Journal of Heuristics, Volume No., 1–16 (1997)
© 1997 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Solving Vehicle Routing Problems using Constraint Programming and Metaheuristics

BRUNO DE BACKER debacker@ilog.fr
VINCENT FURNON furnon@ilog.fr
ILOG S.A, 9 Rue de Verdun, Gentilly, France.

PHILIP KILBY pjk@cs.strath.ac.uk
PATRICK PROSSER pat@cs.strath.ac.uk
PAUL SHAW ps@cs.strath.ac.uk
Department of Computer Science, University of Strathclyde, Glasgow, Scotland.

Editor: Wim Nuijten

Abstract. Constraint Programming typically uses the technique of depth-first branch and bound as the method of solving optimisation problems. Although this method can give the optimal solution, for large problems, the time needed to find the optimal can be prohibitive.

This paper introduces a method for using iterative improvement techniques within a Constraint Programming framework, and applies this technique to vehicle routing problems. We introduce a Constraint Programming model for vehicle routing, after which we describe a system for integrating Constraint Programming and iterative improvement techniques. We then describe how the method can be greatly accelerated by handling *core* constraints using fast local checks, while other more complex constraints are left to the constraint propagation system.

We have coupled our iterative improvement technique with a meta-heuristic to avoid the search being trapped in local minima. Two meta-heuristics are investigated: a simple Tabu Search procedure and Guided Local Search. An empirical study over benchmark problems shows the relative merits of these two techniques. Investigations indicate that the specific long-term memory technique used by Guided Local Search can be incorporated within Tabu Search, resulting in some benefit.

Keywords: Vehicle Routing, Constraint Programming, Tabu Search, Guided Local Search

1. Introduction

This paper describes the use of iterative improvement techniques and meta-heuristics as the search engine within a Constraint Programming framework (ILOG Solver) and its application to the Vehicle Routing Problem (VRP).

The VRP is a practical problem that has been studied widely in the literature [10]. The problem is usually expressed as follows: given a set of customers requiring a visit, and a fleet of vehicles based at a depot that can perform the visits, construct a set of routes for the vehicles that minimises the costs of operation. The objective function is usually expressed as costs related to the number of vehicles and to distance travelled. Constraints include various capacity constraints on weight, volume, length, *etc.*, time constraints on when the customer will accept a visit, and the total length of routes. However, in practical problems we may have many different kinds of additional constraints including legislative restrictions, established work practices and customer preferences, and a complex objective function reflecting complicated pay provisions.

Constraint Programming (CP) is a paradigm for representing and solving a wide variety of problems. Problems are expressed in terms of variables, domains for those variables and constraints between the variables. The problems are then solved using complete search techniques such as depth-first search (for satisfaction) and branch and bound (for optimisation).

The richness of the language used to express problems in CP makes it an ideal candidate for VRPs. However, the VRP is an NP-hard problem [3], and for problems of practical size computing optimal solutions can be too time consuming.

In this paper we investigate how iterative improvement techniques can be used in a CP framework to replace complete search. As simple local search procedures get trapped in local minima, the improvement techniques are used in conjunction with a meta-heuristic. We examine two meta-heuristics: a simple Tabu Search approach [5, 6], and Guided Local Search [21, 23].

The paper is organised as follows. In section 2 we describe the Constraint Programming paradigm, how it can be applied to the VRP, and how local search can be performed within this framework. In section 3 the solution technique using meta-heuristics is discussed. In section 4, computational tests on the two meta-heuristics are reported. Investigations reveal that the performance of simple Tabu Search can be significantly increased by incorporating specific long-term memory features of Guided Local Search.

2. Constraint-Based Modelling of VRPs

Part of the attraction of Constraint Programming is the richness of the language for specifying the model. ILOG Solver [14] allows real and integer variables to be constrained by simple bounds and linear or non-linear constraints. Similarly the objective function can be complex.

Very general expressions can be used as constraints. In addition to expressions involving the usual arithmetic and logical operators, complex symbolic constraints can be used to describe a problem. Examples include the all-different constraint, which is used to express (and efficiently propagate) the fact that each variable in a set must take a different value, and the ability to constrain the index of an array (by using a constrained variable).

CP enhances search using *constraint propagation*. If bounds or constraints on a variable can be inferred, or are tentatively set, these changes are “propagated” through all the constraints to reduce the domains of constrained variables.

ILOG Solver uses depth-first search with constraint propagation. At each node in the search tree, the propagation mechanism removes values from the domains of constrained variables that are inconsistent with other variables. If the propagation mechanism removes all values from any variable, then there cannot be a solution in this subtree and the search backtracks making a different decision at the backtrack point. Backtracking is chronological: decisions can only be undone in the opposite order to which they were made. Moreover, the domains of constrained variables can only be reduced as search progresses down the tree. The domains of all constrained variables can be restored by backtracking to a previous node, but general enlargement (or relaxation) of domains is not supported. This has important implications for the way in which vehicle routing problems are solved in this paper, as discussed in section 2.4.

2.1. *A Constraint Programming Model for the VRP*

A decision variable R_i is associated with each customer visit i, representing the next visit performed by the same vehicle. For each vehicle k, there are additional visits S_k marking the start of the route, and E_k marking the end of the route. To read off the itinerary for a vehicle k, start at S_k and follow the "next" pointers through to E_k.

The set of customer visits will be referred to as N, starting visits as S, and ending visits as E. $A =_{\text{def}} N \cup S \cup E$ is all visits. Regarding R as a function, R maps $N \cup S$ onto $N \cup E$.

To model the VRP and other routing problems, it is desirable to have a special constraint type for dealing with constraints along paths. We use a *path* constraint [1, 8] that allows real-valued quantities accumulated along a route to be bounded. The quantities accumulated can relate to time, load, or other variables. The constraints are of the form

$$R_i = j \Rightarrow Q_j \geq Q_i + q_{ij}$$

Thus, if visit j immediately follows visit i, quantity Q is accumulated. For example, setting q_{ij} equal to the travel time between i and j means Q_j would be the arrival time at j. Setting q_{ij} to be the demand at i would accumulate load on the vehicle. This specification allows other real-world constraints to be expressed succinctly.

In each case a *fixed point* (boundary condition) must be supplied, for example $Q_i = 0 \ \forall\, i \in S$ in the case of load constraints. A single constraint of this type which contains an upper bound also has the effect of eliminating subtours (routes that do not include a S_k and E_k visit).

2.2. *Core Constraints*

The core constraints in vehicle routing problems are related to time and capacity. Time is restricted both by the working day and by time windows at customers. Capacity may be restricted in terms of weight, volume, number of pallet places, *etc.*

A path constraint can be used to propagate the start time of the customer service along each vehicle route. In this case, q_{ij} is the service time at i, plus the travel time from i to j.

Single or multiple service time windows could be set by further constraining the start of service variable for each visit. The start and end of the working day for each vehicle can be represented by imposing time windows on visits in S and E.

A method of modelling capacity constraints on vehicles is to propagate (via the path constraint) the free space in the vehicle along each vehicle route. To model F_i as the free space on arrival at visit i, q_{ij} is the change in free space associated with visit i. To indicate that all vehicles must not be overfilled at any time, constraints of the form $F_i \geq 0 \ \forall i \in A$ are imposed. For vehicle k with capacity c_k, the fixed point $F_k = c_k$ for $k \in S$ can be set. Simultaneous pickups and deliveries can be modelled in a similar way, but this is not discussed here.

2.3. Side Constraints

One of the significant benefits of CP techniques is that side constraints can be incorporated into the model with comparative ease. For example, some visits may require special equipment, or some vehicles may be simply too large to enter the premises. Therefore, we need to be able to disallow a visit being made by particular vehicles.

We model this by propagating a vehicle tag τ_k for each visit k along each route, using a constraint similar to the path constraint for which $R_i = j \Rightarrow \tau_j = \tau_i$ with $\tau_i = i$ for $i \in S$. Thus customer visits served by vehicle k will have tag k. Vehicle k is then forbidden from making visit i by imposing a constraint of the form $\tau_i \neq k$. A conjunction of such expressions can be used to exclude more than one vehicle.

The requirement that two visits i and j must be made by the same (different) vehicles can also be specified using tags: $\tau_i = (\neq) \tau_j$.

Implementing a length capacity for vehicles provides one final example that demonstrates a powerful use of vehicle tags. A length restriction is different from a capacity constraint because it does not accumulate over a route. If visit i involves goods of length l, and L_k is the length of vehicle k, then the constraint $L_{\tau_i} \geq l$ defines the length constraint.

2.4. Search Strategy

Solutions to CP problems are usually found using complete methods such as depth-first search and branch and bound. However, for routing problems of practical size, complete search methods cannot produce solutions in a short and reliable time period. By contrast, iterative improvement methods have proved very successful in this regard.

Iterative improvement methods operate by changing small parts of the solution, for instance moving a visit from one route to another. This type of operation involves retracting previous decisions and making new ones. In contrast to depth-first search, retraction can be done in any order, not simply in the opposite order the decisions were made. In general the overhead of implementing this type of non-chronological retraction mechanism in CP frameworks is very high.

To overcome this problem, the CP system is only used to check the validity of solutions and determine the values of constrained variables, not to *search* for solutions. The search is performed by an iterative improvement procedure. When the procedure needs to check the validity of a potential solution, it is handed to the CP system. As a part of checking, constraint propagation using all constraints still takes place. This adds value to the constraint check, as the iterative improvement method can then take advantage of the reduced domains to speed up search by performing fast legality checks (see later in this section).

The implementation relies on two representations of the solution. There is a passive representation, which is the template against which new solutions are constructed, and which holds the "current solution" (not necessarily the best solution). The second is an active representation that holds the constrained variables, and within which constraint propagation takes place. Variable states in the active representation are "checkpointed" (current domains are saved) before any changes are made, so that domains can be restored later.

When the CP system performs propagation and validity checks it instantiates the set of decision variables R using the itinerary in the passive representation. The constraints then propagate and constrained variables (such as time and capacity variables) have their domains reduced. If any variable has no legal values, the solution is illegal.

Improvement heuristics generally only modify a very small part of a routing plan. Therefore, testing the complete solution in the above manner can be inefficient. We have tried to avoid this inefficiency in two ways: by reducing the amount of work carried out by the CP system to perform each check, and bypassing the CP checks altogether.

The first method applies if vehicle routes are independent, *i.e.* have no constraints acting between visits in them. This is the case in many routing problems. For these problems, only the routes involved in the modification of the plan need be instantiated in the active representation. Normally, this means just one or two routes will be instantiated.

The second method of increasing efficiency is to perform tests on the core constraints in situ (using the domains of constrained variables such as capacity and time, preserved from the last propagation, as described earlier). Propagation precisely computes the remaining slack for each cumulative (Q) variable. This slack is reduced as much as possible using *all* constraints (*i.e.* all side constraints) in the problem. These in situ checks can all be done in constant time using a method derived from Savelsbergh [17]. Only if the constant time check is successful need the route set be passed to the CP system for full checking.

The acceleration techniques just described can increase performance of a VRP search engine by a factor of twenty or more, such that up to 200,000 neighbourhood moves can be examined per second on a high-end PC [1].

3. Solution Methods

Local search operates by considering a set of possible moves. The moves considered in this paper are described in the following section. Local search repeatedly chooses a move which reduces the objective value, and then implements this move, until no further legal cost-reducing moves are available. Two alternatives in choosing a move to perform are “first accept”: choose the first cost-reducing move found, or “best accept”: choose the legal move that reduces the objective by the largest amount. The best accept method is used throughout this paper.

Local search suffers from the problem that it will get caught in the first local minimum that it encounters. In order to escape a local minimum, a controlled method of accepting an up-hill move is required. Meta-heuristics provide the framework for accepting such moves. In this paper, we examine two meta-heuristics, tabu search [5, 6], and guided local search [21, 23]. These are described in more detail later.

3.1. Move Operators

We have considered four move operators (see figure 1):

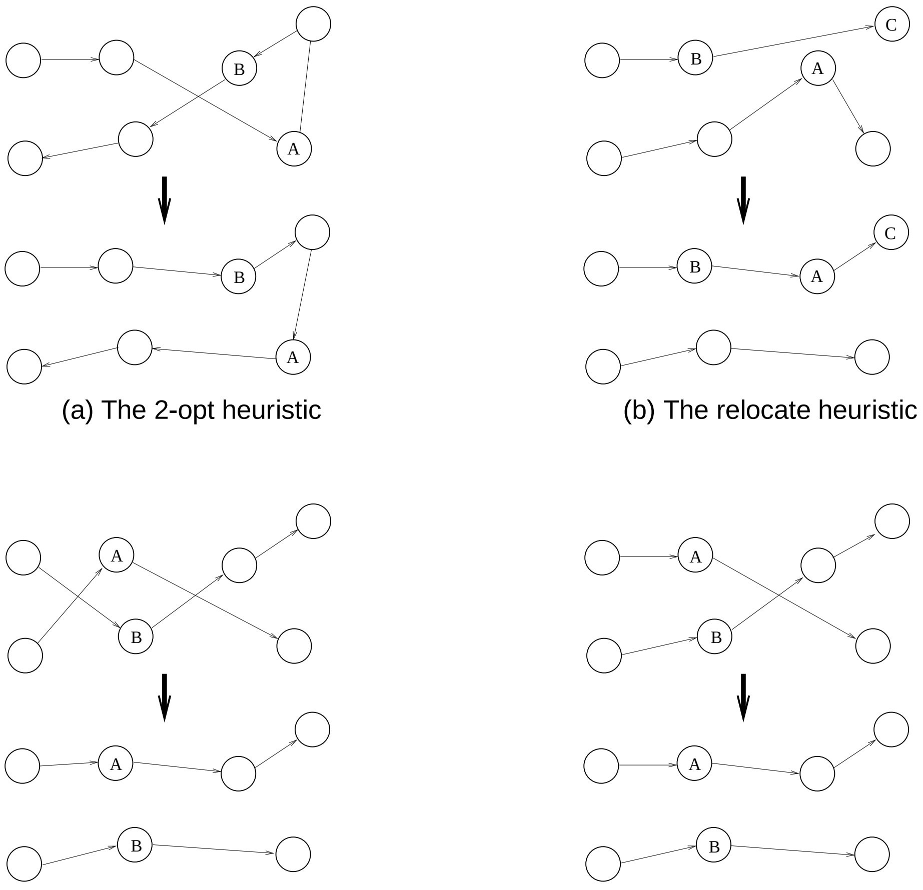
a) 2-opt reverses a section of a route by deleting two arcs, and replacing them with two others to reform the route.

b) Relocate moves a visit from its position in one route to another position in either the same or a different route.

c) Exchange swaps two visits from either the same or different route.

d) Cross swaps the end portions of two route.

2-opt is described in [11], the others in [16].



(a) The 2-opt heuristic (b) The relocate heuristic

(c) The exchange heuristic (d) The cross heuristic

Figure 1. Four move operators

3.2. Tabu Search

Tabu Search is a meta-heuristic that has been widely applied to many combinatorial optimization problems ([7] gives numerous applications) with great success. In particular, Tabu Search been extensively applied to vehicle routing problems, with good results, for example [4, 12, 13, 15, 19]. Tabu Search is sufficiently well known that we do not give a general description, but descriptions can be found in [5, 6, 7].

Table 1. Thresholds for different move operators

Operator	arcs	threshold
cross	4	3
exchange	8	6
relocate	6	5
2-opt	*	3

Our implementation of Tabu Search is a simple one involving tabu moves. It does not use long-term memory for explicit intensification and diversification phases.

For the VRP, moves can be characterised by two sets of directed arcs: arcs added and arcs removed. For example, the relocate operator (figure 1(b)) removes three arcs and adds three new arcs. We use two tabu lists of fixed and equal size: one for storing the most recently added arcs, and the other for storing the most recently removed arcs.

We use the following procedures:

- LocalSearch($\mathcal{S}$) performs a local search, starting at solution $\mathcal{S}$, and returns a new solution. We use a "best accept" algorithm: among all the legal moves defined by the move operators (section 3.1), choose the move that decreases the objective by the greatest amount. Repeat until there is no move that strictly decreases the objective.

- RankMoves($\mathcal{S}$) returns a list of all legal moves from $\mathcal{S}$, ordered according to ascending cost difference. That is, moves reducing the objective by the greatest amount will be near the beginning of the list.

- IsNotTabu(m) is true if the move m is not tabu as dictated by the tabu lists. In our implementation, the tabu status of a move m is decided as follows: The arcs that are removed by m and occur in the "recently added" tabu list are counted. Likewise, the arcs that are added by m and occur in the "recently deleted" tabu list are counted. The sum of these is then compared to a threshold dependent on the move type (see table 1). If the threshold has been met or passed, the move is tabu, and is not tabu otherwise. Note that in table 1, the number of arcs modified in 2-opt is variable because the inverted part of the route may be of any length. In all experiments the length of both lists was 100 arcs. An *aspiration criterion* was used which could change the tabu status of a move. If the move would result in a solution better than the current best, then it was accepted even if tabu.

- Perform(m) performs the move.

- InsertInTabuList(m) inserts the added and deleted arcs into the appropriate tabu lists.

The Tabu Search algorithm is described by Figure 2. $O(\mathcal{S})$ is the objective function. The function head(l) returns the first element in the list l, and removes it from l. InitialSolution() and StoppingCondition() are defined in section 4.

```
$\mathcal{S}$ := InitialSolution()            // $\mathcal{S}$ is current solution
$\mathcal{S}$ := LocalSearch ($\mathcal{S}$)          // Make sure we start at a local min
$\mathcal{S}^*$ := $\mathcal{S}$                        // $\mathcal{S}^*$ is best solution
while not StoppingCondition() do
    $moves$ := RankMoves($\mathcal{S}$)
    $moved$ := false
    while not $moved$ do
        $m$ := head($moves$)
        if IsNotTabu($m$) then
            Perform($m$)
            $moved$ := true
            InsertInTabuList($m$)
            if $O(\mathcal{S}) < O(\mathcal{S}^*)$ then
                $\mathcal{S}^*$ := $\mathcal{S}$
return $\mathcal{S}^*$
```

Figure 2. Tabu Search algorithm

3.3. Guided Local Search

Guided Local Search (GLS) [21, 23] is a meta-heuristic based on penalties. The method works by adding a penalty factor to the objective function, based on the experience the search has gained. Roughly speaking, search is penalised if it strays too close to previously visited local minima, allowing escape from local minima. GLS has proved to be an effective meta-heuristic for a range of combinatorial optimization problems, for example see [20, 22, 23, 24].

In what follows, we describe the general algorithm, and then its application to the vehicle routing problem.

3.3.1. The Guided Local Search Algorithm Guided Local Search moves out of a local minimum by penalizing particular solution features that it considers should not occur in a good solution. It defines a modified objective function, augmented with a set of penalty terms on these features. The usual local search method is then invoked to improve the augmented objective function. The cycle of local search and penalty term update can be repeated as often as required.

GLS requires the following components:

- A set of features F. Let the indicator function for feature $i \in F$ be f_i. $f_i(\mathcal{S}) = 1$ if the feature i is in solution $\mathcal{S}$, 0 otherwise.
- A cost vector c. c_i is the "cost" of feature i.
- A penalty factor λ.

GLS tracks penalties applied via a penalty vector p. p_i is the (integer) number of times feature i has been penalised so far. Assuming $O(\mathcal{S})$ is the the original objective function for the problem, GLS defines an augmented objective function:

$$O'(\mathcal{S}) = O(\mathcal{S}) + \lambda \sum_{i \in F} f_i(\mathcal{S}) p_i c_i$$

and requires a local search procedure that minimises it. The user must provide procedure LocalSearch($\mathcal{S}$) that performs a local search, starting at solution $\mathcal{S}$, and returns a new solution. *The improvement is carried out with respect to the augmented objective O'.*

GLS provides a function called ChoosePenaltyFeatures($\mathcal{S}$, p) which takes a solution and the current spread of penalties, and returns the set of features to be penalised. GLS penalises the most costly features in the current solution, weighted by the number of times the features have already been penalised. This has the effect of concentrating penalties on "bad" features, while tending to accept bad features if they keep appearing in local minima (*i.e.* if they seem inevitable). GLS chooses all features $i \in \mathcal{S}$ for which $c_i/(p_i+1)$ is largest amongst the features in $\mathcal{S}$. Unless the cost values in c are equivalent or low multiples of each other, usually only one feature is chosen to penalise.

Assuming the usual functions of InitialSolution() and StoppingCondition() exist, the GLS algorithm is described in figure 3.

```
$p$ := $\overline{0}$                          // All elements zeroed
$\mathcal{S}$ := InitialSolution()             // $\mathcal{S}$ is current solution
$\mathcal{S}$ := LocalSearch ($\mathcal{S}$)   // Make sure we start at a local min
$\mathcal{S}^*$ := $\mathcal{S}$               // $\mathcal{S}^*$ is best solution
while not StoppingCondition() do
    $f$ := ChoosePenaltyFeatures($\mathcal{S}$, $p$)
    forall g in f do
        $p_g$ := $p_g + 1$
    $\mathcal{S}$ := LocalSearch ($\mathcal{S}$)
    if $O(\mathcal{S}) < O(\mathcal{S}^*)$ then
        $\mathcal{S}^*$ := $\mathcal{S}$
return $\mathcal{S}^*$
```

Figure 3. GLS algorithm

Parallels can be drawn between GLS and the use of *frequency memory* in Tabu Search [5]. Frequency memory records the number of times a particular feature has appeared in previous solutions. It is then used in a *diversification* phase to penalise the occurrence of features that have appeared often. However there are two main differences between the ideas. GLS uses its "memory" throughout the search, rather than in a separate phase. This eliminates the need for "tuning parameters" for the use and duration of a diversification phase. Second, the selection of features to penalise in GLS is based on the cost of the features, as well as the frequency of their occurrence. By penalising features that have appeared often, Tabu Search may attempt to drive out “good” features. GLS uses domain knowledge, in the form of the feature costing, to try to avoid this. In addition, the sense in which the frequency is used is different. In GLS the more often a feature appears and is penalised, the less likely it is to be penalised further. The is not the case in Tabu Search.

3.3.2. GLS for Vehicle Routing We define our implementation of GLS for the VRP with reference to the description above.

- Feature set F: In [21, 23] for the Travelling Salesman Problem, arcs were chosen as the feature to penalise. We use the same method here, but use directed arcs, since for routing problems involving time, direction is important.
- Feature costing: We assume that the cost c_a of directed arc feature a is the length of the arc d_a.
- Penalty factor λ: We have found that values of 0.1 to 0.3 work well, and have used 0.2 for the results in section 4.
- LocalSearch($\mathcal{S}$): We again use a “best accept” algorithm: among all the legal moves defined by the operators in section 3.1, choose the move that decreases the objective by the greatest amount. Stop when there is no move that strictly decreases the objective.

  To implement the objective O', the local search is carried out using a modified distance matrix. In the modified matrix, each directed arc a has a penalised distance $d_a + \lambda p_a d_a$. The matrix is updated each time an element of p is changed.

InitialSolution() and StoppingCondition() are defined in section 4.

4. Computational Experiments

Computational experiments were performed using both the Tabu Search and Guided Local Search meta-heuristics. For both heuristics the StoppingCondition() function used was 1000 forward moves in the search space (these 1000 forward moves took under 10 minutes of CPU time). The InitialSolution() function used was that produced by the Savings method of [2], followed by search to a local minimum. The objective function was the total distance travelled by all vehicles. There was no incentive to reduce the number of vehicles. This means that although the results from different methods are comparable with each other, they are not directly comparable with previously reported best values which emphasise reducing the number of vehicles.

After the 1000 moves, a final phase is entered by both Tabu Search and GLS where restrictions are switched off and a search is performed to a local minimum. For GLS, this corresponds to zeroing all penalties, and finding a local minimum. For Tabu Search, we search to a local minimum using no tabu list. This final improvement can be particularly useful in the case of GLS which may be prevented from reaching a true local minimum during search due to the penalty method employed.

The data for comparison are as used by Solomon [18]. These problems are in 6 classes: C1, C2, R1, R2, RC1, RC2. Each class contains between 8 and 12 individual problem instances. All problems have 100 customers, a central depot, capacity constraints, time windows on the time of delivery, and a total route time constraint. The C problems have customers located in clusters. In the R problems the customers are at random positions. The RC problems contain a mix of both random and clustered customers. The C1, R1 and RC1 problems sets have a short scheduling horizon, and require 9 to 19 vehicles. Problem classes C2, R2 and RC2 are more representative of "long-haul" delivery with longer scheduling horizons and require fewer (2 – 4) vehicles. Both time and distance are given by the Euclidean distance between points.

4.1. Performance of the Algorithms

Figure 4 shows the rate of improvement of Tabu Search and GLS over the six problem classes. The graphs were produced as follows: at each forward move, the lowest travel distance found so far for each meta-heuristic on each problem was recorded. This describes a series of monotonically decreasing curves for each meta-heuristic applied to each problem. Then, all curves corresponding to a single problem class and single meta-heuristic were averaged, resulting in 12 curves. Finally, these curves were normalised against a baseline, which is the best average distance value over the class attained by the best meta-heuristic for the class. Thus, in the graphs, one of the meta-heuristics always attains the 0% point.

In five of the six classes, GLS outperforms simple Tabu Search by a margin varying from 1.2% to 2.7%. In the remaining class, Tabu Search performs 0.5% better than GLS.

4.2. Inside the Search

In an attempt to understand the better performance of GLS over simple Tabu Search, we examined the search process by observing the change in objective function. From this, we hope to see when search stagnates, and how aggressively the search attempts to move out of local minima.

Figure 5 shows the best objective value found so far $O(\mathcal{S}^*)$, and the current objective value $O(\mathcal{S})$, as GLS and Tabu Search proceed through 1000 iterations on problem RC107. Problem RC107 is a typical problem, in that the starting solution can be improved. Problem RC107 demonstrates the gradual fall in the cost of $\mathcal{S}^*$. It also shows how GLS diversifies the search after moving to a new best solution. This can be seen in the rise in the cost of $\mathcal{S}$ following a fall in the cost of $\mathcal{S}^*$. After a period of diversification, GLS moves out of the current local minimum, and finds a path to a new local minimum. This trace also shows how the diversification period tends to become longer during search, suggesting that the search must roam farther from the previous best solution in order to reduce the best cost. In comparison, this simple Tabu Search appears to roam within a relatively small distance from $\mathcal{S}^*$, and as such cannot escape from the local minimum it is trapped it.

Figure 6 shows behaviour for problem C106. The initial solution to C106 is optimal, and search is futile. We conjecture that GLS investigated three distinct local minima, or clusters of local minima. It appears that the search repeatedly returns to the same or a nearby local minimum before escaping to a new part of the solution space with a new, deep local minimum. Simple Tabu Search appears to exhibit a harmonic behaviour, swinging away and then back to $\mathcal{S}^*$with a frequency apparently related to the tabu tenure.

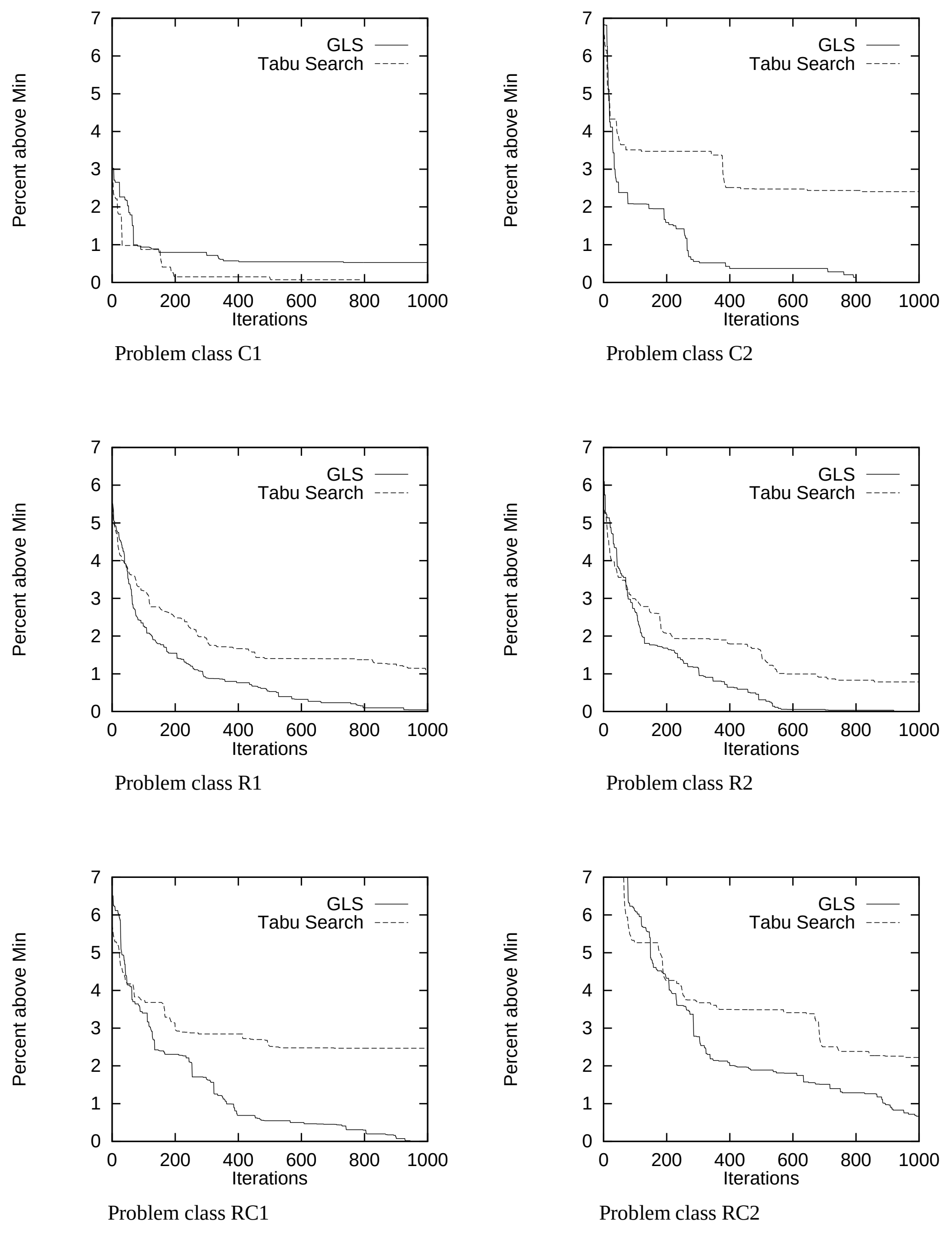
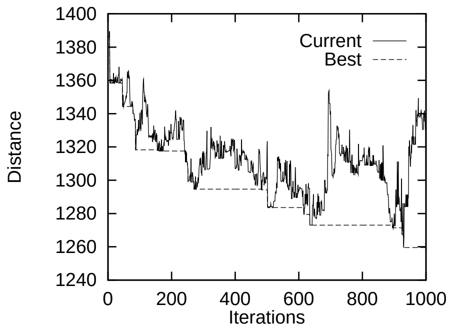


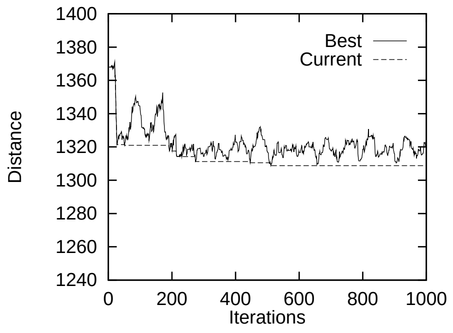
Figure 4. Rate of improvement of Tabu Search and GLS

4.3. Including a GLS-type long-term memory into Tabu Search

The drawback of the simple Tabu Search approach presented is that it is only based on short-term memory. This limits its capability to explore various regions of the search

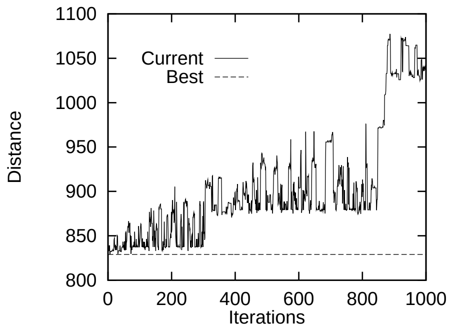


GLS on problem RC107

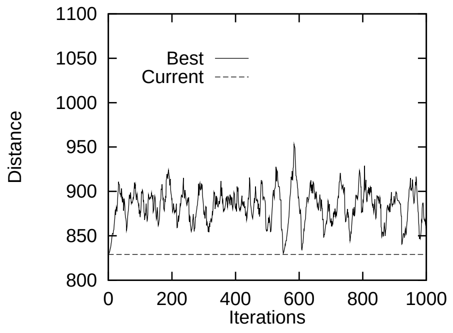


Tabu Search on problem RC107

Figure 5. Inside Search, problem RC107. On the left GLS, and on the right simple Tabu Search. We see that as search proceeds GLS tends to roam progressively further from $S*$ whereas Tabu Search appears to wander within a small distance from $S*$.



GLS on problem C106



Tabu Search on problem C106

Figure 6. Inside Search, problem C106. The starting solution is optimal, and we see on the left, GLS again roams progesssively further from $S*$ and appears to find 2 or 3 new local minima. On the right, a harmonic appears within simple Tabu Search, the frequency possibly a function of the tabu tenure.

space. To better diversify the search, various procedures have already been proposed in the literature including frequency memory as discussed in section 3.3. (For an alternative method that addresses the specific case of the VRP, see [15].)

GLS, because of its simplicity and the excellent results it gives (see [9] for a comparison of GLS with best reported methods) seems to offer an alternative method of implementing long-term memory within Tabu Search with diversification. It has two advantages over other methods: incorporation of domain knowledge through the costing of arcs, and constant diversification without the need for a separate phase.

The resulting Tabu Search still operates by selecting a non-tabu move from the available neighbourhood. However, after each move a GLS-type procedure is called to update the weights in the cost matrix. This method is called Guided Tabu Search (GTS) below.

For the combined method, the length of the tabu lists remained at 100 arcs. However, very slightly better results were achieved by reducing the penalty factor from 0.2 to 0.15.

The performance of GTS relative to GLS and simple Tabu Search is shown in figure 7.

Guided Tabu Search outperforms the other two methods on the “long haul” problems (classes C2, R2 and RC2), performing on average about 1.7% better. For the remaining problem classes, GTS is not quite as convincing, but competitive.

5. Conclusion

We have described a method for combining Constraint Programming with iterative improvement techniques. The combination makes full use of the propagation facilities of the Constraint Programming framework, but does not use a backtracking procedure. The method was accelerated using a system whereby certain core constraints are maintained within the search engine, while other side constraints are handled by the Constraint Programming system. This method was applied to the vehicle routing problem.

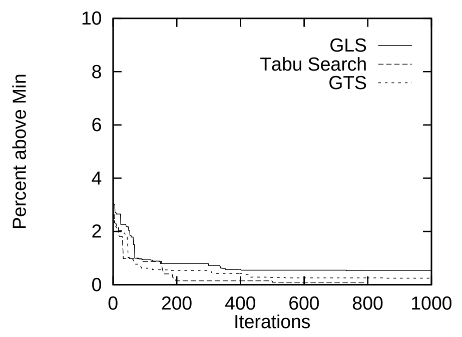
Search engines using two different meta-heuristics were examined: a simple Tabu Search technique, and Guided Local Search. These techniques were compared over a set of standard benchmark problems. Guided Local Search was found to be superior to the simple Tabu Search method. Examination of the behaviour of both algorithms suggested that adding features of Guided Local Search to Tabu Search would improve its performance. A combined method, Guided Tabu Search, was able to outperform the other methods on most problems.

In the future, we wish to examine other diversification methods in Tabu Search, and compare the results to those obtained with Guided Tabu Search. Intensification strategies in Guided Local and Guided Tabu Search are also of interest.

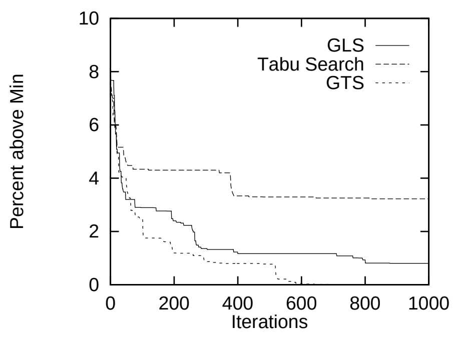
Further testing on real-world problems is in progress. We hope this will give us further insights into ways of improving the methods presented.

Acknowledgments

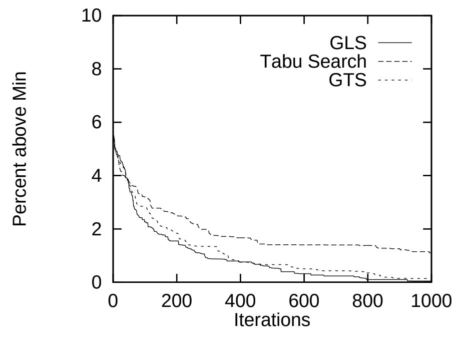
The production of this paper was supported by the GreenTrip project, a research and development undertaking partially funded by the ESPRIT Programme of the Commission of the European Union as project number 20603. The partners in this project are Pirelli (I), ILOG (F), SINTEF (N), Tollpost-Globe (N),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



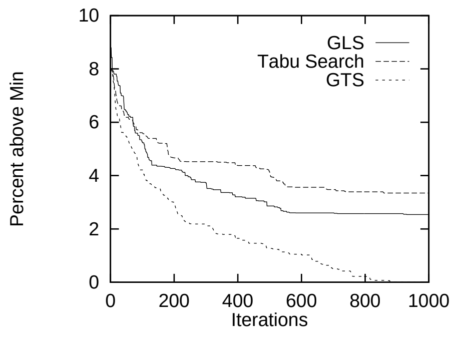
Problem class C1



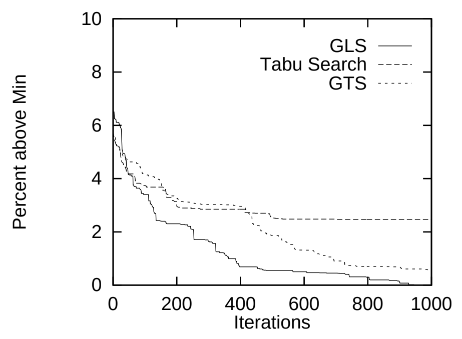
Problem class C2



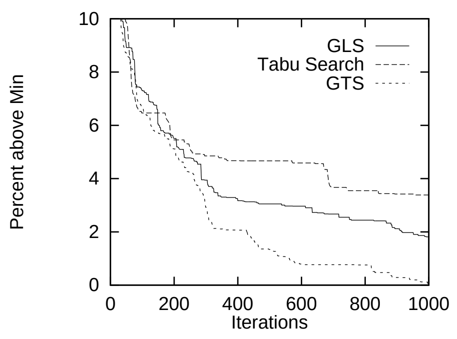
Problem class R1



Problem class R2



Problem class RC1



Problem class RC2

Figure 7. Rate of improvement of basic Tabu Search, GLS, and Tabu Search with GLS long-term memory

References

1. B. De Backer and V. Furnon. Meta-heuristics in constraint programming: Experiments with tabu search on

the vehicle routing problem. In *Proceedings of the 2nd International Conference on Meta-heuristics*, 1997.

2. G. Clarke and G. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.
3. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman, 1979.
4. M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.
5. F. Glover. Tabu search, part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
6. F. Glover. Tabu search, part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
7. F. Glover and M. Laguna. Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–150. McGraw-Hill, 1995.
8. ILOG S.A., 9, Rue de Verdun, Gentilly, France. *ILOG Solver Reference Manual, Version 3.2*.
9. P. Kilby, P. Prosser, and P. Shaw. Guided local search for the vehicle routing problem. In *Proceedings of the 2nd International Conference on Meta-heuristics*, 1997.
10. G. Laporte and I. H. Osman. Routing problems: A bibliography. *Annals of Operations Research*, 61:227–262, 1995.
11. S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Technology Journal*, 44:2245–2269, 1965.
12. I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.
13. J.-Y. Potvin, T. Kervahut, B.-L. Garcia, and J.-M. Rousseau. A tabu search heuristic for the vehicle routing problem with time windows. Technical Report CRT-855, Centre de Recherche sur les Transports, University of Montreal, 1993.
14. J.-F. Puget and M. Leconte. Beyond the glass-box: Constraints as objects. In *Proceedings of ILPS '95*. MIT Press, 1995.
15. Y. Rochat and E. D. Taillard. Probabilitic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, 1995.
16. M. W. P. Savelsbergh. Computer aided routing. Centrum voor Wiskunde en Informatica, Amsterdam, 1988.
17. M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.
18. M. M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35:254–265, 1987.
19. 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.
20. E. Tsang and C. Voudouris. Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. Technical Report CSM-246, Department of Computer Science, University of Essex, August 1995.
21. C. Voudouris. *Guided Local Search for Combinatorial Problems*. PhD thesis, University of Essex, April 1997.
22. C. Voudouris and E. Tsang. Function optimization using guided local search. Technical Report CSM-249, Department of Computer Science, University of Essex, September 1995.
23. C. Voudouris and E. Tsang. Guided local search. Technical Report CSM-247, Department of Computer Science, University of Essex, August 1995.
24. 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.