# New heuristics for the balanced k-Chinese postmen problem

Yasemin Limon

Department of Industrial Engineering,
University of Wisconsin,
Madison, USA
Email: ylimon@wisc.edu

Meral Azizoğlu*

Department of Industrial Engineering,
Middle East Technical University,
Ankara 06800, Turkey
Email: ma@metu.edu.tr
*Corresponding author

**Abstract:** In this study, we consider a directed k-Chinese postmen problem that aims to balance the costs of the postmen, while maintaining the total cost as small as possible. We formulate the problem as a pure integer nonlinear programming model. We propose three solution algorithms that use efficient incorporation of the heuristic subtour elimination constraints. Our computational results have revealed the satisfactory performances of our heuristic solution algorithms.

**Keywords:** k-Chinese postmen problem; pure integer nonlinear program; heuristic solution algorithms.

**Reference** to this paper should be made as follows: Limon, Y. and Azizoğlu, M. (2018) 'New heuristics for the balanced k-Chinese postmen problem', *Int. J. Planning and Scheduling*, Vol. 2, No. 4, pp.331–349.

**Biographical notes:** Yasemin Limon received her BS and MS degrees both in Industrial Engineering, from the Middle East Technical University, Ankara, Turkey. Currently, she is a PhD student and working as a Research Assistant in the Industrial Engineering Department of the University of Wisconsin, Madison.

Meral Azizoğlu is a Full Professor of Industrial Engineering at the Middle East Technical University, Ankara, Turkey. She obtained her BS, MS and PhD degrees in Industrial Engineering from the Middle East Technical University. Her research interests include the design and analysis of manufacturing systems, production planning and scheduling.

This paper is a revised and expanded version of a paper entitled 'Heuristic procedures for K-Chinese postman problem with workload balancing' presented at INFORMS 2015, Philadelphia, USA, 1–4 November 2015.

## 1 Introduction

Routing problems have attracted the attention of many researchers due to their wide range of application areas. Those problems are studied under two main categories: node routing and arc routing. Node routing problems define the routes to serve all nodes or a subset of all nodes of a graph at minimum cost. The traveling salesman and vehicle routing problems are the most known node routing problems that have been the focus of many studies for several decades.

Arc routing problems (ARPs) have their origin in the Königsberg Bridge Problem that was solved by Euler (1736). The ARPs basically define the routes to cover all arcs or a subset of all arcs of a graph at minimum cost.

The basic ARP is known as the Chinese postman problem (CPP). The CPP finds a minimum cost closed walk traversing each arc of a graph at least once. From this basic problem, the research has focused through more difficult and general problems, and the current research is growing enormously. The main motivation behind this increasing research interest is the large number of practical situations like snow removal, street sweeping, garbage collection, delivery of goods, mail distribution, network maintenance.

An ARP is referred to as rural postman problem (RPP) when a defined subset of all arcs is to be traversed. The CPP and RPP have many variants based on their objectives and assumptions. The windy postman problem has different cost of visiting an arc in two directions. The hierarchical postman problem considers precedence relations between the arcs. The maximum benefit postman problem gains benefit at each visit of an arc. The postman problem with time window constraints has time limit on the service of an arc. Similar to the maximum benefit postman problem, the profitable postman problem and the prize-collecting postman problem are introduced with the objective of maximising the benefit gained from the service. Another prominent type of the ARP is the Capacitated ARP in which there are capacitated vehicles to serve a specified arc set. The variants of these ARPs have been arising in response to the practical needs.

The first proposed version of the CPP considers a single postman and aims to minimise the total cost. The objective of minimising total cost is of consequence for the applications like mail delivery, garbage collection, street cleaning, and snow removal.

More practical versions of the CPP have been introduced recently with the perspective that the single CPP is not realistic for some practical applications. There are usually more than one postman for a mail delivery, and similarly more vehicles are necessary for garbage collection. To incorporate multiple postmen to the CPP, the *k*-CPP is first introduced with the aim of minimising the total cost. However, the total cost objective may not be adequate to represent the real life situations when the time limitations are considered. For example, the total cost objective may produce different costs (workloads) for each snow removal vehicle. One of the snow removal vehicles may visit many streets; whereas, the other vehicles may have much less work. Such an allocation results in completing the most loaded work late. A late completion may not be tolerated due to some practical restrictions like maximum working hours and weather conditions. Hence, a solution that balances the costs of the vehicles may be essential. In the literature, the following objectives are defined to address the cost balancing concerns.

- the min-max k-Chinese postman problem (MM k-CPP)
- the minimum absolute deviance *k*-Chinese postman problem (MAD *k*-CPP)

- the minimum square deviance *k*-Chinese postman problem (MSD *k*-CPP)
- the minimum overtime *k*-Chinese postman problem (MOT *k*-CPP).

The MM *k*-CPP minimises the highest cost (workload or time) and does not consider any cost balancing among the postmen. The other three objectives aim to maintain a balancing among the costs of the postmen. The MAD k-CPP minimises the sum of all cost deviations from an allowed service cost; hence, resulting in similar postmen costs. The MSD *k*-CPP minimises the sum of squared deviations from the target cost. The goal of the squared objective function is to penalise higher deviations, and the resulting cost is more balanced compared to that of the MAD *k*-CPP. The MAD *k*-CPP and MSD *k*-CPP may increase the total cost, as not only the over deviations but also the under deviations from the target cost are penalised. The MOT k-CPP aims at minimising the sum of over deviations from the target cost. For this problem, the target cost should be accurately defined, as high target costs result in solutions with unnecessarily high cost assignments to some postmen.

In this study, we define a new cost (workload or time) balancing objective. Our aim is to minimise the cost deviations among the postmen while keeping the total cost over all postmen at a reasonable level. The resulting objective is to minimise the sum of squared costs over all postmen. The sum of the squared costs penalises higher cost deviations at higher extent; hence, gives more evenly distributed cost solution.

The nonlinearity of the objective function increases the complexity of the problem; hence, brings an additional challenge. Recognising this challenge, we present an efficient mathematical model along with efficient approaches for its solution.

The rest of the paper is organised into the following sections. The review of the related literature is given in Section 2. Section 3 defines the problem and settles its complexity. Section 4 explains the heuristic subtour elimination constraints that are used in our solution algorithms. The solution algorithms are explained in Section 5. The results of our computational runs are reported in Section 6. Section 7 concludes with our main findings and suggestions for future work.

## 2 Literature review

A recent overview of the ARPs is given in Dror (2000) where theoretical aspects related to the solution approaches and some real applications are discussed. The survey articles by Eiselt et al. (1995a, 1995b) consider the resolution of the CPP, RPP and capacitated arc routing problem (CARP) on different graph types.

The CPP is first posed by a Chinese mathematician, Guan (1962). The first studies related to the CPP examine the different graph types with the objective of minimising total cost. Euler (1736) and Ford and Fulkerson (1962) present the basics of the undirected and directed CPP, respectively. Edmonds and Johnson (1973) study the directed, undirected and mixed CPPs using the matching theory.

Minieka (1979), Pearn and Liu (1995), and Pearn and Chou (1999) present solution techniques for the CPP on mixed graphs. Corberán et al. (2002) propose a GRASP metaheuristic, Lin and Zhao (1988) examine the directed CPP based on the transportation model. Kappauf and Koehler (1979) and Ralphs (1993) give integer linear programming (ILP) formulations and analyse the polyhedron of its linear programming relaxation.

Malandraki and Daskin (1993) introduce maximum benefit CPP where a benefit is gained by traversing an arc and present an ILP model for its solution. Cabral et al. (2004) study the hierarchical CPP as an RPP, and find an optimal solution by a branch-and-cut procedure.

The *k*-CPP models basically focus on the min-max (MM) *k*-CPP. The MM *k*-CPP is shown to be strongly NP-hard through a reduction from the *k*-partition problem by Frederickson et al. (1978). Frederickson et al. (1978) propose two lower bounds and develop a heuristic procedure together with its worst-case complexity. Ahr and Reinelt (2002) develop several two-step heuristic procedures where construction steps use Augment-Merge and Clustering ideas. The results of their computational study reveal the superiority of their heuristics over those of Frederickson et al. (1978). Ahr (2004) develops improved versions of Frederickson et al. (1978) lower bounds and develops several heuristic procedures. Ahr (2004) also gives an ILP formulation including subtour elimination constraints. He uses valid cuts in his branch and cut algorithm and compares the effect of branching strategies on the performance of the branch and cut algorithm. Ahr and Reinelt (2006) and Willemse and Joubert (2012) propose tabu search algorithms to find high quality approximate solutions in reasonable solution times. Ahr and Reinelt (2006) propose three different neighbourhood structures with linear, quadratic, and cubic running time complexities. Willemse and Joubert (2012) show that the problem of designing patrol routes for security estates can be modeled as MM *k*-CPPs. They propose a tabu search algorithm that is shown to be superior to the existing solution procedures.

Different objectives for the *k*-CPP are presented in the study by Osterhues and Mariak (2005). They provide the *k*-CPP variants using three objective functions: minimising the sum of all deviations from an allowed service time, minimising the sum of squared deviations and minimising the sum of overtime. Similar to the MM *k*-CPP, these variants aim to balance postmen costs. They propose a branch and bound procedure, and find optimal and near-optimal solutions for instances with less than 25 arcs.

Shafahi and Haghani (2015) present a mathematical model for the maximum benefit *k*-CPP and discuss some general cases. Benavent et al. (2009) study the MM windy RPP. They develop a branch and cut algorithm based on the polyhedral description of the problem. Benavent et al. (2010) develop a metaheuristic approach to solve the model stated in Benavent et al. (2009). Benavent et al. (2011) present new valid inequalities for the polyhedron of the MM *k*-WRPP and use them to improve the branch and cut algorithm of Benavent et al. (2009). Benavent et al. (2014) develop a branch-price-and-cut algorithm to solve the MM *k* vehicles windy RPP.

In this study we consider the total squared cost *k*-CPP and propose algorithms for its solution. To the best of our knowledge our study is the first attempt that considers the total squared cost problem and the first optimisation effort to solve MM *k*-CPPs.

## 3 Problem definition and the complexity

Consider a directed graph $G = (N, A)$ where $A$ is the set of arcs and $N$ is the set of nodes. Arc $(i, j)$ connects nodes $i$ and $j$ and is characterised by parameter $c_{ij}$ which might represent the cost of connecting arc $(i, j)$, the distance between node $i$ and node $j$, or the time of traversing arc $(i, j)$. There are $K$ postmen each of whom has to cover at least one arc and each arc should be covered by at least one postman. We assume that $G$ is strongly connected, i.e., there is a path between every pair of nodes $i$ and $j$.

Each postman starts his/her route from the depot and completes the route at the depot. We refer to node 1 as the depot.

A circuit is a sequence of arcs that starts and ends at the same node. A circuit that does reside node 1 is called a subtour. For example, circuit 23→34→42 is a subtour; however, 12→21 is not a subtour.

The main decision of our problem is defined as follows:

$x_{ijk}$ = number of times arc$(i, j)$ is traversed by postman $k$

$\forall(i,j) \in A, k = 1,\dots,K$

The constraints are as defined below:

1 Each arc is visited at least once.

$$\sum_{k=1}^{K} x_{ijk} \geq 1 \qquad \forall(i,j) \in A \tag{1}$$

2 The flow is conserved at each node, i.e. the number of arcs entering to each node is equal to the number of leaving arcs.

$$\sum_{i \in N} x_{ijk} \sum_{i \in N} x_{jik} \qquad \forall j \in N, k = 1,\dots,K \tag{2}$$

3 Each postman covers at least one arc.

$$\sum_{j \in N} x_{1jk} \geq 1 \qquad k = 1,\dots,K \tag{3}$$

The constraint is redundant if there are no less than $K$ departing arcs from the depot or no less than $K$ arriving arcs to the depot.

4 The route of a postman departs from and arrives to a depot and there does not exist any circuit that does not reside the depot.

$$\sum_{(i,j) \in S} x_{ijk} - |S| + 1 \leq M \sum_{(i,j) \in SN} x_{ijk} \qquad \forall S, \qquad k = 1,\dots,K \tag{4}$$

where

| | |
|---|---|
| $S$ | the set of all subtours (subset of arcs that do not reside the depot) |
| $SN$ | the set of all arcs that are incident to Set $S$ |
| $M$ | an upper bound on the total flow value of any subset. |

The constraints in Set (4) are refered to as subtour elimination constraints.

5 The non-negativity and integrality of each flow variable, $x_{ijk}$, are stated below.

$$x_{ijk} \geq 0 \textit{ and integer} \quad \forall(i,j) \in A, k = 1,\dots,K \tag{5}$$

Our objective function is to minimise the sum of the squared costs and is expressed as

$$\text{Minimise}\sum_{k=1}^{K}\left(\sum_{(i,j)\in A} c_{ij}x_{ijk}\right)^2 \equiv \text{Minimise}\sum_{k=1}^{K} w_k^2 \tag{6}$$

Our problem is minimising (6) subject to the constraint sets (1) through (5). We hereafter refer to the problem as (*P*), and constraint sets (1), (2), (3), (5) as $x \in X$. We restate (*P*) in a compact form as follows:

$$(P) \qquad \text{Minimise} \quad \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

Constraint Set(4).

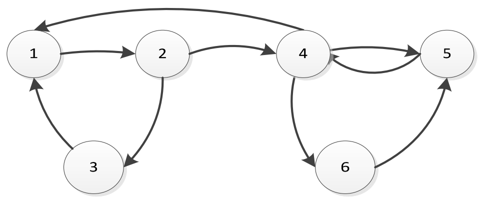
Note that (*P*) is a pure integer nonlinear programming model. Its complexity stems from the integrality requirements on the decision variables and subtour elimination constraints that are defined for each subtour and postman. There are exponential number of subtour elimination constraints as there are exponential number of subtour choices.

## 4 Heuristic subtour elimination constraints

The main difficulty of the model stems from the subtour elimination constraint set that requires enumeration of all possible subtours and a reasonable value for big M. We incorporate the subtour elimination constraints in a restricted way, by eliminating the ones having total flows above a specified value. Recall that a subtour of postman k is a circuit covered by postman *k* that does not reside the depot node, i.e., node 1.

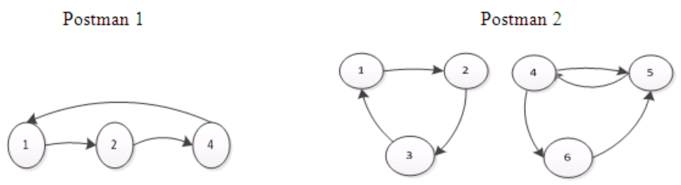
We illustrate the subtours for the graph given in Figure 1.

**Figure 1** An example directed graph *G*



Let $K = 2$ and Figure 2 illustrates the postmen assignments.

**Figure 2** A solution to the example problem



$x_{121} = x_{241} = x_{411} = 1$
and all other $x_{ij1} = 0$

$x_{122} = x_{232} = x_{312} = 1,$
$x_{542} = 2, x_{452} = x_{462} = x_{652} = 1$
and all other $x_{ij2} = 0$

The solution in Figure 2 satisfies all constraints of the k-CPP except that, postman 2 has no connection to the depot; hence, resides a subtour.

Postman 1 has a single circuit 12→24→41. The circuit is not a subtour, as it resides node 1.

Postman 2 has two circuits: 12→23→31, 45→54→46→65→54. Circuit 12→23→31 is not a subtour as it resides node 1. Circuit 45→54→46→65→54 does not reside node 1; hence, it is a subtour with a flow value of

$$FV(S) = x_{542} + x_{452} + x_{462} + x_{652} = 2 + 1 + 1 + 1 = 5.$$

To prevent the creation of any specific subtour, we introduce the following subtour elimination constraint set.

$$\sum_{(i,j)\in S} x_{ijk} - FV(S) \leq \sum_{(i,j)\in SN} x_{ijk} - 1 \qquad \forall\, S, \qquad k = 1,\ldots,K \tag{6}$$

where

| | |
|---|---|
| $S$ | a specified subtour of arcs that does not reside an arc incident to node 1 |
| $SN$ | the set of all arcs that are incident to Set $S$ |
| $FV(S)$ | the total flow value of Set $S$. |

Note that Constraint Set (6) neither requires an extra binary variable nor uses a big M value and it avoids a specified subtour $S$ with a flow value of $FV(S)$ or higher. The constraint set states that if $\sum_{(i,j)\in S} x_{ijk} \geq FV(S)$ then $\sum_{(i,j)\in SN} x_{ijk} \geq 1$, i.e., if the total flow value of Set $S$ is $FV(S)$ or higher then there is an outflow from Set $S$. However, the resulting solution may not be optimal as in any optimal solution the total flow value of Set $S$ may be higher than $FV(S)$.

We illustrate Constraint Set (6) using the solution depicted in Figure 2. The solution resides subtour 45→54→46→65→54 for Postman 2. The total flow value of the associated set is

$$FV(S) = x_{542} + x_{452} + x_{462} + x_{652} = 2 + 1 + 1 + 1 = 5.$$

The set arcs that are incident to Set $S$ is $SN$ = {41, 24}. Constraint Set (6) avoids the creation of the subtour 45→54→46→65→54 as follows:

$$\sum_{(i,j)\in S} x_{ijk} - 5 \leq \sum_{(i,j)\in SN} x_{ijk} - 1 \qquad S = \{45, 54, 46, 65\} \quad SN - \{41, 24\} \quad k = 1, 2$$

Moreover, it prevents the creation of all solutions where $FV(S) > 5$; hence, the optimal solution might be missed. That's why we refer to Constraint Set (6) as heuristic subtour elimination constraint set.

Specifically, we use the below constraints to avoid subtour 45→54→46→65→54.

$$x_{541} + x_{451} + x_{461} + x_{651} - 5 \leq x_{411} + x_{241} - 1$$

$$x_{542} + x_{452} + x_{462} + x_{652} - 5 \leq x_{412} + x_{242} - 1$$

We call a subtour as aggregated if it resides a number of subtours in an added form. Assume $R$ is the set of subtours that appear in any solution. In place of adding each heuristic subtour constraint separately, one may prefer the following heuristic aggregated constraint en route to obtaining a quicker solution however with no guarantee on the individual subtour eliminations.

$$\sum_{S\in R}\sum_{(i,j)\in S} x_{ijk} - FV(S) \le \sum_{S\in R}\sum_{(i,j)\in SN} x_{ijk} - 1 \qquad k = 1,\dots,K \tag{7}$$

An optimal solution to the $Min\sum_{k=1}^{K} w_k^2$ subject to $x \in X$ problem is a lower bound on ($P$). If the resulting solution resides no subtour, it is optimal for ($P$).

Let $Z_1$ be the optimal objective function value of the following problem.

$$\text{Minimise}\sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

Constraint Set (7)

If the resulting solution includes no subtours then $Z_1$ is an upper bound on the optimal objective function value.

Let $Z_2$ be the optimal objective function value of the following problem.

$$\text{Minimise}\sum_{k=1}^{K} w_k^2$$

Subject to $x\in X$

$$\sum_{(i,j)\in S} x_{ijk} - FV(S) \le \sum_{(i,j)\in SN} x_{ijk} - 1 \qquad \forall S \subseteq R, k = 1,\dots,K$$

If the resulting solution includes no subtours then $Z_2$ is an upper bound on the optimal objective function value.

$Z_1$ is likely to produce better estimate than $Z_2$ however at the expense of higher computational effort.

To detect the subtours that appear in the optimal solutions of the relaxed models, we use Hierholzer's algorithm (1873). For the sake of completeness, we state the steps of the algorithm.

*Step 0.* Initialise $i \leftarrow 1, s \leftarrow 0, S \leftarrow \varnothing, C_0 \leftarrow \varnothing$

Starting with node 1, construct a circuit $C_1$ such that the end of an arc is the beginning of the following arc. If $C_1$ contains all arcs on $G$ then stop else mark the arcs in $C_1$.

$$i \leftarrow i + 1$$

*Step 1.* Choose a node included in the unmarked arcs and construct a new circuit.#

*Step 2.* If there is a common node between the constructed circuit and $C_{i-1}$, insert the constructed circuit into $C_{i-1}$ by appending to that node. The resulting circuit is $C_i$.

If $S \neq \emptyset$ and there is a common node between $S$ and $C_i$, insert as many as subtours as possible to the $C_i$ by appending it to that node. The resulting circuit is $C_i$.

Remove the inserted subtours from the set $S$.

If there is no common node between $S$ and $C_i$, it means the previous subtours remain the same.

*Step 3.* If there is no common node between the constructed circuit and $C_{i-1}$, then there is a subtour.

If $S = \emptyset$, add it to the set $S$.

If $S \neq \emptyset$, look for a common node in the previous subtours and new subtour.

If there exists such a node then insert it into the previous subtours and update set $S$, else add the new tour to set $S$.

*Step 4.* If $C_i \cup S$ contains all arcs on $G$, stop, the subtours are found.

Else, mark the arcs used in $C_i \cup S$.

$i \leftarrow i + 1$

Go to Step 1.

## 5 The exact solution algorithm

The exact solution algorithm solves the integer model by relaxing all subtour elimination constraints. If the resulting solution resides no subtours, then it is optimal. If there exists at least one subtour for any postman, then the optimal objective function value of the relaxed model is a lower bound on the optimal cost. In such a case, the algorithm adds subtour elimination constraint set (4) to all generated subtours and resolves the integer model. We determine the M value of each instance as follows:

$$M = \text{Total number of nodes} * \text{Total number of arcs}$$

If the resulting solution has no subtours then we stop, else we add the new subtours while keeping all previously generated ones. We continue until a solution with no subtour is reached. Below is the stepwise description of the algorithm.

*Step 0.* Solve $(P_0)$

$$Minimise \sum_{k=1}^{K} w_k^2$$

$$\text{subject to } x \in X$$

$t = 0$

*Step 1.* Let $S_t$ be the set of subtours generated by the optimal solution of $(P_t)$

If $S_t = \emptyset$ then the resulting solution is optimal, stop

$t = t + 1$

*Step 2.* Solve ($P_t$)

$$Minimise \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

Constraint Set (4) for all subtours in $\bigcup_{j=0}^{t-1} S_j$

Go to Step 1

## 6 Heuristic solution algorithms

We propose three heuristic solution algorithms each of which is based on the optimal solutions of the integer models that make subtour eliminations via constraint set (6) or aggregate constraint (7). As the subtour elimination constraint set (6) may eliminate more than the produced subtours, the resulting solutions may not be optimal. Below is the detailed description of our heuristic algorithms.

### *6.1 Heuristic Algorithm I*

The algorithm first solves the integer model by relaxing all subtour elimination constraints like the exact algorithm. However, it adds the heuristic subtour elimination constraint set (6) in place of the exact subtour elimination constraint set (4). The model is resolved with the new heuristic subtour elimination constraints while keeping all previously generated ones. This procedure is repeated until no subtours are found. The stepwise description of the algorithm is as follows:

*Step 0.* Solve ($P_0$)

$$Minimise \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

$t = 0$

*Step 1.* Let $S_t$ be the set of subtours generated by the optimal solution of ($P_t$)

If $S_t = \varnothing$ then the resulting solution is optimal

$t = t + 1$

*Step 2.* Solve ($P_t$)

$$Minimise \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

Constraint Set (6) for all subtours in $\bigcup_{j=0}^{t-1} S_j$

Go to Step 1

## *6.2 Heuristic Algorithm II*

The algorithm uses the idea of adding subtours to the integer model in a limited extent. It starts like Algorithm I by ignoring all subtour elimination constraints, and treats the subtours as follows:

1 selecting one of the subtours generated by the model and adding the selected subtour using subtour elimination constraint set (6)

2 aggregating all other subtours, and adding the aggregated constraint using subtour elimination constraint (7).

Hence, it adds two types of subtour elimination constraints for each postman, one original and one aggregated constraint.

Each subtour generated by the model is added as an original constraint while all others are treated as aggregated. In doing so, we obtain $R$ solutions if $R$ subtours are generated. We select the solution having the largest objective function value with the hope of reaching the solution quicker. If the selected solution resides no subtour, we stop. If there is at least one subtour then we continue to add one original subtour and one aggregated subtour and make the further selections according to the maximum lower bound rule. Below is the stepwise description of Algorithm II.

*Step 0.* Solve ($P_0$)

$$\textit{Minimise} \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

$t = 0$

$F = \varnothing$

*Step 1.* Let $S_t$ be the set of subtours generated by the optimal solution of $P_t$.

If $S_t = \varnothing$ then the resulting solution is optimal, stop.

$t = t + 1$

For each $r \in S_t$ define ($P_{t,r}$)

$$\textit{Minimise} \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

Subtour $r$ by Constraint Set (6)

Aggregated subtour by Constraint Set (7)

*Step 2* Let $z_{tr}$ be the optimal objective function value.

Select subtour $f$ such that

$$z_{tf} = \max_r \{z_{tr}\}$$
$$F = F \cup \{f\}$$

Let $S_{tf}$ be the set of subtours by the optimal solution of $P_{t,f}$

If $S_{tf} = \varnothing$ then the resulting solution is optimal, stop.

For each $r \in S_{tf}$ define ($P_{t,r}$)

$$Minimise \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

Subtours $r$ and set $F$ by Constraint Set (6)

Aggregated subtour $S_{tf}/\{r\}$ by Constraint Set (7)

Go to Step 2.

## 6.3 Heuristic Algorithm III

The algorithm is similar to Heuristic Algorithm II except that the subtour is selected randomly. In such a case, one spends relatively low computation time, however at the expense of evaluating more problems with different sets of subtour elimination constraints. Below is the stepwise description of Heuristic Algorithm III.

*Step 0.* Solve ($P_0$)

$$Minimise \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

$t = 0$

$F = \varnothing$

*Step 1.* Let $S_t$ be the set of subtours generated by $P_t$.

If St = ∅ then stop.

Let $f$ be a randomly selected subtour from $S_t$.

$t = t + 1$

$F = F \cup \{f\}$

Solve ($P_t$)

$$Minimise \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

Subtour $r \in F$ by Constraint Set (6)

Aggregated subtour $S_{tf}/\{r\}$ by Constraint Set (7)

Go to Step 1.

## 7 Computational experiment

We use eight precedence networks taken from Archetti et al. (2014). The network sizes are as tabulated below:

**Table 1** The network sizes

| *Network* | *# of nodes* | *# of arcs* | *K* | *Number of combinations* |
|---|---|---|---|---|
| d0 | 16 | 36 | 2, 3, 4 | 3 |
| v6 | 31 | 56 | 2, 3, 4 | 3 |
| d11 | 36 | 75 | 2, 3, 4 | 3 |
| p20 | 50 | 109 | 2, 3, 4 | 3 |
| d18 | 64 | 137 | 2, 3, 4 | 3 |
| r16 | 50 | 168 | 2, 3 | 2 |
| g28 | 100 | 186 | 2, 3 | 2 |
| d32 | 100 | 216 | 2, 3 | 2 |
| | | | | Total: 21 |

Note that we have 21 combinations; for each combination we generate 10 instances. Hence, we use a total of 210 instances.

The arc costs are generated from discrete uniform distribution between 1 and 100.

We code the algorithms in C# and solve the models by ILOG CPLEX 12.6. We use computer Intel(R) Core(TM)i7-4770S CPU@3.10 GHz, 16 GB RAM and Windows 7.

For each problem instance and algorithm, we set a termination limit of one hour. The integer programs used for heuristic algorithms are solved using a gap value of 0.01%.

Table 2 reports the CPU times spent by each heuristic algorithm and the mathematical model. In the table the numbers in the parentheses give the number of instances (out of 10) that could not be solved in one hour.

**Table 2** The CPU times (in seconds) of the heuristic algorithms and mathematical model

| *# of nodes* | *# of arcs* | *K* | *Heuristic Algorithm I* | | *Heuristic Algorithm II* | | *Heuristic Algorithm III* | | *Mathematical model* | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | *Aver* | *Max* | *Aver* | *Max* | *Aver* | *Max* | *Aver* | *Max* |
| 16 | 36 | 2 | 0.51 | 1.11 | 0.51 | 1.11 | 0.51 | 1.11 | 0.68 | 1.50 |
| | | 3 | 0.65 | 0.83 | 0.65 | 0.83 | 0.65 | 0.83 | 12.60 | 36.69 |
| | | 4 | 2.56 | 4.74 | 2.56 | 4.74 | 2.56 | 4.74 | 1,906.10 | 3,600(3)* |
| 31 | 56 | 2 | 0.34 | 0.66 | 0.34 | 0.66 | 0.34 | 0.66 | 1.02 | 2.00 |
| | | 3 | 1.64 | 4.77 | 1.64 | 4.77 | 1.64 | 4.77 | 9.50 | 18.33 |
| | | 4 | 13.22 | 50.90 | 10.02 | 25.85 | 13.77 | 46.75 | 2,378.84 | 3,600(6) |
| 36 | 75 | 2 | 0.36 | 1.01 | 0.36 | 1.01 | 0.36 | 1.01 | 0.87 | 1.89 |
| | | 3 | 1.41 | 1.86 | 1.41 | 1.86 | 1.41 | 1.86 | 16.80 | 58.22 |
| | | 4 | 10.99 | 20.87 | 11.94 | 26.07 | 9.04 | 22.01 | 1,661.34 | 3,600(3) |
| 50 | 109 | 2 | 0.37 | 0.86 | 0.37 | 0.86 | 0.37 | 0.86 | 1.13 | 2.56 |
| | | 3 | 3.96 | 9.02 | 3.96 | 9.02 | 3.96 | 9.02 | 53.04 | 273.36 |
| | | 4 | 11.24 | 20.53 | 14.05 | 26.93 | 10.40 | 25.29 | 2,994.43 | 3,600(8) |
| 64 | 137 | 2 | 0.78 | 2.04 | 0.78 | 2.04 | 0.78 | 2.04 | 2.42 | 4.76 |
| | | 3 | 7.12 | 17.53 | 6.97 | 17.53 | 9.73 | 18.95 | 47.98 | 107.42 |
| | | 4 | 14.52 | 32.34 | 15.59 | 34.06 | 27.33 | 95.74 | 3,298.11 | 3,600(8) |
| 50 | 168 | 2 | 0.68 | 2.04 | 0.68 | 2.04 | 0.68 | 2.04 | 6.47 | 20.76 |
| | | 3 | 28.89 | 115.49 | 19.00 | 55.65 | 19.38 | 49.53 | 143.75 | 589.82 |
| 100 | 186 | 2 | 1.03 | 2.79 | 1.03 | 2.79 | 1.03 | 2.79 | 2.77 | 4.90 |
| | | 3 | 5.96 | 11.54 | 5.96 | 11.54 | 5.96 | 11.54 | 21.02 | 61.59 |
| 100 | 216 | 2 | 0.96 | 2.79 | 0.96 | 2.79 | 0.96 | 2.79 | 8.46 | 20.14 |
| | | 3 | 7.22 | 19.78 | 22.85 | 155.10 | 21.59 | 32.67 | 824.06 | 3,600(2) |

Note: *Numbers in the parentheses give the number of instances that could not be solved in 1 hour.

Table 3 reports on the average and maximum deviation of the heuristic solutions from the optimal solutions. We include the instances that we could find an optimal solution in one hour by the mathematical model. For each included problem instance, we measure and report the percent deviation as follows:

$$((\text{Heuristic Solution} - \text{Optimal Solution}) / \text{Optimal Solution})*100$$

**Table 3** The percent deviation of the heuristic algorithms from the optimal solution

| *# of nodes* | *# of arcs* | *K* | *# of instances** | *Heuristic Algorithm I* | | *Heuristic Algorithm II* | | *Heuristic Algorithm III* | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | *Aver.* | *Max* | *Aver.* | *Max* | *Aver.* | *Max* |
| 116 | 36 | 2 | 10 | 0.0031 | 0.0087 | 0.0031 | 0.0087 | 0.0031 | 0.0087 |
| | | 3 | 10 | 0.0058 | 0.0095 | 0.0058 | 0.0095 | 0.0058 | 0.0095 |
| | | 4 | 7 | 0.0045 | 0.0088 | 0.0045 | 0.0088 | 0.0045 | 0.0088 |
| 31 | 56 | 2 | 10 | 0.0032 | 0.0094 | 0.0032 | 0.0094 | 0.0032 | 0.0094 |
| | | 3 | 10 | 0.0053 | 0.0090 | 0.0053 | 0.0090 | 0.0053 | 0.0090 |
| | | 4 | 4 | 0.0063 | 0.0082 | 0.0064 | 0.0085 | 0.0061 | 0.0093 |
| 36 | 75 | 2 | 10 | 0.0030 | 0.0097 | 0.0030 | 0.0097 | 0.0030 | 0.0097 |
| | | 3 | 10 | 0.0042 | 0.0100 | 0.0042 | 0.0100 | 0.0042 | 0.0100 |
| | | 4 | 7 | 0.0060 | 0.0096 | 0.0060 | 0.0096 | 0.0034 | 0.0048 |
| 50 | 109 | 2 | 10 | 0.0029 | 0.0094 | 0.0029 | 0.0094 | 0.0029 | 0.0094 |
| | | 3 | 10 | 0.0044 | 0.0098 | 0.0044 | 0.0098 | 0.0044 | 0.0098 |
| | | 4 | 2 | 0.0073 | 0.0089 | 0.0073 | 0.0089 | 0.0053 | 0.0057 |
| 64 | 137 | 2 | 10 | 0.0024 | 0.0087 | 0.0024 | 0.0087 | 0.0024 | 0.0087 |
| | | 3 | 10 | 0.0054 | 0.0098 | 0.0045 | 0.0094 | 0.0044 | 0.0094 |
| | | 4 | 2 | 0.0044 | 0.0050 | 0.0047 | 0.0079 | 0.0074 | 0.0077 |
| 50 | 168 | 2 | 10 | 0.0027 | 0.0070 | 0.0027 | 0.0070 | 0.0027 | 0.0070 |
| | | 3 | 10 | 0.0041 | 0.0076 | 0.0048 | 0.0099 | 0.0052 | 0.0088 |
| 100 | 186 | 2 | 10 | 0.0040 | 0.0094 | 0.0040 | 0.0094 | 0.0040 | 0.0094 |
| | | 3 | 10 | 0.0053 | 0.0093 | 0.0053 | 0.0093 | 0.0053 | 0.0093 |
| 100 | 216 | 2 | 10 | 0.0032 | 0.0091 | 0.0032 | 0.0091 | 0.0032 | 0.0091 |
| | | 3 | 8 | 0.0034 | 0.0082 | 0.0035 | 0.0082 | 0.0026 | 0.0062 |

Note: *Number of instances with known optimal solutions and used for heuristic deviations.

Table 3 shows the excellent performance of the heuristic algorithms over all problem set. Note that all deviations from the optimal solutions are less than 0.01%.

Table 4 gives the average number of iterations and the number of subtours used for each algorithm. If the number of subtours in each iteration is 0, 1 or 2, three heuristic solution algorithms become same and generate same results. Their differences are resulted from the selection of the subtours which will be added to the mathematical model.

**Table 4** The average number of subtours and iterations by the heuristic algorithms

| *# of nodes* | *# of arcs* | *K* | *Heuristic Algorithm I* | | *Heuristic Algorithm II* | | *Heuristic Algorithm III* | | *Exact algorithm* | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | *Subtour* | *Iteration* | *Subtour* | *Iteration* | *Subtour* | *Iteration* | *Subtour* | *Iteration* |
| 16 | 36 | 2 | 0.00 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| | | 3 | 0.30 | 1.10 | 0.30 | 1.10 | 0.30 | 1.10 | 0.90 | 1.30 |
| | | 4 | 4.20 | 2.10 | 4.20 | 2.10 | 4.20 | 2.10 | 2.29 | 1.57 |
| 31 | 56 | 2 | 0.00 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| | | 3 | 2.10 | 1.80 | 2.10 | 1.80 | 2.10 | 1.80 | 3.90 | 2.00 |
| | | 4 | 26.40 | 6.20 | 16.00 | 4.80 | 22.80 | 6.30 | 12.80 | 4.00 |
| 36 | 75 | 2 | 0.40 | 1.20 | 0.40 | 1.20 | 0.40 | 1.20 | 0.00 | 1.00 |
| | | 3 | 1.20 | 1.40 | 1.20 | 1.40 | 1.20 | 1.40 | 3.00 | 1.90 |
| | | 4 | 17.20 | 4.40 | 12.40 | 3.90 | 11.20 | 3.40 | 6.29 | 1.86 |
| 50 | 109 | 2 | 0.20 | 1.10 | 0.20 | 1.20 | 0.20 | 1.20 | 0.00 | 1.00 |
| | | 3 | 3.90 | 2.20 | 3.90 | 4.90 | 3.90 | 4.90 | 2.70 | 1.70 |
| | | 4 | 14.00 | 3.20 | 9.20 | 3.20 | 7.60 | 2.80 | 0.00 | 1.00 |
| 64 | 137 | 2 | 0.80 | 1.40 | 0.80 | 1.40 | 0.80 | 1.40 | 0.20 | 1.10 |
| | | 3 | 12.00 | 4.00 | 8.70 | 3.70 | 10.80 | 4.30 | 6.40 | 3.10 |
| | | 4 | 26.00 | 5.10 | 14.40 | 4.40 | 21.60 | 5.80 | 10.00 | 2.00 |
| 50 | 168 | 2 | 1.20 | 1.30 | 1.20 | 1.30 | 1.20 | 1.30 | 3.00 | 2.50 |
| | | 3 | 70.50 | 16.40 | 23.40 | 8.60 | 28.20 | 9.90 | 27.30 | 6.40 |
| 100 | 186 | 2 | 0.00 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| | | 3 | 1.20 | 1.30 | 1.20 | 1.30 | 1.20 | 1.30 | 0.30 | 1.10 |
| 100 | 216 | 2 | 1.60 | 1.50 | 1.60 | 1.50 | 1.60 | 1.50 | 2.80 | 1.90 |
| | | 3 | 17.10 | 3.70 | 12.60 | 3.80 | 22.50 | 6.50 | 25.16 | 5.13 |

We observe from the tables that the number of postmen, *K*, plays a dominant role on the performance of the algorithms. As *K* increases the effort spent to solve the integer nonlinear models increase considerably. Both exact and heuristic algorithms could solve few instances of large networks in one hour when *K* = 3 and 4. Note that the effect of *K* becomes more significant as the number of arcs increases.

The number of arcs also significantly affects the performance of the algorithms, as our decision variables are defined on the arcs. Moreover, the network structure is effective in defining the complexity. When the number of arcs coming to and leaving from the nodes increases, more subtours are generated. The number of subtours generated per iteration affects the performance of the algorithm, significantly. Table 2 and Table 4 show that adding more subtours in any iteration makes all the *k*-CPP models harder to solve. Note that when there are 216 arcs and 3 postmen, all algorithms run in reasonable times. The CPU times are 7.22, 22.85 and 21.59 seconds for Heuristic Algorithms I, II and III, respectively.

Based on the discussion on the number of subtours, there is also a connection between the effect of increasing $K$ on the CPU times and number of subtours generated in each iteration. Table 3 shows that the average number of subtours generally increases as $K$ increases. For example, all instances have a smaller number of subtours when there are 2 postmen, and the associated CPU times are relatively short compared to the cases with more postmen. Note that for the same example when there are 168 arcs, the average number of subtours is 70.5 for Algorithm I, 23.4 for Algorithm II, and 28.2 for Algorithm III for $K = 3$. As $K$ increases, more subtours are generated per iteration and the problem becomes more difficult with these subtours. We observe that the exact algorithm could not return any solution in one hour when there are 5 postmen. When there are 4 postmen the exact algorithm could not return an optimal solution for the instances with 168 or more nodes.

The tables also reveal the superiority of Algorithm I in terms of the solution times. Note from Table 2 that the smallest CPU times are due to Algorithm I. This is because considering all subtours simultaneously reduces the number of iterations hence the CPU times.

Algorithm II uses two subtours at a time: one original subtour and one aggregated subtour. It makes more precise evaluation of the subtours when compared to Algorithm III, thereby leading to less iterations. The high CPU times of Algorithm II are due to fact that in each iteration subtour selection is done via an integer program, whereas Algorithm III selects the subtours randomly. Algorithm II makes less iterations however at the expense of higher CPU times. Note from Table 4 that when there are 216 arcs and 3 postmen, Algorithm II adds 12.6 subtours in 3.8 iterations; and Algorithm III adds 22.5 subtours in 6.5 iterations, on average.

## 8 Conclusions

In this study, we consider a directed *k*-CPP with the objective of minimising the total squared costs. Our aim is to balance the costs of the postmen while maintaining the total cost as small as possible. We are unaware of a previous study for the *k*-CPP which addresses our balancing concern.

We develop a pure integer nonlinear programming model and discuss its complexity. We develop three heuristic solution algorithms that make efficient use of the proposed heuristic subtour elimination constraints. The results of our extensive computational study have revealed that the algorithms return solutions that are very close to the optimal ones and can solve instances with up to about 150 arcs with 4 postmen and 200 arcs with 3 postmen in one hour. For larger-sized instances, one may use our subtour elimination constraints in local search algorithms, possibly combined with metaheuristic approaches.

## Acknowledgements

We thank the referees for their helpful and constructive comments.

## References

Ahr, D. (2004) *Contributions to Multiple Postmen Problems*, PhD thesis, University of Heidelberg, Germany.

Ahr, D. and Reinelt, G. (2002) 'New heuristics and lower bounds for the min-max k-Chinese postman problem', in Möhring, R. and Raman, R. (Eds.): *Lecture Notes in Computer Science*, Vol. 2461, pp.64–74, Algorithms–ESA 2002, Springer.

Ahr, D. and Reinelt, G. (2006) 'A Tabu search algorithm for the min-max k-Chinese postman problem', *Computers and Operations Research*, Vol. 33, No. 12, pp.3403–3422.

Archetti, C., Guastaroba, G. and Speranza, M.G. (2014) 'An ILP-Refined Tabu search for the directed profitable rural postman problem', *Discrete Applied Mathematics*, Vol. 163, No. 1, pp.3–16.

Benavent, E., Corberán, Á. and Sanchis, J.M. (2010) 'An heuristic algorithm for the min-max k-vehicles windy rural postman problem', *Computers and Management Science*, Vol. 7, No. 3, pp.269–287.

Benavent, E., Corberán, Á., Desaulniers, G., Lessard, F., Plana, I. and Sanchis, J.M. (2014) 'A branch-price-and-cut algorithm for the min-max k-vehicle windy rural postman problem', *Networks*, Vol. 63, No. 1, pp.34–45.

Benavent, E., Corberán, Á., Plana, I. and Sanchis, J.M. (2009) 'Min-Max k-vehicles windy rural postman problem', *Networks*, Vol. 54, No. 4, pp.216–226.

Benavent, E., Corberán, Á., Plana, I. and Sanchis, J.M. (2011) 'New facets and an enhanced branch-and-cut for the min-max k vehicles windy rural postman problem', *Networks*, Vol. 58, No. 4, pp.255–272.

Cabral, E., Gendreau, M., Ghiani, G. and Laporte, G. (2004) 'Solving the hierarchical chinese postman problem as a rural postman problem', *European Journal of Operational Research*, Vol. 155, No. 2, pp.44–50.

Corberán, A., Martí, R. and Sanchis, J.M. (2002) 'A GRASP procedure for the mixed Chinese postman problem', *European Journal of Operational Research*, Vol. 142, No. 1, pp.70–80.

Dror, M. (2000) *Arc Routing: Theory, Solutions and Applications*, Kluwer Academic Publishers, Boston.

Edmonds, J. and Johnson, E.L. (1973) 'Matching, Euler tours and the Chinese postman', *Mathematical Programming*, Vol. 5, No. 1, pp.88–124.

Eiselt, H.A., Gendreau, M. and Laporte, G. (1995a) 'Arc routing problems, part 1: the Chinese postman problem', *Operations Research*, Vol. 43, No. 2, pp.231–242.

Eiselt, H.A., Gendreau, M. and Laporte, G. (1995b) 'Arc routing problems, part 2: the rural postman problem', *Operations Research*, Vol. 43, No. 3, pp.399–414.

Euler, L. (1736) 'Solutio Problematis ad Geometrian Situs Pertinentis', *Commentarii Academiae Scientarum Petropolitanae*, Vol. 8, No. 1, pp.128–140.

Ford, L.R. and Fulkerson, D.R. (1962) *Flows in Networks*, Princeton University Press, Princeton, N.J.

Frederickson, G., Hecht, M. and Kim, C. (1978) 'Approximation algorithms for some routing problems', *SIAM Journal of Computing*, Vol. 7, No. 2, pp.178–193.

Guan, M.K. (1962) 'Graphic programming using odd or even points', *Chinese Mathematics*, Vol. 1, No. 1, pp.273–277.

Hierholzer, C. (1873) 'Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren', *Mathematische Annalen*, Vol. 6, No. 1, pp.30–32.

Kappauf, H.C. and Koehler, G.J. (1979) 'The mixed postman problem', *Discrete Applied Mathematics*, Vol. 1, No. 1, pp.89–103.

Lin, Y. and Zhao, Y. (1988) 'A new algorithm for the directed Chinese postman problem', *Computers and Operations Research*, Vol. 15, No. 6, pp.577–584.

Malandraki, C. and Daskin, M. (1993) 'The maximum benefit chinese postman problem and the maximum benefit traveling salesman problem', *European Journal of Operational Research*, Vol. 65, No. 3, pp.218–234.

Minieka, E. (1979) 'The Chinese postman problem for mixed networks', *Management Science*, Vol. 25, No. 7, pp.643–648.

Osterhues, A. and Mariak, F. (2005) *On Variants of the k-Chinese Postman Problem*, p.30, Operations Research and Wirtschaftsinformatik, University of Dortmund, Germany.

Pearn, W.L. and Chou, J.B. (1999) 'Improved solutions for the Chinese postman problem on mixed networks', *Computers and Operations Research*, Vol. 26, No. 8, pp.819–827.

Pearn, W.L. and Liu, C.M. (1995) 'Algorithms for the Chinese postman problem on mixed networks', *Computers and Operations Research*, Vol. 22, No. 5, pp.479–489.

Ralphs, T.K. (1993) 'On the mixed Chinese postman problem', *Operations Research Letters*, Vol. 14, No. 3, pp.123–127.

Shafahi, A. and Haghani, A. (2015) 'Generalized maximum benefit multiple Chinese postman problem', *Transportation Research Part C: Emerging Technologies*, Vol. 55, No. 1, pp.261–272.

Willemse, E.J. and Joubert, J.W. (2012) 'Applying min–max k postmen problems to the routing of security guards', *Journal of Operational Research Society*, Vol. 63, No. 2, pp.245–260.