Insertion Heuristics for Central Cycle Problems

John D. Lamb
University of Aberdeen Business School, University of Aberdeen, AB24 3QY, UK

A central cycle problem requires a cycle that is reasonably short and keeps a the maximum distance from any node not on the cycle to its nearest node on the cycle reasonably low. The objective may be to minimise maximum distance or cycle length and the solution may have further constraints. Most classes of central cycle problems are NP-hard. This paper investigates insertion heuristics for central cycle problems, drawing on insertion heuristics for p-centres [7] and travelling salesman tours [21]. It shows that a modified farthest insertion heuristic has reasonable worst-case bounds for a particular class of problem. It then compares the performance of two farthest insertion heuristics against each other and against bounds (where available) obtained by integer programming on a range of problems from TSPLIB [20]. It shows that a simple farthest insertion heuristic is fast, performs well in practice and so is likely to be useful for a general problems or as the basis for more complex heuristics for specific problems.

keywords: tour, cycle, centre, eccentricity, cycle-length

1. INTRODUCTION

In a central location problem we wish to locate a facility or service on a network so that the maximum distance to it from any node is reasonably low. The facility or service might be a fixed point (represented by a single node) or might be an extensive facility [16] represented by a subgraph with some specified properties. One interesting class of problems are those where the subgraph is a circuit or subtour. Applications include the design of mobile services such as libraries, post offices and fishmongers, the design of bus or train routes and touring problems for theatre productions. Such problems may contain constraints on the length of the circuit or subtour, the number of nodes covered by it or the maximum or average distance to it from other nodes. For example, a mobile service typically has limits on the number of stops (nodes) and the total distance travelled (tour length). Additionally the distance travelled to get to the service should be kept reasonably small. Often we can express problems like these as optimisation problems and there are good integer programming formulations when the subgraph is a circuit and the objective is to minimise either the average distance to the subgraph (cycle median problem [13]) or the maximum distance to the subgraph (cycle centre problem [8]). Problems like these are generally **NP**-hard and so we want a simple method that can be applied to a range of problems finding a solution that meets the given constraints while giving good performance on any variables that we might wish to be small.

This paper restricts its attention to problems where we seek a central (as opposed to median) circuit or subtour, and looks for simple fast heuristics that should perform well on a range of problems. In particular, it looks at insertion heuristics in which the circuit or subtour is built up by inserting one node at a time. The motivation is that such heuristics are simple, fast and have been studied for two problems that can be thought of as special cases of the problems we consider. The first of these is the travelling salesman problem. This is a special case in which we require the maximum distance from a node not on the cycle to be zero. Rosenkrantz *et. al.* [21] show performance bounds for several insertion heuristics. The second problem is the p-centre problem. This can be thought of as a special case with unbounded cycle length. Dyer and Frieze [7] find performance bounds for a farthest-neighbour insertion heuristic.

The next section defines various terms. Section 3 reviews the literature on the problems we discuss. Then Section 4 derives some worst-case performance bounds for the heuristics and Section 5 shows how the heuristics perform on a selection of problems derived from TSPLIB [20].

2. DEFINITIONS

Before we can discuss the problems investigated we need some more precise terminology.

We consider a network represented by a graph G with nonnegative edge weights given by $w : E(G) \to \mathbb{R}$. For nodes u and v we define the *distance* $d(u, v)$ to be the length (sum of edge weights) of the shortest path from u to v. More generally, for a subgraph H and node u we define $d(u, H) = \min\{d(u, v) : v \in V(H)\}$. The *eccentricity* $e(H)$ of a subgraph H of G is

$$e(H) = \max\{d(u, H) : u \in V(G)\} \qquad (1)$$

In particular, the eccentricity of a node v is the distance to a node farthest from v.

Given a collection $\mathcal{H}$ of subgraphs of G, an $\mathcal{H}$-*centre* is a subgraph $H \in \mathcal{H}$ of minimum eccentricity. If $\mathcal{H}$ is the set of nodes of G, the $\mathcal{H}$-centre is called a 1-*centre* or just *centre*. If $\mathcal{H} = \{P \subseteq V(G) : |P| = p\}$, the $\mathcal{H}$-centre is called a p-*centre*.

We focus on circuits and subtours. A *cycle* is a graph in which every node has even degree. We define a *subtour* to be a connected cycle and use the term *circuit* to mean a subtour in which every node has degree at most 2. Thus we consider a single node or two nodes joined by parallel edges to be circuits. The length $l(S)$ of a cycle is the sum of its edge weights. Circuits are often called cycles and a *cycle centre* [8] is an $\mathcal{H}$-centre where $\mathcal{H}$ is the collection of circuits of G of length not more than some given bound. Similarly we define the *subtour centre* to be an $\mathcal{H}$-centre where $\mathcal{H}$ is the collection of subtours of G of length not more than some given bound. We use the term subtour rather than connected cycle because it is shorter and emphasises the relationship to the travelling salesman problem.

Centres are closely related to medians. Given a collection $\mathcal{H}$ of subgraphs of G, an $\mathcal{H}$-*median* is a subgraph $H \in \mathcal{H}$ that minimises

$$\sum_{u \in V(G) \setminus V(H)} d(u, H).$$

We assume that distances obey the triangle inequality. That is, for $x, y, z \in V(G)$, $d(x, z) \leqslant d(x, y) + d(y, z)$. We note (following [21]) that if they do not, we can add a sufficiently large constant C to all edge weights to ensure that they do. Then we can modify the problem to one of finding, for example, a subtour centre where the length of the subtour T is bounded by $L - C|E(T)|$ where L is the original bound on the subtour length.

Consider a circuit C. We can write it as a sequence of nodes $x_1, \ldots, x_n$ in the order in which they are encountered. This representation is not unique: we can choose a different starting node or reverse the order of the nodes. However, in the following definitions we assume we have chosen a representation $x_1, \ldots, x_n$ that is unique up to the choice of starting node x_1. We say that y *follows* x in the circuit if we can write it as $y, \ldots, x$. Note that x and y need not be distinct. In the particular case where C contains exactly one node x we have x follows x.

If C is a circuit in which x follows y we can write $C = x, \ldots, y$ and we say we *insert* a node v *between* y *and* x when we create a new circuit $x, \ldots, y, v$. If $C = x, \ldots, p, q, \ldots, y$ is a circuit, we say we *split* C *on* y, x *and* p, q when we create the circuits $x, \ldots, p$ and $q, \ldots, y$. If C and D are circuits that we can write as $C = x, \ldots, p$ and $D = \{q, \ldots, y\}$ we say we *join* C *and* D *on* x, y *and* p, q when we create the circuit $x, \ldots, p, q, \ldots, y$. Note that $x, \ldots, p$ may be identical to x and $q, \ldots, y$ may be identical to y so that, for example, we split x, q, y on x, y and x, q to get x and q, y.

Given a tree T on a subset of the nodes of G, and an edge xy of G joining x and y we call the unique path in T from x to y the *corresponding* path of T. Given a subgraph H of G and an edge xy we denote by $H \cup xy$ the subgraph $(V(H) \cup \{x, y\}, E(H) \cup \{xy\})$ and if $x, y \in V(H)$ we denote by $H \setminus xy$ the subgraph $(V(H), E(H) \setminus \{xy\})$.

We consider problems where there are two variables of interest: eccentricity and cycle length. Thus we are dealing with bi-criteria problems [8]. We focus mainly on subtour centres, but note that the results are relevant to finding cycle centres, to finding minimum-length subtours subject to a bound on eccentricity, to finding subtours satisfying bounds on both criteria or to finding subtours where decreasing one variable is only possible by increasing the other (Pareto efficient solutions).

3. BACKGROUND

There is now over 40 years of research on central structures in networks. Early papers such as [9] concentrate on central nodes and find polynomial algorithms. Perhaps the simplest way to find the centre is to construct a distance matrix and find the row with the least maximum entry. Later papers started to look at extensive central structures such as p-centres [6, 7, 2, 10, 17], paths [24], circuits [8, 13] and tours [5]. Mesa and Boffey [16] provide a survey.

Current and Schilling [5] consider median tours and formulate integer programming and heuristic solution methods but do not consider subtour or cycle centres. Labbé *et. al.* [13] consider a similar problem of locating median cycles on a network and formulate integer programming solution methods, notably testing their methods on instances from TSPLIB

[20]. Foulds *et. al.* [8] consider the problem of finding cycle centres, cycle medians and a third problem, which they call the cycle centroid, on unweighted graphs. They formulate integer programmes for these problems and test a number of problem instances. They also report having tested tabu search methods on the cycle centre problem. I use the integer programming formulations of [8] and [13] to generate integer programmes that generate optimum or feasible solutions for some of the test problems in Section 5.

The p-centre problem is known to be **NP**-complete [12] and several papers describe heuristic approaches to its solution. Dyer and Frieze [7] is of particular interest because it describes a simple farthest-neighbour insertion heuristic. They describe a problem of locating p central points from n on a metric space and show that farthest-neighbour insertion (defined in Section 4) finds a solution whose eccentricity is, at worst, twice the eccentricity of the best solution. Their algorithm runs in time $O(np)$. We note that a set of points on a metric space defines a complete graph with nodes corresponding to the points and distances satisfying the triangle inequality. Drezner [6] also describes a simple heuristic approach for finding p-centres. It is an improvement heuristic rather than a construction heuristic and no bounds are given for its performance. Several more recent papers describe either more complex heuristics [2, 10, 17] or heuristics for more constrained problems [11, 19, 22, 15, 18].

The travelling salesman problem (or TSP) [14] is a well-known problem that is a special case of the cycle centre or subtour centre problem in which either the eccentricity of a solution is required to be zero or the cycle length is large enough to allow a tour covering all nodes and we seek a solution with cycle-length as small as possible. The TSP is well known to be **NP**-hard. Rosenkrantz *et. al.* [21] discuss insertion heuristics applied to the TSP on a network with n nodes and distances satisfying the triangle inequality. They show that a nearest or cheapest insertion heuristic (defined in Section 4) produces a solution that has cycle length at most twice the minimum tour length. They also show that a farthest insertion heuristic produces a solution whose length is bounded by $\lceil \log_2 n \rceil + 1$ times the minimum tour length and suggest the true bound may be much smaller, perhaps four times the minimum. They note that nearest insertion can be made to run in time $O(n^2)$ and cheapest insertion in time $O(n^2 \log n)$. More recently, Cook *et. al.* [4] note that no example is known for which farthest insertion produces a solution more than four times the minimum and report that Johnson, Bentley, McGeoch and Rothberg find farthest insertion gives a solution with an average of 1.16 times the minimum when tested on the TSPLIB problems [20].

There are many other TSP heuristics that may be of value for subtour centre or cycle centre problems. One that is of particular note is Christofides heuristic [3], which is reasonably simple, produces solutions with cycle length at most 1.5 times the minimum and runs in polynomial time.

4. BOUNDS ON INSERTION HEURISTICS

The purpose of this section is to show that the methods we test in the next are likely to be reasonable.

We do this by showing that one farthest insertion heuristic produces a subtour S whose length is at most twice the minimum length tour on $V(S)$ and whose eccentricity is at most twice the minimum eccentricity among all subtours with at most $|V(S)|$ nodes.

An *insertion heuristic* is a heuristic that constructs a sequence of subtours $\{S_1, \ldots, S_p\}$ such that S_1 contains a single node and for $i = 2, \ldots, p$, $|V(S_i)| = |V(S_{i-1})|$. Rosenkrantz *et. al.* [21] describe three insertion heuristics for the TSP. Note that all three assume G is complete so that each S_i is a circuit, but we can modify them, replacing edges with shortest paths, so that they work on any network.

In *nearest insertion*, we construct S_i from S_{i-1} as follows. First, choose a node $v \in V(G) \setminus V(S_{i-1})$ *minimising* $d(v, S_{i-1})$. Then choose nodes x and y such that y follows x in S_{k-1} and $d(v,x) + d(v,y) - d(x,y)$ is as small as possible. Then insert v between x and y.

In *farthest insertion*, we construct S_i from S_{i-1} as follows. First, choose a node $v \in V(G) \setminus V(S_{i-1})$ *maximising* $d(v, S_{i-1})$. Then choose nodes x and y such that y follows x in S_{k-1} and $d(v,x) + d(v,y) - d(x,y)$ is as small as possible. Then insert v between x and y.

In *cheapest insertion*, we construct S_i from S_{i-1} as follows. Choose nodes $v \in V(G) \setminus V(S_{i-1})$ and x and y such that y follows x in S_{k-1} such that $d(v,x) + d(v,y) - d(x,y)$ is as small as possible. Then insert v between x and y.

Although Rosenkrantz *et. al.* obtain better bounds for the nearest and cheapest insertion heuristics, we are interested in farthest insertion for two reasons. First, as noted implicitly by Dyer and Frieze [7], the sequence of improvements in eccentricity is nonincreasing. More precisely, if $i > j > 1$ then $e(S_i) - e(S_{i+1}) \leqslant e(S_j) - e(S_{j+1})$. This means that in problems where eventually we are constrained by subtour length not to insert a farthest neighbour, the heuristic is still likely to be useful because we can be

confident that the first few steps will contribute most to improving eccentricity. Second, although they do not consider subtours, Dyer and Frieze show that a heuristic that starts with a single node and chooses at each step a node at farthest distance from the existing solution creates a set of nodes V such that $e(V)$ is at most twice the eccentricity of a $|V|$-centre. Thus we have the following result.

Theorem 1 *Let S be a subtour constructed using a farthest insertion heuristic starting from the centre of G. Then $e(S) \leqslant 2e(S^*)$ and has length at most $(\lceil \log_2 n \rceil + 1)l(C^*)$ where S^* is the least eccentric subtour on $|V(S)|$ nodes and C^* is the shortest subtour on $V(S)$.*

It is easy to check that, like nearest insertion, farthest insertion can be implemented in time $O(n^2)$.

We now describe a modified insertion heuristic and show that when used with farthest insertion we can reduce the bound on subtour length to that of nearest insertion. We note that this heuristic can be used for the TSP and may provide a practical way of carrying out a single insertion step when other methods create too large an increase in subtour length.

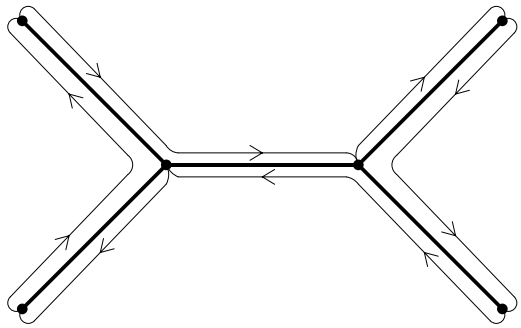

FIG. 1. A tree and circuit.

The idea behind the modified insertion heuristic is that if we can construct a circuit whose edges correspond to paths on a tree, like the circuit and tree of FIG. 1, so that as we trace the circuit we cross each edge of the tree exactly twice then, by the triangle inequality, the circuit length will be at most twice the sum of the weights of the tree edges. The heuristic constructs a minimum-weight spanning tree on the selected nodes step by step, maintaining just such a circuit.

The *modified insertion heuristic* is the heuristic described in FIG. 2. Modified insertion heuristics vary in how the node v inserted at each stage is chosen and in the criterion used to control when the heuristic is finished. As before we assume G is complete and note that we can modify the heuristic, replacing edges with shortest paths, so that it works on any network.

Choose an initial node v_0.
Let $C_1 = v_0$ be the initial circuit.
Let $T_1 = (\{v_0\}, \emptyset)$ be an initial tree.
Repeat until either $|V(T_k)| = |V(G)|$ or some other stopping criterion is met:
  Choose a node v to insert into C_k.
  If $k = 1$:
    Let $T_2 = T_1 \cup v_0 v$.
  else:
    Let x and y be nodes of C_k such that $w(vx) \leqslant w(vy) \leqslant w(vz)$ for all z in C_k.
    If $w(vy) \geqslant w(e)$ for each edge e in the path of T_k joining x to y:
      If $w(vy) \geqslant w(e)$ for each edge e in the path of T_k joining x to y:
        Let $T_{k+1} = T_k \cup xv$.
        Let p be a neighbour of x in C_k.
        Insert v between p and x in C_k to get C_{k+1}.
      else:
        Let e be the edge of largest weight in the path of T_k joining x to y.
        Let $T_{k+1} = (T_k \cup vx \cup vy) \setminus e$.
        Let T_x and T_y denote the components of $T_k \setminus e$ containing x and y.
        Choose $x_1, x_2 \in V(T_x)$ and $y_1, y_2 \in V(T_y)$ so that x_1 follows y_1 and y_2 follows x_2 in C_k.
        Split C_k on y_1, x_1 and x_2, y_2 to get C_x and C_y.
        Let p follow x in C_x and let q follow y in C_y.
        Join C_x and C_y on y, x and p, q to get C.
        Insert v between x and y in C to get C_{k+1}.
      End if.
    End if.
  End if.
  Let $k = k + 1$.
End repeat loop.

FIG. 2. Pseudocode for the modified insertion heuristic.

Lemma 1 *Let T be a tree and label its nodes $v_1, \ldots, v_k$. Let P_i denote the path from v_i to v_{i+1} $(i = 1, \ldots, k-1)$, let P_k denote the path from v_k to v_1 and put $\mathcal{P} = \{P_1, \ldots, P_k\}$. Suppose each edge of T is contained in precisely two paths of $\mathcal{P}$. Then any two paths in $\mathcal{P}$ have at most two nodes in common.*

Proof. First note that for any partition of $V(T)$ into nonempty sets Y and V $\mathcal{P}$ must contain at least two paths joining a node of Y to a node of V.

Suppose $P, Q \in \mathcal{P}$ have three nodes in common and choose common nodes x, y and z encountered in order as we trace P and such that we encounter no other node in Q as we trace P from x to z. Since T is a tree the path from x to z is unique and so must

be a subpath of Q. Moreover, x and y must be adjacent since any node between them in P would also be a node of both P and Q. Similarly y and z must be adjacent. Thus P and Q contain both xy and yz twice.

Now let X be the set of nodes of the component of $T \setminus xy$ containing x, let Z be the set of nodes of the component of $T \setminus yz$ containing Z, and let $Z = V(T) \setminus (X \cup Z)$. Then $x \in X$, $y \in Y$ and $z \in Z$, and so Z and $V(T) \setminus Z$ are nonempty. No $R \in \mathcal{P}$ can join a node of X to a node of Y. For, otherwise it would contain xy. Similarly, no $R \in \mathcal{P}$ can join a node of Z to a node of Y. Hence $\mathcal{P}$ contains no path from Z to $V(T) \setminus Z$—a contradiction.

It follows that any two paths in $\mathcal{P}$ have at most two nodes in common. ∎

Define the *weight* $w(H)$ of a subgraph H of G to be the sum of the weights of its edges. Rosenkrantz *et. al.* [21] show that minimum-weight spanning tree on a complete graph H on k nodes whose edge weights obey the triangle inequality has weight at most $2(1-1/k)w(C_k^*)$ where C_k^* is a a minimum-length circuit on H.

Theorem 2 *At each stage of a modified insertion heuristic, we can select and replace neighbours of C as described. Moreover, at each step C is a circuit and $w(C) \leqslant 2(1-1/k)w(C_k^*)$ where k is the number of nodes inserted and C_k^* is a minimum length circuit on those nodes.*

Proof. Let H_k be the complete graph on $V(T_p)$.

We show first that T_k is a minimum-weight spanning tree on H_k. It is easy to see that this must be true for $k = 1$ and $k = 2$. Suppose it is not true in general and choose a counterexample T_k with k minimal. Then for some $e \in E(T_k)$ and $f \in E(H_k) \setminus E(T_k)$, $w((T_k \cup f) \setminus e) < w(T_k)$ and so $w(f) < w(e)$. Let v denote the last node inserted by the heuristic and let x and y be its nearest neighbours on T_{k-1} with $w(vx) \leqslant w(vy)$. If $e = vx$ then $T_k \setminus e = T_{k-1}$ and so $T_{k-1} \cup f$ is a tree. But this is only possible if $f = vy$ contradicting the minimality of $w(vx)$. If $e = vy$ then f joins two nodes of T_{k-1}, and $(T_k \cup f) \setminus e$ contains $T_{k-1} \cup f$ as a subgraph and so must contain a circuit. If e is not incident with v then either (i)

$$\begin{aligned} w((T_{k-1} \cup f) \setminus e) &= w((T_k \cup f) \setminus e) - w(vx) \\ &= w(T_k) - w(vx) \\ &= w(T_{k-1}) \end{aligned}$$

or (ii)

$$\begin{aligned} w((T_{k-1} \cup f) \setminus e) &= w((T_k \cup f) \setminus e) \\ &\quad -w(vx) - w(vy) + w(rs) \\ &< w(T_k) - w(vx) \\ &\quad -w(vy) + w(rs) \\ &= w(T_{k-1}), \end{aligned}$$

in either case contradicting the minimality of k. It follows that at each step T_k is a minimum-weight spanning tree on H_k.

It is straightforward to check that each iteration of the loop adds one new node to C_k and so C_k comprises k distinct nodes and must be a circuit.

We now show that, for any k, if we write $C_k = c_1^k, \ldots, c_k^k$ and put $\mathcal{P}_k = \{P_1^k, \ldots, P_k^k\}$ where for $i = 1, \ldots, k-1$ P_i^k is the path of T_k joining c_i^k to c_{i+1}^k and P_k^k is the path of T_k joining c_k^k to c_1^k, then each edge of T_k is contained in precisely two paths of $\mathcal{P}_k$. The result is trivially true for $k = 1$ and for $k = 2$ we have $C_2 = c_1^2, c_2^2$, $P_1^2 = c_1^2, c_2^2$, $P_2^2 = c_2^2, c_1^2$ and $c_1^2 c_2^2$ is the only edge of T_2. It follows easily that the result holds for $k = 2$.

Suppose the result above does not hold in general. Choose a counterexample C_k with k minimal. Then the result holds for C_{k-1}.

We consider two cases. First, suppose $T_k = T_{k-1} \cup xv$. Then we may suppose without loss of generality that the neighbour of x in C_{k-1}, $p = c_1^{k-1}$ so that $C_k = c_1^{k-1}, \ldots, c_{k-1}^{k-1}, v$. Then $P_i^k = P_i^{k-1}$ for $i = 1, \ldots, k-2$, P_{k-1}^k is P_{k-1}^{k-1} followed by v, and $P_k^k = \{v, x\}$. T_k has one new edge xv, which is contained in precisely two paths, P_k^{k-1} and P_k^k. Each remaining edge of T_k is contained in P_i^k ($i = 1, \ldots, k-1$) whenever it is contained in P_i^{k-1}. It follows that each edge of T_k is contained in precisely two paths of $\mathcal{P}_k$ and so the result holds for C_k.

Suppose instead that $T_k = (T_{k-1} \cup vx \cup vy) \setminus e$. Construct T_x, T_y, C_x and C_y and choose x_1, x_2, y_1, y_2, p and q as the algorithm does. Let $\hat{x}$ and $\hat{y}$ be the nodes of e in T_x and T_y. Choose i and j so that P_i^{k-1} and P_j^{k-1} are the paths of $\mathcal{P}_{k-1}$ containing e. Let $I = \{1, \ldots, k\} \setminus \{i, j\}$.

Let R_i be the path in T_x from x_1 to x_2, let R_j be the path in T_y from y_1 to y_2, and let $R_t = P_t^{k-1}$ for $t \in I$. Then, since P_i^{k-1} and P_j^{k-1} contain two nodes, $\hat{x}$ and $\hat{y}$ in common and, by Lemma 1, have at most two nodes in common, R_i follows P_i^{k-1} from x_1 to $\hat{x}$ and then follows P_j^{k-1} to x_2. Similarly, R_j follows P_i^{k-1} from y_1 to $\hat{y}$ and then follows P_j^{k-1} to y_2. Thus R_i and R_j contain the same edges with the same multiplicity as P_i^{k-1} and P_j^{k-1} except that they do not contain e. It follows that $\mathcal{R} = \{R_1, \ldots, R_{k-1}\}$ contains each edge of T_x and T_y precisely twice.

Choose a and b so that R_a is the path from x to p and R_b is the path from y to q. Let $J = \{1, \ldots, k\} \setminus \{a, b\}$. Let $P_a = x, v$, let $P_b = v, y$, let P_k be the path from x to y composed of R_a followed by v followed by R_b. And let $P_t = R_t$ for $t \in J$. Then $\mathcal{P} = \{P_1, \ldots, P_k\}$ contains each edge of T_x and T_y precisely twice and also contains xv and vy precisely twice.

It is easy to check that we have constructed $\mathcal{P}$ so

that if t follows s in C_k then $\mathcal{P}$ contains a path from s to t. Hence $\mathcal{P} = \mathcal{P}_k$ and each edge of T_k is contained in precisely two paths of $\mathcal{P}_k$ and so the result holds for C_k.

We have shown that the result holds in either case, contradicting the minimality of k. It follows that, for any k, if we write $C_k = c_1^k, \ldots, c_k^k$ and put $\mathcal{P}_k = \{P_1^k, \ldots, P_k^k\}$ where, for $i = 1, \ldots, k-1$, P_i^k is the path of T_k joining c_i^k and c_{i+1}^k, and P_k^k is the path of T_k joining c_k^k and c_1^k, then each edge of T_k is contained in precisely two paths of $\mathcal{P}_k$.

By applying the triangle inequality repeatedly, we have that $w(xy) \leqslant w(P)$ where P is the path in a spanning tree T corresponding to xy. Applying this for each pair of neighbours of C_k for any k we have

$$w(C_k) \leqslant \sum_{i=1}^{k} w(P_k) = 2w(T_k) \leqslant 2\left(1 - \frac{1}{n}\right) w(C_k^*)$$

as required. ∎

We now consider bounds on the time and space required to carry out a single iteration of a modified insertion heuristic. For now we will ignore the cost of finding a suitable v to insert and of checking stopping conditions other than $|V(T_k)| = |V(G)|$, which requires at most $O(1)$ arithmetic operations.

We start by looking at the requirements for manipulating the circuits. We can record a circuit as an array indexed on $1, \ldots, k$ whose entries represent the nodes in the order in which they are added to the circuit. For each node the array stores the predecessor in the circuit, the successor in the circuit, a second index indicating which circuit the node belongs to (so that we can use one array to store two circuits after splitting) and a third index identifying which subtree the node belongs to (so that we can easily identify T_x and T_y). The benefit of an array is that we can use binary search to find any particular node, requiring at most $O(\log_2 k)$ operations. To insert a node we need at most $O(\log_2 k)$ operations to find x and y and a further $O(1)$ to modify the circuit. To split a circuit we need $O(\log_2 k)$ operations to find (within the array) the nodes we on which we split and $O(k)$ to label the resulting circuits using the second index. Similarly, to join two circuits we need $O(\log_2 k)$ operations to find the nodes on which we join and $O(k)$ operations for relabelling. We need $O(1)$ operations to find the node following or the node that a given node follows.

To create T_{k+1} from T_k we need $O(1)$ operations, whichever way we do it. We need $O(k)$ operations to identify x and y. We can find e (or show that $w(vy) > w(e)$ for each edge in the unique path in T_k from x to y) using a depth-first search algorithm, which requires $O(k)$ operations and $O(k)$ space.

We have shown that in each step in the iteration to find T_{k+1} and C_{k+1} we need at most $O(k)$ operations. It follows that, apart from choosing v and testing a stopping condition, a modified insertion heuristic time requirement is $O(k)$ for the kth iteration. The space requirement is also $O(k)$ because we need $O(k)$ space to store the tree, the circuit array and for depth-first search and no other step requires more than $O(1)$ storage.

Now consider farthest insertion. Following Dyer and Frieze [7], we assume we can find distances from a distance matrix and store the maximum distance $D(v)$ from node v to C_k as an array, which we can update in time $O(n - k)$ at each iteration. Since $e(C_k) = \max_{v \in V(G)} D(v)$ and we can find $l(C_k)$ in time $O(k)$, the time needed to check any simple stopping criterion grows no faster than the time needed for an iteration and so farthest modified insertion needs at most $O(np)$ time, where the network has n nodes and the final circuit p. It is easy to check that the original farthest insertion heuristic also runs in time proportional to np, albeit the constant of proportionality is much smaller for the original heuristic. Thus we can expect the two heuristics to be practical for about the same range of problems.

5. COMPUTATIONAL RESULTS

We now look at how the farthest insertion heuristics perform in practice. We consider two problems. The first is to find a 'good' subtour that has p vertices. Here we wish to find a subtour that has, at the same time, reasonably small eccentricity and reasonably short length. Since this problem is not well defined we consider also a second problem, to find a subtour with eccentricity as small as possible given an upper bound on the subtour length. A third problem would be to find as short a subtour as possible given an upper bound on the eccentricity. We do not test the heuristics on this problem but note that the heuristic of Dyer and Frieze [7] to identify a set of nodes followed by Christofides [3] heuristic to find a subtour on those nodes ought to perform better (see [4]) while retaining much of the simplicity of our heuristics.

We compare the different methods using problems derived from the standard TSPLIB [20] problems. We test each heuristic on each of the 29 TSPLIB problems with 101 or fewer nodes using three instances of problem bounds for each problem.

The heuristics were coded in C++ using the Boost Graph Library [1, 23] to represent the networks, compiled with the GNU g++ compiler using identical optimisation flags and run on a PC with a 3.0 GHz Pentium 4 processor and 1 Gb RAM using the SuSE

Linux 10.1 operating system.

We compare first the performance of the original and modified insertion heuristics on problems of finding a subtour on a subset of p vertices. We test both heuristics on three instances of each of the problems. The first instance has an objective of finding a subtour containing all the nodes of the problem and so is equivalent to the TSP. The second and third instances seek subtours with number of nodes p equal to the nearest integers to $n/2$ and $n/3$. For each instance we test both heuristics starting each from every possible node and recording the best result found. Table 1. summarises the results of the tests. Since the test results are very similar for similar sized problems, the table reports results only 15 TSPLIB problems covering the range of problem sizes.

The first column gives the problem name, which includes the total number of nodes in the network. More details can be found in TSPLIB [20]. The second column gives the number of nodes p in the final subtour. The ECC column gives the eccentricity of the solution found. Since both heuristics choose nodes in the same way they find solutions with the same eccentricity. The UTIM and MTIM columns report the approximate time in CPU seconds taken by the unmodified and modified heuristics. This is the total time for all tests on the given problem instance; so, for example, The MTIM of 212 CPU seconds for problem kroA100, represents an average of 2.12 CPU seconds for each of 100 possible starting nodes. The ULEN and MLEN columns report the subtour lengths for the best subtour found by the unmodified and modified insertion heuristics. In one instance (fri26, in boldface) ULEN was equal to the TSP minimum tour length.

It is notable that the unmodified heuristic is not only much faster, but in most cases actually finds a better solution than the modified heuristic. I have italicised the MLEN entries where modified farthest insertion works at least as well as unmodified farthest insertion. The modified heuristic only performs better on two instances of Ulysses16 and performs less well on the 42 instances tested but not listed in the table.

The reported times for the modified insertion heuristic range from twice to several thousand times as long for each starting node as the unmodified heuristic. However, we note that the CPU time measurements are only approximate and in some small problem instances the reported CPU time was zero, so that a comparison was not possible. The results for the six problems with 100 nodes suggest that unmodified insertion should typically be several hundred times as fast as modified insertion.

TABLE 1. Comparison on problems requiring a subtour on p nodes.

Problem	p	**ECC**	**UTIM**	**MTIM**	**ULEN**	**MLEN**
burma14	14	0	0.00	0.02	3381	4183
burma14	7	168	0.00	0.00	3455	3513
burma14	4	310	0.00	0.00	2752	*2752*
ulysses16	16	0	0.00	0.04	7726	*7705*
ulysses16	8	261	0.00	0.01	7224	*6437*
ulysses16	5	455	0.00	0.00	6010	*6010*
gr21	21	0	0.00	0.13	2753	3455
gr21	10	125	0.00	0.02	2395	3237
gr21	7	160	0.00	0.00	2047	*2047*
gr24	24	0.00	0.00	0.13	1319	1678
gr24	12	49	0.00	0.07	1198	1212
gr24	8	70	0.00	0.02	925	1156
fri26	26	0	0.00	0.17	**937**	1191
fri26	13	29	0.00	0.02	813	974
fri26	8	51	0.00	0.00	755	927
bayg29	29	0	0.00	0.96	1695	2092
bayg29	14	52	0.00	0.05	1319	1469
bayg29	9	71	0.00	0.01	1126	1318
dantzig42	42	0.00	0.01	3.46	777	968
dantzig42	21	15	0.02	2.36	779	850
dantzig42	14	22	0.02	1.06	621	646
att48	48	0	0.05	1.61	11304	14637
att48	24	183	0.02	1.59	10655	12711
att48	16	301	0.01	0.10	9326	10143
berlin52	52	0	0.04	13.8	8639	10535
berlin52	26	125	0.02	0.60	7887	9618
berlin52	17	188	0.03	1.92	7572	8960
brazil58	58	0	0.03	5.48	26803	31384
brazil58	29	287	0.05	6.27	25419	26561
brazil58	19	517	0.01	0.87	23682	25226
st70	70	0	0.12	37.0	763	999
st70	35	9	0.08	4.10	629	714
st70	23	13	0.06	0.99	538	677
eil76	76	0	0.05	59.9	604	805
eil76	38	7	0.04	37.8	460	576
eil76	25	10	0.04	20.0	370	486
gr96	96	0	0.15	203	61234	80069
gr96	48	520	0.16	32.4	58556	73901
gr96	32	759	0.21	27.8	51101	63810
kroA100	100	0	0.40	212	23247	32390
kroA100	50	216	0.22	47.8	21011	27513
kroA100	33	311	0.03	33.8	17489	21416
eil101	101	0	0.08	379	725	984
eil101	50	7	0.05	75.7	548	742
eil101	33	9	0.22	25.3	446	568

The second set of tests are for subtour centre problems. In these the objective is to find a minimum eccentricity subtour of length no greater than some given bound. Again we use the 29 TSPLIB problems with 101 or fewer nodes to construct suitable test instances. For each TSPLIB problem we construct

three test instances. First, we use a bound on subtour length equal to the TSP optimum tour length. This gives a problem whose optimum solution is known to have eccentricity zero. Then we consider instances where the bound on subtour length is 1/2 or 1/3 of the TSP optimum tour length. We test each problem instance using both the unmodified and modified farthest insertion heuristics. For both heuristics, we insert at each iteration the farthest node whose insertion does not create a subtour whose length exceeds the given bound, stopping only when no more nodes can be inserted. This is a slight modification of the heuristics described earlier that is very easy to implement in practice, especially for the unmodified heuristic. It gives a greater time penalty to the modified heuristic because in contrast to the unmodified case, we have to construct a new subtour explicitly to find if its length exceeds the given bound. As before, we test each heuristic with each possible starting node.

We wish to compare the quality of solutions and do this in two ways. First we compare the eccentricity of each solution with the best known eccentricity for the test instance. When we use the TSP bound this is always zero. For the remaining problems I used a modified version of the integer programming formulations of [8] and [13] implemented in Ilog CPLEX 9.1 to look for optimum solutions, stopping each search after 8 hours CPU time if the best solution found had not been shown to be an optimum. A simple comparison with the best known solution is not very instructive, especially when the best solution has eccentricity zero. So we compare with another solution, the best 1-centre, to get two points of comparison.

Table 2. reports the results of the tests on the same range of problems as used for Table 1. The remaining problems show similar results and so are not presented. The *Bound* column reports the upper bound on the subtour length. UFIH and MFIH report the eccentricities of the best result obtained with the unmodified and modified farthest insertion heuristics. The *Best* column reports the best-known eccentricity, which may come from either of the heuristics or be determined by an integer programme. The final colum reports a performance gap recorded to the nearest percentage. It is calculated as $(b-b^*)/(c-b^*)$ where b is the eccentricity of the better of the solutions found by the heuristics, c is the eccentricity of the best 1-centre and b^* is the eccentricity of an optimum solution. Where no optimum solution is known we substitute the best solution found and write $\geqslant$ next to the reported value.

TABLE 2. Comparison on subtour centre problems.

		Eccentricity			
Problem	**Bound**	**Best**	**UFIH**	**MFIH**	**Gap %**
burma14	3323	0	**70**	247	11
burma14	1661	400	*400*	400	0
burma14	1107	406	406	*406*	0
ulysses16	6859	0	287	**237**	16
ulysses16	3429	1122	1122	1122	0
ulysses16	2286	1387	1387	*1387*	0
gr21	1707	225	250	*250*	9
gr21	853	420	420	420	0
gr21	569	495	*495*	495	0
gr24	1272	0	**31**	49	16
gr24	636	74	**82**	109	7
gr24	424	122	136	136	21
fri26	937	0	**9**	26	6
fri26	468	71	81	*81*	11
fri26	312	93	95	95	3
bayg29	1610	0	**36**	49	17
bayg29	805	74	121	**113**	29
bayg29	536	121	151	151	35
dantzig42	699	0	**12**	17	12
dantzig42	349	45	**45**	49	$\geqslant$ 0
dantzig42	233	52	59	59	$\geqslant$ 13
att48	10628	0	**139**	236	10
att48	5314	569	**569**	618	$\geqslant$ 0
att48	3542	739	**739**	807	$\geqslant$ 0
berlin52	7542	0	**127**	151	13
berlin52	3771	390	**402**	475	$\geqslant$ 2
berlin52	2514	426	**584**	606	$\geqslant$ 31
brazil58	25395	0	**209**	287	6
brazil58	12697	1446	1620	**1446**	$\geqslant$ 0
brazil58	8465	1991	*1991*	1991	$\geqslant$ 0
st70	675	0	**8**	11	11
st70	337	24	**24**	26	$\geqslant$ 0
st70	225	43	*43*	43	$\geqslant$ 0
eil76	538	0	**6**	8	14
eil76	269	13	**13**	15	$\geqslant$ 0
eil76	179	22	*22*	22	$\geqslant$ 0
gr96	55209	0	**480**	697	9
gr96	27604	1800	**1800**	2015	$\geqslant$ 0
gr96	18403	2805	**2805**	2875	$\geqslant$ 0
kroA100	21282	0	**179**	289	8
kroA100	10641	621	**635**	702	$\geqslant$ 1
kroA100	7094	1015	**1015**	1046	$\geqslant$ 0
eil101	629	0	**6**	7	13
eil101	314	12	**12**	14	$\geqslant$ 0
eil101	209	21	*21*	21	$\geqslant$ 0

Table 2. emphasises in boldface whether the unmodified or modified insertion heuristic performs better. In some cases both heuristics produces subtours with the same eccentricity. If there is a difference in length the eccentricity of the shorter sub-

tour is italicised. Table 2. does not report CPU times. These range from 0 to 0.002 (kroA100) seconds of CPU time per starting node for the unmodified insertion heuristics and from 0.0002 (ulysses16) to 21.0 (rat99, not shown in table) seconds for the modified insertion heuristic. The average CPU times for both heuristics typically increase with problem size but show substantial variation for problems with similar sizes. Typically the modified insertion heuristic takes several hundred times as long as the unmodified heuristic.

Both heuristics perform well in practice. The performance gap of 35% for bayg29 is the largest found in any of the test instances and the unreported instances show similar performance gaps. The unmodified insertion heuristic typically performs better in practice, finding a better solution than the modified heuristic in 53 of the 87 test instances. Modified insertion worked better in only 8 cases and in 26 cases both heuristics produced solutions with the same eccentricity.

6. DISCUSSION

We have considered insertion heuristics applied to central cycle problems. We chose to look at farthest insertion heuristics because earlier research [21, 7] suggests these might work well for problems where both the eccentricity and the length of a solution are of interest. The empirical results show that, in addition to being very fast, a simple farthest insertion heuristic performs reasonably well in practice. Although we can show better worst case bounds for a modified version of the heuristic, in practice it is slower and usually does not produce solutions as good as the unmodified heuristic.

Rosenkrantz *et. al.* [21] suggest that a simple farthest insertion heuristic performs well because it produces the general outline subtour in the first few iterations and makes only minor modifications in the later stages. We can observe that the modified insertion heuristic may make large modifications to the subtour at each iterations and these are not guaranteed to give a better subtour than that obtained by a simple insertion. Large modifications occur if, in Fig. 2, we have $w(vy) < w(e)$ for some edge e in the current spanning tree. Since we are using farthest insertion it is reasonable to expect this happens often and this is what I have observed in practice.

The main benefits of the unmodified farthest insertion heuristic are that it is a simple, fast, easily implemented method that can be adapted to a variety of problems where we are concerned with both eccentricity and cycle length. These include problems where we seek a solution that is a circuit, where G is not complete, where there are constraints on the number of nodes a solution may contain and where there are weights on the nodes. Either of the heuristics we have investigated could also be used as part of a broader heuristic for a central cycle problem.

REFERENCES

[1] Boost, http://www.boost.org/ Accessed July 2006

[2] C. Caruso, A. Colorni and L. Aloi, Dominant, an algorithm for the p-center problem, Eur J Oper Res 149 (2003), 53–64

[3] N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976

[4] W.J. Cook, W.H. Cunningham, W.R. Pulleybank and A. Schrijver, Combinatorial optimization, John Wiley and Sons, New York, 1998

[5] J.R. Current and D.A. Schilling, The median tour and maximal covering tour problems: formulations and heuristics, Eur J Operat Res 73 (1994), 114–126

[6] Z. Drezner, The p-centre problem—heuristic and optimal algorithms, J Opl Res Soc 35 (1984), 741–748

[7] M.E. Dyer and A.M. Frieze, A simple heuristic for the p-centre problem Oper Res Lett 3 (1985), 285–288

[8] L.R. Foulds, J.M. Wilson and T. Yamaguchi, Modelling and solving central cycle problems with integer programming, Compu and Operat Res 32 (2004), 1083–1095

[9] S.L. Hakimi, Optimum location of switching centers and the absolute centers and medians of a graph, Oper Res 12 (1964), 450–459

[10] R. Hassin, A. Levin and D. Morad, Lexicographic local search and the p-center problem, Eur J Oper Res 151 (2003), 265–279

[11] D.S. Hochbaum and A. Pathria, Generalized p-center problems: complexity results and approximation algorithms, Eur J Oper Res 100 (1997), 594–607

[12] W.L. Hsu and G.L. Nemhauser, Easy and hard bottleneck location problems, Discrete Appl Math 1 (1979), 209–216

[13] M. Labbé, G. Laporte, I. Rodríguez Martín and J.J. Salazar González, Locating median cycles in networks, Eur J Operat Res 160 (2005), 457–470

[14] E.L. Lawler, The traveling salesman problem, John Wiley and Sons Ltd, 1985

[15] A. Lim, B. Rodrigues, F. Wang and Z. Xu, k-center problems with minimum coverage, Theor Comput Sci 332 (2005), 1–17

[16] J.A. Mesa and T.B. Boffey, A review of extensive facility location in networks, Eur J Operat Res 95 (1996), 592–603

[17] N. Mladenović, M. Labbé and P. Hansen, Solving the p-center problem with tabu search and variable neighbourhood search, Networks 42 (2003), 48–64

[18] F.A. Özsoy and M.Ç. Pınar, An exact algorithm for the capacitated vertex p-center problem, Comput Oper Res 33 (2006), 1420–1436

[19] R. Panigrahy and S. Vishwanathan, An $O(\log^* n)$ approximation algorithm for the asymmetric p-centre problem, J Algorithm 27 (1998) 259–268

[20] G. Reinelt, TSPLIB, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ Accessed July 2006

[21] D.J. Rosenkrantz, R.E. Stearns and P.M. Lewis II, An analysis of several heuristics for the traveling salesman problem, Siam J Comput 6 (1977) 563–581

[22] M.P. Scaparra, S. Pallottino and M.G. Scutellà, Large-scale local search heuristics for the capacitated vertex p-center problem, Networks 43 (2004), 241–255

[23] J.G. Siek, L.-Q. Lee and A. Lumsdaine, The boost graph library, Addison-Wesley, Boston, 2002

[24] P.J. Slater, Locating central paths in a graph, Transport Sci 16 (1982), 1–18