ISSN 0143-4543

University of Aberdeen Business School

THE SUBTOUR CENTRE PROBLEM

By

Dr John Lamb

Discussion Paper 2007-19
March 2007

Editor: Dr W David McCausland
www.abdn.ac.uk/business/

The subtour centre problem

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

Abstract

The subtour centre problem is the problem of finding a closed trail S of bounded length on a connected simple graph G that minimises the maximum distance from S to any vertex of G. It is a central location problem related to the cycle centre and cycle median problems (Foulds et al., 2004; Labbé et al., 2005) and the covering tour problem (Current and Schilling, 1989). Two related heuristics and an integer linear programme are formulated for it. These are compared numerically using a range of problems derived from TSPLIB (Reinelt, 1995). The heuristics usually perform substantially better then the integer linear programme and there is some evidence that the simpler heuristics perform better on the less dense graphs that may be more typical of applications.

keywords: Location, Combinatorial optimization, Heuristics, Linear programming

1 Introduction

The *subtour centre problem* is the problem of finding a closed trail S of bounded length on a connected simple graph G so that the maximum distance from S to any vertex of G is as small as possible. More precisely, let G be a connected simple graph in which each edge has a nonnegative *length*. If S is a closed trail in G we can define its *length* to be the sum of the lengths of its edges. We call a closed trail S in G a *subtour*. We use the word subtour to distinguish from tours, which usually visit all the vertices.
We define the *distance* $d(u, v)$ (or sometimes d_{uv}) between vertices u and v of G as the length of the shortest path from u to v. More generally, we define the *distance* between a vertex u and a subgraph H of G as

$$d(u, H) = \min v \in V(H) d(u, v). \tag{1}$$

The subtour centre problem, then, is to find a subtour S of length no greater than a given bound $b \geq 0$ that minimises the maximum value of $d(u, S)$

taken over all $u \in V(G)$. The value that minimises $d(u, H) : u \in V(G)$ for a subgraph H of G is called the *eccentricity* of H. Subgraphs that minimise eccentricity subject to some constraint are often called *centres*.

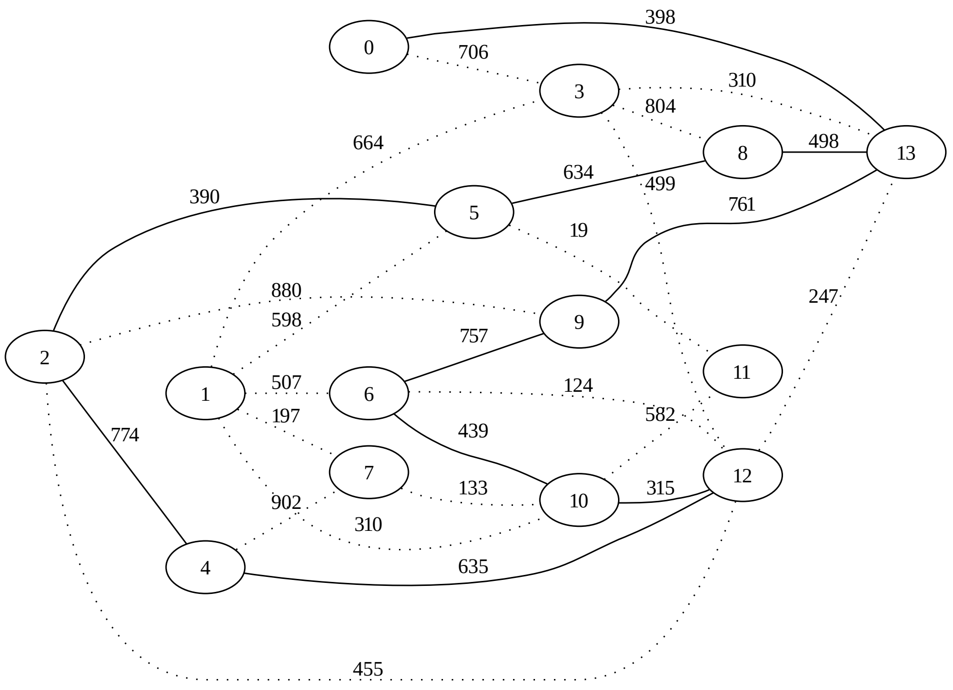

Figure 1: Example of a subtour centre

Fig. 1 shows a graph and its subtour centre. The bound on subtour length is 6000 and the subtour is drawn with solid edges. Vertex 13 appears twice in the subtour.

Typical applications of the subtour centre problem are locating a mobile facility or service on a transport network with constraints on the distance the facility can move. Examples might include the routing of mobile services such as post offices or libraries, the design of bus or train routes and touring problems for theatres and the like. As far as I know, the subtour centre problem has not been studied before except in (Lamb, 2006), which considers only simple insertion heuristics.

Network location problems have been studied since the 1960s. A natural problem is to find a structure that is central in a graph. There are two obvious definitions of centrality. One is the centre, defined above. The other is the *median*, defined like the centre but replacing 'maximum distance' with 'sum of distances'. Both of these and a third, new, definition of centrality, called the centroid, are discussed in detail in (Foulds et al., 2004). The earliest problems to be studied sought 1-medians and 1-centres (Hakimi, 1964): that is, problems in which the median or centre comprises a single vertex. In later studies the central structure was extended to a null graph on p vertices (p-centre) (Drezner, 1984; Dyer and Frieze, 1985), to paths (Slater, 1982), trees (Hutson and ReVelle, 1989) and circuits (Foulds et al., 2004; Labbé et al., 2005; Gendreau et al., 1997; Akinc and Srikanth, 1992; Arkin and Hassin, 1994). Mesa and Boffey (1996) survey the problems studied. Two problems are of particular interest. The first is the cycle centre problem discussed by Foulds et al. (2004). It differs from the subtour centre problem by requiring that solutions are circuits (degree two rather than even degree). They consider only graphs with all edge weights equal to one and find integer linear programming (ILP) solutions. We base our ILP formulation on theirs. The second problem is the covering salesman (Current and Schilling, 1989) or covering tour (Gendreau et al., 1997) problem. (See also (Akinc and Srikanth, 1992) for a generalisation.) This seeks a minimum length tour subject to an eccentricity bound rather than a minimum eccentricity tour subject to a tour-length bound. The solution heuristics for the covering tour problem tackle first a set covering problem to identify a suitable set of vertices for the solution. Then they tackle a travelling salesman problem (TSP) to minimise subtour length. Since we seek a subtour of bounded length rather than bounded eccentricity, we cannot easily adopt such an approach. For, we cannot easily choose vertices for the subtour before we consider its length.

Section 2 develops an ILP formulation for the subtour centre problem. This draws on (Foulds et al., 2004) and (Labbé et al., 2005). Section 3 develops a heuristic for the problem. It uses insertion heuristics similar to those used for p-centre (Dyer and Frieze, 1985) and TSP (Rosenkrantz et al., 1972) problems. It also uses improvement heuristics similar to those used for p-centre (Drezner, 1984) and TSP (Lin and Kernighan, 1973) problems. Section 4 explains how we implement the heuristic efficiently and presents a simplified heuristic. Section 5 presents numerical performance comparisons for the ILP method and the heuristics, using problems derived from TSPLIB (Reinelt, 1995).

We note some features of the subtour centre problem. First, it is $\mathcal{NP}$-hard. For, consider the subtour centre problem in which we seek a subtour of length at most n where G has n vertices and all edges have length one. In this problem, G has a Hamilton cycle whenever there is a solution with eccentricity zero. Since the Hamilton cycle problem is $\mathcal{NP}$-complete, it follows that the subtour centre problem is $\mathcal{NP}$-hard.

The problems we consider do not include any prespecified vertices, often

called hubs, common in location problems. However, the ILP formulation and heuristics are easily be adapted to handle this extra restriction.
We can assume distances obey the triangle inequality: for u, v and $w \in V(G)$,

$$d(u, w) \leq d(u, v) + d(v, w). \qquad (2)$$

Rosenkrantz et al. (1972) note that, if necessary, we can add a sufficiently large constant to every edge length to ensure this condition is met. We test our solution methods on problems that obey the triangle inequality. There is some reason (see (Lamb, 2006)) to believe the heuristics will perform less well on problems that do not satisfy the triangle inequality.
We use an observation commonly used for the TSP. We construct a complete graph $\hat{G}$ from G as follows. Let $V(\hat{G}) = V(G)$, let $E(\hat{G}) = \{uv\colon u, v \in V(\hat{G}), u \neq v\}$ and, for $uv \in E(\hat{G})$, let uv have length $d(u, v)$. Clearly, for any subtour of G there is a corresponding subtour of $\hat{G}$ of the same length. If G satisfies the triangle inequality then so does $\hat{G}$. If S is a closed trail in $\hat{G}$ with a repeated vertex x, we can remove one instance of x from S and join its neighbours to get a subtour of no greater length on the same vertices. We can repeat this process until S has is a circuit. Hence, any shortest subtour S on $U \subseteq V(G)$ corresponds to a circuit of $\hat{G}$ and so we need only consider circuits of $\hat{G}$ when looking for subtours of G.
From here on, we write $\hat{G}$ for the complete graph corresponding to G.

2 An integer linear programming formulation

This section presents an integer linear programming (ILP) formulation of the subtour centre problem. It draws on (Foulds et al., 2004) and (Labbé et al., 2005).
The formulation uses the following observation, used, for example, by Hassin et al. (2003). Given a circuit C of $\hat{G}$, each $u \in V(\hat{G})$ can be *assigned to* a vertex v of C chosen so that $d(u, v) \leq d(u, w)$ for $w \in V(C)$. The assignment is often, but not always, unique. We can assume without loss of generality that if $v \in V(C)$ then v is assigned to itself. For the ILP formulation it is convenient to seek not just a circuit C but also an assignment of each $u \in V(\hat{G})$ to a unique $v \in V(C)$ with v assigned to itself for $v \in V(C)$.
Label the vertices of $\hat{G}$ as $1, \ldots, n$ and the edges $e_1, \ldots, e_m$. If e is an edge joining vertices i and j, we write ij to represent e when it is convenient to do so. We write the length of an edge as d_{ij} (as before) or d_e. Let $b \geq 0$ be the maximum length of a subtour. Let $\mathcal{C}(\hat{G})$ denote all circuits of $\hat{G}$. Then we seek a circuit $C \in \mathcal{C}(\hat{G})$ satisfying

$$\min_{C \in \mathcal{C}(\hat{G})} \max_{i \in \{1,\ldots,n\}} d(i, C) \quad \text{subject to} \quad \sum_{e \in C} d_e \leq b. \qquad (3)$$

We restate this as an ILP. First, let p_{ij} ($i, j = 1, \ldots, n$), x_e ($e = 1, \ldots, m$), z_i ($i = 1, \ldots, n$) and z be variables. These represent the following in a solution.

1. $p_{ij} = 1$ if and only if i is assigned to j.
2. $x_e = 1$ if and only if the solution contains e.
3. Each vertex i is incident with $p_{ii}z_i$ solution edges.
4. z is the eccentricity of the solution.

Following Labbé et al. (2005), we define, for a subgraph H of $\hat{G}$, $\delta(H)$ to be the set of edges with exactly one vertex in H. We write $\delta(i)$ for $\delta(\{i\})$. Also we define

$$\delta(e, H) = \begin{cases} 1 & \text{if } e \in \delta(H), \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

and

$$x(\delta(H)) = \sum_{e \in E(\hat{G})} x_e \delta(e, H). \tag{5}$$

In other words, if $\{e_i : i \in I\}$ is the set of edges joining $V(H)$ to $V(\hat{G}) \setminus V(H)$ then $x(\delta(H)) = \sum_{i \in I} x_i$.
We wish to solve

$$\min z \tag{6}$$

subject to

$$\sum_{e \in E(\hat{G})} d_e x_e \leq b, \tag{7}$$

$$\sum_{i=1}^{n} p_{ij} = 1, \qquad j = 1, \ldots, n \tag{8}$$

$$z \geq \sum_{i=1}^{n} p_{ij} d_{ij}, \qquad j = 1, \ldots, n \tag{9}$$

$$p_{ij} \leq p_{jj}, \qquad i, j = 1, \ldots, n \tag{10}$$

$$x(\delta(i)) = 2z_i p_{ii} \qquad i = 1, \ldots, n, \tag{11}$$

$$x(\delta(S)) \geq 2, \qquad S \subsetneq \{i \in V(\hat{G}) : p_{ii} = 1\}, \tag{12}$$

$$0 \leq x_e \leq 1 \qquad e \in E(\hat{G}), \tag{13}$$

$$0 \leq p_{ij} \leq 1 \qquad i,j \in V(\hat{G}), \tag{14}$$

$$x_e \text{ integer} \qquad e \in E(\hat{G}), \tag{15}$$

$$p_{ij} \text{ integer} \qquad i,j \in V(\hat{G}), \tag{16}$$

$$z_i \text{ integer} \qquad i \in V(\hat{G}). \tag{17}$$

Constraint (7) ensures the circuit length is at most b. Constraints (8) ensure that each vertex is assigned to precisely one solution vertex. Constraint (8) together with constraints (14) and (16) ensure that if k is assigned to j then one of constraints (9) has the form

$$z \geq d_{kj}. \tag{18}$$

Thus the objective is at least as large as the eccentricity of the solution. Constraints (10) ensure that each vertex is assigned to a solution vertex. Constraints (11) ensure that the solution is a subgraph of maximum degree two. And constraints (12) ensure the solution is connected.
Following Foulds et al. (2004, (2.8), (2.9)) we add constraints

$$x_{ij} \leq p_{ii} \qquad i,j \in V(\hat{G}), \tag{19}$$

$$x_{ij} \leq p_{jj} \qquad i,j \in V(\hat{G}), \tag{20}$$

to obtain better performance.
The number of constraints in (12) grows exponentially with problem size. So we remove these constraints. Then we solve the relaxed ILP. Whenever we find a solution that is not connected, we choose a component C of the solution of largest eccentricity. We add a lazy constraint of the form

$$\sum_{i \in I} x_i \leq |I| - 1, \tag{21}$$

where $\{e_i : i \in I\}$ is the set of edges of C. Then we solve the ILP again. This guarantees that the solution is ultimately connected and so satisfies constraints (12). We use constraints (21) rather than the method suggested in Foulds et al. (2004) for two reasons. First, I use the Boost graph library (Siek et al., 2002; Boost, 2006) to test for connectivity. This gives components explicitly and makes it easy to find constraints of the form of (21). Second, in contrast to the method of Foulds et al. (2004), we avoid adding extra variables to the ILP model each time we add a constraint. This lets CPLEX (Ilog, 2006) solve the revised ILP more efficiently because it starts from an existing solution if no new variables are introduced.

3 A heuristic for the subtour centre problem

Since the subtour centre problem is $\mathcal{NP}$-hard, we should reasonably expect that a good heuristic will perform better on average than the ILP method for larger problems. We develop two heuristics, one in this section, one in the next. As in Section 2, we seek circuits in the complete graph $\hat{G}$ corresponding to G rather than subtours directly. We consider the empty set, a single vertex, and a pair of parallel edges together with their vertices to be circuits.

We need some definitions in order to describe the first heuristic. We *remove* a vertex x from a circuit C that has two or more vertices by removing x from the vertices of C and joining its neighbours in C. If C comprises one vertex x, we *remove* x from C by replacing C with $\emptyset$. We *insert* a vertex x into a circuit C with at least one vertex by finding an edge uv of C that minimises $d(ux) + d(xv) - d(uv)$, disconnecting u and v and joining x to u and v. If $C = \emptyset$, we *insert* a vertex x into C by replacing it with the circuit comprising x. And if C comprises a single vertex u, we *insert* a vertex x into C by replacing it with the circuit comprising x, u and parallel edges xu and ux. If C is a circuit, x a vertex of C and $y \neq x$ a vertex not in C we *exchange* x and y by removing x then inserting y. The *cost* of an insertion, exchange or removal is the length of the circuit after the operation minus the length before.

The first heuristic we develop comprises several parts. Each is a smaller heuristic chosen because it modifies an existing circuit. We want a heuristic that increases the number of vertices in the circuit. We use one based on a heuristic of Dyer and Frieze (1985) for the geometric p-centre problem.We call it the *Dyer–Frieze heuristic:*

```
Let $C$ be a (possibly empty) circuit.
While there exists a vertex $x$ such that if we insert $x$ in $C$ we get a circuit of length at most $b$:
    Choose a vertex $x$ that maximises $d(x, C)$ subject to the constraint that if we insert $x$ in $C$ we get a circuit of length at most $b$:
    Insert $x$ in $C$.
```

This is a farthest insertion heuristic. Using results of (Dyer and Frieze, 1985) and (Rosenkrantz et al., 1972), we can show that a circuit constructed using farthest insertion has bounds on length and eccentricity. For details, see (Lamb, 2006), which also demonstrates good empirical performance for this insertion method.

We use a heuristic that seeks to improve on a circuit by exchanging a vertex on the circuit with another not on it. We adapt a heuristic developed by Drezner (1984) for the geometric p-centre problem. Hassin et al. (2003) apply the combination of Dyer–Frieze and this heuristic to p-centre problems. They also note that no polynomial bound on its running time is known and show its error ratio cannot be bounded by a constant. We use it because it has shown good practical performance.

Suppose C is a circuit with vertices $v_1, \dots, v_p$. For each i $(1 \leq i \leq p)$, let $N_i = \{v \in V(G) : d(v, v_i) = \min d(v, v_j)\}$ where the minimum is taken over $j = 1, \dots, n$. In other words, N_i is the set of vertices of G that are at least as near to v_i as to any other vertex of C. Thus N_i contains all the vertices assigned to i. For each i $(1 \leq i \leq p)$, let K_i be the set of vertices that are 1-centres of N_i in G: that is, the set of vertices u of G that minimise $\max d(u, v)$ taken over all $v \in N_i$. An *exchange candidate* is a pair (v_i, k) with $k \in K \setminus V(C)$. The *Drezner heuristic* is as follows:

```
Let $C$ be a circuit.
Repeat
    Let $U$ be the set of exchange candidates $(x, y)$ such that if we exchange
    $x$ with $y$ in $C$ we get a circuit with length at most $b$ and strictly smaller
    eccentricity.
    If $U \neq \emptyset$ choose $(x, y) \in U$ so that the circuit obtained by exchanging $x$
    and $y$ in $C$ has eccentricity as small as possible.
    Exchange $x$ and $y$ in $C$.
until $U = \emptyset$.
```

We can think of this heuristic as a 1-opt local search algorithm that seeks to improve the eccentricity without exceeding the circuit bound. Drezner (1984) originally suggests also a k-opt version of the algorithm. We restrict exchanges to one vertex at a time so that we have only one new circuit to check at each step. An alternative approach would allow k-opt exchanges and solve a TSP problem on each set of vertices tested at each step. Current and Schilling (1989) use a similar method for the covering tour problem.

We can sometimes shorten a circuit by changing the order of its vertices. This does not change its eccentricity. Since we are working with a circuits on a complete graph, changing the order of vertices is a TSP. We try to solve this TSP using one of the most popular and effective TSP heuristics, the *Lin–Kernighan heuristic*. There are many variations of this. We use essentially the version of Cook et al. (1998, pp. 247–249) because it limits the number of rearrangements considered and so terminates reasonably quickly. The documentation and code in (Lamb, 2007) give full details.

A circuit may contain a vertex whose removal does not increase its eccentricity. And its length cannot increase if the triangle inequality (2) holds. We call the following the *shorten circuit* heuristic.

```
Let $C$ be a circuit.
Repeat
    Let $U$ be the set of vertices of $C$ such that if $u \in U$ and removing $u$ from
    $C$ gives a circuit with eccentricity at most $b$ and no increase in length.
    If $U \neq \emptyset$:
        Let $u_1, \dots, u_k$ be $U$ sorted in decreasing order of improvement in
        circuit length
        For $i = 1$ to $k$:
            Remove $u_i$ from $C$ if doing so does not increase the eccentricity of
            $C$.
until $U = \emptyset$.
```

The next heuristic creates a minimally infeasible solution. The idea is that Lin–Kernighan or shorten circuit may be more useful if we increase the length of the circuit before applying it. *Nearest good vertex* is as follows.

```
Let $C$ be a circuit.
Let $v$ be the vertex in $V(G) \setminus V(C)$ nearest $C$.
Insert $v$ in $C$.
```

Heuristic 1, described in Fig. 2, is our first heuristic the subtour centre problem. We consider improvement relative to the best solution so far found. We consider C' an improvement of C if both are feasible and either C' has lower eccentricity than C or C' has the same eccentricity as C and C' is shorter. If a method finds an improvement C' of C we replace C with C'.

```
Let $C$ be an initial candidate circuit.
Use the Dyer–Frieze heuristic to try to improve $C$.
Use the Drezner heuristic to try to improve $C$.
Repeat:
    Use nearest good vertex to insert a vertex in $C$.
    Use Lin–Kernighan to try to shorten $C$.
    Use shorten circuit to try to remove unneeded vertices from $C$.
    Use the Dyer–Frieze heuristic to try to improve $C$.
    Use the Drezner heuristic to try to improve $C$.
Until no improvement is found.
```

Figure 2: Heuristic 1

In practice, Heuristic 1 is very fast and so we repeat it with different candidate circuits as follows. We suppose, as in Section 2, the vertices are labelled $1, \ldots, n$. Our first candidate circuit is the empty circuit. Thereafter, we use candidate circuits containing the vertices $\{1\}, \ldots, \{n\}, \{1, 2\}, \{1, 3\}, \ldots, \{1, n\}$, $\{2, 3\}, \ldots$. We construct each candidate circuit by placing its vertices in random order. We only apply the heuristic if the length of the candidate circuit is at most the bound b.

As with the ILP method, we generate a subtour from the best circuit found by replacing each edge with a shortest path joining its vertices.

4 Implementing the heuristic efficiently

We now consider how we might implement Heuristic 1 efficiently. There are two issues. First, we need efficient coding and data structures. Second, there is no guarantee that each step contributes substantially to performance.This leads us to develop a simplified heuristic, omitting two steps.

The heuristic and ILP algorithm are both coded in C++. I use two libraries to help achieve efficient code. The first is the Boost graph library (Siek et al., 2002; Boost, 2006). It is a template library written in C++. It contains various data structures to handle graphs and algorithms efficiently. In particular, it includes efficient algorithms for depth-first search and Johnson all-pairs shortest paths. We need these to measure eccentricity and find 1-centres. The data structures are created from templates at compile time, reducing the run-time overhead often associated with object-oriented programming. The second library is CPLEX (Ilog, 2006). It is a well-known efficient library for linear and integer programming problems.

The combination of CPLEX and the Boost graph library allows us to quickly develop code for both the ILP algorithm and for heuristics. It simplifies translation from an ILP form to a graph form. So we can easily create ILP constraints of the form of (21), which depend on efficiently finding graph components.

The data structures and algorithms are given in detail in (Lamb, 2007). The most important new data structure is one to hold trails. These represent circuits and subtours. Ideally we want to quickly iterate round a trail, but also to quickly search through it for a given vertex. To achieve both, I implement a trail as a multimap using the C++ standard template library (Austern, 1999). Each vertex is given an integer index and mapped to a pair of iterators. The first iterator points to a predecessor (if there is one) in the trail, the second to a successor. Consider a trail with n vertices (including repeats). Since we represent it as a multimap, we can search for a given vertex in time $O(\log n)$.

And the iterators let us to traverse the trail in order in time $O(n)$.

We now describe a simplified heuristic. Each of the steps in Heuristic 1 can plausibly help improve a solution. However, each also consumes time and may contribute little to finding a solution. So we also consider Heuristic 2 in Fig. 3. It omits the Drezner and Lin–Kernighan steps. Section 5 explains why we choose to omit them.

```
Let $C$ be an initial candidate circuit.
Use the Dyer–Frieze heuristic to try to improve eccentricity.
Repeat:
    Use nearest good vertex to insert a vertex in $C$.
    Use shorten circuit to try to remove unneeded vertices from $C$.
    Use the Dyer–Frieze heuristic to try to improve eccentricity.
Until no improvement is found.
```

Figure 3: Heuristic 2

5 Numerical results

This section presents numerical results. It compares the performance of the ILP algorithm and the heuristics across a range of problems.

We need a range of test problems. The TSPLIB problems (Reinelt, 1995) are a good choice. They are standard test problems for the TSP and include problems known to be difficult. We adapt these in two ways. First, we need length bounds to create subtour centre problems for each TSPLIB graph. The optimum TSP length is one possibility. If the subtour length bound is much greater, the subtour centre problem should become easy. For, then there is a solution with eccentricity zero, and the larger the bound the more should we expect to find such a solution. Similarly, if the subtour length bound is much smaller than the optimum TSP length, we should expect the problem to become easy because we only have to check small subtours. We consider bounds on subtour length that are the optimum TSP length or one half or one third of it.

Second, the TSPLIB problems typically represent distances on a complete graph. But we wish to also consider problems where the graph G is not complete. Clearly, we need only consider G connected. We restrict our attention to G 2-connected so that any vertex might occur in a solution. We construct a new problem instance from each TSPLIB problem we test by adding edges uniformly at random until we get a 2-connected graph. The new problem instances and the code that generated them are available from (Lamb, 2007). The problem instances are in a format used by many TSPLIB files, documented in (Reinelt, 1995).

Before we compare Heuristics 1 and 2, and the ILP algorithm in detail, let us consider why Heuristic 2 might be a reasonable alternative to Heuristic 1. Although each step of Heuristic 1 may improve the solution there is no guarantee that it will do so. Even if it does, we should consider how much CPU time is consumed and how much the step contributes to improving the solution. I investigated these by letting Heuristic 1 run to completion on several problems (up to 29 vertices) and found two things. First, no individual step used substantially more CPU time than any other. Second, more than 99% of the steps that improved eccentricity or circuit length were Dyer–Frieze or shorten-circuit steps. These suggest that Heuristic 2, in Fig. 3 above, as a good alternative to Heuristic 1. We retain the nearest-good-vertex step because it is designed to perturb a solution rather than improve it.

We carry out tests on six problems based on each of the 29 TSPLIB problems with 101 or fewer vertices. We report detailed results for 15 problems that are typical of the whole set.

Tables 1–3 show the results of numerical tests of the ILP algorithm and the two heuristics. Each test ran with a time limit of eight hours CPU time on the same computer and operating system: a personal computer with a 3.0 GHz Pentium 4 processor, 1 Gb RAM and the SuSE Linux 10.1 operating system. The test program recorded the CPU time, eccentricity and length of each feasible solution that improved over the previous best. The heuristics allow feasible solutions with two parallel edges. We exclude these from the results to allow a fair comparison with the ILP algorithm.

Each of Tables 1–3 reports results for five problems. For each problem we consider three bounds and two variants: tsp is the original TSP graph; cc2 is a 2-connected graph obtained from it as described above. The column titled **best** shows the eccentricity of the best solution found by any of the three methods. Entries in it are integer because the lengths of edges in the TSPLIB problems are integer. Bold values indicate values where the solution is known to be optimum, either because the ILP algorithm ran to completion or because the problem is a TSP problem with a known optimum solution. The columns titled **ilp**, **h1** and **h1** show approximately the seconds of CPU time taken to reach a solution with minimum eccentricity by the ILP algorithm, Heuristic 1 and Heuristic 2. There is no guarantee that any method will find a minimum-eccentricity solution and a dash indicates when a method fails.

The last four columns of the tables indicate how quickly each of the three solution methods came to within each of four upper bounds on the eccen-

Problem	**type**	**bound**	**best**	**ilp**	**h1**	**h2**	**25%**	**10%**	**5%**	**1%**
burma14	cc2	3323	**625**	0.284	0.02	0.004	21L	21L	21L	21L
burma14	tsp	3323	**0**	0.468	0.004	0.016	12L	12L	12L	12L
burma14	cc2	1661	**757**	0.224	0.0	0.0	12L	12L	12L	12L
burma14	tsp	1661	**400**	0.812	0.004	0.004	12L	12L	12L	12L
burma14	cc2	1107	**757**	0.02	0.0	0.0	12L	12L	12L	12L
burma14	tsp	1107	**406**	0.408	0.004	0.0	21L	21L	21L	21L
ulysses16	cc2	6859	**904**	1.444	0.06	0.012	12L	12L	12L	21L
ulysses16	tsp	6859	**0**	1.908	0.004	1.3	12L	1L2	1L2	12L
ulysses16	cc2	3429	**1504**	0.052	0.004	0.004	12L	12L	12L	12L
ulysses16	tsp	3429	**1122**	0.188	0.084	0.004	21L	21L	21L	21L
ulysses16	cc2	2286	**1504**	0.012	0.0	0.0	12L	12L	12L	12L
ulysses16	tsp	2286	**1387**	0.016	0.0	0.0	12L	12L	12L	12L
ulysses22	cc2	7013	**1262**	4.268	0.084	0.004	21L	21L	21L	21L
ulysses22	tsp	7013	**0**	239.9	0.02	1.868	12L	12L	12L	12L
ulysses22	cc2	3506	**1581**	11.36	0.0	0.0	12L	12L	12L	12L
ulysses22	tsp	3506	**1122**	0.212	0.008	0.028	12L	12L	12L	12L
ulysses22	cc2	2337	**1772**	0.892	0.004	0.0	12L	12L	12L	21L
ulysses22	tsp	2337	1387	0.04	0.004	0.004	12L	12L	12L	12L
gr24	cc2	1272	**258**	4.532	0.128	0.024	12L	21L	21L	21L
gr24	tsp	1272	**0**	0.232	0.032	0.264	21L	1L2	1L2	1L2
gr24	cc2	636	**282**	1.612	0.004	0.0	21L	21L	21L	21L
gr24	tsp	636	**74**	11.78	0.008	0.588	12L	12L	21L	12L
gr24	cc2	424	**354**	1.948	0.02	0.008	21L	21L	21L	21L
gr24	tsp	424	**122**	16.25	0.012	0.004	21L	21L	21L	21L
fri26	cc2	937	**140**	6.58	0.044	0.016	21L	21L	21L	21L
fri26	tsp	937	**0**	1.848	0.008	0.196	21L	12L	12L	12L
fri26	cc2	468	**188**	2.044	0.004	0.004	12L	12L	12L	12L
fri26	tsp	468	**71**	18.88	0.068	0.0	12L	21L	21L	21L
fri26	cc2	312	**227**	0.308	0.18	0.032	21L	21L	21L	21L
fri26	tsp	312	**93**	1.68	0.02	0.112	12L	12L	12L	12L

Table 1: Numerical results part 1

Problem	**type**	**bound**	**best**	**ilp**	**h1**	**h2**	**25%**	**10%**	**5%**	**1%**
bayg29	cc2	1610	**220**	10.56	0.1	0.016	12L	12L	21L	21L
bayg29	tsp	1610	**0**	2.796	1.616	–	12L	1L	1L	1L
bayg29	cc2	805	**257**	8.617	0.172	0.012	12L	21L	21L	21L
bayg29	tsp	805	**74**	546.7	0.176	0.004	12L	21L	21L	21L
bayg29	cc2	536	**294**	1.844	0.004	1.216	21L	12L	12L	12L
bayg29	tsp	536	**121**	248.7	165.7	60.52	21L	21L	12L	21L
dantzig42	cc2	699	**124**	38.29	0.004	0.004	12L	12L	12L	12L
dantzig42	tsp	699	**0**	166.7	1.936	3852	12L	12L	1L2	1L2
dantzig42	cc2	349	**164**	38.79	0.02	0.0	21L	21L	21L	21L
dantzig42	tsp	349	34	–	0.304	9.037	21	21	12	12
dantzig42	cc2	233	**164**	21.73	0.024	0.008	12L	21L	21L	21L
dantzig42	tsp	233	52	7461	0.344	0.024	21L	21L	21L	21L
att48	cc2	10628	**1043**	517.7	48.73	59.59	21L	12L	12L	21L
att48	tsp	10628	**0**	24957	125.0	–	21L	12L	1L	1L
att48	cc2	5314	**1527**	376.2	8.625	0.048	21L	21L	21L	21L
att48	tsp	5314	401	–	0.188	0.708	21	21	21	12
att48	cc2	3542	**1783**	76.06	71.98	0.416	21L	21L	21L	21L
att48	tsp	3542	739	–	0.056	0.056	12L	12	12	12
berlin52	cc2	7542	**701**	688.0	5.596	104.4	21L	21L	12L	12L
berlin52	tsp	7542	**0**	57.84	2.292	–	12L	12L	1L	1L
berlin52	cc2	3771	**952**	112.1	0.04	0.016	21L	21L	21L	21L
berlin52	tsp	3771	365	–	0.012	0.076	12L	12L	12L	12
berlin52	cc2	2514	**1125**	37.99	0.012	0.016	12L	12L	12L	12L
berlin52	tsp	2514	426	23230	41.55	0.724	21L	21L	21L	21L
brazil58	cc2	25395	**2471**	603.0	3.084	0.86	21L	21L	21L	21L
brazil58	tsp	25395	**0**	9875	1.84	–	21L	21L	1L	1L
brazil58	cc2	12697	2929	262.1	0.564	0.004	21L	21L	21L	21L
brazil58	tsp	12697	1126	–	22.65	18.55	21L	21	21	21
brazil58	cc2	8465	**3178**	159.0	7.548	0.8	21L	21L	21L	21L
brazil58	tsp	8465	1991	–	0.384	0.008	21L	21	21	21

Table 2: Numerical results part 2

Problem	**type**	**bound**	**best**	**ilp**	**h1**	**h2**	**25%**	**10%**	**5%**	**1%**
st70	cc2	675	**59**	7406	–	–	12L	12L	12L	L
st70	tsp	675	**0**	4356	415.0	–	12L	12L	1L	1L
st70	cc2	337	**93**	11232	0.008	0.008	21L	21L	21L	21L
st70	tsp	337	17	–	897.0	–	21L	12	12	1
st70	cc2	225	**93**	1519	0.008	0.0	21L	21L	21L	21L
st70	tsp	225	28	–	0.732	0.112	21	12	21	21
eil76	cc2	538	**36**	3381	–	–	12L	L	L	L
eil76	tsp	538	**0**	73.59	301.5	–	12L	1L	L1	L1
eil76	cc2	269	**46**	3300	1.284	10.01	21L	12L	21L	12L
eil76	tsp	269	11	–	0.348	104.5	21L	21L	12	12
eil76	cc2	179	**50**	1306	0.288	0.12	21L	21L	21L	21L
eil76	tsp	179	15	–	9.409	19.49	21	12	12	12
gr96	cc2	55209	5906	–	13.92	14.92	21L	21	12	12
gr96	tsp	55209	**0**	–	1527	–	12L	12L	1L	1
gr96	cc2	27604	7551	–	489.9	1071	21L	21L	12L	12
gr96	tsp	27604	1224	–	1435	–	12L	12	12	12
gr96	cc2	18403	8651	4601	11.6	1.784	21L	21L	21L	21L
gr96	tsp	18403	2465	–	1.048	0.208	21	21	21	21
kroA100	cc2	21282	2062	–	–	27128	12L	12	2	2
kroA100	tsp	21282	**0**	–	248.8	–	21L	12L	1L	1
kroA100	cc2	10641	2789	–	39.99	82.85	21L	21	12	12
kroA100	tsp	10641	417	–	4453	5885	21L	12	21	12
kroA100	cc2	7094	3104	15515	25.25	7.973	21L	21L	21L	21L
kroA100	tsp	7094	794	–	1842	269.1	21	12	21	21
eil101	cc2	629	33	–	606.0	–	12L	12	12	1
eil101	tsp	629	**0**	1270	1335	–	12L	12L	1L	L1
eil101	cc2	314	41	15757	3464	11144	21L	21L	21L	12L
eil101	tsp	314	10	–	3.22	82.37	21L	21	12	12
eil101	cc2	209	45	27678	10880	12680	21L	12L	12L	12L
eil101	tsp	209	14	–	373.3	20386	21	21	21	21

Table 3: Numerical results part 3

tricity of the best solution. These are calculated as $b^* + p(c - b^*)$ where b^* is the eccentricity of the best solution found, c is the eccentricity of the 1-centre and p is 0.25, 0.1, 0.05 or 0.01. The 1-centre is easily found and has length zero. So any reasonable solution should have eccentricity between b^* and c and the bounds give a crude indication of how close we are to the best solution we know of. The entries in the columns show from left to right the order in which the three methods found a solution: L indicates ILP, 1 indicates Heuristic 1 and 2 indicates Heuristic 2. If times are identical then ILP is listed before Heuristic 1 and Heuristic 1 before Heuristic 2. If a method fails to reach a bound it is not listed.

We see immediately from the results that both Heuristics 1 and 2 perform well, often finding an optimum solution even in the tsp problems that are equivalent to travelling salesman problems because they have eccentricity bound equal to zero. Only in two cases, one with problem st70 and one with eil76, do neither of the heuristics find the best solution. There is clearly some increase in solution time as problem size increases, but also dramatic differences in solution time for similar problems. I have considered regression models using results for all 29 TSPLIB problems on 101 to estimate how solution time increases with problem size but have found no model that can predict with any reasonable confidence how solution time and problem size are related.

			method					method			
bound			**ilp**	**h1**	**h2**	total	bound	**ilp**	**h1**	**h2**	total
0%	type	tsp	3	62	31	96	10%	0	51	48	99
		cc2	6	37	60	103		2	39	64	105
	total		9	99	91	199		2	90	112	204
1%	type	tsp	3	62	31	96	25%	0	36	76	112
		cc2	6	36	61	103		0	33	76	109
	total		9	98	92	199		0	69	152	221
5%	type	tsp	1	58	37	96					
		cc2	4	39	61	104					
	total		5	97	98	200					

Table 4: Summary of results

It is difficult to see from Tables 1–3 performance differences between Heuristic 1 and Heuristic 2. Table 4 summarises the data in a way that should make differences clearer. It shows for each type of problem, for each bound (25%, 10%, 5% and 1%) and for the best solutions found (0%) the numbers of times each method is first to reach the bound. Where two or more methods find a

	Df	Sum Sq	Mean Sq	F value	Pr(> F)
problem	28	20.56	0.73	3.5166	1.891e-09*
pbound	2	0.81	0.40	1.9345	0.144808
type	1	0.13	0.13	0.6192	0.431468
method	1	7.60	7.60	36.3930	1.974e-09*
bound	1	2.17	2.17	10.4023	0.001282
type:method	1	22.76	22.76	108.9753	< 2.2e-16*
type:bound	1	0.24	0.24	1.1724	0.279066
method:bound	1	15.71	15.71	75.2321	< 2.2e-16*
type:method:bound	1	5.40	5.40	25.8510	4.095e-07*
Residuals	1702	355.46	0.21		

Selected coefficients	Estimate	Std Error	t value	Pr(> |t|)
(Intercept)	0.39588	0.06686	5.921	3.86e-09*
type tsp	0.29043	0.04170	6.966	4.66e-12*
method h2	0.29012	0.04170	6.958	4.91e-12*
bound	-0.17402	0.24057	-0.723	0.46955
type tsp:method h2	-0.65807	0.05897	-11.160	< 2e-16*
type tsp:bound	-0.96266	0.34021	-2.830	0.00472
method h2:bound	0.86345	0.34021	2.538	0.01124
type tsp:method h2:bound	2.44627	0.48113	5.084	4.09e-07*

Table 5: Analysis of covariance

solution in the same CPU time, both methods are counted, which is why the totals exceed 174 (29×6), the number of test cases. We see immediately that in each case the proportion of successes for the ILP algorithm is much fewer than we should expect if all three methods were equally good.

We construct an analysis of covariance model using *R* (R Development Core Team, 2006) to make further inferences from the result. Its dependent variable takes the value 1 for a method that is the first to reach a solution bound and zero otherwise. It includes the a factor (pbound) representing the eccentricity bound b (1, 1/2 or 1/3 of the TSP bound), and all interactions between the method, problem type and solution bound. We exclude the ILP method from the analysis. Table 5 summarises the results: * denotes effects with p-value less than 0.001. We can infer from this model that there are strong interactions between problem type, solution method and solution bound.

From the selected coefficients, we can infer that Heuristic 2 performs better, but its performance declines on tsp problems and as the solution bound (25%, 10%, 5%, 1%, 0) decreases. This is consistent with Table 4 and further

supported by an analysis of variance (not reported in detail) excluding cases with nonzero solution bound. In this the interaction between problem type and solution method is highly significant while all individual effects have p-value above 0.2. The coefficient of this effect is also highly significant and we can infer that Heuristic 1 is better at finding the best solution for tsp problems while Heuristic 2 is better for cc2 problems.
Although statistical tests are clearly useful, at least for identifying differences that might appear by chance alone, we must use caution when trying to generalise from the tests above. The statistical tests assume an independent random sample of problems. It is difficult to define what a randomly chosen subtour centre problem might be.

6 Conclusion

We have looked at two heuristics and one ILP algorithm for the subtour centre problem. As we might reasonably expect for an $\mathcal{NP}$-hard problem, the heuristics typically outperform the ILP algorithm.
For practical applications, Heuristic 2, described in Fig. 3, is probably the most useful. It is simpler and easier to implement than Heuristic 1 and shows similar performance. There is some evidence that it performs better on less dense graphs, such as we might expect in practical location problems.
The component heuristics of Heuristics 1 and 2 are reasonably general and it may be useful to investigate how easily they could be adapted to similar problems. In particular, Heuristic 2 might be adapted so that it can be used for the cycle centre problem of (Foulds et al., 2004).

References

Akinc, U., Srikanth, K., 1992. Optimal routing and process scheduling for a mobile service facility. Networks 22, 163–183.

Arkin, E. M., Hassin, R., 1994. Approximation algorithms for the geometric covering salesman problem. Discrete Applied Mathematics 55, 197–218.

Austern, M. H., 1999. Generic Programming and the STL. Addison–Wesley.

Boost, 2006. http://www.boost.org/, accessed July 2006.

Cook, W. J., Cunningham, W. H., Pulleybank, W. R., Schrijver, A. J., 1998. Combinatorial Optimization. John Wiley and sons, New York.

Current, J. R., Schilling, D. A., 1989. The covering salesman problem. Transportation Science 23 (3), 208–213.

Drezner, Z., 1984. The p-centre problem—heuristic and optimal algorithms. Journal of the Operational Research Society 35 (8), 741–748.

Dyer, M. E., Frieze, A. M., 1985. A simple heuristic for the p-centre problem. Operations Research Letters 3 (6), 285–288.

Foulds, L. R., Wilson, J. M., Yamaguchi, T., 2004. Modelling and solving central cycle problems with integer programming. Computers & Operations Research 31, 1083–1095.

Gendreau, M., Laporte, G., Semet, F., 1997. The covering tour problem. Operations Research 45 (4), 568–576.

Hakimi, S. L., 1964. Optimum location of switching centres and the absolute centres and medians of a graph. Operations Research 12, 450–459.

Hassin, R., Levin, A., Morad, D., 2003. Lexicographic local search and the p-center problem. European Journal of Operational Research 151, 265–279.

Hutson, V. A., ReVelle, C. S., 1989. Maximal direct covering tree problems. Transportartion Science 23, 288–299.

Ilog, 2006. CPLEX, http://www.ilog.com/products/cplex/, accessed November 2006.

Labbé, M., Laporte, G., Rodríguez Martín, I., Salazar González, J. J., 2005. Locating median cycles in networks. European Journal of Operational Research 160, 457–470.

Lamb, J. D., 2006. Insertion heuristics for cycle centre problems, Working Paper Series, Business School, Aberdeen University Research Archive, http://hdl.handle.net/2164/96.

Lamb, J. D., 2007. http://www.abdn.ac.uk/~cms127/code.html, accessed February 2007.

Lin, S., Kernighan, B. W., 1973. An effective heuristic algorithm for the traveling salesman problem. Operations Research 21, 498–516.

Mesa, J. A., Boffey, T. B., 1996. A review of extensive facility location in networks. European Jounal of Operational Research 95, 595–603.

R Development Core Team, 2006. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
URL http://www.R-project.org

Reinelt, G., 1995. TSPLIB, http://www.iwr.uni-heidelberg.de/groups/comopt/software/tsplib95, accessed July 2006.

Rosenkrantz, D. J., Stearns, R. E., Lewis II, P. M., September 1972. An analysis of several heuristics for the traveling salesman problem. Siam Journal on Computing 6 (3), 563–581.

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

Slater, P. J., 1982. Locating central paths in a graph. Transportation Science 16, 1–18.