Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems

Paul Shaw*
Department of Computer Science,
University of Strathclyde, Glasgow.

April 1998

Abstract

We use a local search method we term Large Neighbourhood Search (LNS) for solving vehicle routing problems. LNS meshes well with constraint programming technology and is analogous to the shuffling technique of job-shop scheduling. The technique explores a large neighbourhood of the current solution by selecting a number of customer visits to remove from the routing plan, and re-inserting these visits using a constraint-based tree search. We analyse the performance of LNS on a number of vehicle routing benchmark problems. Unlike related methods, we use Limited Discrepancy Search during the tree search to re-insert visits. We also maintain diversity during search by dynamically altering the number of visits to be removed, and by using a randomised choice method for selecting visits to remove. We analyse the performance of our method for various parameter settings controlling the discrepancy limit, the dynamicity of the size of the removal set, and the randomness of the choice. We demonstrate that results produced by our technique are very competitive with Operations Research meta-heuristic methods, indicating that constraint programming technology is directly applicable to vehicle routing problems.

1 Introduction

The vehicle routing problem (VRP) is one of visiting a set of customers using a fleet of vehicles, respecting constraints on the vehicles, customers, drivers, and so on. The goal is to minimize the costs of operation, which normally involves reducing a combination of the number of vehicles and the total distance travelled or time taken. Because of the size of routing problems that need to be routinely solved in industry, local search techniques (sometimes combined with meta-heuristics) are generally used to produce solutions to routing problems. These methods scale well and can produce reliably good solutions.

Constraint programming (CP) appears to be good technology to apply to VRPs because of the ubiquity of side constraints in real-world problems. A natural conclusion is that if one can use constraint programming technology within a local search framework, one should have a technique that will work well for real-world vehicle routing problems. Unfortunately, the traditional view of local search does not integrate well with the tree-based view of search within constraint programming. The challenge is to try to get the best of both of these methods: exploration of a neighbourhood from local search, and the power of propagation and heuristics from constraint programming.

In this paper we use a technique we refer to as Large Neighbourhood Search (LNS) which makes moves like local search, but uses a tree-based search with constraint propagation to evaluate the cost and legality of the move. As a "heavyweight" technique such as constraint programming is used to evaluate the move, many less moves per second can be evaluated than is normally the case

*Member of the APES Group.

for local search. However, these moves can be more powerful, moving the search further at each step. When such far reaching moves are possible there are normally many more of them available than would be the case if the moves were very localized: hence the name Large Neighbourhood Search.

LNS is applied to the vehicle routing problem by defining a move to be the removal and re-insertion of a set I of customer visits. The set of customer visits chosen for removal and re-insertion defines the move. We define a "relatedness" measure between customer visits and use this as a basis for choosing the set I. We use Limited Discrepancy Search to re-insert the customer visits into the routing plan. The size of the set I is increased over time, stepping up when the search is deemed to have stagnated at the current size of set I.

Experiments are carried out on a number of classic benchmark problems from the literature, with and without time windows. We analyse the quality of solutions produced by LNS over a range of parameter settings by comparison to the best published solutions. Despite its simplicity in comparison with Operations Research techniques, LNS is shown to have excellent average performance and produces many new best solutions to these benchmark problems.

The paper is organized as follows: Section 2 describes LNS as applied to the vehicle routing problem. The method is then compared with related work, and its benefits are discussed within a vehicle routing context. Section 3 presents computational experiments on vehicle routing problems with and without time windows. Finally, section 4 draws some conclusions.

2 Large Neighbourhood Search

LNS is based upon a process of continual relaxation and re-optimization. For the VRP, the positions of some customer visits are *relaxed* (the visits are removed from the routing plan), and then the routing plan is *re-optimized* over the relaxed positions (by re-insertion of these visits back into the routing plan). The re-insertion process can use the full power of constraint programming via constraint propagation and heuristics. For instance, the *minimum* cost re-insertion can be evaluated via branch and bound, or incomplete techniques can be applied that only explore part of the search tree in the hope of finding a good solution. One iteration of removal and re-insertion can be considered as the examination of a powerful neighbourhood move.

Two factors can affect the way in which the search operates: how the set of customer visits are chosen for removal, and the process used to re-insert the visits. These two aspects are examined in more detail in the remainder of this section.

2.1 Choosing Customer Visits

We describe a possible method for choosing the set of customer visits that should be removed and re-inserted together. It is one of myriad such methods that could be used, and we would not be surprised if many better techniques could be found. However, we do believe in a general *strategy* for choosing visits to remove together: that of choosing *related* visits.

Related, of course, has to be suitably defined. A measure should be used that results in good opportunities for the re-insertion to achieve some improvement in the routing plan. One obvious criterion is that visits that are geographically close to one another will be more related than visits that are more distanced. If visits close to one another are removed from the routing plan together, there is opportunity for interchange of positions and so on. If visits distanced from each other are removed, the best re-insertion point for each will probably be close to where it came from. If this is the case, the re-insertion of these visits has been "independent" and the same effect could have been achieved by removing less customer visits from the routing plan and performing these re-insertions sequentially. (No more customer visits than necessary should be removed from the routing plan, as the re-insertion process is more expensive for larger numbers of visits.)

If two visits occur in the same route, we can also consider them to be related. Removing multiple visits from the same route should be encouraged, as removing all visits from a route is the only way of reducing the number routes. (This happens when all visits are removed from a route,

```
RemoveVisits(RoutingPlan plan, Integer toRemove, Real D)
    VisitSet inplan := getVisits(plan)
    v := ChooseRandomVisit(inplan)
    inplan := inplan - v
    RemoveVisit(plan, v)
    VisitSet removed := v
    while |removed| < toRemove do
        v := ChooseRandomVisit(removed)
        // Rank visits in plan with respect to relatedness to v
        VisitList lst := RankUsingRelatedness(v, inplan)
        // Choose a random number, rand, in [0, 1)
        Real rand := Random(0,1)
        // Relax the visit that is rand^D of the way through the rank
        v := lst[Integer(|l|*rand^D)]
        removed := removed + v
        inplan := inplan - v
    end while
end RemoveVisits
```

Figure 1: How visits are chosen for removal

and are then re-inserted into existing routes.) Related visits might also have similar allowable visiting hours, or be visited at similar times in the current routing plan.[1]

We assume a relatedness function $\mathcal{R}(i, j)$ exists that takes any two visits i and j and delivers a non-negative value indicating how closely these visits are related. This function might include some of the features described above, and perhaps some more problem specific features. For instance, in a pickup and delivery problem where the pick up and delivery visits have to be performed by the same vehicle, with pick up before delivery, these two visits should be highly related. This ensures that when one visit is removed from the routing plan the other is too, so that both can be moved to a different vehicle if this reduces cost. If only one visit of the pair were removed, it would be highly restricted in where it could be re-inserted.

In this paper, we do not address benchmark problems with side constraints[2] and use the simple relatedness function:

$$\mathcal{R}(i, j) = \frac{1}{c_{ij} + V_{ij}}$$

where c_{ij} is the cost of getting to j from i (travel distance in this paper), and V_{ij} evaluates to 1 if i and j are served by the same vehicle and 0 otherwise. We further assume that c_{ij} is normalized in the range $[0..1]$. Hence, relatedness as used here depends only on the proximity of the visits and whether they are currently visited by the same vehicle.

Figure 1 describes how visits are chosen. To discourage stagnation (*i.e.* to avoid the situation where similar set of visits are continually chosen based solely on relatedness) we also include a random element. In the algorithm, D is a parameter that controls determinism. With $D = 1$, relatedness is ignored and visits are chosen randomly. With $D = \infty$, visits relaxed are maximally related to some other relaxed visit. We have found that D can lie in quite a wide range. $5 \leq D \leq 20$ works reasonably well, but we have found that values of D less than 3 or greater than 30 work poorly.

There are, of course, many other ways in which customer visits could be chosen. In figure 1, a visit is chosen that is related to one visit in the already chosen set. Alternatively, one could rank the visits by relatedness to all (or some) visits in the chosen set. Additionally, the simple ranking system is not ideal when the relatedness of some pairs of visits is much larger than others.

[1] In a study of the job-shop scheduling problem, [3] uses a shuffling technique (analogous to LNS) that relaxes the start times of all operations within a certain time window.

[2] See [13] for a study of LNS and traditional local search methods when side constraints are added.

2.1.1 Control of the Neighbourhood Size

There is also the question of deciding on the *number* of visits to remove at each move of LNS. For efficiency, one wants to remove the smallest set that will yield an improvement in the cost when the visits are re-inserted. We use the following scheme to attempt to ensure this: Initially, start the number of visits r to remove at 1. Then, during search, if a consecutive attempted moves have not resulted in an improvement of the cost, increase r by one. An upper limit of 30 was placed on the value of r. This scheme has the benefit of increasing the number of visits for re-insertion only when the search has deemed to have become "stuck" for the smaller value of r. The value of the parameter a thus determines how stubbornly LNS seeks improving moves at smaller values of r.

2.2 Re-inserting Visits

The re-insertion process uses a branch and bound technique, with constraint propagation and heuristics for variable and value selection. In its simplest form, the technique examines the whole search tree for the re-insertion of minimum cost (which amounts to examination of every combination of re-insertions for all removed visits).

We view each of the relaxed (removed) visits as the constrained variables, which can take values corresponding to their available insertion points. An insertion point is an arc in the routing plan that can accommodate the visit in question. For any particular visit, some insertion points may immediately be ruled out as illegal (for instance due to time or capacity constraints) via simple propagation rules. There is one propagation rule for load, and one for time. For load, a visit cannot be inserted into *any* position on a route if doing so would violate the capacity constraint of the vehicle. For time, a visit v cannot be inserted between visits i and j if this would cause the vehicle to arrive at v or j after their latest deadlines. Additionally, a simple propagation rule maintains the earliest and latest arrival times for all visits along a route, based on the travel times between visits, and time windows on these customer visits.

Insertion positions for visits can also be ruled out if they would take the lower bound on the cost of the plan over the upper bound defined by the best solution found so far. We form the lower bound completely naively as the *current* cost of the routing plan. No attempt is made to compute a lower bound on the cost of including as yet unrouted visits. This makes the procedure very fast, but means that the search tree will be larger than if it would be if a better bounding procedure was used. Improving this bound is a subject of future work.

2.2.1 Branching Heuristics

The efficiency of branch and bound can be increased by the use of variable and value ordering heuristics. Variable ordering heuristics are used to reduce the size of the search tree. Value ordering heuristics are used to guide the search to a solution. It is generally believed, and experimental evidence suggests [9, 10] that an effective variable ordering strategy is to instantiate more constrained variables first. In our context, a variable (visit) could be considered constrained if it is far removed from the rest of the routing plan (and so will constrain the plan when inserted, due to high consumption of distance and time resources). Less work has been performed on value ordering heuristics, but a study in [8] improved performance of solution procedures for constraint satisfaction problems using a "promise" heuristic. In our context, we could consider a value (insertion point) more promising if it increases the cost of the routing plan less.

Assume that visit v has a set of insertion points $\mathcal{I}_v = \{p_1, \ldots, p_n\}$, and that the cost $\mathcal{C}_p$ of an insertion point p is the increase in cost of the routing plan resulting from inserting v at p. We then define the *cheapest* insertion point $c_v \in \mathcal{I}_v$ of v as the one for which $\mathcal{C}_{c_v}$ is a minimum. As heuristics, we choose visit v to insert for which which c_v is maximized, and then try inserting it in each of its insertion points, cheapest to most expensive, in increasing order. This choice of visit is known as the *farthest insertion* heuristic. The sub-problem of inserting the remainder of the removed visits is solved after each insertion of v. Note that if any visit has only one legal insertion

```
Reinsert(RoutingPlan plan, VisitSet visits, Integer discrep)
    if |visits| = 0 then
        if cost(plan) < cost(bestplan) then
            bestplan := plan
        end if
    else
        v := ChooseFarthestVisit(visits)
        i := 0
        for p in rankedPositions(v) and i ≤ discrep do
            InsertVisit(plan, v, p)
            Reinsert(plan, visits - v, discrep - i)
            RemoveVisit(plan, v)
        end for
    end if
end Reinsert
```

Figure 2: How visits are re-inserted

point, it is immediately inserted at this point. This can be considered to be a propagation rule: a so-called "unit propagation".

For richer routing problems, other domain knowledge could be included in the variable and value ordering heuristics. For instance, if one particular visit has many side constraints imposed upon it, then it is likely to be highly constrained and a candidate for early insertion.

2.2.2 Limited Discrepancy Search

In many cases, the branch and bound re-insertion procedure can find a better solution or prove that none exists for about 25 visits in a few seconds for problems with time windows.[3] However, the distribution of solution times has a long tail, and the result is that some problems take a very long time to solve. To avoid this problem, we included Limited Discrepancy Search [12] (LDS) within our branch and bound procedure. LDS explores the search tree in order of an increasing number of *discrepancies*, a discrepancy being a branch against the value ordering heuristic. We count a single discrepancy as the insertion of a customer visit at it's second cheapest position. We count as two discrepancies either one insertion at the third cheapest position, or two insertions at their second cheapest positions, and so on. We use only one phase of LDS, with the discrepancy limit set to a pre-defined value d. In this way, we explore all leaf nodes from 0 to d discrepancies, without revisiting nodes. Our re-insertion algorithm is shown in figure 2. Note that the management of legal insert positions is not mentioned. We assume they are handled by automatically triggered propagation rules. The parameter d represents a way of trading the intensity of the exploration of the search tree with how many LNS moves can be applied per second. When d is small, we opt for large numbers of attempted moves with little exploration of possible insertions for each move. For high values of d, the opposite situation hold. One might conjecture that there will be some trade off point. This premise is investigated in section 3.

2.3 Related Work

LNS is exactly analogous to the shuffling technique used in job-shop scheduling [2, 3] which is itself inspired from the shifting bottleneck procedure of [1]. Shuffling is a technique whereby the start times for operations on the majority of machines are relaxed, and then a tree-based search procedure is used to reconstruct the schedule. We have not used the term "shuffling" in this paper to avoid confusion with job-shop scheduling. Moreover, the basic technique is easily generalizable

[3] For problems without time windows, only around 15 visits can be re-optimized in this time as the reduced number of constraints results in less pruning of the search space.

to other problem classes, where the natural visualization of the move is not a shuffling of positions, but something else entirely.

In [2], a fairly simple shuffling technique is presented, but in [3], various different types of shuffle are used. The different shuffles are based on different criteria for selecting the operations which will have their start times relaxed. These selections are based on common machines, common start times, and so on. Interestingly, each of the shuffles can be seen as exploiting the relatedness of operations. The authors also use an incomplete search technique to complete the schedule based on limiting the number of backtracks available. We tried out such an approach, but it proved markedly inferior to the LDS procedure used.

Other related work on routing problems has been carried out in [17, 19] which advocates the use of constraint programming within local search. Here, traditional move operators from the routing literature are used (for instance k-opt), but a constraint programming branch and bound search is used to evaluate the neighbourhood to find the best legal move. One can see the similarities with the shuffling/LNS approach, but there are some differences. The main difference is that only traditional move operators are being used, and so constraint programming is used not to improve the power of moves, but only the efficiency of evaluation. Secondly, the whole neighbourhood is being explored, requiring a complete branch and bound search. With the shuffling/LDS approach, any method can be used to perform the re-insertion of visits, for instance, a heuristic method, complete search, LDS, or another discrepancy-based approach (*e.g.* [16]).

Finally, some work has been performed in solving quadratic assignment problems using a very similar technique [15].

2.4 Discussion

There are many advantages to using constraint-based local search (in the form of LNS) over traditional local search approaches, both in the general case, and in the case of the VRP.

The main advantage of using LNS is that the addition of side constraints can be handled much better. For instance, in the case of the VRP, different models such as the pickup and delivery problem can be handled with ease. Using standard local search move operators, coping with this model is more difficult: special-purpose operators are required to move both the pickup and delivery to a different route simultaneously. For LNS, we simply make the pickup and delivery strongly related (to encourage both to be removed together), and let the constraints of *same vehicle* and *time(pickup)* $<$ *time (delivery)* constrain where the visits can be inserted. In [14], Kindervater and Savelsbergh discuss ways of efficiently introducing side constraints into local search. Their methods, however, are extremely complex and dedicated to particular move operators and particular side constraints. Moreover, no suggestion is made of how to extend their methods to different move operators or side constraints.

A difficulty with problems with many side constraints is that many of the simple move operations will be illegal due to violation of the side constraints. Increasing numbers of side constraints constantly reduce the number of feasible moves. This can make search difficult, as the search space can become pitted with local minima or even disconnected. LNS alleviates this problem somewhat by providing more powerful far-reaching move operators that allow the search to move over barriers in the search space created by numerous side constraints. This ability is demonstrated in a forthcoming paper [13].

Evaluation of cost differences can be a time consuming phenomenon in local search techniques. In idealized models, one often uses travel distance as the cost function, since for edge exchange moves, the difference in cost can be computed quickly in constant time. Savelsbergh [22] has also introduced methods of constant time costing of various route time measures. However, for real vehicle routing problems, cost functions are often much more complicated than this. In LNS, the full cost of a move will be evaluated during constraint propagation. Cost differences are not used, and there is no need to invent clever methods to compute them. We did mention, however, that our heuristics for choosing the next visit to insert and its favoured insert position operate on cost differences. Since this information is just a hint to the search, and most cost functions are

generally related loosely to distance, we could (without fear of losing much efficiency) simply use distance as a good approximation for the variable and value ordering heuristics.

3 Computational Results

We now report the results of applying LNS to various benchmark problems in the literature both with and without time windows. We examine the quality of solutions obtained by our technique over various parameter settings, as compared to the best published solutions. We also report new best solutions obtained by the use of LNS. Finally, we perform longer runs of LNS in order to compare results with the best Operations Research methods for solving VRPs with time windows. We used a 143 MHz Ultra Sparc running Solaris for all our experiments. The LNS code was written in C++ and compiled using the Sun C++ compiler.

3.1 Capacitated VRPs

The capacitated VRP problems we examine have a single depot, an unlimited number of vehicles, with each vehicle having identical capacity. A set of customers must to be visited, each having a specified load. The sums of the loads of customers visited by a single vehicle must not exceed its capacity. The objective is to minimize the total distance travelled by all vehicles. The customers are embedded in a plane, and all distances are calculated using the Euclidean metric.

Following [21], we use three flavours of capacitated VRP: classic test problems, non-uniform problems, and those derived from real-world data. The classic problems (C50, C75, C100, C100B, C120, C150, C199) are due to [5]. The non-uniform problems (TAI100A–TAI100D, TAI150A–TAI150D) were created by Rochat and Taillard [21] to attempt to capture structure inherent in real problems. In these problems, loads are exponentially distributed, and customers are clustered. The sizes and compactness of the clusters can be quite variable. Finally, some problems reflecting real-world data are taken from [7] (F71 and F134) and [25] (TAI385). In all of these problems, the number in the name indicates the number of customers.

LNS as presented has 3 parameters that can be varied. First, one can adjust the number of discrepancies d given to LDS in the re-insertion. Second, we can vary the number of consecutive failed re-insertion attempts a needed to increase the number of visits being re-inserted at each stage. Finally, we can vary the determinism parameter D. We chose $d \in \{0, 1, 2, 3, 5, 10, \infty\}$ ($d = \infty$ corresponding to complete search), $a \in \{250, 500, 1000\}$, and $D \in \{5, 10, 15\}$. We ran LNS three times (with different random seeds) on all problems with each combination of parameter settings. A time limit of 900 seconds was placed upon LNS. When $d = \infty$, the distribution of re-insertion times has a long tail, and so some re-insertions can take a very long time. As such, we placed a time limit of 20 seconds on the re-insertion process, which is in force for all experiments reported. We chose an initial solution with the number of vehicles equal to the number of customers, with one customer visit performed by each vehicle. Initially, the neighbourhood size is set so that only one visit is removed and re-inserted.

Table 1 shows the results of running LNS over all problems without time windows: three times for each parameter combination. We show the percentage difference in cost between solutions obtained by LNS and the best published solution. We computed these percentages as follows: for each combination of parameters, we take the costs of all the solutions provided by LNS and divide by the cost of the best published solution for the problem in question. This delivers a set of *cost ratios*. We then form a ratio which is the geometric mean of this set: the global cost ratio. By subtracting one and multiplying by 100, we attain the average percentage figure above the best published solutions. We used this method to produce all averages of percentages.

The average solutions produced by LNS are close to the best published solutions. For the range of parameters chosen, average solutions are all within 4% of the best published solution, and for the best parameter settings, 2.2% from the best published solution on average. Thus, LNS appears to be relatively robust in how its parameters can be set. However, the difference between the best and worst parameter settings is significant, as at this level of quality, improving the solution quality

		discrepancies						
attempts	determinism	0	1	2	3	5	10	∞
250	5	3.0	2.5	2.3	2.2	3.3	3.7	2.6
	10	2.8	2.3	2.2	2.3	2.5	3.5	3.2
	15	2.8	2.4	2.2	2.7	2.6	3.4	3.8
500	5	3.0	2.4	2.9	2.7	2.2	2.9	3.8
	10	2.8	2.1	2.3	2.6	2.9	3.9	3.7
	15	2.9	2.5	2.5	2.3	2.6	3.3	3.6
1000	5	2.9	2.9	2.9	3.0	2.7	3.4	3.6
	10	3.1	3.1	2.9	3.1	3.1	3.2	3.9
	15	3.1	2.8	2.7	2.9	2.6	3.0	3.6

Table 1: Performance of LNS for simple capacitated problems using various parameter settings. Each problem without time windows was solved three times. Mean *percentages* above the best published solutions are shown.

by around 2% can take a significant period of time for local search procedures. Table 1 shows that the most important parameter is d, the discrepancy limit, which for the problems without time windows, should be kept low. The quality of results begins to tail off for $d > 3$, with complete search performing the worst. The values of the other parameters seem unimportant within this range, except that the results for $a = 1000$ are very slightly worse than for the other two settings of a. We have also observed that values of the determinism parameter determinism parameter D below 3 or over 30 give poorer results.

3.1.1 Best Published Solutions

Table 2 compares the performance of the best solutions obtained by LNS compared with the best published solutions. A +, -, or = is used to indicate whether LNS bettered, could not match, or matched the best published solution. LNS has tied the best published solution in 8 of the 18 cases, bettered it in 3, and not attained it in 7 of the cases. The maximum deviation (around 1.5%) from the best was in problem TAI385, with 385 customers. We conjecture that because of the size of this problem, a longer running time is required to reduce cost to nearer the optimum than we allowed in our experiments.

3.2 VRPs with Time Windows

We performed experiments on some of Solomon's instances [23], which are the classic benchmark vehicle routing problems with time windows. Each problem has a single depot, an unlimited number of identical vehicles, 100 customers, time windows and capacity constraints. Additionally, a scheduling horizon is defined by placing a time deadline on the return time to the depot.

The problems are divided into two main classes. These classes, which we shall term "series 1" and "series 2" have different scheduling horizons. The series 1 problems have a shorter scheduling horizon than those of series 2, and so vehicle routes in series 2 are longer than those of series 1. On average, around 3 vehicles are required to serve the 100 customers in the series 2 problems, whereas around 12 are needed for series 1.

Experiments were performed only on the series 1 instances, of which there are 29 individual problems. For the series 2 problems, the re-insertion procedure was not able to optimize the insertion of the large number of visits required to reduce the number of routes to 4 or under.[4]

The series 1 Solomon's problems are split into subclasses R1, C1, and RC1. The R1 class consists of 12 problems with customers distributed at random. The C1 class has 9 problems with

[4] As future work, we plan to tackle this problem by starting the search from a point delivered by a good construction heuristic (not one with one visit per route), and provide some guidance in the cost function to encourage at least one short route. In this way, less visits will need to be re-optimized in order to reduce the number of routes by one.

Problem	Best	LNS	
C50	524.61	524.61	=
C75	835.26	835.26	=
C100	826.14	826.14	=
C100B	819.56	819.56	=
C120	1042.11	1042.97	-
C150	1028.42	1032.61	-
C199	1291.45	1310.28	-
TAI100A	2047.90	2047.90	=
TAI100B	1940.61	1939.90	+
TAI100C	1407.44	1406.86	+
TAI100D	1581.25	1586.08	-
TAI150A	3055.23	3055.23	=
TAI150B	2727.99	2732.27	-
TAI150C	2362.79	2361.62	+
TAI150D	2655.67	2661.72	-
F71	241.97	241.97	=
F134	1162.96	1162.96	=
TAI385	24435.5	24816.0	-

Table 2: Comparison of best solutions obtained by LNS against best published solutions for simple capacitated problems.

customers clustered in defined areas. The RC1 class has 8 problems with a mixture of clustered and randomly placed customers. Travel times are equal to the Euclidean distance between customer visits. The objective function for vehicle routing problems with time windows is normally a hierarchical one: minimize the number of vehicles, and within this, minimize total travel distance. We did this by associating a high cost (1000) with the use of each vehicle. LNS then automatically reduces vehicles when it can.

We performed the same parametric analysis as for problems without time windows. Again, a time limit of 900 seconds was placed upon LNS. We chose as an initial solution one customer visit assigned to each of 100 vehicles. Initially, the neighbourhood size was such that one customer visit was removed and re-inserted.

The average percentages above the best published values for problems with time windows are shown in table 3. Since we now take vehicles into account, we show the average percentages of vehicles (left) and distance (right) above the best published solution. Only results for classes $R1$ and $RC1$ are included in the table. Results for class $C1$ were not included as these problems turned out to be very easy. All runs over all parameter settings produced the best published solution to all C1 problems bar one: problem C104. For problem C104, 84% of the solutions generated were the best published solution. Including class C1 in the table would have skewed the results.[5]

Results again indicate that LNS performs well, attaining average solutions (in terms of numbers of vehicles) from just over 3% to just over 6% from the best published solution, depending on parameter settings. Distances were generally under around 1% from the best published solution, for all but the smallest discrepancy limits. Again, we see that by setting parameters judiciously, one can almost half the difference between the objective attained by LNS and that of the best published solutions.

For problems with time windows, we now see that larger numbers of discrepancies provide good solutions, the best solutions being produced by discrepancy limits of 5 or 10. There are

[5] Further evidence of the easiness of some of the C1 benchmarks is that the problems known as C101 and C102 are so easy that our branch and bound procedure applied to the whole 100 customer problem can prove the optimal solution to both problems as 10 vehicles, distance 828.94 in a few seconds. This refutes previous results in [6] which claim the optimal has distance 827.3 using 10 vehicles. However, this work uses distances truncated at the first decimal place, leading to the error.

		discrepancies													
att.	det.	0		1		2		3		5		10		∞	
250	5	5.4	1.9	4.4	1.0	3.7	0.7	4.2	0.6	3.8	0.9	3.3	0.4	3.7	0.4
	10	5.6	2.0	4.2	1.5	3.8	1.0	3.8	0.8	3.2	0.6	4.1	0.4	3.4	0.4
	15	5.0	2.8	3.5	1.6	4.1	0.9	3.6	0.7	3.3	0.6	3.6	0.3	3.3	0.6
500	5	5.6	1.4	4.4	0.8	4.3	0.7	4.2	0.5	3.8	0.6	3.6	0.2	3.9	0.4
	10	5.2	2.1	3.8	1.3	3.5	0.7	3.7	0.6	4.4	-0.1	4.4	0.1	4.2	0.4
	15	5.0	2.2	4.7	0.7	3.7	0.5	3.7	0.7	3.4	0.4	3.4	0.1	4.0	0.1
1000	5	6.1	1.3	5.0	0.8	5.1	0.3	5.3	0.4	5.0	0.5	4.2	0.1	5.0	0.4
	10	6.1	1.3	4.8	0.6	5.1	0.4	4.3	0.8	5.4	0.2	3.4	0.4	4.1	0.6
	15	5.8	2.1	4.7	0.8	5.1	0.3	4.2	0.6	5.7	0.1	3.5	1.1	4.4	0.3

Table 3: Performance of LNS for VRPs with time windows using various parameter settings. Each problem was solved three times. Mean *percentages* of vehicles (left) and distance (right) above the best published solutions are shown.

simple reasons for this. First, when more constraints are present, more pruning occurs, making more intensive search cheaper than for problems with no time windows (this effect can also been observed in [4, 18]). Second, our farthest insertion heuristic is likely to work much better when time windows are not present, as it does not take these constraints into account. (This is important, as LDS finds solutions with less discrepancies when using a good heuristic which makes few mistakes.) Finally, the objective is (primarily) to reduce the number of vehicles, which can cause problems with a low discrepancy limit. To reduce vehicles, often a packing of visits into a route is required that respects time windows, but does not necessarily result in a low distance route. In this case, our farthest insertion heuristic provides poor guidance for re-inserting visits in order to reduce the number of routes.

The limit a on the number of consecutive unsuccessful attempts needed before increasing the number of customer visits to re-insert again plays a role. Here, it can be seen that with $a = 1000$ results are poorer than for $a = 250$ or $a = 500$. The value of a has a greater impact for problems with time windows, as the re-insertion process is slower for these problems. (The propagation methods for time windows are operating, and this slows things down.) With $a = 1000$, only a small amount of search is done when medium to large numbers (> 15) of visits are being re-inserted. When this is so, the methods are not competitive with runs using smaller values of a. This implies that removal and re-insertion of smaller numbers of visits is insufficient for very good solutions to the VRP with time windows. The determinism parameter was less important to the end results, but more good solutions are produced for $D = 10$ and $D = 15$ than for $D = 5$.

3.2.1 Best Published Solutions

Table 4 compares the best solutions obtained by LNS with the best published cost values. The best published results are taken from [6, 20, 21, 24, 26]. We show in the table the best published solution, and the best solution obtained by either Rochat and Taillard [21] (hereafter referred to as RT) or Taillard *et al.* [24] (hereafter referred to as TAI) if the best published solution was not generated by RT or TAI. We do this as RT and TAI use real, double precision distances, as opposed to some of the other methods. As also stated in [24], a consequence of using limited precision distances is that solutions found using these methods may not be feasible when higher precision distances are used to check their validity. A +, -, or = is used to indicate whether LNS bettered, could not match, or matched the best solution from RT or TAI.

LNS has tied RT or TAI in 16 of the 29 cases, bettered them in 10, and not matched them in 3 of the cases. In two of these three cases, the number of vehicles could not match those generated by TAI. All new best solutions produced by LNS are available on the GreenTrip web page at http://www.math.sintef.no/GreenTrip.

Problem	Best Published		Best of RT & TAI		LNS		
C101	10	827.3	10	828.94	10	828.94	=
C102	10	827.3	10	828.94	10	828.94	=
C103	10	828.06			10	828.06	=
C104	10	824.78			10	824.78	=
C105	10	828.94			10	828.94	=
C106	10	827.3	10	828.94	10	828.94	=
C107	10	827.3	10	828.94	10	828.94	=
C108	10	827.3	10	828.94	10	828.94	=
C109	10	828.94			10	828.94	=
R101	18	1607.7	19	1650.80	19	1650.80	=
R102	17	1434.0	17	1486.12	17	1486.12	=
R103	13	1207	13	1294.24	13	1292.68	+
R104	10	982.01			9	1007.31	+
R105	14	1377.11			14	1377.11	=
R106	12	1252.03			12	1252.03	=
R107	10	1126.69			10	1104.66	+
R108	9	968.59			9	963.99	+
R109	11	1214.54			11	1197.42	+
R110	11	1080.36			10	1135.07	+
R111	10	1104.83			10	1096.73	+
R112	10	953.63			10	953.63	=
RC101	14	1669	14	1696.94	14	1696.95	-
RC102	12	1554.75			12	1554.75	=
RC103	11	1110	11	1262.02	11	1261.67	+
RC104	10	1135.83			10	1135.48	+
RC105	13	1643.38			14	1540.18	-
RC106	11	1448.26			12	1376.26	-
RC107	11	1230.54			11	1230.48	+
RC108	10	1139.82			10	1139.82	=

Table 4: Comparison of best solutions obtained against best published solutions for Solomon's problem

3.2.2 Comparison of Improvement Over Time

In both RT and TAI, tables of the mean number of vehicles and distances as the search progresses are given. We use this opportunity to compare LNS with these two approaches. However, their approaches use more CPU time than the experiments we have conducted so far (even accounting for differences in machine performance). We therefore performed a longer run using parameter settings we considered as reasonable from examination of table 3. We solved all problems in R1 and RC1 (we did not examine class C1 as it has already proved to be so easy) using a time limit of 1 hour. We solved each problem 6 times, with different random seeds: for three of these we used a discrepancy limit d of 5, and for the others, we set $d = 10$. We set $a = 250$ and $D = 15$.

The results are shown in tables 5 and 6. The tables show, for each method, the CPU time used at three points during the algorithm, and the mean solution quality for each class at that point. This quality is expressed as the mean number of vehicles used per problem over the class, and the mean distance travelled per problem over the class. We use a faster machine than either RT or TAI, and so to provide a fairer comparison, we have divided their times by the ratio of our clock rate to theirs.[6]

We can see that the results for $d = 5$ are significantly better than those for $d = 10$, indicating

[6] This is by no means perfectly correct, but should give a better comparison of the resources used by each method.

that a smaller discrepancy is adequate to provide very good solutions. Clearly, just as for problems with no time windows, there is a balance to be struck between the effort expended in the re-insertion procedure and the number of re-insertions applied per second. However, it does appear that the problems with time windows require a higher discrepancy limit than those without (around 5 as opposed to around 2). Table 5 shows how well a simple constraint programming technique such as LNS performs in comparison with the best Operations Research meta-heuristic techniques: the number of vehicles and distance is reduced to approximately the same level as TAI using a roughly equivalent amount of CPU time.

	RT			TAI			LNS		
Class	CPU	Quality		CPU	Quality		CPU	Quality	
R1	315	12.83	1208.43	803	12.64	1233.88	900	12.45	1198.37
	909	12.58	1202.31	2408	12.39	1230.48	1800	12.35	1201.47
	1888	12.58	1197.42	4816	12.33	1220.35	3600	12.33	1201.79
RC1	301	12.75	1381.33	656	12.08	1404.59	900	12.05	1363.67
	909	12.50	1368.03	1969	12.00	1387.01	1800	12.00	1363.68
	1818	12.38	1369.48	3938	11.90	1381.31	3600	11.95	1364.17

Table 5: Comparison of solution quality over time for Solomon's problems (long run, with a discrepancy limit of d=5)

	RT			TAI			LNS		
Class	CPU	Quality		CPU	Quality		CPU	Quality	
R1	315	12.83	1208.43	803	12.64	1233.88	900	12.48	1196.07
	909	12.58	1202.31	2408	12.39	1230.48	1800	12.45	1195.30
	1888	12.58	1197.42	4816	12.33	1220.35	3600	12.42	1195.71
RC1	301	12.75	1381.33	656	12.08	1404.59	900	12.05	1360.89
	909	12.50	1368.03	1969	12.00	1387.01	1800	12.03	1358.40
	1818	12.38	1369.48	3938	11.90	1381.31	3600	12.00	1358.26

Table 6: Comparison of solution quality over time for Solomon's problems (long run, with a discrepancy limit of d=10)

4 Conclusion

A method analogous to the shuffle of job-shop scheduling, which we term Large Neighbourhood Search has been applied to vehicle routing problems. The method operates by continually removing and re-inserting customer visits using a constraint programming technique. This type of search is very naturally suited to constraint programming technology, which allows very general models of combinatorial problems to be specified. The search method thus seems ideal for models involving complex real-world constraints.

We have found that both the choice of customer visits for re-insertion, and the re-insertion process affect the quality of solutions obtained. We used a “relatedness” method which is a simple way of controlling which visits are re-inserted, and Limited Discrepancy Search within the re-insertion process.

The power of LNS has been demonstrated by its application to vehicle routing benchmark problems. LNS has been shown to be extremely competitive with the best Operations Research meta-heuristic methods, while being significantly simpler. LNS is competitive both in its average performance, and in terms of its ability to produce new best solutions not found by other methods.

We have demonstrated that constraint programming methods are equally as good as Operations Research methods on these standard benchmark problems. We believe, however, that the constraint programming method holds more promise for tackling real-world problems due to its ability to effectively address side constraints. We now wish to move away from expending vast amounts of time and energy on slightly improving the solutions to some benchmarks. Effectively addressing side constraints faced by industry is a far more challenging goal.

Acknowledgement

I wish to thank members of the APES group for their thought provoking conversations, and Ian Gent in particular for encouraging me to write this paper.

The production of this paper was supported by the GreenTrip project, a research and development undertaking partially funded by the ESPRIT Programme of the Commission of the European Union as project number 20603. The partners in this project are Pirelli (I), ILOG (F), SINTEF (N), Tollpost-Globe (N), and University of Strathclyde (UK).

References

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

[2] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal On Computing*, 3:149–156, 1991.

[3] Y. Caseau and F. Laburthe. Disjunctive scheduling with task intervals. Technical report, LIENS Technical Report 95-25, École Normale Supérieure Paris, France, July 1995.

[4] Y. Caseau and F. Laburthe. Solving small TSPs with constraints. In L. Naish, editor, *Proceedings the 14th International Conference on Logic Programming*. The MIT Press, 1997.

[5] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. *Combinatorial Optimization*, pages 315–338, 1979.

[6] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problems with time windows. *Operations Research*, 40(2):342–354, 1992.

[7] M. Fisher. Optimal solution of vehicle routing problems using minimum K-trees. *Operations Research*, 42:626–642, 1994.

[8] P. A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction probems. In *Proceedings of the 10th ECAI*. John Wiley & Sons, 1992.

[9] I. Gent, E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proceedings of CP-96*. Springer, 1996.

[10] I. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In *Proceedings of the 13th AAAI*. The MIT Press, 1996.

[11] I. Gent and T. Walsh. From approximate to optimal solutions: Constructing pruning and propagation rules. In *Proceedings of the 15th IJCAI*, 1997.

[12] W. D. Harvery and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the 14th IJCAI*, 1995.

[13] P. Kilby, P. Prosser, and P. Shaw. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. Submitted to the Special Issue of the Journal "Constraints" on Industrial Scheduling, 1998.

[14] G. A. P. Kindervater and M. W. P. Savelsbergh. Vehicle routing: Handling edge exchanges. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 337–360. Wiley, Chichester, 1997.

[15] T. Mautor and P. Michelon. MIMAUSA: A new hybrid method combining exact solution and local search. In *Proceedings of the 2nd International Conference on Meta-heuristics*, 1997.

[16] Pedro Meseguer and Toby Walsh. Interleaved and discrepancy based search. In *Proceedings of the 13th European Conference on AI—ECAI-98*, 1998. To appear.

[17] G. Pesant and M. Gendreau. A view of local search in constraint programming. In *Proceedings of CP '96*, pages 353–366. Springer-Verlag, 1996.

[18] G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 1998. To appear.

[19] G. Pesant, M. Gendreau, and J.-M. Rousseau. GENIUS-CP: A generic single-vehicle routing algorithm. In *Proceedings of CP '97*, pages 420–433. Springer-Verlag, 1997.

[20] J.-Y. Potvin and S. Bengio. A genetic approach to the vehicle routing problem with time windows. Technical Report CRT-953, Centre de Recherche sur les Transports, University of Montreal, 1994.

[21] Y. Rochat and E. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, 1995.

[22] M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

[23] M. M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35:254–265, 1987.

[24] E. Taillard, P. Badeau, M. Gendreau, F. Guertain, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 32(2), 1997.

[25] E. D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–676, 1993.

[26] S. R. Thangiah, I. H. Osman, and T. Sun. Hybrid genetic algorithm, simulated annealing, and tabu search methods for vehicle routing problems with time windows. Working paper UKC/OR94/4, Institute of Mathematics and Statistics, University of Kent, Canterbury, 1994.