Journal of the Operational Research Society (2013) 64, 34–47

www.palgrave-journals.com/jors/

Investigating the use of metaheuristics for solving single vehicle routing problems with time-varying traversal costs

K Harwood[1*], C Mumford[1] and R Eglese[2]

[1]*Cardiff University, Wales, UK; and* [2]*Lancaster University Management School, Lancaster, UK*

Metaheuristic algorithms, such as simulated annealing and tabu search, are popular solution techniques for vehicle routing problems (VRPs). These approaches rely on iterative improvements to a starting solution, involving slight alterations to the routes (ie, neighbourhood moves), moving a node to a different part of a solution, swapping nodes or inverting sections of a tour, for example. When working with standard VRPs, where the costs of the arcs do not vary with advancing time, evaluating changes to the total cost following a neighbourhood move is a simple process: simply subtract the cost of the links removed from the solution and add the costs for the new links. When a time-varying aspect (eg, congestion) is included in the costs, these calculations become estimations rather than exact values. This paper focuses on a single vehicle routing problem, similar to the Travelling Salesman Problem, and investigates the potential for using estimation methods on simple models with time-variant costs, mimicking the effects of road congestion.

Journal of the Operational Research Society (2013) **64,** 34–47. doi:10.1057/jors.2012.17
Published online 4 April 2012

Keywords: vehicle routing; time varying; congestion; Road Timetable; 2-Opt

1. Introduction

The vehicle routing problem (VRP) is a combinatorial optimisation problem where a set of customers are visited by a fleet of vehicles (in some cases, the 'fleet' may consist of only one vehicle). Each vehicle starts at a depot, traversing arcs from customer to customer forming a tour, returning to the depot at the end of the tour. The customers and depots are represented by nodes with arcs connecting them forming a connected graph. Each of the arcs connecting the nodes has an associated cost, and the goal is to visit all the customers in such a way that the overall cost is minimised. The cost can be measured in various ways and may refer to distance, time, or the number of vehicles, for example, or weighted combinations of these. There are a number of different variants on the VRP. For example, customers may have to be visited at certain times (ie, within specified time windows), and this may involve deliveries, pickups or both. This paper considers a fairly new variant, in which the costs (in this case, travel times) between customers vary according to the time of day or the day of the week, whereas in standard VRPs they are fixed.

*Correspondence: K Harwood, Computer Science & Informatics, Cardiff University, Queen's Buildings, 5 The Parade, Roath, Cardiff, Wales CF24 3AA, UK.
E-mail: K.G.Harwood@cs.cardiff.ac.uk

Given the predominance of heuristic and metaheuristic methods for solving large real-world VRPs, we concentrate on the important research issues surrounding the concept of the 'neighbourhood move', which is a central feature of all solution methods that rely on local search. Assessing whether or not a neighbourhood move produces an improvement is not so straightforward in a time-varying environment in which travel times between locations are likely to vary depending on the level of congestion.

To keep things simple, we will focus on the single vehicle routing problem (SVRP). This is functionally a slight modification of the standard Travelling Salesman Problem (TSP). TSPs feature a single vehicle that takes a tour of a number of customers, visiting each customer exactly once and returning to its starting location at the end of the tour. With the TSP, the tour can begin with ANY city. With the SVRP, the single tour must start and end with a single depot. The goal is to try to minimise the total cost (in this case, time taken) by finding 'cheaper' routes for the vehicle while still visiting all the customers within any constraints of the problem. Although the SVRP is essentially identical to the TSP under normal circumstances, when travel times along arcs are subject to variation, selecting alternative starting/finishing nodes for a particular tour can significantly affect the total travel times (ie, tour lengths). For this reason, we will refer to our problem as the SVRP rather than the TSP throughout this paper.

In the next section, we introduce time-varying travel networks and discuss the difficulties that arise when implementing heuristic or metaheuristic search. A basic problem formulation is also included. Section 3 covers the background to the problem in more detail, examining related literature, and considering various approaches to modelling the problem and some of the difficulties that have been encountered. In Section 4, we scope our experiments, explaining in detail the neighbourhood move we will be using, and describing our test instances and how we construct starting solutions. Our experimental plan is presented in Section 5 along with an explanation of how the results will be evaluated. The results themselves are also presented in this section, and a discussion of our main findings follows. Finally, in the last section, we draw conclusions and talk about future work that is planned.

2. Time-varying travel networks

In standard VRPs and TSPs, the costs of traversing the arcs are the same throughout the lifetime of the problem. However, in the real world, this is not usually the case when we are considering travel times, as many roads are quicker to traverse at some times of day than at others. There are a number of factors that contribute to this, but congestion is a major cause. By congestion we mean 'crowding' on the road, leading to queuing and speed reduction. However, congestion itself is impossible to predict exactly, even when reliable data on traffic volume are available. Weather conditions and 'random' effects caused by accidents, broken-down vehicles and so on also have an impact. Nevertheless, route planners can benefit from historical data for traffic volume and road speed, to help them avoid using certain roads at regularly congested times (Eglese *et al*, 2006).

Normally, when using a heuristic or metaheuristic improvement method to solve a VRP, small changes or *neighbourhood moves* are made to a starting solution in an attempt to improve the solution in a step-by-step fashion. At each iteration, a neighbourhood move is applied and the quality of the new solution is assessed. Over a period of time, the search will focus on the most promising solutions and the poorest will be discarded. The exact procedure chosen will depend on the precise search algorithm adopted.

Solution quality can be evaluated in a number of ways, but we will simply take the total time required to visit all the customers. For time invariant SVRPs the value of the objective function for a new solution generated by a neighbourhood move is easily calculated from the previous solution. This is done by simply subtracting the costs of the arcs that have been removed from the solution and adding the costs of the arcs that are new to the solution. However, with the time variant model this is not so straightforward. Simply subtracting the costs of the deleted arcs and adding the costs of the newly created arcs will not produce an accurate result, because changes that are made early on in a tour will have a knock-on effect, with arcs traversed later in the tour being reached at a different time of day than they were in the previous solution. If some roads are more congested than others, for example during the rush hour, a detour later on in a tour may save time, even if a longer travel distance is involved.

In order to calculate the precise effect of a neighbourhood move on a tour with time-varying traversal costs, every arc from the first change in the tour until the end of that tour will need to be examined and the cumulative cost of the path containing all these arcs from the latter part of the tour must be re-computed. Once the total tour length for the new solution has been calculated, it can be compared with the tour length of its predecessor. The computational effort required to re-evaluate large parts of a route every time a neighbourhood move is tried is obviously considerably higher when compared with the simple procedure of adding and deleting a small number of edges. Clearly, applying the simple calculations for adding and deleting edges in the time-varying case will give an 'estimate' of the actual change, which may be acceptable in certain situations, especially if congestion is very light or if it uniformly affects most roads in a similar way.

The main focus of this paper is to consider ways in which it is possible to determine, with reasonable accuracy, whether a neighbourhood move is likely to lead to an improvement of the current objective value, while keeping the computing time to a minimum.

We will examine the accuracy and potential usefulness of applying some simple estimates of tour length changes in order to guide heuristic and metaheuristic search procedures applied to time-dependent SVRPs. For this preliminary work, we will base our SVRP instances on benchmark TSP instances from TSPLIB (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, accessed 10 January 2011), for which we will identify one of the nodes as the depot. To the distance matrix for the arcs of the TSP, we will apply speed matrices that will vary according to the time of day and in this way we will obtain our travel times. We will study the trade-off between computation speed (estimates are faster) *versus* computation accuracy (re-evaluations of all affected arcs will give more reliable results). We expect that in some situations an estimate will be good enough to guide a heuristic or metaheuristic algorithm towards better solutions, and in other cases that it will not be, requiring a fuller evaluation of the new tour. We eventually aim to establish some useful rules of thumb, which could be used to guide a heuristic or metaheuristic algorithm and inform it at each stage whether or not an estimate is appropriate.

2.1. Problem formulation

Although congestion levels are a constantly changing value, and a perfect model would reflect this, it is much easier to simplify our model by discretising the data. The method we have used is to divide the day up into 'time bins', periods of time during which the speed on each arc is assumed to be the same throughout. Obviously, the more time bins that are used, the closer the model is to the real world, but also the more calculations and lookups are needed. Time bins are explained further in the next section.

A network for the single vehicle, time variant VRP that we will investigate consists of:

- a connected graph, $\mathcal{G}$ (which may or may not be a complete graph);
- a set of n vertices, $\mathcal{V} = \{v_1 ... v_n\} \in \mathcal{G}$ with v_1 the depot and $\{v_2 \dots v_n\}$ the customers;
- a set of directed edges, $\mathcal{E} = \{e_{v_i v_j} \in \mathcal{G} | v_i, v_j \in \mathcal{V};\ v_i \neq v_j\}$;
- a set of r time bins, $\mathcal{P} = \{p_1 ... p_r\}$, we will assume each has equal width w;
- a set of traversal times for every edge $e_{v_i v_j} \in \mathcal{E}$ for each time bin $p_k \in \mathcal{P}$, $\mathcal{T}(e_{v_i v_j}, p_k)$.

A solution is an ordered set $\mathcal{S} = \{s_1 \dots s_{n+1}\}$ beginning and ending with the depot, $s_1 = s_{n+1} = v_1$, with $\{s_2 \dots s_n\}$ a permutation of the customer vertices, $\{v_2, \dots, v_n\}$.

The objective is to minimise the total traversal time of the tour, that is, the sum of the traversal times of each link. A simplified pseudocode version of the method we used for calculating the total traversal time is shown in Algorithm 1.

TT = total traversal time and t = current time.

Algorithm 1 Total Traversal Time Computation
$t \leftarrow 0$ {The start time is 0 in all our experiments, but this need not be the case}
$TT \leftarrow 0$ {Initialise total travel time}
for $i \leftarrow 1$ to n **do**
$j \leftarrow (\lfloor t/w \rfloor \bmod r) + 1$ {Possibly roll over to next day}
$TT \leftarrow TT + \mathcal{T}(e_{s_i s_{i+1}}, p_j)$ {Lookup from table}
$t \leftarrow t + \mathcal{T}(e_{s_i s_{i+1}}, p_j)$ {Any loading times would also be added here}
end for
Print TT {The total time spent to traverse the graph}

Assigning TT involves looking up the traversal times for each link in a table, using the edges's source and destination nodes and the current time bin. This pseudocode ignores the first in first out (FIFO) problem, which we will cover later.

3. Background

As far as we are aware, previous research in the field of time-varying traversal costs is rather limited, and most of this effort has concentrated on shortest path computations. Efficient methods for evaluating shortest paths in a time-varying environment can be incorporated into more complex VRPs, and thus form essential components of algorithms designed to solve real-world problems. We will start by highlighting key publications covering shortest path algorithms for time-variant travel. Next, we will extend our discussion to time-variant costs for VRPs.

The underlying approach is based on Dijkstra's label setting algorithm (LSA) (Dijkstra, 1959), which finds the shortest path between any two nodes in a network. The original paper noted that the algorithm also works when the arcs are directed, that is, the traversal times are based on the direction of travel between the nodes. Over the years, various researchers have adapted this algorithm: applying it to time-variant networks (Dreyfus, 1969) and examining the limitations of finding the shortest path within a time-varying environment and proving that it is valid for a network that maintains the FIFO property (Kaufman and Smith, 1993), also referred to as the non-passing property (Sung *et al*, 2000). There are a number of ways that this can be achieved, which we will explain later.

There are many approaches to solve standard versions of TSPs and VRPs. However, it is invariably the case that these methods need some modification before they can be applied to VRPs with time-varying traversal costs. We will now look at some approaches that are possible.

We begin with a brief mention of a very simple construction heuristic: a greedy nearest neighbour (NN) algorithm. In its simplest form, NN will begin with a starting node (in the case of our SVRP, this will be the depot) and build a path in an iterative fashion by adding the closest unvisited node to one or the other end of the partial path, eventually producing a path that passes through all the nodes. Finally, the two ends of the path are joined to form a tour. The only modification that needs to be made to NN when used in a time-varying system is that it becomes a directed path, and new nodes can only be added to one end of the path. Although we are not aware of any publications that use NN for solving the time-variant TSP, it clearly has potential for creating an initial solution for a heuristic algorithm.

An exact method for solving VRPs is Dynamic Programming (DP). Its run time complexity of $O(n^2 2^n)$ is a considerable improvement over other exact techniques. Some authors (Kok *et al*, 2009; Malandraki and Dial, 1996) have used a restricted form of DP for problems with time-varying traversal costs that, while no longer exact, gives good results.

By far the most popular solution methods for VRPs involve heuristic and metaheuristic techniques, although very little previous work has been done using these approaches in a time-varying environment. Our aim in the next two subsections is to identify the difficulties that stem from time-varying aspects, and adapt our solution

methodology to accommodate them. One major issue with a time-varying system is how to model the time variance. Another problem, which stems from the modelling issue, is the FIFO problem.

3.1. Modelling time-varying travel

Time-varying travel costs are difficult to model. Although discrete models are desirable, real roads are not uniform and vehicle speeds tend to vary continuously. Most authors (Eglese *et al*, 2006; Ichoua *et al*, 2003; Sung *et al*, 2000) nevertheless resolve the issue by simply assuming that the speed is the same across the entirety of a specified road section. Fortunately, providing the road sections are not too long, this does not seem to cause too much error. Most researchers also discretise the time, such that the speed of a vehicle will remain constant until a new section of road is reached or a new period of time entered.

There are many methods that can be used for modelling congestion. The simplest method used by some authors (Fisher *et al*, 1982; Hill *et al*, 1988) is that of incorporating congestion in a time-dependent way, which we will refer to as *uniform multiplicative congestion*. This is based on two simplifying assumptions: (1) that vehicles travel at the same speed on all roads and (2) that congestion uniformly affects all roads in exactly the same way (eg, between 8 a.m. and 10 a.m. all roads take twice as long to traverse).

In their paper, Ichoua *et al* (2003) introduced and explained the concept of a *Travel Speed Matrix* (TSM). The idea is similar to the uniform multiplicative approach mentioned above, but with a heterogeneous set of roads, rather than homogeneous (eg, some roads are faster to traverse than others, representing the difference between a motorway and a minor road). In the simplified TSM that Ichoua *et al* used, the roads are still affected to the same degree by congestion (so the second assumption of uniform multiplicative congestion is retained). An in-depth review can be found in the next subsection (the FIFO problem).

Lastly, Eglese *et al* (2006) detailed the construction of a Road Timetable using Dijkstra's LSA to create a complete graph of shortest times between the customers. They illustrated this process using a real-world example, which had a base (incomplete) graph with 3326 arcs and 1666 nodes. Of these nodes, 18 were designated as customers and one as the depot. Dijkstra's LSA was used to construct the Road Timetable between these 19 primary entities (a table with the traversal time between every pair of primary nodes during every time bin) and then that was used to form an illustrative instance of the time-varying VRP. Because the times come from samples of actual road speeds, the congestion does not follow a simplistic multiplicative pattern. The instance featured capacitated vehicles, time windows and demand on each customer (randomly generated).

3.2. The FIFO problem

Essentially, the FIFO property states that if a vehicle is traversing a link, then the later it leaves the start node the later it will arrive at the end node. Without the FIFO property, a situation may arise, for example, where the fastest way to get from A to B is to wait at A for 5 min before heading to B. There are some scenarios, such as air travel (Malandraki and Dial, 1996), where this can be appropriate.

In the present paper, however, we are considering only a single vehicle type visiting a predefined set of customers. It has been shown (Horn, 2000) that if the speed of a vehicle is correctly updated whenever it enters a new time bin, then the FIFO property is maintained. Note also that, in reality, it is possible for a vehicle leaving the depot at a later time to completely catch-up with an earlier vehicle. However, in discretised models this can never happen because the model has only one speed for an entire arc's length, whereas in reality there can be traffic queued at one end of a road and not the other.

Some authors (Ichoua *et al*, 2003; Sung *et al*, 2000; Horn, 2000) resolve the problem of FIFO by using a step function. This method involves modelling the congestion levels by splitting the day into discrete time bins, referred to as intervals or periods in Ichoua's work. If a vehicle enters a new time bin while traversing an arc, the amount of time that is spent in each time bin is calculated separately and averaged. The result is that the FIFO property is maintained.

Eglese *et al* (2006) used a method equivalent to the one used by Ichoua *et al*, but coded to reduce computation time when the network is made up of many short arcs. For this paper, we will update vehicle speeds whenever travel across an arc spans more than one time bin in order to model the effects of the time-variant congestion.

4. Scoping our experiments

Recall that the main purpose of this paper is to explore efficient ways to assess neighbourhood moves for heuristic and metaheuristic algorithms operating in a time-varying traversal cost environment. We will tackle this in two stages: first by speeding up the assessment of single neighbourhood moves (ie, the microscopic level), and second by speeding up a complete heuristic/metaheuristic framework (ie, the macroscopic level).

4.1. Overview of experiments

The main idea for our first set of experiments is to test how well estimates can be used to predict whether a given neighbourhood move will produce an improvement to the solution or not, avoiding the need for full evaluations of a neighbourhood tour wherever possible. In the experiments,

we plan to compare estimates with full computations and measure the accuracy of our estimates simply by counting how many times they are correct in their predictions *versus* how many times they are wrong. It is likely that estimates may work well in some situations and not in others, and establishing some simple 'rules of thumb' to provide adaptive guidance to an iterative improvement scheme is our long-term goal. In the second set of experiments, we propose to assess the run time *versus* solution quality trade-off obtained by using estimates during the execution of a simple hillclimbing algorithm.

For the present work, we aim to keep our scope fairly narrow, so that the number of experiments is manageable. Our plan is to use the results of the preliminary experiments we carry out here to guide our future work. Thus for simplicity, we will focus on just one type of neighbourhood move, limit our congestion models to two types and our problem instances to two small ones for the first set of experiments, with two extra instances added for the second set of experiments.

4.2. Assessing individual neighbourhood moves (microscopic level)

Given a particular problem instance, we will begin each experimental run by generating starting tours that are either *random* or *greedy* (further details can be found under Starting Solution Construction). The *random* starting solution is included to see whether estimates will work as effectively when used on a poor solution as they do when used on a rather better solution. A neighbourhood move will then be performed on this starting solution and an 'estimate' of the comparative quality of the new candidate solution will be produced. This estimate will be derived from the time invariant model, that is, we simply take the cost of the arcs that are added and subtract those that are deleted. A 'cumulative tour cost (CTC)' is kept that represents the cost of traversing the arcs up to each node, so the difference in the CTC before and after the arc's traversal is the cost of traversing that arc. The new arcs that are being added are simply looked up in the cost matrix, matching the time slot in which they are traversed. As a brief example, given a complete tour ABCDA, if the time taken to reach node B in the tour is 104 min and the time taken to reach node C is 152 min then the time taken to traverse the arc between B and C is $152-104=48$ min.

Once the 'estimate' has been made, the programme then calculates the actual change that the time-varying cost matrix determines. In order to perform this evaluation, the travel time for every arc that occurs in the tour following the first change is measured (clearly, the first part of the tour is unchanged, so the CTC up until then can be used). The difference between the original solution and the new solution's value is then compared with the 'estimate' calculated earlier and the results are plotted on a graph.

4.3. Assessment in a heuristic/metaheursitic framework (macroscopic level)

When it comes to solving a VRP, there will always be a trade-off between solution quality and resource utilisation, particularly run time. In the first set of experiments, we examine how often our estimation tool makes correct predictions *versus* how often it is wrong. The second set of experiments will focus on trading off solution quality *versus* run time by testing our estimation tool within a simple heuristic framework.

We will use a simple hill climber and test it with and without our estimation tool. At each cycle of the hill climber, a neighbourhood move will be performed and its quality estimated and/or calculated. If the tour is judged to be shorter following the change, then the new tour will replace the current tour as the focus of the search.

4.4. 2-opt and other neighbourhood moves

For all of our experiments, we will be using only one type of neighbourhood move: 2-opt. With this method, two arbitrary nodes are selected, and the path between these two points (inclusive) is inverted and re-attached (see Figure 1). In the time invariant scenario, such a rearrangement will not cause much disruption on a symmetric problem, simply the removal of the preceding arcs to both nodes and the addition of the new connecting arcs. In the time-varying scenario. However, the fact that the intervening nodes are traversed in reverse order can have a much greater effect on the solution quality.

With a 2-opt operation, the resulting tour can be considered in three parts: *pre-change*, *changed* and *post-change*. Illustrative examples are included below (please refer to Figure 1).

Pre-change: The part of the tour from the depot (A) until just before the first change (C). The traversal times will be the same for this section of tour, as the change only takes effect after C.

Changed: This part of the tour is from C to G. This is the section of the tour that has been inverted and will thus be traversed in the opposite direction.

Post-change: Assuming that the FIFO property is held, if the tour is an improvement at node G then it will be an improvement overall and vice versa (the first to enter into the final part of the tour will be the first one to complete it).

Taking into account this division of the problem, it can be seen that only the *changed* section needs to be

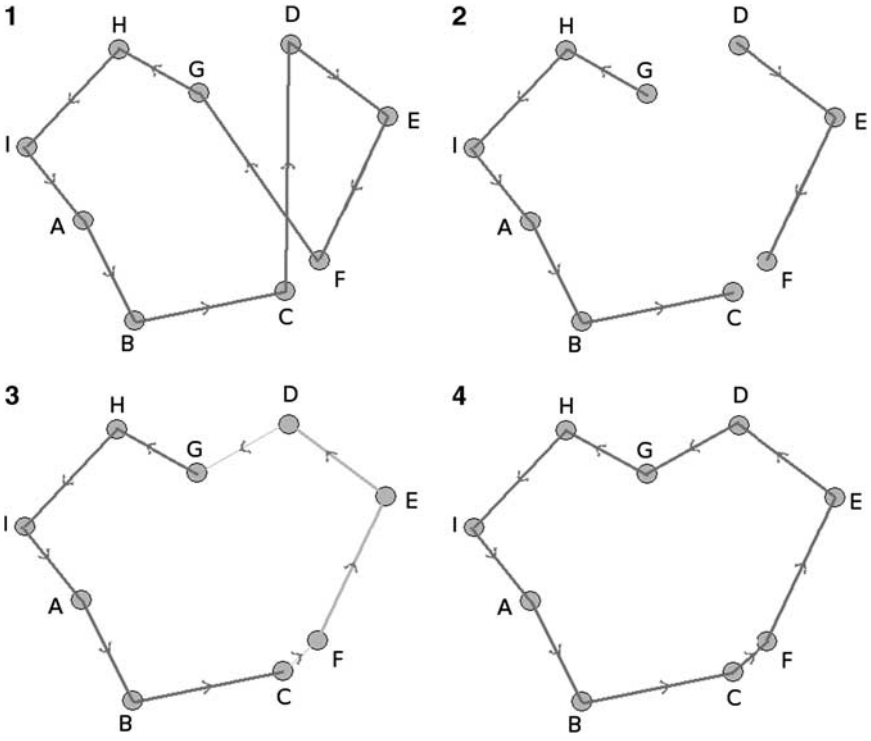

Figure 1 Four step process of 2-opt.

re-calculated in order to find out whether the neighbourhood move will lead to an improvement, although the *post-change* section must be calculated in order to evaluate the magnitude of any improvements.

4.5. Problem instances

The problem instances that we will be using for the first experiment set are based on two instances from TSPLIB (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95, accessed 10 January 2011), bier127 (127 beer gardens in Augsburg (Bavaria) by Juenger/Reinelt) and a280 (drilling problem by Ludwig). These instances are converted into SVRP instances by assigning one city as a depot. Five different variants of each instance are produced by selecting a different node as the depot in each case.

These are symmetric SVRP instances, with the distances between the nodes represented as the Cartesian distances between the points. For the two methods of congestion modelling used in this paper, we will assign travel times to each link and minimise those, rather than travel distance, in our objective function. Furthermore, these travel times will be affected by the speed that the congestion model enforces on the arc, so that there are different speeds on different roads at different times of day. The two TSPLIB instances were chosen because they are quite different in appearance. a280 has an even distribution of nodes in neat lines, whereas bier127 has a tight cluster of points in the centre and then outliers spread out around the centre (see Figure 2).

For the second set of experiments (using the hill climbing framework), we retain the two problem instances used in the first experiments (a280 and bier127) for continuity, and add two more from TSPLIB: a much smaller problem (bayg29) featuring 29 nodes that are irregularly but fairly evenly distributed, and a much larger problem (gr666), which is irregular and clustered, based loosely on the distribution of airports around the globe (but converted into a two-dimensional Cartesian problem).

4.6. Producing congestion values

In their paper, Ichoua *et al* used a simple set-up with three different road classifications (these represent types of road, such as 'motorway' or 'A roads') and three time bins. The congestion in the first and third time bin was the same, representing morning and evening congestion, while the second time bin represented the uncongested travel in the middle of the day. Three different scenarios using this set-up were performed by Ichoua *et al*, with the ratio between the two congestion levels different for each scenario, leading to the situations having different degrees of 'time dependency'. Table 1 illustrates different speeds on different road types at different times of day. A high number represents a faster (and preferable) route.

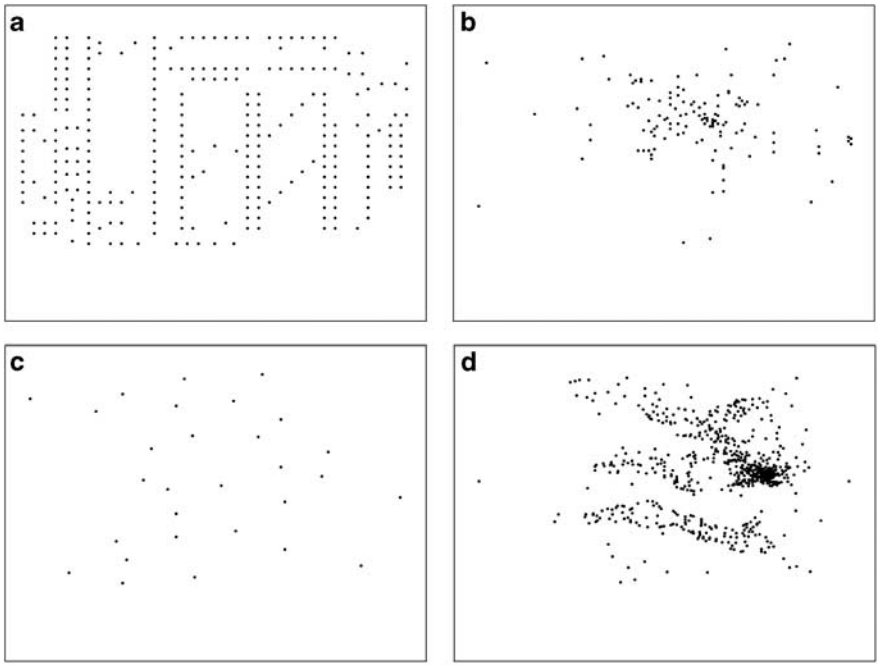

Figure 2 The four problem instances used in our study.

Table 1 Ichoua's TSM 1: Example showing speeds on three road types at different times of day

		Time bins (t)		
Three road type (c)	A	0.54	0.81	0.54
	B	0.81	1.22	0.81
	C	1.22	1.82	1.22

Table 1 is created by assigning speeds to each classification of road and then applying multiplicative congestion. A simple way to show how this works is by using a column matrix to represent the speeds and a row matrix to represent the multiplicative congestion values:

$$\begin{vmatrix} 0.81 \\ 1.22 \\ 1.82 \end{vmatrix} \times | 2/3 \quad 1 \quad 2/3 | = \begin{vmatrix} 0.54 & 0.81 & 0.54 \\ 0.81 & 1.22 & 0.81 \\ 1.22 & 1.82 & 1.22 \end{vmatrix}$$

It is worth noting that, similar to the uniform multiplicative method described earlier, in the TSM implementation used by Ichoua *et al*, all the roads are affected by congestion in exactly the same way. It is only when time windows are involved that the congestion becomes disruptive. For our experiments, the TSM will be using road classification factors of 0.8, 1 and 1.5 to represent hypothetical B roads, A roads and motorways, respectively. The multiplicative factor will then be applied to the roads in the same way as it is in the basic, multiplicative problem.

The uniform multiplicative method uses a homogenous set of roads, on which the speeds are all the same, whereas the TSM introduces heterogeneous roads, upon which there are three different speeds. Therefore, from now on these methods will be referred to as *speed1* and *speed3* to indicate that the roads have all one speed, or three different speeds, respectively. Modelling like this means that the roads are symmetrical, that is, it takes as long to traverse them one direction as it does the other direction. Although this is not reflective of real-life situations, it does simplify the experiments.

Through our experiments, we propose to investigate the effects of congestion in order to help model real-life situations. We will focus on just two congestion models for the first experiment set (see Figure 3). The first model, which we will call *stepped*, is a simple stepped decrease, from 5 (high congestion) down to 1 (no congestion) over the course of the 'day', then an equally steady increase from 1 back to 5 over the 'night'. The other one we will refer to as *twin peak* congestion, starting at 1 at the beginning of the 'day' and changing quite rapidly throughout the day, with a medium level of congestion in the middle of the day and two 'peaks' of high congestion (to simulate the morning and evening rush hours). Both of these congestion models will be repeated from one day into consecutive days, although the runs should not go very far into the second day, if at all. For the second set of experiments, we only plan to use one model, twin peak, with *speed3* roads, to represent morning and afternoon 'rush hour' congestion.

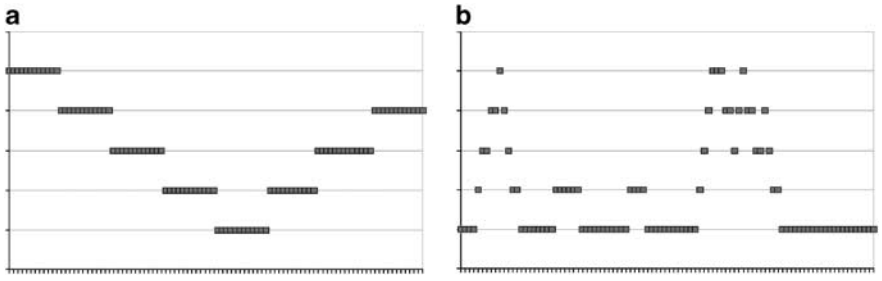

Figure 3 Left: Stepped congestion; Right: Twin peak congestion.

4.7. Starting solution construction

For the first set of experiments, we will produce *greedy* and *random* starting solutions as follows:

1. A greedy NN algorithm was run on each instance, starting from the depot. The algorithm has no random element to it, so only one tour will result in each case. This produces a (comparatively) good solution.
2. A randomised tour was produced for each instance, starting with the corresponding greedy solution, by randomly shuffling the non-depot nodes and then completing the tour by adding the depot to the end. This produces a *random* solution.

Of particular note is that the two types of starting solution will have substantially different tour lengths. *Greedy* tours are generally three times faster to traverse than the *random* tours. Sometimes *random* solutions are so slow to traverse that travel will overflow into a second day.

For the second set of experiments, random tours will be produced using the same method as above. We will be creating 20 starting solutions, five for each of the four problem instances that we use. For each problem instance, five different nodes will be designated as the depot, and these will be chosen in a methodical way; for example, with bier127 the starting nodes will be 1, 26, 51, 76 and 101.

Clearly the effectiveness of our estimation method on greedy starting tours is likely to be of more interest than its performance on random starting tours, given that a heuristic or metaheuristic search spends most of its time enhancing good solutions to make them even better, and little (or no) time at the start of the search dealing with very poor solutions. Indeed, a greedy construction algorithm, such as NN, is frequently applied to produce a starting solution for real-world problems, and the heuristic or metaheuristic search applied to that rather than to a random starting solution. However, we believe that it is important to assess the validity of our estimates in a variety of situations.

4.8. Tour evaluation methods

We will now expand on the alternative methods for evaluating tour quality, referring back to Figure 1. We will consider three approaches, described below:

Naïve: This method is the simplest of the methods we will be using. The pre-change tour does not need to be re-calculated, so this approach starts at C and then calculates the traversal time of each arc from C until it reaches A (ie, all the arcs in both the changed and the post-change sections). If the final result is an improvement on the original then this new tour is used, otherwise it is discarded.

Standard: This method is similar to the naïve method, but with an added calculation that should speed it up, relying on the FIFO property. It calculates every arc of the changed section (from C to G inclusive) and then, upon reaching G, it compares the current CTC with the CTC of the original tour at G. If it is an improvement then it calculates the post-change section in order to find out the overall tour length (and thus determine how much of an improvement it is). If it is not an improvement, it discards it.

Estimate: This method is based on the standard method, but uses the estimation tool first and only calculates the changed section if the estimation tool suggests that it will lead to an improvement, for example, the method first looks at the traversal times of CD and FG in the old tour, then calculates CF and guesses at DG (using the assumption that DG will be traversed at the same time in the new tour as FG was in the old tour). It then compares CD + FG with CF + DG, if the former is quicker then it calculates the changed section exactly as the standard method does, and then the post-change section if it turns out to be an improvement, if it is slower then it discards the new tour without any further calculation.

The tests will involve randomised starting solutions and random choices of nodes upon which to perform the 2-opt operation. The Final Solution Quality (FSQ) should be about the same for the naïve and standard solutions, as they will differ in FSQ only due to experimental error. We would expect the estimation method to produce a poorer FSQ on average.

5. Experimental work

First, we will examine the potential of using estimates, by looking in detail at how effective they are when assessing 2-opt moves. Next, we will look at overall performance when estimates are incorporated into a simple hill-climbing framework.

All coding was implemented in MATLAB Version 7.8.0.347 using a PC running Linux Red Hat on an Intel Quad 2.83 GHz processor with 12MB Advanced Level 2 cache, and 4GB of 800 MHz RAM.

5.1. Assessing the use of estimates in individual neighbourhood moves

We use a 2D graph (Figure 4) to help us assess the usefulness of our estimation method, with each of the points on the diagram representing the estimation of the effect of a neighbourhood move compared with the actual change that is calculated using the time-varying traversal model. We have divided the diagram into four quadrants, which correspond to *true positive* (TP), *false positive* (FP), *true negative* (TN) and *false negative* (FN). To simplify our analysis, we will focus only on membership of the four quadrants, and principally on whether or not the predictions are correct. Using the axes as dividers, the results are easy to interpret (see Table 2).

Table 2 Properties of the four quadrants

Name	*Estimate and reality*	*Description*
FN	Worse and better	Of the most interest, as they are the beneficial solutions that would be ignored if the 'estimate' was used as a guide for the heuristic without any modification
TP	Better and better	Using the estimate as a guide would find these improvements
FP	Better and worse	These would be investigated fruitlessly if the estimate was used as a guide
TN	Worse and worse	Those tours whose needless investigating is being avoided by using the estimate as a guide, and thus saving calculation time

5.2. Results for individual neighbourhood moves

In total we are using two customer distributions, two speed models, two congestion models and two starting solutions (16 experiments) each with five variants based on different choices for the depots, giving 16 × 5 runs in total. Table 3 gives the average results for runs on each of the 5 runs variants for the 16 different experiments. The left half of the table represents the experiments run on a280 and the right shows those run on bier127. The numbers represent the percentage of total solutions that lie in each of the four quadrants on our 'estimate' *versus* 'reality' graph.

The main purpose of these experiments is to investigate to what extent an estimation method can be relied upon. In order to help understand this, a comparison of 'true' results (those in the TP or TN quadrants, which represent the points for which the estimate correctly predicted whether the change would be an improvement) against 'false' results (those in the FN or FP quadrants) needs to be made. A simplified view is that FP costs calculation time, FN costs solution quality, TN saves calculation time and TP contributes to solution quality. Other observations will also be made. Each of the parameters will be looked at in turn.

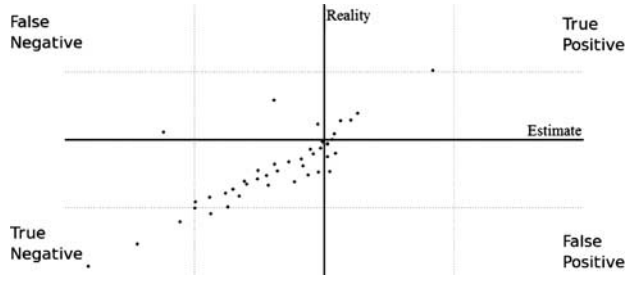

Figure 4 The four quadrants of the 'estimate' *versus* 'reality' graph.

5.2.1. Congestion instance: stepped versus twin peak. Pairing up each of the stepped results with its equivalent twin peak result we find that, in the experimental pairs involving *random* starting solutions, stepped congestion produced more FP and more FN results than twin peak. With *greedy* starting solutions, however, all the pairs had more false results with twin peak (stepped had less FN than twin peak but more FP in every instance).

As may be noticed, the '*greedy* stepped' combination produces few FP results. In total, four of the five runs of a280 speed1 had two FP results (out of a total of 38 781 different node pairs) and bier127 had three runs producing a single FP result (out of 8001). For speed3 neither problem instance had any FP results.

One possible reason why the estimate has more FP on stepped congestion for *random* but virtually none for *greedy* may be because of the initial construction algorithm used. The low number of FP results from the *greedy* runs may be an artifact of the NN construction, which tends to use short edges at the start of the tour, and long edges increasingly as the greedy choice is reduced towards the end of the tour.

From the above observations, we can tentatively conclude that the estimation method, if given a *greedy* starting tour, is able to cope with gradually changing congestion much better than with congestion that alters

Table 3 Results for individual neighbourhood moves (%)

Congestion	*a*280				*bier*127			
	FN	*TP*	*FP*	*TN*	*FN*	*TP*	*FP*	*TN*
Random start								
Speed1 and step	6.49	42.27	6.57	44.67	6.68	41.55	6.65	45.13
Speed1 and twin	4.02	45.67	4.13	46.18	5.09	44.65	5.15	45.11
Speed3 and step	5.55	46.13	5.56	42.76	6.68	41.55	6.65	45.13
Speed3 and twin	3.79	46.34	3.21	46.65	4.73	44.65	4.83	45.79
Greedy start								
Speed1 and step	0.29	0.14	0.00	99.57	0.64	0.42	0.01	98.94
Speed1 and twin	0.09	0.30	0.35	99.26	0.25	0.97	1.02	97.77
Speed3 and step	0.18	0.19	0.00	99.63	0.79	0.19	0.00	99.02
Speed3 and twin	0.07	0.36	0.25	99.32	0.24	0.65	0.74	98.38

more rapidly. This seems sensible: with rapidly changing congestion the quality of the estimate is likely to be poor, but it is reassuring to see this effect in our results. On the results obtained on the 2-opt moves for the *greedy* starting solutions, there were few or no FP results for stepped congestion.

5.2.2. Congestion type: homogeneous (speed1) versus heterogeneous (speed3). In all the experiments with *random* starting solutions, the congestion model used has little noticeable effect on the ratio of FP, FN and TP obtained in our experiments, certainly less than any of the other parameters (such as problem instance). The results from the *greedy* starting solutions have more of a noticeable difference in the ratios. However, this could be explained (at least in part) by the small sample size, as it is much more difficult to improve a good solution than a poorer one. Thus, most of the results for the neighbourhood moves applied to the *greedy* starting solution are in the TN quadrant. For the rest of this section, we will be referring only to the *greedy* results, as there is no statistically significant difference between the two congestion modelling methods for the *random* starting solutions.

In every case, the estimates made for neighbourhood moves for heterogeneous road networks produce less FP results than estimates made on homogenous networks. The heterogeneous routes also have less TP, FP and FN results but more TN results. For the a280 results, the heterogenous routes have a decrease in both FP and FN results and an increase in both TP and TN results. So the estimate method is, in fact, more useful when the roads are heterogeneous. For bier127 it is not quite as good, with less FP for the heterogenous routes and more FN for one of the two congestion models (the other leads to slightly less). In both cases, the ratio of FN to TP is worse for heterogenous (meaning that fewer of the moves that would be improvements are identified by the estimate as being such).

In conclusion, the congestion type of the road networks had little effect in these scenarios. It may still be the case that using a more realistic congestion model based on actual speed measurements will result in the congestion type having more effect. Within the structured SVRP instance, it seemed that the more complex congestion modelling system (with heterogeneous roads) actually worked in favour of the estimation method.

5.2.3. Distribution of nodes: a280 versus bier127. In all the scenarios, there are more FP and FN for bier127 than a280, except for the occasions where neither have any FP results. For the *random* starts, these extra incorrect predictions seem to be at the cost of TP. For *greedy* starting solutions, bier127 has more TP results for twin peak congestion, but about the same for stepped congestion.

The ratio of FP to FN between the two problem instances for twin peak congestion is approximately the same (bier127 has roughly three times as many results of each compared with a280), independent of the congestion type.

From these observations, it seems clear that the problem instance has a fairly important effect on the estimation quality. It seems plausible that the clustered nature of bier127 is leading to the increased inaccuracy in the estimates compared with the more ordered and evenly spread a280. A more extensive study is needed to verify this effect, involving many more problem instances.

5.3. Assessment in a heuristic framework

For these experiments, there are two issues we will investigate: calculation method and problem instance. These will be measured within a simple heuristic framework, using a

basic hill climbing algorithm. Starting with a *random* starting solution, we will perform 1 000 000 2-opt neighbourhood moves on random pairs of nodes/arcs, incorporating any moves that lead to an improvement, timing the whole process and recording the FSQ. For each pair of calculation methods (three methods) and problem instance (four instances), we will run the experiment 25 times in order to get a representative spread of results.

In this section, we present a comparison between the *naïve*, *standard* and *estimation* methods for evaluating neighbourhood moves incorporated within a simple hill climbing heuristic method. We examine the trade-offs between FSQ and run time.

Figure 5 illustrates the results for the hill climber on the four problem instances. In all cases, the estimation method is the fastest, with the standard method second and naïve evaluation slowest.

Tables 4 and 5 give the minimum, average (mean) and maximum for both 'FSQ' and 'time taken to calculate' for each of the problems, along with the average number of improvements (out of a possible 1 000 000). All times are measured in seconds, all FSQs and tours have arbitrary units. Table 6 shows the average percentage change between standard and estimate for each problem (note that both FSQ and time are minimisations, so a negative number represents an improvement for the Estimate over the Standard method).

5.3.1. Interpreting the results. As mentioned before, there are two aspects to a successful heuristic: Run time and FSQ. We can examine this trade-off in Figure 5. The estimation method is much faster in all instances, and gives very good solutions, except for bier127. It is not surprising that the standard method matches the solution quality produced by the naïve method, given that potential improvements are not missed by either of these methods.

Tables 4 and 5 present the results in more detail. The benefits of using the estimation method clearly grow as the instance size becomes larger. For bayg29, the estimate takes 19% less time than the standard method, for bier127

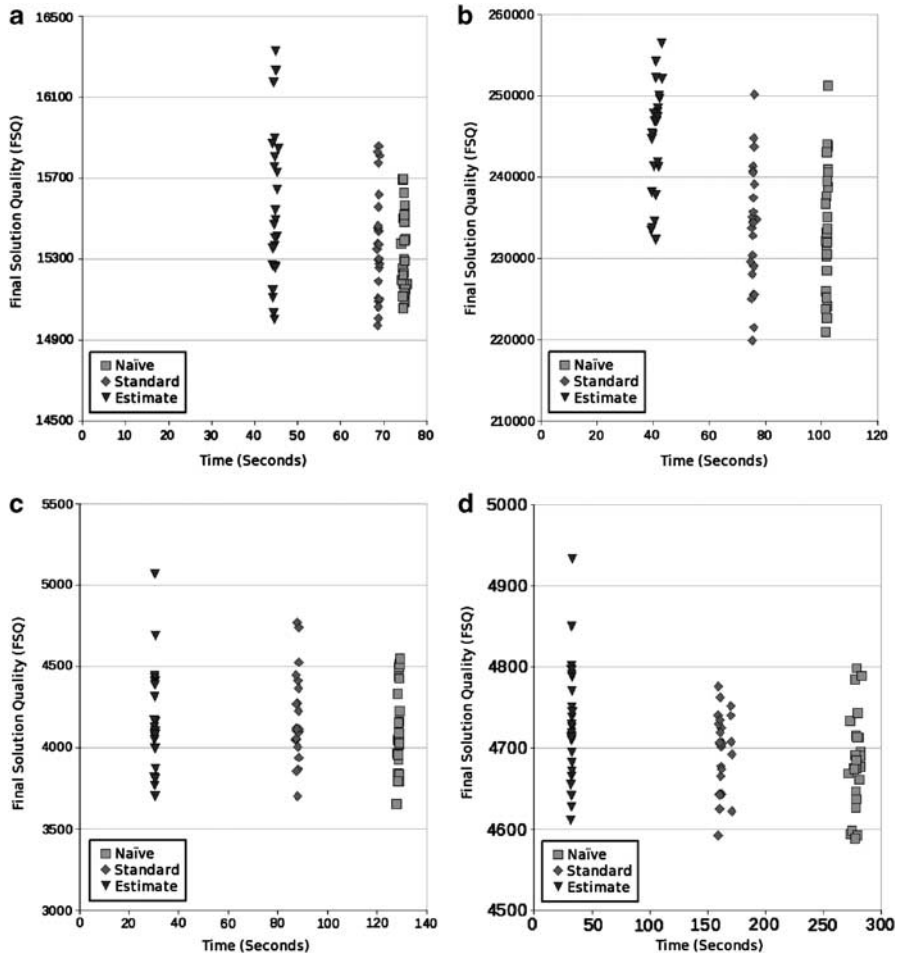

Figure 5 Run time *versus* final solution quality: Comparing *naïve*, *standard* and *estimation* methods for replicated runs of a hill climber. (a) bayg29; (b) bier127; (c) a280; (d) gr666.

Table 4 bayg29 and bier127 results

	*bayg*29			*bier*127		
	Naïve	*Standard*	*Estimate*	*Naïve*	*Standard*	*Estimate*
Minimum FSQ	15 056.86	14 971.98	15 000.88	220 993.97	219 912.75	232 306.76
Average FSQ	15 338.01	15 363.13	15 536.61	233 850.43	233 742.77	244 338.49
Maximum FSQ	15 695.06	15 858.51	16 325.76	251 247.15	250 233.42	256 479.54
Minimum time	74.08	68.46	44.08	101.38	74.62	39.21
Average time	74.66	68.75	44.63	101.86	75.61	40.83
Maximum time	75.47	69.01	45.50	102.35	76.75	43.06
Average improvements	45	43	39	401	398	357

Table 5 a280 and gr666 results

	*a*280			*gr*666		
	Naïve	*Standard*	*Estimate*	*Naïve*	*Standard*	*Estimate*
Minimum FSQ	3655.24	3703.11	3703.39	4588.35	4592.18	4611.18
Average FSQ	4117.28	4177.71	4168.19	4682.86	4695.77	4732.70
Maximum FSQ	4548.36	4768.78	5070.44	4798.69	4776.03	4933.34
Minimum time	127.87	87.22	30.08	271.96	159.00	31.56
Average time	128.65	87.91	30.37	278.49	162.80	32.35
Maximum time	129.31	88.51	30.68	283.39	171.09	32.90
Average improvements	1052	1065	990	3384	3357	3184

Table 6 Average percentage change

	*bayg*29 *(%)*	*bier*127 *(%)*	*a*280 *(%)*	*gr*666 *(%)*
Average FSQ	1.13	4.53	−0.23	0.79
Average time	−18.73	−46.00	−65.46	−80.13

it is 46% faster, a280 sees a saving of 65% and, in the case of gr666, using the estimation method results in a saving of 80% compared with the standard method. These are considerable savings.

One other notable aspect of these results, however, is that the run times (over naïve, standard and estimate) grow rather more slowly than one may expect, in relation to the size of the instance. This can be largely explained because we currently have quite a large computational overhead in our implementation. Nevertheless, we can observe a steady growth in computation time with an increasing number of nodes for the naïve and standard methods. On the other hand, the run times for the estimation method actually reduce as the number of nodes increases. This is indeed somewhat counter-intuitive, and is in part due to a more complex (and time consuming) computation required to implement 2-opt when nodes adjacent to the depot are involved. The smaller the instance, the more likely one of these nodes is selected. In any case, given the small number of improvements found out of 1 000 000 neighbourhood trials in each test run, we would not expect the run time to grow very fast with instance size (see the final row in Tables 4 and 5). Recall that the estimate simply evaluates the difference between the cost of the two arcs added and the two arcs taken away, and also that this operation is performed in constant time, regardless of the number of nodes, with very few complete tour evaluations needed.

Comparing the average FSQ for the estimate method with the FSQs for the standard and naïve method: for bayg29 it was 1.13 and 1.29% worse than standard and naïve, respectively, which is statistically significant. FSQ for bier127 was 4.53 and 4.48% worse than standard and naïve with over a third of the results for both standard and naïve giving better results than the best of the estimates for this instance. This instance is clearly the worst for the estimation method in terms of FSQ, although why this may be is not yet known. FSQ for a280 was 0.23% better and 1.24% worse—by random chance the estimate method actually gives better results than the standard method. The results for bier127 and a280 demonstrate once more that the estimation tool is more accurate when applied to a structured problem like a280 than it is with the clustered bier127. Lastly, the estimate's FSQ for gr666 was 0.79 and 1.06% worse than standard and naïve.

To sum up, it is indeed possible to considerably reduce computation times (from 35% with small problems up to over 80% with large instances) without compromising solution quality in the scenarios explored. Clearly, the

standard method can be used, in circumstances where the estimation method is not sufficiently accurate. A number of questions currently remain unanswered:

- How repeatable are all the results?
- In the second experiment, why was a280's estimate better compared with standard and the worst (apart from bier127) compared with naïve?
- How quickly (in terms of both moves performed and time) do the methods converge on their 'final' results?
- Why was bier127 so much worse for the estimate than the other instances?—Over twice as many non-TP results for all the greedy results in the first experiment and more than 4.5% worse results in the second experiment, compared with just over 1% worse for the next worst.

6. Conclusions and future work

The long-term goal of this work is to find rules of thumb that can be applied when working with heuristic and metaheuristic methods on VRPs in a time-varying environment, so that quick estimates can be used to focus on potentially good solutions without the need to make full computations. Ideally, this should involve self-adaptation, so that the degree of reliance on estimates is automatically adjusted to dynamically maintain an equitable balance between computation time and solution quality. The experiments carried out for the present study are aimed at investigating how plausible this may be.

The results of this initial study are very encouraging, suggesting that considerable savings in run time can be made using the estimation method, with little loss of solution quality (savings of 30–80% calculation time with only 1% loss of quality for the majority of problem instances, even at its worst we saw 4.5% loss of quality and a 45% saving in time). Nevertheless, we have to be cautious regarding these findings, given this study has been quite narrow in its scope: looking at a simplified version of congestion modelling and a single vehicle problem. Future work will include expanding this investigation into the field of multiple vehicles, and looking at real-world data. Clearly, multiple vehicle instances on the same problem size will contain fewer nodes in the individual tours, meaning less calculations for the standard method, so the focus will be more on the smaller problems that we have investigated here.

From this preliminary work, it would appear that the nature of the problem instance is perhaps the most relevant aspect that influences how useful estimates can be. Both sets of experiments showed that the clustered bier127 led to worse performance for the estimation tool than the structured a280. Thus, it would seem prudent to carry out our further investigations based on different benchmark TSPs, looking at how various features of the instance (such as clustering of customers) affect the accuracy of the estimation tool. Further experiments will perhaps be limited to investigating only *greedy* solutions, as these are of most relevance to real-world optimisation.

Another aspect that requires further work is the investigation of different neighbourhood moves, other than 2-opt (eg, delete and insert). With our preliminary results on the effects of 2-opt, the other neighbourhood moves can now be compared to see how effective our estimates can be. With the expansion into multiple vehicle VRPs, other heuristic methods will need to be investigated, such as the merging of routes, and these are quite different to the neighbourhood moves typical of TSPs.

Future plans also include the assessment of our estimation methods within more sophisticated metaheuristic frameworks, such as simulated annealing. We also plan to investigate changing the 'acceptance criteria' for assessing results. Instead of simply investigating results that appear better than the current tour, we can change the parameters so that the estimation tool investigates tours that are apparently slightly poorer. It is likely that fewer improvements will be erroneously discarded, although this will be at the expense of longer run times.

The results of our second set of experiments clearly indicate that our estimate method is faster than the standard method that we have used here. The loss of solution quality in most cases seems to be acceptably small. Further experiments will need to be performed to establish whether similar savings can be made if more complex metaheuristics are used.

References

Dijkstra EW (1959). A note on two problems in connexion with graphs. *Numerische Mathematik* **1**(1): 269–271.

Dreyfus SE (1969). An appraisal of some shortest-path algorithms. *Operations Research* **17**(3): 395–412.

Eglese R, Maden W and Slater A (2006). A road timetable™ to aid vehicle routing and scheduling. *Computers & Operations Research* **33**(12): 3508–3519.

Fisher ML, Greenfield AJ, Jaikuman R and Lester JT (1982). A computerized vehicle routing application. *Interfaces* **12**(4): 42–52.

Hill A, Mabert V and Montgomory D (1988). A decision support system for the courier vehicle scheduling problem. *Omega International Journal of Management Science* **16**(4): 333–345.

Horn MET (2000). Efficient modeling of travel in networks with time-varying link speeds. *Networks* **36**(2): 80–90.

Ichoua S, Gendreau M and Potvin JY (2003). Vehicle dispatching with time-dependent travel times. *European Journal of Operations Research* **144**(2): 379–396.

Kaufman DE and Smith RL (1993). Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *Journal of Intelligent Transportation Systems* **1**(1): 1–11.

Kok AL, Hans EW and Schutten JMJ (2009). *Vehicle routing under time dependent travel times: The impact of congestion avoidance*. Internal Report, University of Twente.

Malandraki C and Dial RB (1996). A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operations Research* **90**(1): 45–55.

Sung K, Bell MGH, Seong M and Park S (2000). Shortest paths in a network with time-dependent flow speeds. *European Journal of Operations Research* **121**(1): 32–39.

Received February 2011;
accepted January 2012 after one revision