On the Reformulation of Vehicle Routing Problems and Scheduling Problems*

J. Christopher Beck[1], Patrick Prosser[2], and Evgeny Selensky[2]

[1] ILOG SA, Paris, France. cbeck@ilog.fr
[2] Department of Computing Science, University of Glasgow, Scotland.
pat/evgeny@dcs.gla.ac.uk

Abstract. In the capacitated vehicle routing problem with time windows (CVRPTW) we have a set of customer visits, each with a demand and time window in which the visit must be serviced, and a fleet of vehicles, each vehicle of limited capacity. The problem is to visit all customers whilst minimising total travel distance and the use of vehicles. In the job-shop scheduling problem (JSSP) we have a set of jobs, composed of a sequence of activities, and a set of resources. Each activity requires exclusive use of a resource for a given amount of time. The problem is then to sequence activities on resources such that all precedence constraints are respected and the makespan is minimised. We can reformulate a VRP to an open shop scheduling problem by representing visits as activities, vehicles as resources on the factory floor, and travel as set up costs between activities. We also have the inverse reformulation. In this paper we present two reformulations: from VRP to open shop, and the inverse, from JSSP to VRP. Not surprisingly, we show that VRP technology performs poorly on reformulated JSSP, as does scheduling technology on reformulated VRPs. We then present a pre-processing transformation that "compresses" the VRP, transforming an element of travel into the duration of the visits. The compressed VRPs are then reformulated as scheduling problem, to determine if it is primarily distance in the VRP that causes scheduling technology to degrade on the reformulated problem. This is a step towards understanding the features of a problem that make it more amenable to one technology rather than another.

1 Introduction

Problem reformulation research is represented by an increasingly large body of work [15, 3] with a long term aim of better understanding the interplay between features of problems and how we might formulate them. The eventual goal of such work is to significantly ease, and maybe even automate the process of finding efficient formulations for interesting problems. In this paper, we begin with the observation that, on a conceptual level, vehicle routing problems (VRP) and factory scheduling problems are very similar. Consider the following:

* This work was supported by EPSRC research grant GR/M90641 and ILOG SA.

– They both involve the execution of tasks (activities in the factory and visits in the VRP).
– A task can only be completed through the use of one or more resources (tools and machines on the shop floor, drivers and vehicles in the VRP).
– Resources are often constrained by capacity that specifies, for example, the number of tasks that can be processed by that resource.
– Often, there is a set of alternative resources to choose from (similar machines on the shop floor, a fleet of vehicles in the VRP).
– When a task is completed there may be some interval of time that must expire before the resource can be used for another task, and this interval may depend on the pair of consecutive tasks (a transition time or set up cost on a machine in the factory, a travel distance between visits in the VRP).
– There may be further temporal constraints among the tasks, specifying time windows when they can and cannot be executed and/or specifying a necessary relationship between tasks (e.g., task B must not be executed until after task A is finished).
– The problem is solved when the resources have been assigned to tasks and the tasks have been assigned start times or ordered such that all temporal and capacity constraints are respected.
– There is an optimisation criterion involving the minimisation of various definitions of the length of time necessary to complete all the activities. For example, one may seek to minimise the sum of all transition times.

With these similarities, one might expect that similar technology is used to solve vehicle routing and scheduling problems. But this isn't the case. For example, local search techniques (tabu search, simulated annealing, guided local search) [11, 13, 22] are more popular for vehicle routing problems whereas complete and quasi-complete search techniques [12] are used more in scheduling. As a single data point, ILOG markets two commercial C++ libraries (ILOG Scheduler and Dispatcher) for solving scheduling and vehicle routing problems respectively. Though both products are built on constraint programming technology, the core technology in the scheduling product is global constraint propagation [14, 16] while the core technology in the vehicle routing product is local search [8].

What are the characteristics of vehicle routing problems that make them more amenable to local search techniques allied to construction methods? What properties of scheduling problems make them more suitable to systematic search and powerful constraint propagation? In this paper, we take a first step toward answering these questions. After defining the vehicle routing and scheduling problems in the next section, we present reformulations between the VRP and factory scheduling problems. Section 4 presents initial experiments of applying scheduling and VRP technology to these reformulated problems. A preprocessing transformation of VRPs, designed to better exploit the strengths of the scheduling techniques, is then presented in Section 5. In Section 6 we present an empirical study of how this transformation influences solution quality and discuss the results. Section 7 concludes the paper.

2 Vehicle Routing and Factory Scheduling Problems

In the delivery variant of the capacitated vehicle routing problem with time windows (CVRPTW), m identical vehicles initially located at a depot are to deliver discrete quantities of goods to n customers. The locations of the depot and customers are known. Each customer has a demand for goods and each vehicle has a capacity. A vehicle can make only one tour starting at the depot, visiting a subset of customers, and returning to the depot. It is assumed that travel time equals travel distance, computed using the Euclidean metric. Time windows define an interval for each customer within which the visit must be made[1]. A solution is a set of tours for a subset of vehicles such that all customers are served only once and time window and capacity constraints are respected. The objective is to minimise distance travelled, and sometimes additionally to reduce the number of vehicles used. The problem is NP-hard [10].

An $n \times m$ job shop scheduling problem (JSSP) consists of n jobs and m resources. Each job consists of a set of m completely ordered activities. Each activity requires a resource and has an execution duration on that resource. The complete ordering defines a set of precedence constraints, such that an activity cannot begin execution until the preceeding activity has completed. Each of the m activities in a job requires exclusive use of one of the m resources. No activities that require the same resource can overlap in their execution and once an activity has started it must be executed for its entire duration (i.e. no preemption is allowed). The job shop scheduling decision problem is then to decide if all activities can be scheduled, given for each job a release date of 0 and a due date of the desired makespan D, while respecting the resource and precedence constraints. The job shop scheduling decision problem is NP-complete [10].

The open shop scheduling problem (OSSP) [5] is a relaxed form of the job shop. Activities may be performed on any resource in a set of alternatives, activities in a job may be partially ordered, and there may be transition times (set up costs) between activities when performed on the same resource. Such problems have recently been studied by Focacci [9]. For the remainder of the paper we will refer to the CVRPTW as the VRP and the OSSP as the SSP (shop scheduling problem, job or open).

3 Transformations: VRP $\leftrightarrow$ SSP

VRP $\rightarrow$ SSP: We reformulate the VRP into a SSP as follows. Each vehicle is represented as a resource, and each customer visit as an activity. The distance between a pair of visits corresponds to a transition time between respective activities. Each activity can be performed on any resource, and is constrained to start execution within the time window defined in the original VRP. Each activity has a demand for a secondary resource, corresponding to a visit's demand within a vehicle. For each resource R there are two special activities $Start_R$ and

[1] In this paper we assume that a visit has to be started within a time window. We do not assume that it has also to be completed within the window.

End_R. Activities $Start_R$ and End_R must be performed on resource R. $Start_R$ must be the first activity performed on R and End_R the last. The transition time between $Start_R$ and any other activity A_i corresponds to the distance between the depot and the i^{th} visit. Similarly the transition time between End_R and A_i corresponds to the distance between the depot and the i^{th} visit. The processing time of $Start_R$ and End_R is zero. In addition, we associate a consumable secondary resource with every (primary) resource. This models the capacity of vehicles. Consequently a sequence of activities on a resource corresponds to a vehicle's tour in the VRP. Therefore, in the resultant SSP each job consists of only one activity[2], each activity can be performed on any resource, and there are transition times between each pair of activities. The problem is then to minimise the sum of transition times on all machines (and maybe also to minimise the number of resources used).

SSP $\rightarrow$ VRP: To reformulate the SSP as a VRP, we have for each resource a vehicle, and for each activity a customer visit. The visits have a duration the same as that of the corresponding activities. Each visit can be made only by the vehicles corresponding to the set of resources for the activity. Any ordering between activities in a job results in precedence constraints between visits. Transition times between activities correspond to travel distances between visits. The deadline D imposes time windows on visits. Assuming we have m resources, and therefore m vehicles, we have $2m$ dummy visits corresponding to the departing and returning visits to the depot. A vehicle's tour corresponds to a schedule on a resource. For the $n \times m$ JSSP we have a VRP with $m(n+2)$ visits and m vehicles. Each visit can be performed only by one vehicle. Since there are no transition times in the JSSP, there are no travel distances between visits, but visits have durations corresponding to those of the activities. There are precedence constraints between those visits corresponding to activities in a job. The decision problem is then to find an ordering of visits on vehicles that respects the precedence constraints and time windows.

4 A Study of Mutual Reformulations

In [19] Selensky investigated the two extreme cases identified above, i.e. the reformulation of benchmark VRPs as SSPs (and their subsequent solution using ILOG Scheduler) and the reformulation of JSSP benchmarks as VRPs (solved using ILOG Dispatcher). The VRPs used were the 56 Solomon benchmarks [21] and jobshop problems of size 6×6 [24] and 15×15 [23].

VRPs as OSSPs The 56 Solomon benchmark problems were reformulated as open shop scheduling problems and solved using ILOG Scheduler. Discrepancy-bounded depth first search [2] was used, with 3 hours of cpu time allowed for each instance, running on a 933MHz Pentium III processor with 1Gb of RAM.

[2] Alternatively, we might think that there are no jobs, just activities.

This was compared against a direct encoding of the problems into ILOG Dispatcher i.e. VRPs formulated as VRPs and solved with VRP technology, again using the same computational resources. Unsurprisingly, it was observed that the reformulation severely degraded solution quality. Typically, the Scheduler solutions required twice as much travel as the Dispatcher solutions, and often used more vehicles. This clearly suggests that ILOG Dispatcher is indeed well suited to VRPs and symmetrically, that ILOG Scheduler is not.

JSSPs as VRPs Six JSSP benchmark problems were formulated as VRPs and solved using ILOG Dispatcher. The first solution was built using the savings heuristic [6] and then improved using GLS. The problems were then solved as JSSP in ILOG Scheduler. Both techniques were limited to 60 seconds cpu, and 180 seconds for the larger problem instances. The quality of the Dispatcher solutions ranged from a factor of 2 to a factor of 6 worse than the Scheduler solutions. Again, this convincingly demonstrates that ILOG Scheduler is better suited to jobshop scheduling than Dispatcher.

Nevertheless, we should add the caveat that these two classes of problems are extreme and we should expect that such specialised tools would demonstrate extreme differences. The VRP instances reformulate to strange open shop problems, having essentially single activity jobs, a vast selection of resources for each activity, vanishingly small durations and comparatively enormous transition times. Conversely the JSSP instances map to VRPs where a visit can be performed only by one vehicle and there is no travel! While these results do not come as a great surprise, they do lend support to the belief that there are characteristics of vehicle routing and scheduling problems that make them more suitable to their standard resolution technology. Three differences stand out:

1. Alternative resources - The transformed VRPs have many more alternative resources than are typically studied in scheduling problems. Although the vehicles might not all be used, the benchmark problems start with 20 or more vehicles. In contrast, existing scheduling research on resource alternatives typically have many fewer alternatives. For example, Focacci et al. [9] experimented on problems with up to three alternatives while Davenport & Beck [7] used problem instances of up to eight alternatives.
2. Transition time *vs* activity duration - In scheduling problems the duration of an activity is typically much larger than the transition time. This is not the case in the transformed VRPs where, in fact, the transition time is many times the activity duration. Furthermore, much of the research in scheduling has focused on techniques that reason directly about the durations of activities while putting less emphasis on the transition time. Certainly, the state of the art in constraint-based scheduling is based to a large extent on global constraint propagation [16, 17] and heuristic search techniques [20, 1] that do not directly take into account transition time. Therefore, we expect that scheduling problems where the transition time appears to be a key aspect of finding a solution, will be difficult for current scheduling technology.
3. Complex temporal relationships - Scheduling problems tend to have more complex temporal relationships among activities. As we have seen, prece-

dence constraints among the activities in a job are part of the JSSP. More complex metric temporal constraints have been widely explored [4] and strong global constraint propagation techniques that take such relationships into account exist [14]. The reformulated VRPs have no temporal constraints other than the time windows. It is again reasonable to suppose that the lack of such relationships means that the state-of-the-art in scheduling cannot be exploited to solve these problems.

Can we find some less drastic reformulation of the VRP which would tend to minimise these differences? If we can, we might then identify the problem features that most influence the performance of the technologies. In the balance of this paper, we present a transformation that attempts to reduce the difference between the transition time and activity duration in VRPs, while preserving the cost of solutions.

5 Compressing the VRP

The basic idea of this transformation is to reduce travel within a VRP by adding a portion of travel costs to the costs of visits. Consequently, when the VRP is reformulated as a SSP that SSP will have activities with relatively large durations and small set up costs. That is, the problem should look more like a SSP than a VRP. Intuitively, we might think that we are *compressing* the VRP, forcing travel between visits into the duration of those visits. We present this transformation as follows. First we show how to compress a travelling salesman problem (TSP), then we take into consideration time windows, i.e. TSPTW. Finally, we show how we can apply our transformation to the VRP.

5.1 Compressing the TSP

Consider a travelling salesman problem where nodes have costs as well as edges[3]. In a solution we must visit each node once and once only. The cost of visiting a node is then the cost of the edge entering the node plus the cost of the edge exiting the node, plus the cost of the node itself. We can transform this cost such that the node cost is increased and the costs on the entering and exiting edges are reduced.

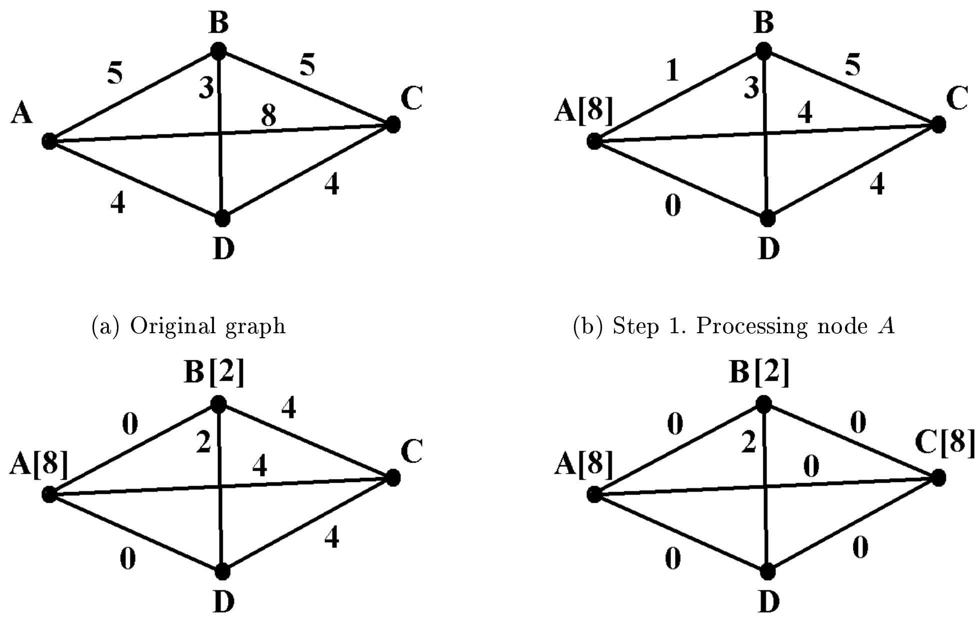
Consider node j, with entering edge (i, j) and exiting edge (j, k), with costs C_j, $C_{i,j}$ and $C_{j,k}$ respectively. Let $C_{min} = min(C_{i,j}, C_{j,k})$. We can reduce the cost of both edges by C_{min}, such that one of these becomes zero, and add to the node cost $2 \times C_{min}$. This preserves the cost of entering, visiting, and exiting j. More generally, for a TSP we can process each node i as follows

1. let C_{min} be the cost of the cheapest edge incident on node i;
2. for each edge incident on node i subtract C_{min} from the edge's cost;
3. add a cost of $2 \times C_{min}$ to C_i, the cost of node i;

[3] For a conventional TSP node costs would then be zero.

4. the node now has at least one incident edge of zero cost.

We need only process each node once, i.e. after processing, a node has at least one zero cost incident edge and re-processing will have no effect. Figure 1 shows the sequence of transformations of a four-node clique. All nodes start with zero cost and are processed in alphabetic order. We start with the initial problem and then process node A. This removes a cost of 4 from A's incident edges and adds a cost of 8 to node A (Figure 1a). We then process node B. This reduces the cost of incident edges by 1 and adds a cost of 2 to node B (Figure 1b). Then we move on to node C and reduce the cost of incident arcs by 4 and add a cost of 8 to node C. Finally, processing D has no effect because there is a zero-cost incident arc. Note, that if we processed the vertices in a different order we might end up with a different final graph, i.e. the transformation is order dependent.



(a) Original graph

(b) Step 1. Processing node A

(c) Step 2. Processing node B

(d) Step 3. Processing node C

Fig. 1. Transformation of a 4-node clique. New node costs are shown in square brackets.

We might consider the above process as transforming a node X into a double node[4] $X - Y$, with a travel time between X and Y. All routes to and from X now have to go via Y.

[4] And in that sense, it would have similarities to the transformation of [18] from arc-routing to node-routing

In a Hamiltonian path we have two distinguished nodes, i.e. the start node s and end node e, and the transformation is then modified as follows. Since s is not entered and e is not exited, we do not add to those vertices twice the cost of its minimum incident edge. Instead, we add the minimum cost once only, and process all other vertices as above.

5.2 Compression with Time Windows

When time windows are associated with nodes the compression can change the lower and upper bounds on the time of entering and exiting a node. Consider the problem of finding a Hamiltonian path from s to e, where each node i has a time window $[ES_i, LS_i]$, i.e. we must arrive at node i no earlier than ES_i and no later than LS_i. Assume we have a graph with edges (s,i), (s,j), and (i,j) where $C_{s,i} < C_{s,j}$ and $C_{i,j} < C_{s,j} - C_{s,i}$. Further assume that we process nodes in the order s, then i, then j. On processing start node s, the transformed costs of node s becomes $C'_s = C_s + C_{s,i}$, and transformed edges $C'_{s,j} = C_{s,j} - C_{s,i}$, and $C'_{s,i} = 0$. Node i is then processed, and this has no effect since it has a zero cost incident edge $C'_{s,j}$. On processing node j we get the transformed costs $C'_j = C_j + 2 \times C_{i,j}$, $C''_{s,j} = C'_{s,j} - C_{i,j}$, and $C'_{i,j} = 0$. Note that via substitution $C''_{s,j} = C_{s,j} - C_{s,i} - C_{i,j}$. Consequently, on the transformed graph the cost of travelling directly from s to j is $C_s + C_{s,j} - C_{i,j}$, i.e. we arrive earlier on the transformed graph and might now have to wait before entering node j. A symmetric argument holds for leaving the node.

Theorem 1. *On transforming an arbitrary node i, with time window $[ES_i, LS_i]$, its earliest start time becomes $ES_i - C_{i,j}$ and its latest start time becomes $LS_i - C_{i,j}$, where $C_{i,j}$ is the cost of the cheapest edge incident on node i.*

Proof. We need to prove that on travelling from i to j, or from j to i in the transformed graph we maintain any slack that existed in the original graph. The proof is by cases. On travelling from i to j we can leave as early as possible ES_i, process node i with cost C_i, and then traverse the edge with cost $C_{i,j}$, arriving at node j with slack $S1$ after node j's earliest start ES_j. Similarly we can leave as early as possible and have slack $S2$ with respect to the latest start time of node j, leave as late as possible and have slack $S3$ with respect to ES_j, or as late as possible and have slack $S4$ with respect to LS_j. The four slack equations are then as follows:

$$ES_i + C_i + C_{i,j} = ES_j + S1 \quad (1)$$
$$ES_i + C_i + C_{i,j} = LS_j + S2 \quad (2)$$
$$LS_i + C_i + C_{i,j} = ES_j + S3 \quad (3)$$
$$LS_i + C_i + C_{i,j} = LS_j + S4 \quad (4)$$

There are three cases to consider when transforming node i and then node j. Assume that we have edges (h,i), (i,j), and (j,k). We then have the three cases below:

$$C_{h,i} < C_{i,j} \wedge C_{j,k} < C_{i,j} - C_{h,i} \quad (5)$$
$$C_{h,i} < C_{i,j} \wedge C_{i,j} - C_{h,i} \leq C_{j,k} \quad (6)$$
$$C_{i,j} \leq C_{h,i} \quad (7)$$

We must then consider twelve cases. The first case is when we process i and then j in a graph with the property of equation (5). We must then show that the four slack equations hold. We repeat this process for the graphs (6) and (7). Via the technique of substitution used earlier, i.e. used to prove that time windows must be adjusted, it can be shown that the slack equations hold for each of the graphs above, when travelling from i to j or from j to i. *QED*

The distinguished node e is processed as an arbitrary node i.e. we subtract from ES_e and LS_e the cost of the cheapest incident edge. The time window for start node s is not altered, retaining its original values of ES_s and LS_s.

5.3 Compressing the VRP

As noted above, the transformation is order dependent. In Figure 1, if we processed nodes in some other order we might end up with a different graph. In this paper we will investigate three transformations. The first, and most obvious, uses a lexicographic ordering of vertices. We will refer to this as a *lex* ordering. The second, we will call *maxMin*; when selecting a node i to process next we choose a node such that its cheapest incident edge is a maximum. For example, in Figure 1 this would initially select node A or node C, as their cheapest incident edges are largest, with a cost of 4. The intuition behind this is that it will attempt to make the biggest reduction in edge costs. The third ordering, is *minMin*. This might be thought of as the anti-heuristic, selecting to process a node i with smallest minimum incident edge cost. In Figure 1 this would initially choose node B or D.

After the VRP has been compressed, we can then reformulate the VRP as a SSP and solve it using scheduling technology[5]. We now attempt to determine if any of these compressions (of VRP benchmarks) result in better solutions, with or without the reformulation to a SSP.

6 An Empirical Study

The experiments were performed on the 56 Solomon benchmarks [21]. These problems fall into six classes: C1, C2, R1, R2, RC1, and RC2. All problems in a class have the same set of customer visits, i.e. there is one set of 100 x/y coordinates corresponding to the customer visits. What differentiates problems within a class is the distribution of time windows and demands. In C1 and C2

[5] We could also take an arc-routing problem and transform it using the procedure defined in [18], and then compress that problem and reformulate it as SSP

visits are clustered, and in C2 visits have larger time windows and increased vehicle capacity, i.e. C2 is a less constrained set than C1. The locations of visits in R1 and R2 are random, and again, each R2 instance has larger time windows and increased vehicle capacities. The RC set is made up of clustered groups of randomly generated visits, and again RC2 is a less constrained set than RC1.

The 56 benchmark VRP were transformed as above, using the lex ordering, maxMin ordering, and minMin ordering. This gives us a total 224 problems, i.e. the original problems and each problem transformed using the three orderings. The problems were then solved using ILOG Dispatcher, each instance being allowed 10 minutes cpu time. The purpose of this was to determine how VRP solving technology is influenced by the amount of travel within the problem. That is, will Dispatcher perform worse or better as we compress problems, transforming travel between nodes into the cost of processing those nodes? The same set of 224 problems were then reformulated as scheduling problems and solved using ILOG Scheduler, and again given 10 minutes cpu time per instance. Would the transformation result in an improvement in Scheduler's performance on the reformulated VRPs?

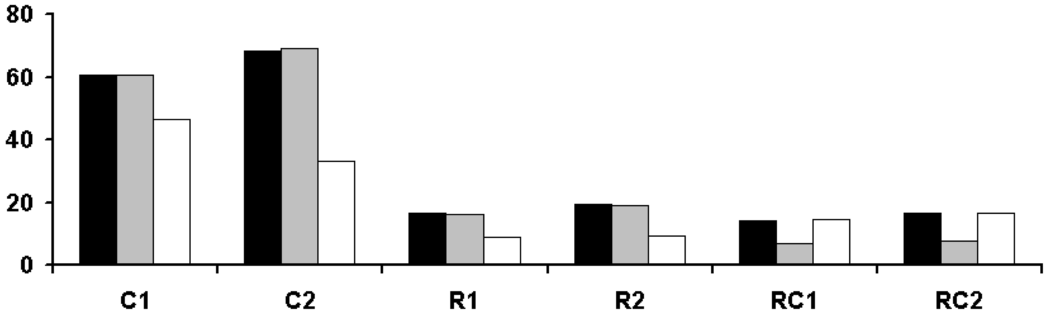


Fig. 2. Percentage increase in cost as a result of transforming VRPs and solving with Dispatcher

Solving with Dispatcher: Figure 2 shows the percentage change in cost as a result of the transformation when solving the VRPs with Dispatcher, i.e. without reformulation. The black bars represent lex ordering, the grey bars maxMin ordering, and minMin ordering is in white. Results are expressed as the average percentage change in cost compared to the original problem solved with Dispatcher. All of the orderings result in a large increase in cost, between 10% and 70%. An analysis of the first solutions found, prior to improvement with local search, showed that the first solutions were worse for the transformed problems, and the difference between the transformed and original problem increased even more as local search proceeded. That is, the transformation degraded the initial construction and the performance of local search. This demonstrates the sensitivity of the VRP solving technology to the mix of travel and processing time. The maxMin ordering attempts to make the largest compression of the problem, and this also corresponds to the greatest increase in cost of solution. This tends to suggest that the VRP technology degrades as the VRP becomes more like a SSP. This might also suggest that an inverse transformation, one that was able to *stretch* the VRP by converting the cost of processing a visit into additional travel, might improve the VRP technology.

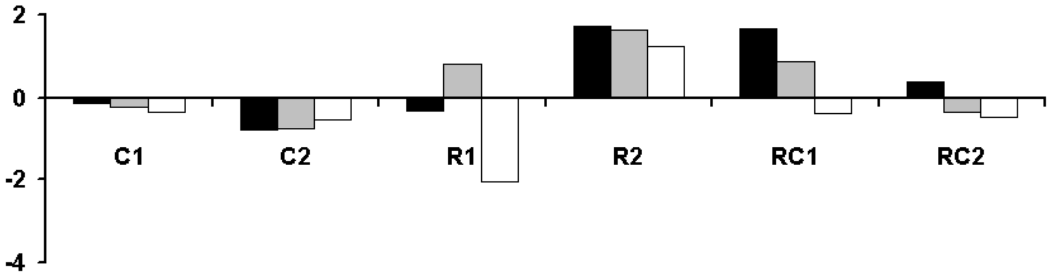


Fig. 3. Percentage increase in cost as a result of transforming VRPs, reformulating them, and solving with Scheduler

Solving with Scheduler: Each of the 224 problems were then reformulated as SSP and solved using ILOG Scheduler. In Figure 3 we see again the percentage change in cost as a result of the transformation (again with black bars for lex ordering, grey for maxMin ordering, and white for minMin ordering). The picture is now not so clear. In the clustered problems, C1 and C2, we see a reduction in cost, and a comparatively larger improvement in the relaxed C2 problems. In the random problems, R1 and R2, there tends to be an increase in cost, even greater in the less constrained R2 problems. This might suggest that the transformations do not fare well in unstructured problems. In the RC problems it appears that the transformations reduce cost but only when the problems are less constrained (i.e. the RC2 problems).

The strongest conclusion that we dare to take from these experiments is that distance appears to be a crucial factor when solving VRPs with VRP technology, and that when reformulated as scheduling problems, distance, i.e. transition times, have more of an impact when problems have structure and time windows and capacity are relaxed.

7 Conclusion and Future Work

We have presented reformulation between VRPs and SSP. By solving the reformulated problems with domain specific technologies we have demonstrated that indeed the vehicle routing technology appears to be well suited to VRPs and not at all suited to reformulated SSP. The converse also appears to hold. However we must add the caveat, that this is no surprise because the benchmark problems used are extreme cases and should indeed be well fitted to their solving technologies.

We then presented a transformation process for the VRP, which can be used as a pre-processing step before solving a problem, or reformulating and solving a problem. This transformation attempts to *compress* the VRP by adding an element of travel into the processing of each node. This was done in the hope that the reformulated problem would appear to be more like a SSP than a VRP, i.e. duration of activities would be increased and transition times would be decreased. Our experiments showed that the transformations degraded the VRP technology, suggesting that indeed travel is an important problem feature. When reformulated as SSP it was less clear. Although the activities now have increased duration and decreased transition times, they are still peculiar problems: they remain as single activity jobs that can be performed on any resource.

More study is required. In particular, more control needs to be exercised over our data sets. In our ongoing studies we are producing VRPs with more structure, and structure that we can control i.e. we can increase structure gradually. This is done by gradually varying time windows, vehicle capacity, the composition of the fleet (such that it is no longer homogenous), specialisation of visits (i.e. visits can only be performed by certain classes of vehicles), sequencing constraints between visits, etc. By gradually increasing the richness of these VRPs we expect that we might reach a point where SSP technology competes with VRP technology, and our study will then try to determine just what features bring about this competition.

In addition, we plan to investigate the inverse transformation i.e. rather than compress a VRP we might stretch it. Might this be a pre-processing step that will improve VRP solving? Also, when there are set up costs in the SSP, might a further compression improve solving?

Could the transformations and reformulations be of any real use? We believe so. As we add more constraints to the VRP, such as sequencing constraints between visits, restrict time windows, and specialise visits to a subset of the fleet, VRPs will tend to be more like the SSP. Similarly, as the tooling on the shop floor becomes more flexible, such that machines can perform many functions, and the cost of re-configuring tools increases, the SSP will also tend to have features similar to the VRP. Therefore on a spectrum that has the VRP at one end and the JSSP at the other, we might expect that as problems become richer it will be less clear as to just what technology is most appropriate. At that time we might expect that transformations and reformulations will become an important tool.

Acknowledgements

We would like to thank Barbara Smith and Vincent Furnon.

References

1. J. C. Beck and M. S. Fox. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, 117(1):31–81, 2000.
2. J.C. Beck and L. Perron. Discrepancy-bounded depth first search. In *Second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CP-AI-OR'00)*, 2000.
3. J. Borrett and E. Tsang. A Context for Constraint Satisfaction Problem Formulation Selection. *Constraints Journal*, 6:299–327, 2001.
4. A. Cesta, A. Oddi, and S.F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 2000. to appear.
5. Bo Chen and Vitaly A. Strusevitch. Approximation algorithms for three-machine open shop scheduling. *ORSA Journal on Computing*, 5(3):321–326, 1993.
6. G. Clake and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.
7. A.J. Davenport and J.C. Beck. An investigation into two approaches for constraint directed resource allocation and scheduling. In *INFORMS*, 1999.
8. B. DeBacker, P. Furnon, P. Shaw, P. Kilby, and P. Prosser. Solving vehicle routing problems using constraint programming and metaheuritics. *Journal of Heuristics*, 6:5001–523, 2000.
9. F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 2000.
10. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.
11. F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic, 1995.
12. W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference onf Artificial Intelligence (IJCAI-95)*, pages 607–613, 1995.
13. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
14. P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. In *Proceedings of the 6th European Conference on Planning (ECP;01)*, 2001.
15. Bernard A. Nadel. Representation Selection for Constraint Satisfaction: A Case Study Using n-Queens. *IEEE Expert*, pages 16–23, 1990.
16. W. P. M. Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach.* PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.
17. C. La Pape. Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.
18. W.L. Pearn, A. Assad, and B.L. Golden. Transforming arc routing into node routing problems. *Comput. Opns. Res*, 14(4):285–288, 1987.

19. Evgeny Selensky. On mutual reformulation of shop scheduling and vehicle routing. In *Proceedings of the 20th UK PLANSIG*, pages 282–291, 2001.
20. S. Smith and C. Cheng. Slack based heuristics for constraint satisfaction scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 139–144, 1993.
21. M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35:254–365, 1987.
22. C. Voudouris and E.P.K. Tsang. Guided Local Search. *European Journal of Operational Research*, 113(2):80–110, 1998.
23. Jssp benchmarks, 15 by 15. http://www.dcs.gla.ac.uk/pras/resources.html.
24. Jssp benchmarks, 6 by 6. http://www.ms.ic.ac.uk/jeb/pub/jobshop1.txt.