen relativitatstheorie. *Sitzungsber. preuss. Akad. Wiss*, 47:831–839.

Event Horizon Telescope Collaboration and others (2019). First m87 event horizon telescope results. i. the shadow of the supermassive black hole. *arXiv preprint arXiv:1906.11238*.

Harvey, W. D. and Ginsberg, M. L. (1995). Limited discrepancy search. In *IJCAI (1)*, pages 607–615.

Hooker, J. N. (1994). Needed: An empirical science of algorithms. *Operations research*, 42(2):201–212.

Hooker, J. N. (1995). Testing heuristics: We have it all wrong. *Journal of heuristics*, 1(1):33–42.

Hooker, J. N. and Vinay, V. (1995). Branching rules for satisfiability. *Journal of Automated Reasoning*, 15(3):359–383.

Lenstra, J. K. and Kan, A. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227.

Russell, S. J. and Norvig, P. (2010). Artificial intelligence: A modern approach.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer.

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

van Wijk, J. (2021a). Large neighbourhood search. `https://github.com/JaccovanWijk/LargeNeighbourhoodSearch`.

van Wijk, J. (2021b). Large neighbourhood search framework. `https://github.com/JaccovanWijk/LNSFramework`.