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

Paul Shaw*

ILOG S.A.
9, rue de Verdun, BP 85
94253 Gentilly Cedex, FRANCE.
shaw@ilog.fr

Abstract. We use a local search method we term Large Neighbourhood Search (LNS) to solve vehicle routing problems. LNS is analogous to the shuffling technique of job-shop scheduling, and so meshes well with constraint programming technology. LNS explores a large neighbourhood of the current solution by selecting a number of "related" customer visits to remove from the set of planned routes, and re-inserting these visits using a constraint-based tree search. Unlike similar methods, we use Limited Discrepancy Search during the tree search to re-insert visits. We analyse the performance of our method on benchmark problems. We demonstrate that results produced are competitive with Operations Research meta-heuristic methods, indicating that constraint-based technology is directly applicable to vehicle routing problems.

1 Introduction

A 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 produce a low cost routing plan specifying for each vehicle, the order of the customer visits they make. (In academic problems cost is generally proportional to the number of vehicles, or total travel distance/time.) Industrial VRPs tend to be large, and so local search techniques are used extensively as they scale well and can produce reliably good solutions.

Constraint programming appears to be a good technology to apply to VRPs because of the ubiquity of complex constraints in real problems, such as legislation on driver breaks, or complex pay provisions. However, search in constraint programming is usually based upon complete tree-based techniques, which can at the present moment only solve problems of up to 30 customers reliably.

A natural conjecture is that a combination of local search and constraint programming should work well for VRPs. Such a method would hopefully provide the advantages of both: *exploration* and *propagation*.

* This work was carried out while the author was working in the Department of Computer Science, University of Strathclyde, as part of the APES research group. The author wishes to thank all members of APES for their help and support.

M. Maher and J.-F. Puget (Eds.): CP'98, LNCS 1520, pp. 417-431, 1998.

We apply a technique we refer to as Large Neighbourhood Search (LNS) to VRPs. LNS makes moves like local search, but uses a tree-based search with constraint propagation to evaluate the cost and legality of the move. The moves made are generally very powerful, changing a large portion of the solution. The potential for changing large parts of the solution gives LNS its name, as a neighbourhood's size typically varies exponentially with the number of basic elements of the solution changed by the move.

One way of applying LNS to a VRP is by defining a move to be the removal and re-insertion of a set I of customer visits. We define a "relatedness" measure between customer visits and use this as a basis for choosing the set I at each step. We use Limited Discrepancy Search (LDS) to re-insert the customer visits into the current set of routes. The size of I increases over time, stepping up when search is deemed to have stagnated at the current size of set I.

Experiments are carried out on benchmark problems, with and without time windows. We analyse solutions produced by LNS over a range of parameter settings. LNS is shown to have excellent average performance and produces many new best solutions to these benchmark problems.

The paper is organised as follows: Section 2 describes LNS as applied to the VRP, compares it with related work, and assesses its benefits. Section 3 presents computational experiments on benchmark problems. Section 4 concludes.

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 *re-optimised* over the relaxed positions (by re-inserting these visits). One iteration of removal and re-insertion can be considered as the examination of a neighbourhood move. If a re-insertion is found that results in a cost below that of the best routing plan found so far, this new solution is kept as the current one.

The re-insertion process uses heuristics and constraint propagation. The *minimum* cost re-insertion can be evaluated via branch and bound, or techniques that only partially explore the search tree can be used.

Two factors affect the way in which LNS operates when applied to the VRP: how customer visits are chosen for removal, and the re-insertion process. These are now examined in more detail.

2.1 Choosing Customer Visits

We describe a possible method for choosing the customer visits that are removed and re-inserted. We would not be surprised if better techniques are found. However, we believe in a general choice strategy: that of choosing *related* visits. *Related* has to be suitably defined. A good measure is one that results in opportunities for the re-insertion to improve the routing plan. *i.e.* the measure should discount (by labelling as unrelated or loosely related), sets of visits that

are likely to maintain their previous positions when re-inserted. There is no point in removing visits whose re-insertion is independent of the others', and the relatedness concept attempts to capture this.

One observation is that visits geographically close to one another are more related than remote ones. (Alternatively, visits that it is cheap to travel between should be more related than those with a high travel cost.) It is unlikely that remote visits will have inter-changes in position due to the high costs involved.

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 when reducing the number of vehicles used is important, as removing all visits from a route is the only way to reduce the number of routes. (This happens when all visits are removed from a route, 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]

Here, for simplicity, we assume a *binary* relatedness operator $\mathcal{R}(i,j)$ taking two visits and delivering a non-negative value indicating how closely they are related. Ideally, this function should include domain knowledge about side constraints (*e.g.* see section 2.4 for a discussion of a pickup and delivery example).

We do not address problems with side constraints[2] here, and define:

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

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 different vehicles *and* reduction of the number of vehicles is important in the cost function (see section 3.2 on problems with time windows). V_{ij} evaluates to 0 otherwise. We assume that all c_{ij} are normalised in the range [0..1].

Figure 1 describes how visits are chosen. If the relatedness measure $\mathcal{R}(i,j)$ perfectly captured which visits should be removed together, then one would imagine that visits should be drawn from the routing plan using only the relatedness concept. In reality, the relatedness measure is never perfect, and relying on it too heavily can cause search to be too short-sighted (for example, see section 3.1). We therefore include a random element. In the algorithm, D 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. In between, there is a mixture. (Also note that there is always *some* random element to the search even at $D = \infty$, as a *random* visit maximally related to a previously relaxed visit is chosen.)

There are other ways that $\mathcal{R}(i,j)$ could be used to choose customer visits. 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. Moreover, the ranking system is not ideal when the relatedness of some pairs of visits is much larger than others (this is addressed in [10]).

[1] For the job-shop scheduling problem, [4] uses a shuffling technique (analogous to LNS) that relaxes the start times of all operations within a certain range.

[2] See [10] for a study of LNS when side constraints are added.

```
RemoveVisits(RoutingPlan plan, integer toRemove, real D)
    VisitSet inplan := GetVisits(plan)
    Visit 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
        // Visits are ranked in decreasing order of relatedness
        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(|lst|*rand^D)]
        removed := removed + {v}
        inplan := inplan - {v}
    end while
end RemoveVisits
```

Fig. 1. How visits are removed using relatedness

Control of the Neighbourhood Size For efficiency, one wants to remove the smallest set of visits that will improve 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 in the cost, increase r by one. An upper limit of 30 was placed on the value of r. This scheme increases r only when the search has deemed to have become "stuck" for the smaller value of r. The value of a determines how stubbornly LNS seeks improvements at smaller values of r. [3] used a similar technique for shuffling in job-shop scheduling.

2.2 Re-inserting Visits

The re-insertion process uses branch and bound, with constraint propagation and heuristics for variable and value selection. The upper bound is set to the cost of the best solution found so far. In its simplest form, the search examines the whole tree for the re-insertion of all visits at minimum cost.

We view each of the relaxed (removed) visits as the constrained variables, that can take values corresponding to their available insertion points. (An insertion point is a point between two adjcacent visits in the same route that may accommodate the visit.) For any particular visit, some insertion points may be ruled out as illegal via simple propagation rules. For example, 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, propagation rules maintain the load

on the vehicle, and bounds on start of service time for all visits along a route, based on the pickup quantity, the travel times between visits, and customer time windows. For a detailed description of such rules, see [15].

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 as the *current* cost of the routing plan. (We do not compute a lower bound on the cost of including as yet unrouted visits.) This makes the procedure fast, but the search tree is larger than it would be if the bound was better. Improving the bound is a subject of future work.

Branching Heuristics We follow the general rules of of "most constrained variable", "least constrained value" to choose a visit to insert and its insertion point. A visit could be considered constrained if it is far from the rest of the routing plan (and so will bring the cost of the plan more quickly towards the upper bound when inserted). When choosing a position, we could consider an insertion point less constraining 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 a heuristic, we choose visit v to insert for which the cost $\mathcal{C}_{c_v}$ of its cheapest insertion is largest. This choice of visit is known as the *farthest insertion* heuristic. We then try to insert this visit at each of its insertion points, cheapest to most expensive, in increasing order. The sub-problem of inserting the remaining visits is solved after each insertion of v. If any visit has only one legal insertion point, it is immediately inserted at this point. This is performed as a propagation rule: a so-called "unit propagation".

The farthest insertion heuristic works well, but like any heuristic, has its problems. The main one is it only addresses one constraint: the bound on the cost function. When other constraints are added, its guidance is poorer. Ideally, one wants the heuristic to take account of all (or the more important) constraints.

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 removed visits in a few seconds for problems with time windows. For problems without time windows, the optimal re-insertion for only around 15 visits can be computed in this time as the reduced number of constraints results in less pruning of the search space. Unfortunately, the distribution of solution times has a heavy tail, and some re-insertions take a long time to compute. To alleviate this problem, we used Limited Discrepancy Search [9] (LDS). 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 its *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 re-visiting leaf nodes. Our re-insertion algorithm is shown in figure 2. The management of legal insert positions is not mentioned—we assume they are handled by automatically triggered propagation rules, as previously discussed. The parameter d trades the coverage of the the search tree with the speed of re-insertion. When d is small, we opt for large numbers of attempted re-insertions with little search tree coverage for each one. For high values of d, the opposite situation holds. The presence of a "trade off" is investigated in section 3.

```
Reinsert(RoutingPlan plan, VisitSet visits, integer discrep)
    if |visits| = 0 then
        if Cost(plan) < Cost(bestplan) then
            bestplan := plan
        end if
    else
        Visit v := ChooseFarthestVisit(visits)
        integer i := 0
        for p in rankedPositions(v) and i ≤ discrep do
            Store(plan) // Preserve plan on stack
            InsertVisit(plan, v, p)
            Reinsert(plan, visits - v, discrep - i)
            Restore(plan) // Restore plan from stack
            i := i + 1
        end for
    end if
end Reinsert
```

Fig. 2. How visits are re-inserted

2.3 Related Work

LNS is analogous to the shuffling technique used in job-shop scheduling [1, 4]. To perform a shuffle, start times for operations on the majority of machines are relaxed, and a tree-based search procedure reconstructs the schedule. We have not used the term "shuffling" in this paper to avoid confusion with job-shop scheduling. Moreover, the basic idea is easily generalizable to other problem classes, where the natural visualization of the move is not a shuffling of positions.

In [1], a simple shuffle is presented, but in [4], various types of shuffle are used. The shuffles differ by the criteria for selecting the operations that will be relaxed. These selections use 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 reconstruct the schedule by limiting the number of backtracks available. We tried

such an approach, but it proved inferior to LDS, with virtually no advantages in implementation simplicity or efficiency.

Other work on routing problems [14, 16] also advocates the use of constraint programming within local search. Here, move operators from the routing literature are used (for instance generalised insertion), but a constraint programming branch and bound search evaluates the neighbourhood to find the best legal move. One can see the similarities with LNS, but there are differences. The main one is that only *traditional* move operators are being used, and so constraint programming only improves the efficiency of the evaluation of the neighbourhood, and not the power of the move operators themselves. Secondly, the whole neighbourhood is being explored, requiring a complete branch and bound search. With LNS, any method can be used to perform the re-insertion of visits, for instance, a heuristic method, local or complete search, LDS, or another discrepancy-based approach (*e.g.* [13]).

Some work has been performed in solving quadratic assignment problems using a similar technique to LNS [12].

Constraint programming and local search were applied to routing problems in [2], using a technique of filtering out certain moves which violate core constraints, allowing the constraint engine to check the remainder. The operation of the method is therefore unlike LNS.

2.4 Discussion

There are advantages to using LNS over traditional local search. The main advantage is that side constraints can be better handled. For instance, in the VRP, different models such as the pickup and delivery problem (PDP) can be easily dealt with. Using traditional local search, this is more difficult: special-purpose operators need to move both the pickup and delivery to a different route simultaneously. With LNS, the pickup and delivery are simply made strongly related. The normal PDP constraints of *same vehicle* and *time(pickup)* $<$ *time (delivery)* then constrain where the visits can be re-inserted. Making these visits highly related is important here. If only one visit of the pair was removed, it would be highly restricted in where it could be re-inserted.[3]

Kindervater and Savelsbergh [11] discuss ways of efficiently introducing side constraints into local search. Their methods, however, are complex and dedicated to particular move operators and side constraints. Moreover, they do not suggest 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 local search move operations normally used (such as moving a single visit to a new position) will be illegal due to violation of these constraints. Increasing

[3] The above discussion brings about a difficulty with LNS. When many different *types* of side constraints are operating, how strongly should any constraint relate visits in comparison to the others? This question, one of tuning, seems to plague all sufficiently complex heuristic algorithms. Automatically determining relative relatedness values is a subject of future work.

numbers of side constraints constantly reduce the number of feasible moves. This can make local search difficult, as the search space becomes more restricted or even disconnected. LNS alleviates this problem somewhat by providing far-reaching move operators that allow the search to move over barriers in the search space created by side constraints.

In local search, evaluation of cost differences is time consuming. In idealised models, one often uses travel distance as the cost function, since for most simple moves, cost differences can be computed in constant time. (Savelsbergh [19] has also introduced ways of computing route time differences for such operators in constant time.) However, for real VRPs, cost functions are seldom this simple. With LNS, the full cost of a move is 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 position operate on cost differences. Since this information is just a *hint* to the search, and most cost functions are generally related to distance, we can simply use distance as an approximation.

3 Computational Results

We report the results of applying LNS to benchmark problems with and without time windows. All problems have an unlimited number of identical limited capacity vehicles located at a single depot. Time and distance between customers is Euclidean. Each customer has a specified load, and for problems with time windows, a service time and a time window during which it must be visited. For all problems, we chose an initial solution with the number of vehicles equal to the number of customers, with one customer visit performed by each vehicle. The initial neighbourhood size is set so that only one visit is removed and re-inserted. We examine the quality of solutions obtained by LNS over various parameter settings. We also report new best solutions obtained. Finally, we perform CPU-intensive runs of LNS to compare results with the best Operations Research methods. We used a 143 MHz Ultra Sparc running Solaris for all experiments. All code was written in C++ and compiled using the Sun C++ compiler.

3.1 Capacitated VRPs

Following [18], we use three types of capacitated VRP: classic test problems, non-uniform problems, and those derived from real data. The classic problems (C50, C75, C100, C100B, C120, C150, C199) are due to [6]. The non-uniform problems (TAI100A–TAI100D, TAI150A–TAI150D) were created by Rochat and Taillard [18] to capture structure inherent in real problems: loads are exponentially distributed, and customers are realistically clustered. Finally, problems reflecting real data are taken from [8] (F71 and F134) and [22] (TAI385). In all these problems, the number in the name indicates the number of customers. The objective is to minimise the total distance travelled.

LNS as presented has 3 parameters. First, we can vary the number of discrepancies d used by LDS. Second, we can vary the number of unsuccessful moves a that must be made to increase the number of visits to re-insert. Finally, we can change the determinism parameter D. We chose $d \in \{0, 1, 2, 3, 5, 10, \infty\}$ ($d = \infty$ performs complete search), $a \in \{250, 500, 1000\}$, and $D \in \{1, 5, 10, 15, 30, \infty\}$ ($D = 1$ ignores relatedness, while $D = \infty$ uses maximal relatedness).

For each combination of parameter settings, we ran LNS three times (with different random seeds) on all problems, with a time limit of 900 seconds. When $d = \infty$ some re-insertions can take a long time, and so a time limit of 20 seconds was placed on the re-insertion process, which is in force for all experiments. (The number of timeouts that occurred for $d \leq 10$ was negligable.)

Table 1 shows the results of running LNS over all capacitated problems: 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 them by the cost of the best published solution for the corresponding problem. 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 costs produced by LNS are close to the best published ones. All average results are within 7% of the best published solution, and for the best parameter settings, 2.2% from the best published solution on average. The number of attempts a has the smallest impact on the quality, but results for $a = 1000$ are slightly worse. From examination of, for instance, the results for $a = 250$, it is clear that the worst results are produced at the extremities of the ranges of discrepancies and determinism. Determinism set at 5 or 10 and a discrepancy limit around 2 appear to give the best results. Relatedness is useful, as at unit determinism (visits chosen for re-insertion at random), results are poorer. However, over reliance on relatedness also produces a degradation of results (seen at infinite determinism). LDS is also playing a role: at 0 and infinite discrepancies, results are worse than for values such as 2 or 3.

Best Published Solutions Table 2 compares the lowest costs obtained by LNS with the best published ones. A +, -, or = indicates whether LNS bettered, could not match, or matched these solutions. LNS has tied the best in 8 cases, bettered it in 3, and not attained it in 7 of the cases. We attribute the largest deviation of around 1.5% in problem TAI385 to the large problem size. We believe a longer running time is required than we allowed in our experiments.

3.2 VRPs with Time Windows

We performed experiments on some of Solomon's instances [20], the classic benchmark VRPs with time windows. Each problem has 100 customers, time

		discrepancies						
attempts	determinism	0	1	2	3	5	10	∞
250	1	4.3	3.8	4.8	4.2	5.0	5.6	6.0
	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
	30	3.3	2.8	2.7	2.6	3.4	3.2	4.2
	∞	5.5	3.6	4.1	4.3	5.1	5.3	4.5
500	1	5.1	4.4	3.8	4.6	5.0	5.3	5.3
	5	3.0	2.4	2.9	2.7	2.4	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
	30	2.6	2.7	3.0	2.6	3.2	3.9	4.2
	∞	4.9	5.0	4.6	3.9	4.2	5.8	4.6
1000	1	4.8	5.3	5.2	4.6	5.3	6.0	5.4
	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
	30	3.5	3.0	2.6	3.7	3.2	3.4	4.1
	∞	6.1	4.4	5.0	4.6	5.7	6.0	5.2

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

windows and capacity constraints. 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: "series 1" and "series 2" with different scheduling horizons. The series 1 problems have a shorter scheduling horizon than those of series 2. 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 problems, of which there are 29. For the series 2 problems, the re-insertion procedure was not able to optimise the insertion of the large number of visits required to reduce the number of routes to 4 or under. As future work, we plan to tackle this problem by providing some guidance in the cost function to encourage at least one short route. In this way, less visits will need to be re-optimised to reduce the number of routes.

The series 1 problems are split into subclasses: R1, with customers distributed randomly, C1, with customers in well-defined clusters, and RC1, with a mixture. The objective function for VRPs with time windows is normally a hierarchical one: minimise the number of vehicles, and within this, minimise total travel distance. We associate a high cost with the use of each vehicle, and then LNS automatically reduces vehicles when it can. We performed the same analysis as for problems without time windows.

The average percentages above the best published values are shown in table 3. Since we now take vehicles into account, we show the average percentages of

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	+

Problem	Best	LNS	
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	=

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

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 are easy. Nearly all runs produced the best known solution to their corresponding problem and so including class C1 in the table would have skewed the results. Further evidence of the easiness of some of the C1 benchmarks is that problems C101 and C102 can be solved to optimality by our branch and bound insertion procedure in a few seconds. The optimal solution to both problems is 10 vehicles, distance 828.94, correcting previous results in [7] that claim the optimal has distance 827.3 using 10 vehicles. [7] uses distances truncated at the first decimal place, leading to the error. We use real-valued distance and time values.

Results again indicate that LNS performs well, attaining average solutions (in terms of numbers of vehicles) as good as just over 3% from the best published solution. However, some quite bad solutions (up to around 12% from the best published solution) are produced when relatedness is ignored ($D = 1$).

One can see that good solutions are created when the number of discrepancies is higher than for the problems without time windows. There are 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 [5, 15]). Second, our farthest insertion heuristic makes more mistakes for these problems for two reasons: time windows are not taken into account, and the heuristic provides poor guidance in reducing the number of routes. Thus, more discrepancies are necessary to repair heuristic errors.

The attempts a plays an important role—results are poorer for increasing a. This is because the neighbourhood size is still low by the end of the search. For instance, with $a = 1000$, little search is done with medium to large numbers (> 15) of visits are being re-inserted. The worsening of results as a increases is probably due to the fact that to reduce the number of routes by one, often two or even three routes have to be removed from the routing plan. For the series 1 problems, this means that around 20 visits or more must be removed.

		discrepancies											
att.	det.	0		1		2		3		5		10		∞	
250	1	9.4	2.1	8.5	1.3	9.5	1.4	10.5	1.5	9.5	1.5	12.7	1.4	10.6	1.6
	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
	30	6.6	4.2	3.1	2.2	3.6	0.8	3.3	0.8	3.8	0.1	5.0	-0.0	4.3	0.5
	∞	6.0	4.2	6.0	1.4	4.5	1.6	4.2	0.4	3.7	0.5	3.6	0.6	4.2	0.2
500	1	9.9	1.8	9.0	1.0	10.5	1.4	10.2	1.1	9.2	1.2	8.1	1.7	10.8	1.1
	5	5.6	1.4	4.4	0.8	4.3	0.7	4.2	0.5	4.1	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
	30	5.1	3.0	3.8	1.4	4.7	0.7	3.2	0.7	3.9	0.4	3.4	0.7	4.6	0.0
	∞	6.1	4.2	5.1	1.4	3.9	0.8	4.1	0.5	3.8	0.6	3.7	0.8	3.6	0.7
1000	1	9.9	2.3	11.6	2.1	9.3	1.9	10.4	1.9	13.8	2.1	11.0	1.9	10.0	1.8
	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
	30	6.2	2.4	5.7	0.7	3.9	0.3	4.9	0.4	5.0	0.3	3.9	0.1	5.7	0.1
	∞	7.2	2.7	5.4	2.2	4.1	1.7	4.3	0.8	5.4	0.1	5.2	0.3	5.2	0.3

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

Finally, unlike the problems without time windows, relying heavily on relatedness does not appear to be as detrimental to the quality of results. Without time windows, cost rose noticeably when D was too high, but only a mild increase (if any) can be seen for the problems with time windows. It would thus appear that the relatedness function for problems with time windows (which concentrates on relating visits in the same route) is a good guide.

Best Published Solutions Table 4 compares the best solutions obtained by LNS with the best published ones taken from [7, 17, 18, 21, 23]. We show in the table the best published solution, and the best solution obtained by either Rochat and Taillard [18] (hereafter referred to as RT) or Taillard *et al.* [21] (hereafter referred to as TAI) if the best published solution was not generated by RT or TAI. We do this as RT and TAI (unlike the others) use real, double precision distances. As stated in [21], 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 = indicates 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, LNS could not match the number of vehicles used by TAI. All new best solutions produced by LNS are available at http://www.math.sintef.no/GreenTrip.

Prob.	Best Pub.	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	=

Prob.	Best Pub.	RT & TAI	LNS	
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 problems

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 approaches. However, these approaches use more CPU time than the experiments reported so far. We therefore performed longer runs of LNS using parameter settings of $a = 250$ and $D = 15$ (which we considered reasonable from examination of table 3). We solved all problems in R1 and RC1 6 times with different random seeds, using a time limit of 1 hour. For half of these runs we set $d = 5$, and for the other half, $d = 10$.

Table 5 shows, 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 to give a better comparison of resources used, have divided their times by the ratio of our clock rate to theirs.

We can see that the results for $d = 5$ are better than those for $d = 10$, and so a smaller discrepancy is better here. LNS performs well 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.

4 Conclusion

Large Neighbourhood Search, a method analogous to the shuffle of job-shop scheduling, has been applied to VRPs. LNS operates by making powerful reinsertion based moves, which are evaluated using constraint programming.

	RT		TAI		LNS				
Class	CPU	Quality	CPU	Quality	CPU	Quality	$(d = 5)$	Quality	$(d = 10)$
R1	315	12.83	1208.43	803	12.64	1233.88	900	12.45	1198.37	12.48	1196.07
	909	12.58	1202.31	2408	12.39	1230.48	1800	12.35	1201.47	12.45	1195.30
	1888	12.58	1197.42	4816	12.33	1220.35	3600	12.33	1201.79	12.42	1195.71
RC1	301	12.75	1381.33	656	12.08	1404.59	900	12.05	1363.67	12.05	1360.89
	909	12.50	1368.03	1969	12.00	1387.01	1800	12.00	1363.68	12.03	1358.40
	1818	12.38	1369.48	3938	11.90	1381.31	3600	11.95	1364.17	12.00	1358.26

Table 5. Comparison of solution quality over time for Solomon's problems

Selecting visits for re-insertion based upon a "relatedness" concept leads to significantly better results than random selection. LDS was used to re-insert visits, giving better results than complete search or limiting the the number of backtracks in depth-first search.

On benchmark problems, LNS is highly competitive with leading Operations Research methods, while being much simpler. Furthermore, we believe LNS holds more promise for real problems than traditional local search methods due to its ability to better address side constraints.

Acknowledgment

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. D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal On Computing*, 3:149–156, 1991.
2. B. De Backer, V. Furnon, P. Prosser, P. Kilby, and P. Shaw. Local search in constraint programming: Application to the vehicle routing problem. In A. Davenport and C. Beck, editors, *Proceedings of the CP-97 workshop on Industrial Constraint-based Scheduling*, 1997.
3. P. Baptiste, C. Le Pape, and W. Nuijten. Constraint-based optimization and approximation for job-shop scheduling. In *Proceedings of the AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems, IJCAI-95, Montreal, Canada*, 1995.
4. Y. Caseau and F. Laburthe. Disjunctive scheduling with task intervals. Technical report, LIENS Technical Report 95-25, École Normale Supérieure Paris, France, July 1995.

5. 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.
6. N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. *Combinatorial Optimization*, pages 315–338, 1979.
7. 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.
8. M. Fisher. Optimal solution of vehicle routing problems using minimum K-trees. *Operations Research*, 42:626–642, 1994.
9. W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the 14th IJCAI*, 1995.
10. 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 *Constraints* Special Issue on Industrial Scheduling, 1998.
11. 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.
12. 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.
13. Pedro Meseguer and Toby Walsh. Interleaved and discrepancy based search. In *Proceedings of the 13th European Conference on AI—ECAI-98*, 1998. To appear.
14. G. Pesant and M. Gendreau. A view of local search in constraint programming. In *Proceedings of CP '96*, pages 353–366. Springer-Verlag, 1996.
15. 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.
16. 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.
17. 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.
18. Y. Rochat and E. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, 1995.
19. M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.
20. M. M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35:254–265, 1987.
21. 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.
22. E. D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–676, 1993.
23. 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.