A new local search algorithm providing high quality solutions to vehicle routing problems

Paul Shaw
APES Group,
Department of Computer Science,
University of Strathclyde,
26 Richmond Street,
Glasgow, Scotland.
email: ps@cs.strath.ac.uk

July 1997

Abstract

This paper describes a new local search algorithm that provides very high quality solutions to vehicle routing problems. The method uses greedy local search, but avoids local minima by using a large neighbourhood based upon rescheduling selected customer visits using constraint programming techniques. The move operator adopted is completely generic, in that virtually any side constraint can be efficiently incorporated into the search process. Computational results show that a naive implementation of the method produces results bettering the best produced by competing techniques using minima-escaping methods.

1 Introduction

In recent years, the method of choice for solving vehicle routing problems has been to use a local search technique. These local search methods have been favoured since they quickly provide solutions to problems of practical size that have not been solved by exact methods. However, because local search techniques only make small changes to the solution, they can only navigate through the state space in constrained ways. This leads to the problem of search becoming trapped in local minima, and hence the widespread use of *minima-escaping* procedures to move the search out of these traps.

The most used minima-escaping procedures (often called meta-heuristics) are simulated annealing [18], tabu search [13, 14], genetic algorithms [16], and hybrids or extensions of these. All these methods have been successfully applied to vehicle routing problems at some stage ([20] gives a bibliography). The power of these types of techniques is that simple heuristics (move operators), coupled with meta-heuristics such as those mentioned can produce good solutions to problems of practical size. The techniques have become so popular that they are used in great numbers of vehicle routing papers published in the OR literature.

However successful the meta-heuristic methods are, one should be careful of applying them universally. For instance, a recent exposition by Gent [12] has shown that bin packing problems solved using a hybrid genetic algorithm [6] are better solved by a human using pencil and paper. This result suggests that the problems are ill-suited to solution by the meta-heuristic, although easy for a human. Humans are generally poorly adapted to making many seemingly meaningless moves (that meta-heuristic methods indulge in), but better adapted to making the *right* moves, perhaps using some foresight. Additionally, where

local search generally uses a fixed neighbourhood, a human optimizer will tend to make up the neighbourhood as they go along. They may sometimes make simple moves and at other times, perhaps when the problem is highly constrained, make more complex or composite moves. These more complex moves allow more search space to be reached from any point, and so reduce the need for minima-escaping or plateau-exploring.

This paper uses the idea of solving vehicle routing problems using a powerful move operator with a large neighbourhood and so less reliance on minima-escaping procedures. The move operator is so powerful, the experiments in this paper have been carried out with no minima-escaping method at all. The paper is organized as follows. Section 2 describes the optimization algorithm by first giving an overview, and then describing both the search method and the move operator. The genericity of the method is then discussed within a vehicle routing context. Section 3 discusses computational experiments on some benchmark routing problems with time windows. Finally, section 4 draws conclusions on the method, discusses its generality as an optimization method for combinatorial problems, and proposes directions for future work.

2 Method

The optimization method used is a simple greedy local search procedure where moves within the neighbourhood are generated in a randomized way. A move is accepted if it reduces the cost, and is rejected otherwise. The method stops after a specified time period. Greedy search has been used because it is simple, fast, and is adequate to provide very good results to the benchmark problems studied.

The move operator is one based on relaxation and reoptimization: the positions of some customer visits within the schedule are *relaxed* and then those positions are *reoptimized*. This is done as follows: A set of customer visits are chosen and removed from the schedule. This set of visits characterizes the move. The visits are then reinserted back into the schedule at optimal cost. This reinsertion is performed using a branch and bound procedure, with the limit on the bound set at the cost of the solution before relaxation took place.

To recap, the search operates by choosing, in a randomized fashion, a set of customer visits. These are removed from the schedule, and then reinserted at optimal cost. The new solution is retained as the best if it is better. This process is repeated as many times as desired. All that now remains to be described is the method for choosing visits to remove and reoptimize, and how the branch and bound insertion procedure operates.

2.1 Choosing Customer Visits

This section describes a possible method for choosing customer visits for relaxation. 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: that of choosing *related* visits. This can be coupled with a diversification element by injecting randomness into the choice.

Related, of course, has to be suitably defined. A measure must be fixed that allows most opportunity for the reinsertion to achieve some improvement in the schedule. One criterion that springs to mind is that related visits will be geographically close to one another. If visits close to one another are removed from the schedule, there is opportunity for interchange and so on. If visits distanced from each other are removed, the best reinsertion point for each will probably be exactly where it came from.

Related visits could also have similar time windows, or be visited at similar times in the current schedule. Likewise, they could have loads which are of similar weight. If two visits occur in the same tour, they are also related. Removing visits from the same tour should be encouraged since removing all visits in a tour is the only way of reducing the number tours. Likewise, close or adjacent positions in the tour could be a factor in the relatedness.

Choose a visit randomly and relax it
while not enough visits relaxed **do**
    Randomly choose a visit v from those already relaxed
    Rank unrelaxed visits with respect to relatedness to v
    Choose a random number r in $[0, 1)$
    Relax the visit that is r^p of the way through the rank
end while

Figure 1: How Visits are Chosen for Removal

We define a relatedness function $\mathcal{R}(i, j)$ between any two visits i and j. 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 two visits (pick up and delivery) are required to complete each order, these two visits should be highly related. This ensures that when one visit is removed from the schedule 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 reinserted.

We have observed that the relatedness function can be varied significantly while still providing good solutions. For the results produced in this paper, we defined $\mathcal{R}(i, j)$ using:

$$\frac{1}{\mathcal{R}(i,j)} = \alpha c_{ij} + \beta|t_i - t_j| + \gamma T_{ij} + \delta|q_i - q_j|$$

where c_{ij} is the cost of getting to j from i (travel distance in this paper), t_i is the start of service at visit i, T_{ij} evaluates to 1 if i and j are served by the same vehicle and 0 otherwise, and q_i is quantity of goods at i. We further assume that c_{ij}, q_i, and t_i have been normalized in the range $[0..1]$. The results in section 3 were produced with $\alpha = 0.75$, $\beta = \delta = 0.1$, and $\gamma = 1$. That is, it was deemed important to remove visits in the same tour and/or visits that are close together, with less emphasis on time and capacity.

The relatedness function can then be used to choose customer visits from the schedule. To encourage exploration, we also include a random element. Figure 1 describes how visits are chosen. In the algorithm, p is a parameter that controls diversification. With $p = 1$, relatedness is ignored and visits are chosen randomly. With $p = \infty$, visits relaxed are maximally related to some other relaxed visit. We have found that $3 \leq p \leq 5$ works well, and used $p = 4$ to produce the results in section 3.

There can, of course, be many other ways in which customer visits are chosen. In what was described above, 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 ranking system is not ideal when the relatedness of some visits is much larger than others.

2.2 Branch and Bound

The branch and bound method is based on techniques used in constraint programming (CP). When performing branch and bound in CP, at each step in the search, a variable is chosen to instantiate and (in the usual case) subproblems are explored in turn with the variable assigned to each of its values. After each assignment, *constraint propagation* occurs. This propagation takes into account all of the problem constraints, and the bound constraint on the cost function. The propagation usually results in the reduction of the domains of the constrained variables, reducing the search space. When one or more constrained variables have no legal values remaining, the current subproblem is insoluble, and the search backtracks. This view is ideally suited to vehicle routing problems with side constraints, since propagation will occur on all of the side constraints, as well as the core constraints such as capacity. Recently, CP propagation methods for the TSP were described in [4].

For our implementation, the branch and bound search used is straightforward. 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 schedule 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). Others can be ruled out if they take the lower bound on the cost of the plan over the allowed limit.

The branch and bound procedure relies on the cost function never decreasing as visits are added to the schedule. The lower bound is formed completely naively as the *current* cost of the schedule. No attempt is made to compute a lower bound on the cost of including as yet unrouted visits. Improving this bound will be a subject of future work.

The efficiency of CP tree search 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. (In the case of branch and bound this will tighten the bound limit early, and make the remainder of search faster.) It is generally believed, and recently been shown [9] that an effective variable ordering strategy is to instantiate more constrained variables first. Of course, *constrainedness* is a problem dependent attribute, but some mathematical descriptions of it can be found for simple random problems [10, 11]. In our context, a variable (visit) could be considered constrained if it has very few available insertion points, or that many of its insertion points incur a high cost.

Less work has been performed on value ordering heuristics, but a study in [7] provides strong evidence to indicate that instantiating a variable with values in order of their "promise" works well. In our context, we could consider a value (insertion point) more promising if it increases the cost less.

Assume that visit v has p_v insertion points, and that its cheapest insertion point results in a cost increase of c_v. In our implementation, we choose a variable for which c_v/p_v is maximized, and then assign it each of its values in order of increasing cost. Note that if for any v, $p_v = 1$, the visit is immediately inserted at its only available point.

Other domain knowledge can also 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 instantiation.

In most cases, the branch and bound reinsertion procedure can find a better solution or prove that none exists for about 25 visits in a few seconds. 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 the search becoming hampered by such problems, a time out of 30 seconds is placed upon the branch and bound search. After this time has expired, the search terminates retaining the best solution it has found so far (if one has been found).

Finally, we would like to make some notes on efficiency. The solution technique has so far been described as relaxing a part of the problem, and then reoptimizing that part. This implies a relaxation procedure, which in general is not easy to perform in a CP framework without incurring a significant overhead. To remove the need for a relaxation procedure, we can instead start from "scratch", and assign all variables *not* in the relaxed set to their previous values, allowing propagation to restrict the domains of the relaxed variables. In the VRP context, this means rebuilding the schedule by inserting visits one at a time that are not in the removed set. This approach may sound inefficient, but it takes an insignificant period of time compared to that consumed by branch and bound.

Another point on efficiency is that as problem sizes grow, there will be very many insertion points to consider for each visit. Therefore, it may be necessary to perform an *incomplete* search by using a heuristic to prune unpromising branches (for instance, those of high cost). Alternatively, one could use a *discrepancy*-based approach [15, 19, 33] to attempt to reach a good solution more quickly. In fact, any search procedure can be used for the re-insertion process, even a local search procedure if so desired. The method does not *rely* on the move operator using complete search, but the backtracking search method

does allow constraints to be efficiently handled via propagation.

2.3 Discussion

The method outlined has some very distinct advantages over methods that make small changes to the schedule, and rely on minima-escaping procedures.

The main advantage is that the technique is completely general. For instance, 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 operators are required to move both the pickup and delivery to a different route. In our case, we simply make the pickup and delivery strongly related (to encourage both them to be reoptimized together), and let the CP constraints of *same vehicle* and *time(pickup)* $<$ *time (delivery)* constrain where the visits can be inserted.

Just about any type of side constraint can be handled efficiently with the new method. All that is needed is a *propagation* method for the side constraint in question. Roughly speaking, one can achieve this by supplying the answer to the question: given visit v has just been inserted at point p, is it still legal (according to this side constraint) to insert w at point q? In this way, after each insertion, illegal insertion points for those visits not yet inserted can be pruned. Implementing such a propagation method is very much easier than constructing a special move operator for visits involved in certain particular constraints. In fact, the move operators themselves may be obscure. The constraint-based method is also a much more general solution.

A difficulty with vehicle routing 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. With the new approach, the large moves allow the search to move over barriers in the search space created by numerous side constraints. Moreover, side constraints produce more constraint propagation, allowing the branch and bound search to narrow the search space even better, so becoming more efficient on these highly constrained problems. In [17], 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.

Evaluation of cost 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 [25] 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 our method, the full cost of a move will be evaluated during constraint propagation (the cost function can be written as a function of constrained variables representing times/distances/weights and so on). This propagation will only re-evaluate parts of the cost function that have been changed by the previous insertion. Cost differences are not used, and there is no need to invent clever methods to compute them.

We did mention, however, that cost differences could be used to *guide* search in variable and value ordering heuristics. Since this guidance 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.

The final advantage of using CP techniques is that one can immediately benefit from advances in constraint technology. There have been advances in recent years in the efficiency of propagation algorithms. This has been both in the general case [2, 3, 26] and in the case of specific common types of constraint, for example [22, 23]. There has also

been progress made on variable ordering heuristics both for search [9], and for constraint propagation algorithms [8].

Although our method works extremely well (see results in section 3), it can still be combined with minima-escaping procedures if desired. Techniques such as simulated annealing and tabu search could be applied, but the fit is not good as it's not obvious how to move to a poorer solution using the branch and bound procedure. (Methods can be constructed, but they are rather *ad hoc*.) A better meta-heuristic would be guided local search (GLS) [29, 31] that includes a penalty in the cost function dependent on solution features. GLS would integrate better with the branch and bound procedure, since only the cost function, and not the local search procedure itself is affected. GLS has already shown to be powerful on routing problems [1, 21] and other combinatorial problems [30, 32].

3 Computational Results

To assess the new method in terms of solution quality, we performed experiments on some of Solomon's instances [27]. Each problem has a single depot, 100 customers, time windows and capacity constraints. We choose as an initial solution one customer visit assigned to each of 100 vehicles.

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 branch and bound procedure was not efficient enough to optimize the insertion of the larger number of visits required to reduce the number of routes to 4 or under. As future work, we will 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 tour. In this way, less visits will need to be reoptimized in order to reduce the number of tours by one.

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 customers clustered in defined areas. The RC1 class has 8 problems with a mixture of clustered and randomly placed customers. Distance and travel time are defined by the Euclidean distance between points.

The algorithm was executed 3 times with different random number seeds on a somewhat outdated DEC Alpha workstation, which performs around 100 MIPS. The optimization was stopped after two hours. The cost function used was based on first, number of vehicles, and second, distance travelled. For each move, twenty-five visits were initially relaxed. This figure was decreased by one for every branch and bound procedure that timed out, and was increased by one for every twenty that did not. This allowed the search to adjust the power of the move operator to the resources available.

In table 1, we show the average results attained by our method compared to the averages reported in [24] and [28], which are, to date, the best two heuristic approaches to the VRP reported. The above references use an identical cost function to ourselves: reduce vehicles, and within this, reduce distance. For each class, an average is taken over all problems in the class, over each of the three runs. This is done both for the number of vehicles, and distance travelled. The CPU times used as checkpoints by [24] and [28] are somewhat arbitrary, and correspond to the time taken for various numbers of iterations of their algorithm. For this reason, they are different for each problem. We have reported results at 7.5 minutes, 15 minutes, 30 minutes, one hour, and two hours. For reference, [24] uses a 100 MHz machine (around the same speed as ours), while [28] use a 50MHz machine (around half

Problem Class	CPU Time (s)	Result	
		Vehicles	Distance
C1	540	10.00	832.59
	1600	10.00	829.01
	3200	10.00	828.45
R1	450	12.83	1208.43
	1300	12.58	1202.31
	2700	12.58	1197.42
RC1	430	12.75	1381.33
	1300	12.50	1368.03
	2600	12.38	1369.48

Rochat and Taillard [24]

Problem Class	CPU Time (s)	Result	
		Vehicles	Distance
C1	2926	10.00	830.41
	7315	10.00	828.59
	14630	10.00	828.45
R1	2296	12.64	1233.88
	6887	12.39	1230.48
	13774	12.33	1220.35
RC1	1877	12.08	1404.59
	5632	12.00	1387.01
	11264	11.90	1381.31

Taillard *et al.* [28]

Problem Class	CPU Time (s)	Result	
		Vehicles	Distance
C1	450	10.00	828.38
	900	10.00	828.38
	1800	10.00	828.38
	3600	10.00	828.38
	7200	10.00	828.38
R1	450	13.19	1259.99
	900	12.61	1214.07
	1800	12.44	1205.76
	3600	12.36	1204.42
	7200	12.31	1205.06
RC1	450	13.21	1479.54
	900	12.33	1392.89
	1800	12.08	1363.04
	3600	12.00	1361.27
	7200	12.00	1360.40

Our method

Table 1: Comparison of Average Performance on Each Class

the speed of ours).

The results show that, as well as our method being a very general one, it also produces extremely good results to benchmark problems. Immediately one can see that for class C1, our method quickly provides solutions that it then cannot improve upon during the rest of the search process. In fact, in every case, the best known solutions to all problems in class C1 were found. This demonstrates that the C1 class is very easy and virtually worthless as a set of benchmarks. Two of the problems known as C101 and C102 are so easy that our naive branch and bound procedure applied to the whole 100 customer visits can prove the optimal solution to both problems as 10/828.94 (10 vehicles, distance 828.94). This refutes previous results in [5] which claim the optimal to be 10/827.3, but use distances truncated at the first decimal place, obviously leading to the error. Because the C1 problem class is so easy, most reasonable local search algorithms can approach or attain the best known solutions, as demonstrated by the good performance of the other two methods for class C1.

For classes R1 and RC1, our method is slow to improve compared to [24], which produces better results up to around the 15 minute mark, after which point our method is superior. We are aware of the reason for this initial slow improvement. Near the beginning of search, it is relatively easy to reduce the cost because there are so many cost improving moves available. However, at this stage, the search is still burdened with reoptimizing over 20 visits, making improvement slow, even when it need not be. A simple fix would be to gradually increase the number of visits to relax and reoptimize as the search progresses.

Problem Name	Previous Best Solution			New Best Solution	
	Vehicles	Distance	Reference	Vehicles	Distance
R104	10	982.01	[24]	9	1007.31
R107	10	1126.69	[28]	10	1104.66
R108	9	968.59	[28]	9	963.99
R109	11	1214.52	[28]	11	1205.96
R110	11	1080.36	[24]	10	1135.07
R111	10	1104.83	[28]	10	1096.73
RC104	10	1135.83	[24]	10	1135.48
RC107	11	1230.54	[28]	11	1230.48

Table 2: New Best Solutions

Investigation of such strategies will be the subject of future work.

Results for class R1 indicate that after the initial period, our method outperforms the other two. Vehicles are reduced over [24], although travel distance is slightly greater. This is to be expected in routing problems with time windows: to save time, one may have to travel out of one's way to next serve a customer who is ready, rather than waiting for the time window to open at a nearby customer. This phenomenon can also be seen in the improvement over time of our method on class R1. Between 3600s and 7200s, a decrease in the number of vehicles used is accompanied by an *increase* in travel distance. [28] reduces vehicles to around the same level as our method, but the distance travelled is slightly greater.

For the class RC1, both our method and [28] outperform [24]. Given sufficient CPU resource, the method of [28] appears to be slightly better at reducing the numbers of vehicles than our method. Our method is extremely competitive, however, reducing distance to around twenty units less than [28]. We anticipate that our method could be improved in its guidance to reduce vehicles, as discussed earlier in this section.

Finally, in table 2 we report the objective values of some new best solutions. (Appendix A contains the solutions themselves.) In our tests, we have not been able to match or better the best results reported in [24] and [28] in only 4 of the 29 problems. Note that in two of the new best solutions (R104 and R110), vehicles have been reduced over previously published solutions. Each of these reductions has been accompanied by an increase in the travel distance.

4 Conclusion

This paper has introduced a new search method featuring a large neighbourhood and a powerful move operator based on relaxing and reoptimizing small parts of the problem. 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 is thus ideal for models involving complex real-world constraints.

The power of the technique has been demonstrated by its application to vehicle routing problems. Although the implementation of the branch and bound procedure used is extremely naive, initial results indicate that the performance of the algorithm is better than the best techniques using minima-escaping methods. These very good results demonstrate that minima-escaping procedures are not the only way to tackle combinatorial problems—very good results can be obtained with greedy search. Since these are the very first results published by this method, we expect them to improve via better branch and bound methods, better relaxation choices, and extensions or hybrids.

Looking to the future, many choices lie open. The immediate work is to improve the bound of the branch and bound algorithm. The performance on problems with long vehicle

routes should also be improved by the methods discussed in section 3. Side constraints can be added and their impact on performance and solution quality investigated.

More broadly speaking, the idea of using a large neighbourhood with relaxation and reoptimization should apply to many other combinatorial optimization problems, just as the traditional meta-heuristics such as simulated annealing and tabu search do currently.

We have called the more general method Large Neighbourhood Search. To apply Large Neighbourhood Search to a new problem class, all that is required is a definition of constrained variables and values (usually obvious), and some measure of the relatedness of variables. The optimization can then be carried out by a method identical to that already described. We look forward to investigating these possibilities in the near future.

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] B. De Backer, V. Furnon, P. Kilby, P. Prosser, and P. Shaw. Solving vehicle routing problems with constraint programming and meta-heuristics. Submitted to the Journal of Heuristics.

[2] C. Bessiere. Arc consistency and arc consistency again. *Artificial Intelligence*, 65:179–190, 1994.

[3] C. Bessiere, E. C. Freuder, and J.-C. Regin. Using inference to reduce arc consistency computation. In *Proceedings of the 14th IJCAI*, 1995.

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

[5] 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.

[6] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

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

[8] I. Gent, E. MacIntyre, P. Prosser, P. Shaw, and T. Walsh. The constrainedness of arc consistency. In *Proceedings of CP-97*. Springer, 1997.

[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] I. P. Gent. Heuristic solution of open bin packing problems. Submitted to the Journal of Heuristics. Available from http://www.cs.strath.ac.uk/Contrib/ipg/papers/-binpacksol.ps.gz.

[13] F. Glover. Tabu search, part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[14] F. Glover. Tabu search, part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

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

[16] J. H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.

[17] 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.

[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[19] R. Korf. Improved limited discrepancy search. In *Proceedings of the 13th AAAI*. The MIT Press, 1996.

[20] G. Laporte and I. H. Osman. Routing problems: A bibliography. *Annals of Operations Research*, 61:227–262, 1995.

[21] P. Prosser and P. Shaw. Guided local search for the vehicle routing problem. In *Proceedings of the 2nd International Conference on Meta-heuristics*, 1997.

[22] J.-C. Regin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 11th AAAI*. The MIT Press, 1994.

[23] J.-C. Regin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the 13th AAAI*. The MIT Press, 1996.

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

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

[26] T. Schiex, J.-C. Regin, C. Gaspin, and G. Verfaille. Lazy arc consistency. In *Proceedings of the 13th AAAI*. The MIT Press, 1996.

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

[28] E. Taillard, P. Badeau, M. Gendreau, F. Guertain, and J.-Y. Potvin. A new neighbourhood structure for the vehicle routing problem with time windows. Technical Report CRT-95-66, Centre de Recherche sur les Transports, University of Montreal, 1995.

[29] C. Voudouris. *Guided Local Search for Combinatorial Problems*. PhD thesis, University of Essex, April 1997.

[30] C. Voudouris and E. Tsang. Function optimization using guided local search. Technical Report CSM-249, Department of Computer Science, University of Essex, September 1995.

[31] C. Voudouris and E. Tsang. Guided local search. Technical Report CSM-247, Department of Computer Science, University of Essex, August 1995.

[32] C. Voudouris and E. Tsang. Partial constraint satisfaction problems and guided local search. Technical Report CSM-250, Department of Computer Science, University of Essex, September 1995.

[33] T. Walsh. Depth-bounded discrepency search. In *Proceeedings of the 15th IJCAI*, 1997.

A New Best Solutions

We report the new best solutions (table 2) found by our method below. For each solution, the problem name, travel distance, and number of vehicles are given. Each vehicle route is shown on a line of its own, with the numbers of the visits made in order.

R104
Distance: 1007.31057682
Vehicles: 9
18-82-48-46-8-47-36-49-19-7
52-88-62-11-64-63-90-32-10-31-27
6-94-96-59-93-99-84-17-45-83-5-60-89
28-1-69-76-53-40-2-13-95-97-58
42-43-15-87-57-41-22-74-73-21-26
92-98-14-44-38-86-16-61-85-91-100-37
70-30-20-66-9-35-65-71-51-50
72-75-56-23-67-39-55-4-25-54
68-80-24-29-79-78-34-81-33-3-77-12

R107
Distance: 1104.65597961
Vehicles: 10
50-33-65-71-9-66-20-51-1
47-36-46-8-84-61-85-98-100-93-96
53-40-12-68-29-24-80
26-21-39-23-67-55-4-25-54
27-76-3-79-78-34-35-81-77-28
69-30-88-31-10-63-90-32-70
92-42-43-15-41-22-75-56-74-72-73-58
52-7-62-11-64-49-19-48-82-18-89
83-45-99-6-87-57-2-13-97-95-94
59-37-91-44-14-38-86-16-17-5-60

R108
Distance: 963.993343022
Vehicles: 9
31-88-62-10-63-90-32-66-20-30-70
2-57-15-43-42-87-41-22-74-73-21-26
27-1-69-76-3-79-29-24-80-68-77-12
72-75-56-23-67-39-55-4-25-54
6-5-61-17-84-8-46-45-83-60-89
94-95-59-93-99-96-53-13-58-40
28-50-33-81-51-9-71-65-35-34-78
52-7-19-11-64-49-36-47-48-82-18
92-98-85-44-14-38-86-16-91-100-37-97

R109
Distance: 1205.95664202
Vehicles: 11
28-12-76-29-3-79-78-34-35
92-14-44-16-86-38-43-58
52-88-19-47-36-49-48
39-67-23-75-22-41-74-56-4
2-72-73-21-40-53-26-54-55-25
5-98-59-99-84-61-85-91-100-37-93
83-45-82-7-18-8-46-17-60-89
27-69-31-30-51-81-50-77-68-24-80
33-9-71-65-66-20-70-1
62-11-64-63-90-32-10
95-42-15-57-87-94-6-96-97-13

R110
Distance: 1135.07089578
Vehicles: 10
47-19-11-64-49-46-36-48
95-98-44-16-86-38-14-43-42-13-58
21-72-73-22-75-23-67-39-56-74
2-41-15-57-87-97-92-37-100-91-93
28-76-12-53-40-26-54-55-25-4
70-30-9-71-65-35-34-78-24
83-5-61-85-59-99-94-96-6-89
29-79-33-51-81-3-50-77-68-80
27-52-88-7-82-18-8-84-17-45-60
69-31-62-63-90-10-32-20-66-1

R111
Distance: 1096.72658305
Vehicles: 10
83-18-45-8-84-61-91-100-85-93-96
28-50-76-53-40-4-54-12-26
7-19-64-49-36-46-47-48-82
2-57-15-43-41-22-74-56-72-21-58
52-31-88-62-11-63-10-90-32-20-70
1-51-81-33-79-9-78-29-68-3-77
92-98-37-59-99-97-87-6-94-13
73-75-23-67-39-55-25-24-80
95-42-14-44-38-86-16-17-5-60-89
27-69-30-65-66-71-35-34

RC104
Distance: 1135.47939614
Vehicles: 10
88-60-78-73-79-7-6-2-55
85-63-89-76-84-51-64-83-66
90-65-99-97-75-58-77
14-47-15-11-9-87-59-74-86-57-52
80-91-56-95-62-67-94-93-71-72-54-96-81
42-44-43-38-37-35-36-40-39-41
20-49-19-23-21-18-48-25-24-22
92-50-33-32-30-28-26-27-29-31-34
69-98-53-12-17-16-13-10-82
61-70-1-3-5-45-8-46-4-100-68

RC107
Distance: 1230.47745014
Vehicles: 11
2-6-7-8-5-3-1-45-46-4-100
90-81-94-93-96-54
92-95-84-85-63-51-76-89-56
38-41-72-71-67-50-62-91-80
69-98-88-53-78-73-79-60-55-70-68
11-12-14-47-17-16-15-13-9-10
61-42-44-43-40-37-35-36-39
64-22-19-23-21-18-48-49-20-24
31-29-27-28-26-30-32-34-33
82-99-52-87-59-86-57-66
65-83-25-77-75-97-58-74