Central European Journal of Operations Research (2020) 28:337–366
https://doi.org/10.1007/s10100-018-0598-8

ORIGINAL PAPER

A time-dependent hierarchical Chinese postman problem

Merve Kayacı Çodur[1] · **Mustafa Yılmaz**[1]

Published online: 13 December 2018

Abstract
The hierarchical Chinese postman problem (HCPP), a variant of the Chinese postman problem, is an arc routing problem. HCPP is NP-hard and several methods have been developed to solve this problem. Many studies in the literature have taken the minimum distance covered between nodes into consideration. All of these studies ignore time-dependent travel speeds between two locations. Travel speeds (and time) in almost all metropolitan areas change drastically during the day due to a variety of different factors, such as weather condition, peak traffic hours and accidents, along with the distance. Spending minimum time to travel streets is of great importance to ensure traffic flow and road safety, particularly in many practical implementation areas of HCPP, such as snow plowing winter, garbage collection and routing security patrol vehicles. This study introduces a new problem called the time-dependent hierarchical Chinese postman problem that aims to minimize the total travel time, while obeying precedence relationships between edges. This problem differs from most arc routing problems because the duration of traversing a street changes depending on the time of day. The problem was formulated as a mixed integer linear programme. Two metaheuristics were built: a genetic algorithm (GA) and a hybrid simulated annealing (hSA). The proposed model and metaheuristics were tested on modified benchmark instances and randomly generated problem instances with as many as 490 edges. The performance of GA and hSA were compared in terms of solution quality and computational time.

Keywords Arc routing problems · Hierarchical Chinese postman problem · Time dependent routing · Genetic algorithm · Simulated annealing

✉ Merve Kayacı Çodur
mkayaci@atauni.edu.tr

Mustafa Yılmaz
mustafay@atauni.edu.tr

[1] Department of Industrial Engineering, Faculty of Engineering, Atatürk University, 25240 Erzurum, Turkey

1 Introduction

The hierarchical Chinese postman problem (HCPP) is a variant of the Chinese postman problem (CPP) with the edges of a graph partitioned into priority classes by a precedence relation that is defined in clusters. Thus, the problem is to determine the shortest tour that starts from and ends at a depot obeying the hierarchy mentioned, or the precedence relationship between clusters. Formally, HCPP is defined on an undirected graph $G(V, E)$, where V is the set of nodes, E is the set of edges and $c(i, j) \geq 0$ is the distance of traversing the edge $(i, j) \in E$. If the precedence relationship is linear, the edge set is partitioned into h classes, and each subnetwork represents a hierarchical level. That is, $G(V, E) = G_1(V_1, E_1) \cup G_2(V_2, E_2) \cup \cdots \cup G_h(V_h, E_h)$, $E_i \cap E_j = \emptyset$, $i \neq j$ and no edge $(i, j) \in E_h$ can be traversed before the traversal of all edges in E_{h-1} that has been completed. This variant of HCPP has been used in real life applications, such as snow removal operations where streets have different priority levels, street sweeping, garbage collection and routing of security control vehicles etc.

Lemieux and Campagna (1984) developed a heuristic method for the snow plowing problem where streets were classified as major and secondary. Alfa and Liu (1988) proposed a technique to obtain a good solution for the type of the problem in which a group of arcs must be traversed prior to another group in a directed, hierarchical network. In this problem, arcs of the same hierarchy are grouped together and the CPP is applied to each group to obtain the optimal routing strategy for that hierarchy. Furthermore, as the CPP cannot be applied unless all arcs in a hierarchy are connected, they developed a heuristic method that selected an arc from another hierarchy to establish a connection.

HCPP was firstly defined by Dror et al. (1987). The authors developed an algorithm $O(kn^5)$ that found the optimum postman tour for a network where roads were divided according to priority relations. They indicated that the HCPP is an NP-hard in general, but under the sum criteria is polynomial time solvable if the precedence relation, or ordering, is linear, and each cluster is connected. This algorithm did not give results unless the above-mentioned conditions were met. Another heuristic method combining subnetworks with no interconnection was presented by Damodaran (1997). This proposed heuristic was found to have better performance than the method developed by Alfa and Liu (1988).

Ghiani and Improta (2000) defined an algorithm that provided an exact solution for the first time. They presented a lower complexity computational algorithm that solved the problem of matching an auxiliary graph with $O(k^3n^3)$. Cabral et al. (2004) presented a transformation of the HCPP into an equivalent rural postman problem. An optimal solution was obtained by applying an exact branch-and-cut algorithm to the transformed problem. Korteweg and Volgenant (2006) described an alternative algorithm that minimized the costs of the first priority class, then the costs of the second class, etc. The hierarchical network example used in this study is given in Fig. 1. In general, the CPP tour solution is a known lower bound for the HCPP. Damodaran et al. (2008) proposed a heuristic method that improved the lower bounds for the HCPP. Better lower bounds allow an optimal solution to be found in a shorter computational

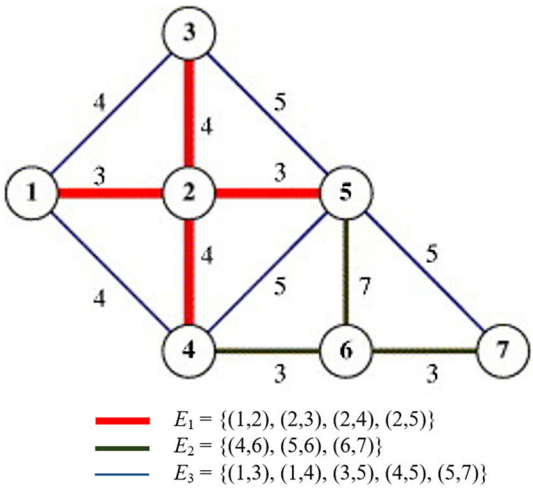


Fig. 1 Hierarchical network example

time. Sayata and Desai (2015) gave an alternative approach using Kruskal's method to reduce the number of edges in the graph that has a time complexity of $O(k^2n^2)$, where k is the number of layers in the graph and n is the number of nodes. The results were compared to the simple HCPP instance, where the number of hierarchical levels is less than 3, and the number of nodes is less than 20.

Although it is encountered frequently in real life, there are a limited number of studies in published literature about the HCPP. Furthermore, there is still a lack of modelling approaches that represent real life conditions more closely. A practical issue that is rarely dealt with in studies is the notion of time dependency, where travel time between nodes depends on the time of the day. Almost all current studies take the cost to be a fixed value of distance within the objective function, without taking travel time into consideration. Unfortunately, this assumption is a weak approach for representing real world conditions.

In recent years, availability of data for travel time or travel speed at different times of the day has led researchers to develop different routing models using this additional information. Modeling of time-dependent travel time has become an important issue in time-dependent routing literature. In a time-dependent routing problem, travel time in the network depends on the time of the day. In addition to this, travel time between two locations is changed over the distance covered along with different factors, such as weather condition, traffic congestion and accidents. The purpose of this problem is generally to minimize the total travel time.

Although the time-dependent vehicle routing problem and the time-dependent traveling salesman problem have been widely studied in the literature (Malandraki 1989; Malandraki and Daskin 1992; Schneider 2002; Helvig et al. 2003; Albiach et al. 2008; Maden et al. 2009; Harwood et al. 2013; Cordeau et al. 2014), time-dependent arc routing problems have rarely been addressed.

Tagmouti et al. (2007) studied an arc routing problem with capacity constraints and time-dependent service costs adapted to winter gritting applications. They proposed an exact problem solving approach that transformed the arc routing problem into an equivalent node routing problem, and the problem was then solved through a column generation approach. Tagmouti et al. (2010) developed a new variant of the capacitated arc routing problem in which a time-dependent piecewise linear service cost was considered. This problem arises again in winter gritting where, if the service of an arc is too early or too late, there is a cost in time and the authors adapted a variable neighborhood heuristic method. A dynamic variant of the capacitated arc routing problem is tackled by Tagmouti et al. (2011) using a Variable Neighborhood Descent heuristic. Black et al. (2013) introduced a new problem called the time-dependent prize-collecting arc routing problem that arises whenever one has to choose between a number of full truckload pick-ups and deliveries on a road network where travel time change over the day. For this problem, two heuristic methods, a variable neighborhood search and a metaheuristic tabu search were proposed and tested using a set of benchmark problems. In time-dependent Chinese postman problems, Fan et al. (2002) focused on a directed CPP model and then they presented this problem with both crisp and fuzzy time window constraint. The first integer programming formulation modeling the time varying CPP directly was presented by them. A new integer programming formulation for solving all the general instances of the Chinese postman problem with time dependent travel time (CPPTDT) was proposed by Sun et al. (2011). Sun et al. (2011) proved that the Chinese Postman Problem defined on the time dependent network is NP-hard and they proposed a dynamic programming algorithm for the problem with the First-In-First-Out property. For the CPPTDT, a linear integer programming formulation, namely, the cycle-path formulation, which can be viewed as an extended version of the formulation given by Fan et al. (2002) was presented by Sun et al. (2015).

To the best of our knowledge, time-dependent travel time (and travel speed) has not been addressed in HCPP literature. To consider more realistic representations of real life problems, the time-dependent hierarchical Chinese postman problem (TD-HCPP), a new variant of HCPP, is proposed in this study.

TD-HCPP considers both hierarchical rules and changeable travel time in a graph, where the cost of servicing an edge depends on the time. The objective is to find a minimum travel time route where every edge must be traveled at least once under precedence constraints. This problem can be stated as follows: assume that a vehicle or postman (usually vehicle is accepted due to the sizes of networks) is available to travel to edges with the time period, or interval, partitioned into p time intervals of T_1, T_2, *and* Tp. Given a network $G(V, E)$ of h hierarchy, a symmetric distance matrix $c(i, j)$, a $h \times p$ travel speed matrix $V_{Tk} = (v_{ijTk})$ contains the travel speeds for all edges (i, j) when the vehicle departs from node i within the time interval T_k, where $k = 1, 2, \ldots, p$. The problem considered here is motivated by a service application encountered in local snow plowing operations, where roads are cleaned beginning with the busier streets then continuing with the less busy ones. In this case, snow covered roads should be cleaned in an appropriate time period to avoid traffic congestion and ensure safe transportation. For all these reasons, the addition of time dependent travel time to the basic HCPP is a significant step to model and solve realistic routing problems.

This paper is organized as follows: Sect. 1 gives comprehensive information in literature about the handled problem. Section 2 includes the problem definition, and Sect. 3 defines the TD-HCPP mathematical model. Section 4 presents the solution approaches that would be used to solve the problem, since it is NP-hard and describes the proposed GA heuristic. Section 5 provides technical details of the proposed hSA heuristic and its implementation. Section 6 reports the computational results for the main problems presented in published literature and others generated randomly with regard to the HCPP. Finally, Sect. 7 shows the conclusions for this study and proposals for potential future studies.

2 Problem definition

2.1 A time-dependent travel speed model

In time-dependent routing problems, the travel time of a vehicle is dependent on the time of the day, which means it varies based on when the travel takes place. The travel time between two nodes in a specific moment can be determined using a speed model. In this section, the speed model used in this paper, necessary data and assumptions will be discussed.

Malandraki (1989); Malandraki and Daskin (1992); Hill and Benton (1992) defined one of the first approaches to the discrete travel time function that describes the relationship between the time of day and the travel time. This technique was used in all studies prior to a study conducted by Ichoua et al. (2003). The major weakness of this technique was that the defined travel time between two nodes depended on the starting time of each interval and it was constant throughout that interval. As the vehicle's speed is not fixed, considering the constant travel time between two points in a city where the traffic is congested is far from realistic. This is because the changing density of traffic flow in a day leads to changes in the vehicle's speed, which, in turn, results in changes in travel time. Time-dependent speed function, an approach that is quite close to real life, was introduced by Ichoua et al. (2003). They showed that travel time between two nodes is not only dependent on the vehicle's departure time from a node, but also on its speed during the time interval. This approach to calculating the travel time preserves the first-in-first-out (FIFO) principle. This speed model is applicable to many real-world applications.

The speed model used for TD-HCPP is based on the speed model of Ichoua et al. (2003) and Verbeeck et al. (2014). Ichoua's speed model, the arcs were divided into three different categories randomly. But it would not be an appropriate approach to categorizing the edges by random assignment because of the hierarchical classification of TD-HCPP. Verbeeck et al. presented a speed model that different arc categories and different time intervals with non-equal width were used. This speed model was adapted to the our problem in this present study. More importantly, during the construction of the problem instances, the edges are assigned to an arc category taking the hierarchical relationship between them into account.

In our speed model in this study, the travel speed, and therefore the travel time of a vehicle, is indexed by the time intervals and arc categories. A speed v_{ijk} is defined

Table 1 Speed matrix

Arc categories	Time periods			
	Morning peak 7 am–9 am	Normal 9 am–5 pm	Evening peak 5 pm–7 pm	Normal 7 pm–7 am
First priority	0.5	0.81	0.5	0.81
Secondary priorities	0.5	0.7	1	1.5
	0.5	1.5	0.5	1.5
	1	1.5	0.5	0.7
Last priority	1.5	1.5	1.5	1.5

for every time interval $k \in \{T_1, T_2, T_3, \ldots T_K\}$ and arc category C_{arc}. Table 1 shows the 5×4 speed matrix used, where each row corresponds to a category of arcs, and each column is divided into one of four discrete time periods so that $C_{arc} = 5$ and $k = 4$. Four unequal time periods indicate that there is heavy traffic in the morning and evening, and normal traffic for the remaining time. As operations such as snow plowing can be carried out 24 h a day, there is no restriction on time intervals. For this, it is assumed that the vehicle leaves the depot after 7 a.m. and returns back when the operation is finished.

Roads are categorized based on the hierarchical relationship between them. Thereafter, edges were assigned to one of the arc categories, again based on hierarchical rules. There are three main rules that were used to associate hierarchical edges with arc categories defined by Verbeeck et al. (2014):

- First, priority roads—mostly located in city centers—always have heavy traffic. Therefore, first-priority roads are assigned to the 'always busy' edge category that is in the first row.
- Roads in the last hierarchical level represent roads in rural and less traveled areas. As these roads were rarely used, there is little traffic if any, and the speed values defined in the speed model are higher. Therefore, roads belonging to the last hierarchical level were added to the 'seldom traveled' arc category that is the last row.
- The secondary hierarchical roads (E_m, $m = 2, \ldots, H-1$) were randomly assigned to 'morning peak', 'evening peak' or to 'two peak categories that represent both morning and evening peaks'. To clarify: These roads can represent morning density, evening density or average density.

In brief, the number of hierarchical classes is not fixed every time and can vary according to the type of problem. For this reason, a number of arc categories equal to the number of total hierarchies defined for the $G(V, E)$ network are taken into

consideration. For each problem type, first- and last-priority roads are respectively assigned to the first and last rows.

To calculate the travel time between the i and j nodes, the data for distance matrix between these two nodes (c_{ij}), the travel speed matrix (v_{ijk}) and the departure time (t_i) from node i are required. The departure time determines the time period and together with the arc category, the starting speed (v_{ijk}) can be looked up in the speed matrix. During routing, the vehicle starts from a time period and continues to travel from one time period to another. Changing roads and time intervals allows speed to change frequently. Calculation of the travel time between two nodes is directly associated with the distance and speed between the two nodes. This procedure was explained by Ichoua et al. (2003) as given Algorithm 1. A vehicle leaves node i at $t_0 \in [\underline{t_k}, \overline{t_k}]$, and that arc (i, j) belongs to category c. The time t denotes the current time, and $\overline{t'}$ denotes the arrival time.

Algorithm 1. Travel time calculation procedure
Set t to t_0
Set c to c_{ij}
Set t' to $t + (c/v_{ijk})$
while $t' > \overline{t_k}$ do
$c \leftarrow c - v_{ijk}(\overline{t_k} - t)$
$t \leftarrow \overline{t_k}$
$t' \leftarrow t + (c/v_{ijk+1})$
$k \leftarrow k+1$
end
Return travel time $= (t' - t_0)$

In this study, TD-HCPP model has some assumptions including: The route is designed for a single machine. The origin for the vehicle is given and the destination for the vehicle after one cycle is the same as its origin. Travel time of each edge depends on the time of day and the distance between the nodes.

3 Mathematical formulation

This section describes a Mixed Integer Programme (MIP) formulation for TD-HCPP. The problem is defined based on the undirected graph $G(V, E)$. In a set of given nodes $V = \{i, j = 1, 2, \ldots, n\}$, i_0 is the starting point and the route starts and ends in i_0. The edge set $E = \{(i, j)\colon i, j \in V \text{ and } i \neq j\}$, the distance matrix $\mathrm{C} = \{c_{ij}\colon i, j \in V\}$. In the undirected case, $c_{ij} = c_{ji}$. In the HCPP, the edge set is partitioned into $h = 1, 2, \ldots, H$ priority classes $E_1, E_2, \ldots, E_h$ with $E_i \cap E_j = \emptyset$ for $i \neq j$, the classes obey a linear order ($\ll$), generality is numbered as $E_1 \ll E_2 \ll \cdots \ll E_h$. All edges of class E_i must be traversed before those of class E_{i+1}. Table 2 lists the sets, parameters and decision variables used in this model.

Table 2 Mathematical model indexes, sets, parameters and variables

Indexes	
i	Starting nodes
j	Ending nodes
t	Index of the edge traversal at step t in the route
k	Index of the edge traversal at kth time period
h	Index of the edge traversal in the hth hierarchy
Sets	
V	Set of vertices $i, j = \{1, 2, \ldots, n\}$
H	Set of hierarchies $\forall\, h \in H$
K	Set of time periods $\forall\, k \in K$
E_h	Set of edges of hth hierarchy class
E	Set of all edges in the network, $E = \{E_1 \cup E_2 \cup \cdots \cup E_h\}$
$\|E_h\|$	The total number of edges of hth hierarchy class
$\delta_h(i)$	The edges of E_h set that exist from node i, $\delta_h(i) = \{j, (i, j) \in E_h\}$
Parameters	
c_{ij}	Distance of the edge (i, j)
l_k	Start time of kth time period
u_k	End time of kth time period
v_{ijk}	Travel speed from node i to node j at kth time period
w_0	Travel start time
M	Is a sufficiently large number
Variables	
x_{ij}^t	$\begin{cases} 1, & \text{if traversed from node } i \text{ to node } j \text{ at } t\text{th step} \\ 0, & \text{otherwise} \end{cases}$
y_{ij}^t	$\begin{cases} 1, & \text{if traversed from node } i \text{ to node } j \text{ at } t\text{th step for the first time} \\ 0, & \text{otherwise} \end{cases}$
ϕ_{ht}	$\begin{cases} 1, & \text{if traversed from all edges of } h\text{th hierarchy in } t\text{th step completed} \\ 0, & \text{otherwise} \end{cases}$
z_{ijt}^k	$\begin{cases} 1, & \text{if traversed from node } i \text{ to node } j \text{ at } t\text{th step in the } k\text{th time period} \\ 0, & \text{otherwise} \end{cases}$
wd_i^t	Continuous decision variable that contains the departure time from node i at step t
wa_i^t	Continuous decision variable that contains the arrival time to node i at step t
T_{ij}^t	Travel time between node i and node j at step t

The formulation of TD-HCPP is given by

$$Minimize \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{L} T_{ij}^t \tag{1}$$

Subject to

$$\sum_{t}\left(x_{ij}^{t}+x_{ji}^{t}\right)\geq 1 \quad \forall(i,j)\in E \tag{2}$$

$$\sum_{i}x_{ij}^{t-1}=\sum_{s}x_{js}^{t} \quad \forall j,t \quad t>1, \quad i,s\in V \tag{3}$$

$$\sum_{(i,j)}x_{ij}^{1}\geq 1 \quad (i,j)\in\delta_1(i_0) \tag{4}$$

$$\sum_{j}x_{ji}^{t}\geq 1 \quad t=t_{end}, \quad i=i_0 \tag{5}$$

$$\sum_{(i,j)}x_{ij}^{t}=1 \quad \forall t \tag{6}$$

$$x_{ij}^{t}+x_{ji}^{t}\geq y_{ij}^{t} \quad \forall i,j,t \quad i<j \tag{7}$$

$$\sum_{t'<t}(x_{ij}^{t'}+x_{ji}^{t'})\leq M*(1-y_{ij}^{t}) \quad \forall i,j,t \quad i<j \tag{8}$$

$$\sum_{t}y_{ij}^{t}=1 \quad \forall(i,j) \quad i<j \tag{9}$$

$$\sum_{t'\leq t}\sum_{(i,j)\in E_h}y_{ij}^{t'}\geq\|E_h\|*\phi_{ht} \quad \forall h,t \tag{10}$$

$$\sum_{t'\leq t}\sum_{(i,j)\in E_{h+1}}(x_{ij}^{t'}+x_{ji}^{t'})\leq M*\phi_{ht} \quad \forall t, \quad h=1,\ldots,H-1 \tag{11}$$

$$wd_i^{'1'}=w_0 \quad i=i_0 \tag{12}$$

$$v_{ijk}*(wa_j^t-wd_i^t)+M*x_{ij}^t+M*z_{ijt}^k\leq c_{ij}+2*M \quad \forall i,j,t,k \tag{13}$$

$$v_{ijk}*(wa_j^t-wd_i^t)\geq c_{ij}*z_{ijt}^k \quad \forall i,j,t,k \tag{14}$$

$$wd_j^t=wa_j^{t-1} \quad \forall j,t \quad t>1 \tag{15}$$

$$T_{ij}^t\leq M*x_{ij}^t \quad \forall i,j,t \tag{16}$$

$$T_{ij}^t\leq(wa_j^t-wd_i^t) \quad \forall i,j,t \tag{17}$$

$$T_{ij}^t\geq(wa_j^t-wd_i^t)-M*(1-x_{ij}^t) \quad \forall i,j,t \tag{18}$$

$$wd_i^t+M*z_{ijt}^k\leq u_k+M \quad \forall i,j,t,k \tag{19}$$

$$wd_i^t\geq z_{ijt}^k*l_k \quad \forall i,j,t,k \tag{20}$$

$$\sum_{k}z_{ijt}^k=x_{ij}^t \quad \forall i,j,t \tag{21}$$

$$x_{ij}^t,y_{ij}^t,\phi_{ht},z_{ijt}^k\in\{0,1\},T_{ij}^t,wd_i^t,wa_i^t\geq 0 \quad \forall i,j,t,h,k \tag{22}$$

The objective function (1) minimizes the total travel time. Constraints (2) and (3) are the general constraints developed for CPP. Constraint (2) guarantees traversing each edge at least once. Constraint (3) ensures the connectivity between edges. If a

route enters node j at step $t-1$, it must leave node j at step t. Constraint (4) along with constraint (5) forces the route to begin at the starting node, i_0, at the first step (t=1), and end at the same node on the last step. Constraint (6) enables only one edge at step t to be traversed. Constraints (7) to (9) fix the definition of y_{ij}^t, which describes the first-time traversal of the edge (i, j) at step t. Note that if x_{ij}^t or x_{ji}^t binary variables are equal to 0, the edge (i, j) is not traversed in the tth step. Therefore, Constraint (7) ensures that the symbol y_{ij}^t will be given the value 0, depending on x_{ij}^t. Regarding the steps prior to t, the information on whether edge (i, j) is traversed before is given by Constraint (8). This is illustrated with an example: Consider an edge (i, j). The set of $[x_{ij}^t]_{1xt}$ can be represented as a vector whose elements are established by the index t. Suppose that, $1 \leq t \leq 10$,

$$\left[x_{ij}^t\right] = [0010001001] \tag{23}$$

$$\left[x_{ji}^t\right] = [0001000100] \tag{24}$$

$$\left[x_{ij}^t\right] + \left[x_{ji}^t\right] = [0011001101] \tag{25}$$

Equality (23) shows that edge (i, j) was traversed from node i to node j at step 3, 7 and 10, and Equality (24) shows that edge (j, i) was traversed from node j to node i at steps 4 and 8. Equality (25) shows that edge (i, j) was traversed 5 times in total and the first traversal was performed in step 3. In Constraint (7), if $x_{ij}^t + x_{ji}^t = 0$, then $y_{ij}^t = 0$. Otherwise, if $x_{ij}^t + x_{ji}^t = 1$ then constraint (8) requires that $\sum_{t'<t} (x_{ij}^{t'} + x_{ji}^{t'}) \leq M * (1 - y_{ij}^t)$. If $\sum_{t'<t} (x_{ij}^{t'} + x_{ji}^{t'})$ is equal to 1, which means that the edge (i, j) was traversed before, then $y_{ij}^t = 0$. If the summand is equal to 0, then $y_{ij}^t \in \{0, 1\}$. The summand is equal to 0 only at steps 2 and 3. However, due to Constraint (7), $y_{ij}^t = 1$ only when t=3rd step. So, the variable $y_{ij}^t = 0$ for all values of t except for t=3, in which case constraint (9) applies. This gives the following results:

$$\left[y_{ij}^t\right] = [0010000000] \tag{26}$$

The y_{ij}^t decision variable plays an important role, since there is a hierarchical structure between edges and it is impossible to traverse from the next hierarchy without traveling all roads in the current hierarchy. In this way, the notion for traversing the edges for the first time is used to control whether all roads in a hierarchy have been traversed or not. Constraint (10) fixes the definition of ϕ_{ht}, which describes that all edges $(i, j) \in E_h$, in the hth hierarchy are passed at step t. Constraint (11) ensures that (h+1). Priority edges cannot be traversed without traversing h. priority edges. The main time-dependency notions are given in Constraints (12) to (21) as the fixed distance is replaced by the time-dependent travel time. Constraint (12) shows the departure time from starting node (i_0). Constraint (13) and (14) correlate the traversal time between two nodes to the speed and the distance covered. Then, constraint (15) guarantees that the departure time of any node except i_0 is equal to the arrival time of this node at the previous step. The travel time for each step t, is calculated by multiplying the value

of differences between the arrival and departure times in the nodes at each step t and x_{ij}^t binary variable. In other words, it is expressed as $(wa_j^t - wd_i^t) * x_{ij}^t$. Since only one (i, j) edge is traversed in each t step, the travel time naturally becomes equal to 0 when $x_{ij}^t = 0$. Since the objective is to minimize the total travel time, the formulation is shown as follows:

$$Minimize \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{L} (wa_j^t - wd_i^t) * x_{ij}^t \tag{27}$$

In order to solve the model (27) we need to linearize it. Suppose that this expression is equal to T_{ij}^t positive variable,$T_{ij}^t = (wa_j^t - wd_i^t) * x_{ij}^t$. The product of a binary (x_{ij}^t) and a continuous variable $(wa_j^t - wd_i^t)$ is linearized with Constraints (16) to (18).

Constraints (19) and (20) determine the appropriate time interval according to the departure time from the originating node. If (i, j) edge is traversed, Constraint (21) guarantees that the vehicle absolutely starts in a time period. Finally, constraint (22) represents the type of variable.

4 Solution approaches

Metaheuristic algorithms are general algorithmic frameworks that have significantly capable and effective methods for tackling many types of complex optimisation problems (Glover and Kochenberger 2003). These methods have been successfully used for solving difficult combinatorial optimisation problems in operations research. Hence, these algorithms could be potential alternatives, because the solution time increases non-polynomially according to the enlargement of the problem scale by solving the problem with exact solvers.

Metaheuristics include a range of algorithms with different characteristics. Generally, determination of an efficient metaheuristic is an experience dependent process requiring extensive analyses on different metaheuristics and their basic parameters and operators. In this study, a genetic algorithm (GA) and hybrid simulated annealing are selected for tackling the proposed TD-HCPP.

There have been a number of previous papers related to arc routing. Multi-objective genetic algorithms were proposed by Lacomme et al. (2006) for solving capacitated arc routing problems. Morgan and Mumford (2009) presented a weight-coded genetic algorithm approach to the capacitated arc routing problem. Dhein et al. (2018) presented a genetic local search algorithm for a new bi-objective arc routing problem with profit collection and vehicle dispersion. The use of a hybrid GA for solving open capacitated arc routing problems was described by Arakaki and Usberti (2018). However, the number of studies in which a GA has been applied to the Chinese postman problem is limited, and a summary of these is presented in Hua and Li-shan (2003), Jiang et al. (2010) and Ma et al. (2015). The Simulated Annealing (SA) heuristic was proposed by Kirkpatrick et al. (1983) and Cerny (1985). Both conceptually and practically, it is one of the simplest meta-heuristics and has been applied successfully

to many hard combinatorial optimisation problems. In addition, it has recently been applied to arc routing problems (Tirkolaee et al. 2016; Wøhlk 2015).

The following subsections discuss in further detail the solution representation, objective function calculation, and main steps of the proposed GA and hybrid SA (hSA) heuristics.

4.1 Solution representation

In meta-heuristic algorithms, the solution representation method has profound effects on the quality of results. Because TD-HCPP has a difficult and complex structure, approaches using meta-heuristics to produce new solutions have become complex. Each route in the TD-HCPP represents an appropriate solution. However, the lengths of the routes may differ, because they can travel along the edges multiple times. Moreover, the routes are determined based on their hierarchical level, and any disruption of this structure results in infeasible solutions. Therefore, representation of the solution as a route is less appropriate for designing the crossover, mutation, and neighbourhood solution producing operators.

Both GA and SA heuristics have been applied to many scheduling problems in the literature (Wang and Zheng 2003; Hart et al. 2005; Gaafar and Masoud 2005). They are also frequently used to solve the travelling salesman and vehicle routing problems (Jung and Haghani 2001; Moon et al. 2002; Jiang et al. 2005; Razali 2015; Kumar and Panneerselvam 2015; Mukhairez and Maghari 2015; Mohammed et al. 2017; Samadi-Dana et al. 2017). In most of the scheduling and node routing studies, solutions have been demonstrated using permutation coding techniques. Further, in some arc routing problems discussed in the literature, the arcs/edges have also been indicated by a string of numbers comprising a permutation of n arcs/edges. Moreira and Ferreira (2010) described a new GA to solve the rural postman problem, where it was transformed into a travelling salesman problem. Here, all required edges were numbered, and the genes of each chromosome were represented by these numbers. Yu and Lin (2015) developed an iterated greedy heuristic algorithm for the time-dependent prize-collection arc routing problem. A solution was represented by a string of numbers comprising a permutation of n prize arcs, denoted by the set {1, 2, …, n}. Rabbani et al. (2016) proposed a multi-objective simulated annealing algorithm for solving the capacitated k-vehicles windy rural postman problem. They developed a solution representation where the length of the chromosome was considered equivalent to the number of required edges plus the number of vehicles minus one. Each solution was represented by a random permutation vector of required edges for all vehicle routes. Tirkolaee et al. (2016) proposed a hybrid metaheuristic algorithm for a capacitated arc routing problem. Solution representation was used as a string to represent the traversing sequence of required edges. In other words, each chromosome was shown as a sequence of the required edges.

In this study, both the complexity of TD-HCPP and due to the widespread use of permutation code technique for combinatorial optimizations problems in the literature, we have considered the TD-HCPP as a scheduling problem for which all edges are defined as a task. A TD-HCPP solution is represented by a string of numbers com-

Table 3 Graph transformation

Task no (τ)	Edge (i, j)	Hierarchical level (h)	Task priority
1	(1, 2)	1	–
2	(2, 3)	1	–
3	(2, 4)	1	–
4	(2, 5)	1	–
5	(4, 6)	2	1, 2, 3, 4
6	(5, 6)	2	1, 2, 3, 4
7	(6, 7)	2	1, 2, 3, 4
8	(1, 3)	3	5, 6, 7
9	(1, 4)	3	5, 6, 7
10	(3, 5)	3	5, 6, 7
11	(4, 5)	3	5, 6, 7
12	(5, 7)	3	5, 6, 7

prising a permutation of n edges. The jth number indicates the jth edge to be visited. Thus, the first element of a solution indicates the first edge to be visited. Other edges are added to the tour, one by one, from left to right to represent the visiting sequence, provided that the hierarchical precedence is not violeted. Based on this sequence, the meta-heuristic operators can be applied easily and obtained feasible solutions. Therefore, the proposed solution representation is very effective and efficient.

The steps for converting process as follows:

Step 1. Given G(V, E) graph, every edge (i, j) is defined as a "task" and denoted as $\tau^{(i,j)}$. The number of tasks is equal to the total number of edges ($\tau = 1, 2, \ldots, Z$. $Z = \|E\|$).

Step 2. Assign all priority tasks of $\tau^{(i,j)}$ to the set of Z(pt) according to the hierarchical level h of the edge (i, j). If $(i, j) \in E_1$ then, $\tau^{(i,j)}$ task (correspond to this edge) has no predecessor. In the case of $(i, j) \in E_h$, $h > 1$, the priority tasks set of $\tau^{(i,j)}$ are composed of tasks that are assigned to the edges $(i, j) \in E_{h-1}$. The transformation of the graph in Fig. 1 is shown in Table 3.

The representation of a solution is an integer string of length E, where E is the number of edges. The value of each gene identifies the tasks. The sequence of the genes in the solution is the order of the tasks assigned according to their priorities. Thus, the task numbers {1, 2, 3, 4} are initially assigned since they represent the edges that belong to the first hierarchical level and do not have predecessors. Then, the task numbers {5, 6, 7} are assigned, eventually followed by the assignments of the task numbers {8, 9, 10, 11, 12}. Therefore, solutions are of fixed length by ensuring the hierarchical sequence of the TD-HCPP.

By considering the priorities of the tasks, different solutions can be formed through the random assignments of the tasks. However, completely random assignments yield feasible but poor solutions. For instance, in a P = [1 2 3 4 5 7 6 9 8 11 10 12] solution sequence, the tasks are arranged according to their priorities; thus, the solution is a feasible. However, the edges to which successive tasks correspond are not directly

1	3	2	4	6	5	7	12	11	9	8	10
(1,2)	*(2,4)*	*(2,3)*	*(2,5)*	*(5,6)*	*(4,6)*	*(6,7)*	*(7,5)*	*(5,4)*	*(4,1)*	*(1,3)*	*(3,5)*

Fig. 2 An example of solution representation

connected to each other. For example, the task number, $\tau^{(5,6)} = 6$, is followed by the task number $z^{(1,4)} = 9$ indicating that after the edge (5, 6), the edge (1, 4) will be passed. These paths are not directly connected to each other in graph. Hence, a feasible but poor solution is obtained. In heuristics, starting an algorithm from a good solution has an important effect both in terms of computation time and in the approaching the best solution. Therefore, the steps of the method developed for an efficient initial solution include:

Step 1. A task is assigned to the first gene of the solution set P. Since the node $i_0 = 1$ is the starting point, only the edge/edges connected to this node are assigned. In Table 3, there is only the edge (1, 2) and $\tau^{(1,2)} = 1$ is assigned to the P set.
Step 2. Create $\mathbf{S_T}$ set that composed of tasks which have no predecessors.
Step 3. Before assigning a task to the tth gene of the solution P, the task assigned to the previous $(t-1)$th gene is examined. The $\mathbf{S_C}$ set of the tasks that are directly connected to this task (assigned to $(t-1)$th gene) is formed. Thus, at every step t, a $\mathbf{S_C}$ set is created wherein the tasks are directly connected with task $\tau = P(t-1)$.
Step 4. Select a random task $\tau^{(i,j)} \in \mathbf{S} = \{\mathbf{S_T} \cap \mathbf{S_C}\}$ and assign to P(t). If $S = \emptyset$, a task will be selected from the $\mathbf{S_T}$ set randomly.
Step 5. Repeat Step 2–4 until P is complete.

An example of the solution representation for twelve tasks can be seen in Fig. 2.

The necessity for and advantages of the developed approach is:

- An initial solution/solutions can be produced easily in a much shorter time.
- Solution lengths in a population are fixed and equal to E, given the G(V, E) graph.
- The permutation coding method prevalently applied to the routing problems in previous studies was adopted to the TD-HCPP.
- Meta-heuristic operators (Crossover, mutation, produce new solution) are performed without corrupting the feasibility of the individuals.

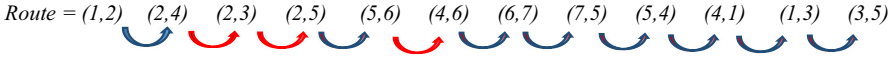
4.2 Objective function value calculation

The fitness function value of each solution is calculated using the objective function in Eq. (1). The travel time of traversing an edge $(i, j) \in E$ is directly associated with the distance (c_{ij}) and speed (v_{ijk}) between the two nodes. Throughout the route, the vehicle starts to travel in a time interval and continues to travel from one time interval to another during its travel. The changing edges and time intervals lead to the changing speed of the vehicle. The departure time (wd_i) determines the time interval in which the vehicle is located and together with the arc category, it can be looked up the starting speed (v_{ijk}) in the speed matrix. The vehicle starts to travel with an identified starting speed and continues at the same speed during its travel. The fitness function calculate pseudocode is shown in Algorithm 2.

Algorithm 2. ObjectiveFunctionValue method

R ← route
w_{d0} ←travel start time
$t^{(R)}$ ←0, total travel time of R
for n=1:2:(length(R)-1)
i=R(n)
j=R(n+1)
c=$c(i,j)$
wd_i ← departure time from node i
h ← define arc categori of edge (i,j), h=$h(i,j)$.
k ← define time interval according to wd_i, k=1,…,4
v ← SpeedMatrix(h,k)
$t^{(i,j)}$←c/v
wd_j= $wd_i + t^{(i,j)}$, calculate arrival time to node j, which is equal to the departure time the next step.
$t^{(R)} = t^{(R)} + t^{(i,j)}$
end
return $t^{(R)}$

Since the TD-HCPP is a routing problem, direct calculation of the objective function value of the solution formed by the arrangement of the tasks is not possible because the edges to which the consecutively arranged tasks correspond do not form a complete route. For example, when the current route of the solution given in Fig. 2 is formed using the arrangement of the edges (i, j), the flow between nodes is disrupted. The flow of the route cannot be sustained in the regions denoted by the red arrow.



To calculate an objective function of a solution, the Route Construction method is proposed. All initial solutions are converted to the feasible routes by using this method that is introduced in Sect. 4.2.1.

4.2.1 Route construction

The process to convert a solution **P** to route **R** are given in Algorithm 3. This algorithm runs with the solution provided by the objective function value procedure in Algorithm 2. All symbols used in this stage are given;

P_{size}: Length of solution P.
R: Final route.
$R_{current}$: Incomplete set R.
R_T: Temporary route set.
R_{back}: The return path from the final node of the $R_{current}$ to the starting node $i_0 = 1$.
$\tau^{(i,j)}$: Task no that assigned to edge (i, j), $\tau^{(i,j)} = 1, \ldots, Z$.
p_{end}: The last element of $R_{current}$, $(p_{end} \in V)$.
$h_{current}$: Hierarchical level of $R_{current}$. It means that, when $\forall(i, j) \in E_h$ are passed in $R_{current}$, $h_{current} = h + 1$. if there are missing edge $(i, j) \in E_h$ *in* $R_{current}$, it means that, not all of the edges in the hth hierarchical level are passed and $h_{current}$=h. HCPP contains a hierarchical arrangement, and therefore, without completing the edges

in the hth hierarchical level, the edges with the hierarchical levels of $h+1$, $h+2$, …, H are not passed. Hence, the $h_{current}$ information is needed in the route completion process.

$t(p_{end}, i)$: Travel time from node p_{end} to i.

$t(p_{end}, j)$: Travel time from node p_{end} to j.

$t^{(R)}$: Total travel time of route R.

$h(p_{end}, i)$: Hierarchical level of edge $(p_{end}, i) \in E$.

$h(p_{end}, j)$: Hierarchical level of edge $(p_{end}, j) \in$ E.

I: Adjacency matrix $(n \times n)$. if there is a direct connection between node i and j, I(i, j) = 1, otherwise I(i, j) = 0.

Each task $\tau^{(i,j)}$ belonging to the solution P is put into the process in turn. The edge (i, j) to which the task is assigned is added to the $R_{current}$ by ensuring the integrity of the flow. The steps of the proposed Route Construction approach can be summarized as follows.

Step 1. Select first gene P(1). Add edge (i,j) correspond to task $\tau^{(i,j)}$ = P(1) to the $R_{current}$ set, initially $R_{current}$ = [].

Step 2. Select edge (i,j) that correspond to $\tau^{(i,j)}$ = P(t), (t=2, …, P_{size}).

Step 2.1 Define the p_{end} and $h_{current}$. When creating the $R_{current}$, both connectivity between nodes and hierarchical sequence should be provided. Hence, the paths from the p_{end} to node i and j is created and the hierarchical levels of these paths are determined ($h(p_{end},i)$, $h(p_{end},j)$). Then, which path will be added to the $R_{current}$ is decided.

Step 2.1.1 if $p_{end} = i$, add (i,j) to $R_{current}$. So, there is no need of additional paths to ensure connectivity.

Step 2.1.2 if $p_{end} = j$, add (j,i) to $R_{current}$. So, there is no need of additional paths to ensure connectivity.

Step 2.1.3 if both $p_{end} \neq i$ and $p_{end} \neq j$, then the paths should be found from p_{end} to i and j respectively to ensure connectivity. According to the travel time of paths, less travel time path is added to $R_{current}$.

Step 2.1.3.1 Find the path from p_{end} to i. This path is added temporary route set R_{Ti}. At this stage, two scenarios emerge:

(I) There is a direct connection between the p_{end} and i, I($p_{end,}$ i)=1. In this case, the decision is made after performing a hierarchical control. If $h(p_{end,}\, i) \leq h_{current}$ then the path ($p_{end,}$ i)∪(i,j) is assigned to the temporary route R_{Ti}. If $h(p_{end,}\, i) > h_{current}$ then, the ($p_{end,}$ i) path cannot be added. Therefore, additional routes from the p_{end} to node i that do not disrupt the hierarchical arrangement are found. For this purpose, a new graph of G'=(V',E') is created. This new graph consists of edges such as $\forall(i,j) \in E_h$, $h \leq h_{current}$. Then, using the dijkstra algorithm on the new G' network, the shortest path from $p_{end,}$ to i is found. This path ($p_{end,}$ i)∪(i,j) is assigned to the temporary route R_{Ti}.

(II) There is no direct connection between the p_{end} and i, I($p_{end,}$ i)=0. In this case, a new graph of G'=(V',E') is created. The shortest path from $p_{end,}$ to i is found by dijkstra algorithm on G'. This path ($p_{end,}$ i)∪(i,j) is assigned to the temporary route R_{Ti}.

The travel time $t^{(R\,Ti)}$ of the route R_{Ti} is calculated.

Step 2.1.3.2 Find the path from p_{end} to j. The processes in Step 2.1.3.1 are repeated to find the route R_{Tj}. The travel time $t^{(R\,Tj)}$ of the thereby obtained route R_{Tj} is calculated.

Step 2.2 The $t^{(R\,Ti)}$ and $t^{(R\,Tj)}$ values are compared. The route with the shortest travel time is added to the $R_{current}$ sequence.

Step 3. Repeat Step 2 as much as the solution length (P_{size}).

Step 4. The shortest path from the final node of the $R_{current}$ to the node i_0=1 is found and added to the set of R_{back}.

Step 5. The final route set is completed, R={$R_{current}$ ∪ R_{back}}.

Algorithm 3. Route construction method

w_{d0}, P, P_{size}
R, $R_{current}$, $R_{back} \leftarrow \varnothing$
$R_{current}(1) = i$, $R_{current}(2) = j$, $(i,j) = \tau^{(i,j)} = P(1)$
calculate $t^{(i,j)}$
$w_d \leftarrow w_{d0} + t^{(i,j)}$
for k=2: P_{size}
$\tau^{(i,j)} \leftarrow P(k)$
$p_{end} \leftarrow$ last node of $R_{current}$
if $p_{end} = i$
$R_{current} \leftarrow R_{current} \cup \{(i,j)\}$
elseif $p_{end} = j$
$R_{current} \leftarrow R_{current} \cup \{(j,i)\}$
elseif $((p_{end} \neq i)$ & $(p_{end} \neq j))$
$h_{current} \leftarrow$ current max hierarchy level of $R_{current}$
R_{Ti}, $R_{Tj} \leftarrow \varnothing$

if $I(p_{end},i) = 1$
$h_{control} \leftarrow h, (p_{end},i) \in E_h$
if $h_{control} \leq h_{current}$
$R_{Ti} \leftarrow \{(p_{end},i)\}$
elseif $h_{control} > h_{current}$
G' $\leftarrow$ create new graph
$R_{Ti} \leftarrow$ dijkstra(G',p_{end},i)
end
elseif $I(p_{end},i) = 0$
G' $\leftarrow$ create new graph
$R_{Ti} \leftarrow$ dijkstra(G',p_{end},i)
end
$R_{Ti} \leftarrow R_{Ti} \cup \{(i,j)\}$
$t^{(RTi)} \leftarrow$ *ObjectiveFunctionValue*(R_{Ti})
if $t^{(RTi)} < t^{(RTj)}$
$R_{current} \leftarrow R_{current} \cup \{R_{Ti}\}$
$w_d \leftarrow w_d + t^{(RTi)}$
elseif $t^{(RTj)} < t^{(RTi)}$
$R_{current} \leftarrow R_{current} \cup \{R_{Tj}\}$
$w_d \leftarrow w_d + t^{(RTj)}$
else
$R_{current} \leftarrow R_{current} \cup$ randomly select$\{R_{Ti}/R_{Tj}\}$
end

if $I(p_{end},j) = 1$
$h_{control} \leftarrow h, (p_{end},j) \in E_h$
if $h_{control} \leq h_{current}$
$R_{Tj} \leftarrow \{(p_{end},j)\}$
elseif $h_{control} > h_{current}$
G' $\leftarrow$ create new graph
$R_{Tj} \leftarrow$ dijkstra(G',p_{end},j)
end
elseif $I(p_{end},j) = 0$
G' $\leftarrow$ create new graph
$R_{Tj} \leftarrow$ dijkstra(G',p_{end},j)
end
$R_{Tj} \leftarrow R_{Tj} \cup \{(j,i)\}$
$t^{(RTj)} \leftarrow$ *ObjectiveFunctionValue*(R_{Tj})

end
Output;
$R_{current}$, w_d
end
$p_{end} \leftarrow R_{current}$
$R_{back} \leftarrow$ dijkstra (G,p_{end},1)
$R \leftarrow R_{current} \cup \{R_{back}\}$

4.3 Main steps of the proposed GA heuristic

In this section, a GA approach is proposed to solve the TD-HCPP. Generally speaking, in the first stage, we apply the solution representation method as described in Sect. 4.1 to generate initial feasible solutions, then GA is employed to improve the initial solutions. To make the solution escape from the local optima advance and to accelerate the convergence, mutation and crossover operators are designed.

4.3.1 Initial population

In the GA, a chromosome represents a feasible solution. The population consist of a set of chromosomes which are created randomly. Each iteration of the outer loop provides one chromosome P, generated as given in Algorithm 4. All of the chromosomes fitness function values are calculated by using the objective function calculation method described in Sect. 4.2.

Algorithm 4. Generate initial population

```
Pop_size ← population size
Pop ←[ ]
while ||Pop|| < Pop_size
        Create a new chromosome P
        P(1) ←randomly select a task τ^(1,j)
                for t=2:E
                        Set S_T, S_C, S={S_T ∩ S_C}
                        Randomly select a task τ from S
                            if S = Ø
                              Randomly select a task τ from S_T
                            end
                         Insert τ in chromosome P(t)
                         S_T ← S_T \ τ
                end
        Pop ← Pop ∪ {P}
end
return Pop
```

4.3.2 New population generation

New population generation is described in Algorithm 5. Two different chromosomes are selected by the Roulette Wheel method. In this method, the probability of selecting a chromosome is proportional to its fitness and the chances of the fitter individual to be selected as a parent for crossover are higher. With respect to minimization of the objective function in our problem, the fitness value of each chromosome is determined to be $1/f_i$, $i=1, \ldots, \text{Pop}_{size}$. Only one child is generated through crossover reproduction, according to a crossover probability p_c. A mutation operator is applied to a child according to a mutation probability p_m. The selection, reproduction and mutation sequence is repeated during the iterations of GA.

Algorithm 5 Method to generate a new population

```
Pop ← current population
Pop_Fonk ← set of ObjectiveFunctionValue of all chromosomes in Pop
P1 ← SelectIndividual(Pop)
P2 ← SelectIndividual(Pop)
if GenerateRandomNumber (∈(0,1)) < p_c
    Offspring ← crossover(P1,P2)
    if GenerateRandomNumber (∈ (0,1)) < p_m
        Mutate(Offspring)
    end
    child ← Offspring
end
child_fonk ← calculate ObjectiveFunctionValue(child)
After crossover and mutation operators, update Pop
control=0
for i=1:length(Pop_Fonk)
    if child_fonk < Pop_Fonk(i)
      Pop(i,:) ← child
      indice=i
      control=1
      break;
    end
end
if control =1
   for j=(indice+1): length(Pop_Fonk)
        Pop(j,:) ← Pop(indice,:)
        indice = indice +1
   end
end
Return Pop, Pop_Fonk
```

Crossover

Crossover is a very important way to reproduce and produce offspring, which can increase the diversity of the population. In this study, a single-point crossover method was employed. However, in this method, the exchange of the parents after being cut from a single point yields infeasible solutions. The crossover method developed to produce feasible solutions is as follows: with $p \neq 1$ and $p \neq E$, a random crossover point (p) is selected and all genes of the first parent until this point are exactly copied to the offspring. Starting from 1, all genes of the second individual are controlled in turn, and the genes that the offspring do not have are transferred to the offspring starting from the ($p+1$)th point. The process continues until there are no genes left to assign to the offspring. Only a single offspring is formed from the two individuals selected with this method and the hierarchical sequence of the new individual is not disrupted, indicating that a new feasible solution is produced. An example of crossover type is illustrated in Fig. 3.

Mutation

The mutation operation is used to increase the diversity of the population by preventing entrapment of the solution to a local optimum. The process of mutation happens after the crossover process, and in its most common form involves producing new chromosomes by replacing the gene (or several genes) of existing chromosomes. However, for

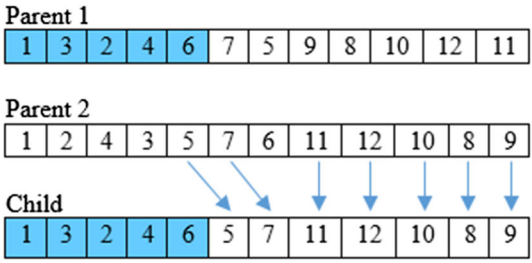


Fig. 3 Crossover process

the TD-HCPP, the mutation operation is performed by selecting, and then replacing, tasks (genes) that only belong to the same hierarchical level. Thus, infeasible solutions due to replacing the edges from different hierarchical levels are avoided. In this study, to solve the TD-HCPP, we use the neighbourhood solution producing method in SA (described in Sect. 5.2) combined with a mutation operator. Here, $\|Eh\| - 1$ individuals are produced, and then only one individual is selected among those having a minimum objective function value. For instance, for C1 = (1, 3, 2, 4, 5, 7, 6, 9, 10, 8, 11, 12), an offspring of C1 has three sub-sets; $C1_1 = \{1, 2, 3, 4\} \in Z_1$, $C1_2 = \{5, 6, 7\} \in Z_2$, and $C1_3 = \{8, 9, 10, 11, 12\} \in Z_3$. If we assume that task no 6 (7th gene) is selected, then task nos 5 and 7 are replaced by task no 6. Therefore, $\|Eh\| - 1 = 2$ individuals are created. These are; $C1' = (1, 3, 2, 4, 6, 7, 5, 9, 10, 8, 11, 12)$, $C1'' = (1, 3, 2, 4, 5, 6, 7, 9, 10, 8, 11, 12)$. Finally, the best solution among the generated solutions is chosen.

5 Main steps of the proposed hybrid heuristic algorithm: hSA

In this section, to improve the performance of the basic SA heuristic further, a greedy search heuristic is applied to generate the initial solution. The proposed hSA algorithm is basically composed of two steps. In the first step, an initial solution is obtained by the greedy algorithm. Starting an algorithm from a good solution in heuristic methods particularly helps in reducing computation time. Therefore, a good initial solution is produced by using the greedy algorithm as described in Sect. 5.1. The solution used in SA is the second step of hSA. The SA method is a strong local search strategy that continues to improve the solution until the initial solution meets the stop criterion. The neighbours of the initial solution are devised by this strategy, because the optimal solution may be close to this solution. When the algorithm stops, the final solution and the cost are given as outputs.

5.1 Greedy algorithm

The proposed greedy algorithm is based on the nearest neighbour algorithm. In this algorithm, the rule is always to go next to the nearest as-yet-unvisited edge, subject to the following restrictions: we start with a starting point, and then traverse to all the edges closest to the starting city that do not create any sub tour and violate the hierarchical precedence. If the next nearest edge makes a sub tour or exceeds the hierarchical rules, then we visit another unvisited edge that is not the nearest. In some cases, unvisited edges are not directly connected with the last visited edge. To ensure

the connectivity in the route, it may be necessary to pass through other specified edges instead of the nearest edges for traversing the unvisited edges. This process continues until all cities are visited at least once. The proposed algorithm is easy to implement and executes quickly, but it can sometimes miss shorter routes, due to its greedy nature.

The steps of the algorithm are as follows:

Step 1. Start from the starting point ($i_0 = 1$) and assign the R route set (initially R = []).
Step 2. Create $\mathbf{S_R}$ edges set that should be visited.
Step 3. Define the neighbours of the last node of R and assign the **S** edges set.

- If $\mathbf{S} \cap \mathbf{S_R} \neq \emptyset$, select an edge having the least distance from the last node and assign the R route.
- If $\mathbf{S} \cap \mathbf{S_R} = \emptyset$, this means that the unvisited edges ($\in S_R$) are not directly connected with the last node ($\boldsymbol{n_{last}}$) of R route. Hence, the set of N nodes that can be visited from the $\boldsymbol{n_{last}}$ (under hierarchical constraints) is identified. If these node/nodes $\in S_R$, select one of them with the least distance and add to R route. If $(N \cap S_R) = \emptyset$, it is checked whether each node in the N is adjacent with the nodes in the S_R. One of these nodes connected with nodes in the N set is selected randomly and added to R route. After ensuring the connectivity between $\boldsymbol{n_{last}}$ and each node of $\mathbf{S_R}$, an appropriate edge ($\in S_R$) is added to the R route.

Step 4. If all edges are being passed, go to Step 5. Otherwise, go to Step 2.
Step 5. Stop the algorithm and report the solution.

5.2 Simulated annealing

Simulated annealing is a stochastic search technique that uses a probability function to improve the quality of the solutions. The general framework of SA is such that it starts from an initial solution (S), then goes to a new solution (S′) by using a specific neighbourhood generation procedure. Subsequently, a neighbourhood will be replaced with the initial solution if its objective function value is better than the initial solution. Otherwise, the amount of change in the objective function value is calculated (Δ) and a random number is generated from the interval [0, 1]. If the random number is less than the value of the probability equation $\left(\exp^{-\Delta/T}\right)$, the worse solution is accepted. Commonly, the temperature (T) is uniformly reduced and the cooling equation is $\alpha T \rightarrow T$. This process is repeated until the stopping criterion is reached.

In this algorithm, the solution representation is treated as a permutation sequence of the tasks described in Sect. 4.1. There are various methods for achieving the neighbourhood solution. The neighbourhood solution producing method is based on a modified generation mechanism introduced by Ishibuchi et al. (1995). In this technique, several neighbours of a current solution are evaluated, and the move to the best of these neighbours is examined.

The steps of simulated annealing can be summarised as follows:

Step 1. Heuristic parameter values are defined (initial temperature T_o, cooling rate α, frozen temperature T_{end}), let $i = 0$.

Step 2. Choose an initial solution S from the greedy algorithm. The obtained solution is a feasible route. For this reason, the route is first converted into a permutation sequence as described in Sect. 4.1. (The necessity of this transformation, to produce more efficient, faster, and new feasible solutions in simulated annealing, has been mentioned previously). Then, the objective function is calculated, which was introduced in Sect. 4.2.
Step 3. Randomly and independently generate M solutions from the neighbourhood of the current solution S. Select the best solution S' among the generated M solutions. Let $i = i + \mathrm{M}$.

- If the objective value is better, the neighbourhood solution is considered the current solution.
- If the objective value is worse, the neighbourhood solution is considered the current solution based on a probability $P = \left(\exp^{-\Delta/T}\right)$. Otherwise, the current solution will not change.

Step 4. The counters and parameters are updated, and steps 3–4 are repeated until the stopping criterion is reached.

Only Step 3 is different from the standard simulated annealing algorithm. The best solution among the generated M solutions is chosen. In the TD-HCPP, the value of M varies according to the number of hierarchical levels of each problem. That is, a randomly selected task belonging to the hth hierarchical level is replaced with all the other tasks belonging to the same hierarchical level. Thus, $M = \|E_h\| - 1$ neighbourhood solutions are produced at each iteration. The temperature value is updated when the maximum number of iterations N is reached ($i \geq \mathrm{N}$). The number of neighbourhood search was taken as low ($\mathrm{N} = 10$) in each temperature. A higher N value results in higher computational time because of M neighbourhood solutions are produced at each iteration of N.

6 Results and discussion

The computational results of the proposed mathematical model, GA and hSA will be described below.

6.1 TD-HCPP datasets

To evaluate the proposed mathematical model and metaheuristics, adequate test instances and appropriate performance measures are required. No benchmark test instances have been developed for the HCPP so far. This study examined two classes of HCPP instances:

- Illustrative. There are two small-size test instances of HCPP in the literature. These are the test instances used in the studies conducted by Dror et al. (1987) and Korteweg and Volgenant (2006). The number of nodes in these instances are 8 and 7 respectively, and the number of hierarchies in both instances is $h = 3$.

- Random. The second class of instances was randomly generated with varying values of parameters: number of nodes, edges and hierarchies. The set of edges that corresponds to each node was given in two different forms as simple and medium-level. The hierarchy level for each instance was generated respectively between 2 and 5.

These instances were further adapted with varying travel speeds (and time) at different time periods. Each instance has a time span from 7 a.m. to 7 a.m. in which the day (24 h) is divided into four different time intervals. The starting node is taken as $i_0=1$ in all test instances. The number of time intervals is taken as $k=4$ for each instance. Departure time from node 1 for all instances is taken as 8:30 a.m. The details of test instances characteristics can be seen in Table 4.

6.2 Computational results

In this section, we assess the performance of Cplex, GA and hSA by comparing the minimum costs found by them. The instances were solved by Cplex solver in GAMS 24.2.3 and the proposed GA and hSA in Matlab 2016a. We carry out all experiments using a system with intel i7 3.40 GHz processor and 8 GB memory. The results are represented in Table 4. The first column of the table includes the instance name, while the second, third and fourth columns keep the numbers of vertices, hierarchies and the edges (which are denoted by $|V|$, $|h|$ and $|E|$), respectively. The next three columns are related to Cplex results. These columns demonstrate objective value of the best solution found by Cplex, cpu time Cplex uses, and the percent deviation between the best solution found by Cplex and the best possible objective function value reported by Cplex (which are denoted by O^{Cplex}, T^{Cplex}, Gap), respectively. The maximum time to solve the instances was given 3 h (10,800 s). The computational scalability of the problem has been demonstrated in the random test instances. In the instance class illustrative, and the random instances which are less than 10 nodes, optimum results were obtained within seconds. Instances with 10 nodes and 13 edges take less than half a minute of computation time to compute an optimal route. Although most small size instances can be solved to optimality, the computation time scales up quickly as the number of edges and number of hierarchy increase. A feasible solution has been found for three of four instances with 10 nodes and 18 edges after 3 h, but the % gap is over 50%. For the other example, no feasible solution has been found in 3 h. No results were recorded for the instances with 20 nodes as no feasible solutions were found after 3 h. Based on the literature, HCPP problem is NP-hard in the strong sense (Dror et al. 1987). Since TD-HCPP is an extension of HCPP, this problem is NP-hard as well. Indeed, the computational time increases as the problem size grows and quality of the solution decreases critically. This means that the performance of Cplex solver is not very satisfactory, even in small problems. Results of GA and hSA are also shown in Table 4. The columns 8–10, respectively, keep the objective value of the mean solution, the objective value of the best solution found by GA and the cpu time of best solution (which are denoted by M^{GA}, O^{GA} and T^{GA}). Finally, the mean and the best possible objective function value found by hSA, and the cpu time of best solution hSA uses (which are denoted by M^{hSA}, O^{hSA} and T^{hSA}) are given under columns eleventh, twelfth and thirteenth, respectively. Random experiments on test instances

Table 4 Results of Cplex, GA and hSA

Test instances	$\|V\|$	$\|h\|$	$\|E\|$	O^{CPLEX}	T^{CPLEX} (s)	Gap (%)	M^{GA}	O^{GA}	T^{GA} (s)	M^{hSA}	O^{hSA}	T^{hSA} (s)
HCPP1	8	3	10	**41.90**	1.17	0	41.90	**41.90**	0	41.90	**41.90**	0
HCPP2	7	3	12	**70.64**	17.06	0	70.64	**70.64**	0	70.64	**70.64**	1.15
Rnd_inst1	7	2	7	**93.65**	0.13	0	93.65	**93.65**	0	93.65	**93.65**	0
Rnd_inst2			10	**100.72**	1.52	0	100.72	**100.72**	0	100.72	**100.72**	0
Rnd_inst3		3	7	**102.74**	0.09	0	102.74	**102.74**	0	102.74	**102.74**	1.23
Rnd_inst4			10	**83.31**	0.19	0	83.31	**83.31**	0	83.31	**83.31**	0
Rnd_inst5		4	7	**108.13**	0.09	0	108.13	**108.13**	0	108.13	**108.13**	0
Rnd_inst6			10	**96.16**	0.08	0	96.16	**96.16**	0	96.16	**96.16**	1.03
Rnd_inst7		5	7	**140.83**	0.09	0	140.83	**140.83**	0	140.83	**140.83**	3.19
Rnd_inst8			10	**100.46**	0.17	0	100.46	**100.46**	1.81	100.46	**100.46**	0
Rnd_inst9	8	2	8	**122.57**	0.11	0	122.57	**122.57**	0	122.57	**122.57**	1.25
Rnd_inst10			10	**113.09**	0.05	0	113.09	**113.09**	0	113.09	**113.09**	100
Rnd_inst11		3	8	**125.26**	0.03	0	125.26	**125.26**	0	125.26	**125.26**	3
Rnd_inst12			10	**102.64**	0.03	0	102.64	**102.64**	0	102.64	**102.64**	6.64
Rnd_inst13		4	8	**158.86**	0.08	0	158.86	**158.86**	0	158.86	**158.86**	0
Rnd_inst14			10	**168.95**	1.38	0	168.95	**168.95**	0	168.95	**168.95**	0
Rnd_inst15		5	8	**108.17**	0.16	0	108.17	**108.17**	0	108.17	**108.17**	0
Rnd_inst16			10	**108.37**	0.17	0	108.37	**108.37**	0	108.37	**108.37**	0
Rnd_inst17	9	2	10	**113.185**	0.47	0	113.185	**113.185**	0	113.185	**113.185**	1.34
Rnd_inst18			12	**131.98**	501.42	0	131.98	**131.98**	0	131.98	**131.98**	7.89
Rnd_inst19		3	10	**95.58**	0.11	0	95.58	**95.58**	0	95.58	**95.58**	0
Rnd_inst20			12	**128.22**	174	0	128.22	**128.22**	0	128.22	**128.22**	1.18

Table 4 continued

Test instances	$\|V\|$	$\|h\|$	$\|E\|$	O^{CPLEX}	T^{CPLEX} (s)	Gap (%)	M^{GA}	O^{GA}	T^{GA} (s)	M^{hSA}	O^{hSA}	T^{hSA} (s)
Rnd_inst21		4	10	**141.25**	0.42	0	141.25	**141.25**	0	141.25	**141.25**	0
Rnd_inst22			12	**115.20**	0.09	0	115.20	**115.20**	0	115.20	**115.20**	0
Rnd_inst23		5	10	**132.27**	2.53	0	132.27	**132.27**	0	132.27	**132.27**	0
Rnd_inst24			12	**129.02**	0.55	0	129.02	**129.02**	0	129.02	**129.02**	23.23
Rnd_inst25	10	2	13	**1405.88**	14.75	0	1405.88	**1405.88**	0	1405.88	**1405.88**	0
Rnd_inst26			18	1749.73	10.800	87	1757.6	**1738.25**	500.38	1855.98	1832.3	451.45
Rnd_inst27		3	13	**1330.77**	21.0	0	1330.77	**1330.77**	0	1373.43	1373.43	223.55
Rnd_inst28			18	1652.62	10.800	93	1631.03	**1610.20**	412.42	1666.47	1666.47	384.67
Rnd_inst29		4	13	**1154.25**	22.0	0	1154.26	**1154.25**	0	1154.25	**1154.25**	10.69
Rnd_inst30			18	1452.63	10.800	61	1446.14	**1439.22**	51.41	1469.60	1447.50	303.95
Rnd_inst31		5	13	**1147.60**	0.53	0	1147.60	**1147.60**	0	1147.60	**1147.60**	0
Rnd_inst32			18	–	10.800	–	2037.89	**2014.59**	403.23	2039.85	**2014.59**	390.26
Rnd_inst33	20	2	55	–	–	–	5090.23	4955.06	1534.3	4547.41	**4531.80**	1532.0
Rnd_inst34			76	–	–	–	6379.91	6178.44	1505.0	5830.50	**5770.90**	2736.4
Rnd_inst35		3	55	–	–	–	4384.66	4162.59	389.3	4082.95	**3891.70**	755.1
Rnd_inst36			76	–	–	–	5849.10	**5663.70**	1517.0	6009.81	5787.10	1606.4
Rnd_inst37		4	55	–	–	–	5526.38	5376.51	1354.0	5272.82	**5150.20**	1401.1
Rnd_inst38			76	–	–	–	7163.84	7108.46	1716.9	7008.27	**6824.48**	1469.2
Rnd_inst39		5	55	–	–	–	5253.84	**5054.89**	425.2	5835.96	5694.00	847.2
Rnd_inst40			76	–	–	–	6337.77	6284.85	1168.8	6139.39	**5861.46**	750.6
Rnd_inst41	30	2	125	–	–	–	10.441.95	10.381.20	5796.1	9055.50	**8996.30**	8297.9
Rnd_inst42			174	–	–	–	14.324.43	13.606.70	11.967.0	12.163.67	**12.148.00**	16.181.0
Rnd_inst43		3	125	–	–	–	10.943.48	10.788.10	2625.8	10.248.45	**10.138.27**	3569.3

Table 4 continued

Test instances	$\|V\|$	$\|h\|$	$\|E\|$	O^{CPLEX}	T^{CPLEX} (s)	Gap (%)	M^{GA}	O^{GA}	T^{GA} (s)	M^{hSA}	O^{hSA}	T^{hSA} (s)
Rnd_inst44			174	–	–	–	13.756.43	13.624.10	4253.5	12.213.83	**12.146.35**	7652.7
Rnd_inst45		4	125	–	–	–	12.332.20	12.125.50	3980.2	10.707.38	**10.603.00**	3349.1
Rnd_inst46			174	–	–	–	16.285.28	16.082.60	4099.2	14.573.51	**14.233.00**	3743.7
Rnd_inst47		5	125	–	–	–	11.526.68	11.386.90	2760.8	10.950.81	**10.779.86**	1713.8
Rnd_inst48			174	–	–	–	16.864.38	16.786.30	3827.5	15.341.61	**15.209.00**	4809.5
Rnd_inst49	40	2	223	–	–	–	19.853.23	19.731.60	11.104.0	16.822.00	**16.758.00**	24.526.0
Rnd_inst50			312	–	–	–	29.242.13	29.196.20	29.493.0	24.654.00	**24.595.00**	52.740.0
Rnd_inst51		3	223	–	–	–	19.464.18	19.192.70	10.401.0	17.060.30	**17.042.47**	14.118.8
Rnd_inst52			312	–	–	–	25.926.50	25.788.90	14.188.0	22.687.23	**22.560.42**	21.330.1
Rnd_inst53		4	223	–	–	–	20.366.95	20.095.80	8466.3	18.447.78	**18.400.42**	8052.0
Rnd_inst54			312	–	–	–	25.252.33	25.049.00	10.186.0	22.139.97	**22.057.40**	14.629.7
Rnd_inst55		5	223	–	–	–	19.751.18	19.466.00	4031.8	19.405.99	**19.346.34**	7265.2
Rnd_inst56			312	–	–	–	27.236.65	27.079.00	9982.3	24.437.71	**24.222.05**	17.325.4
Rnd_inst57	50	2	350	–	–	–	31.447.65	31.333.70	43.548.0	27.233.27	**27.099.58**	62.880.9
Rnd_inst58			490	–	–	–	37.923.37	37.869.20	42.089.0	32.041.33	**31.978.00**	130.625.0
Rnd_inst59		3	350	–	–	–	34.763.28	34.282.40	28.752.0	30.203.92	**30.013.36**	43.064.6
Rnd_inst60			490	–	–	–	38.762.50	38.420.00	31.194.0	34.026.94	**33.999.68**	64.734.9
Rnd_inst61		4	350	–	–	–	32.437.18	32.336.50	31.584.0	29.816.30	**29.772.36**	25.974.2
Rnd_inst62			490	–	–	–	48.155.73	47.741.10	56.180.0	41.343.29	**41.099.54**	53.704.3
Rnd_inst63		5	350	–	–	–	34.935.15	34.802.00	10.326.0	31.856.22	**31.733.51**	16.233.4
Rnd_inst64			490	–	–	–	49.027.53	48.620.20	34.854.0	43.498.50	**43.479.00**	46.781.0

Bold: solutions of Cplex, GA and hSA are equal. Bold and underline: best solution

have been implemented to determine the best set of parameters for both algorithms. Regarding the parameters of the GA, the population size is fixed at $Pop_{size}=50$. The crossover and mutation probabilities are taken as 0.8 and 0.05 respectively. We evolve the population for 500 generations. Regarding the parameters of the hSA, the initial temperature $T_o=50$, cooling rate $\alpha=0.9$ and frozen temperature $T_{end}=1$ are taken. Every instance was run more than five times on the same platform conditions. Each experiment is terminated when the solution reach the exact solution in which Cplex obtains. As can be understood from Table 4, Cplex, GA and hSA are able to find the optimal solution for instances with 7, 8 and 9 nodes. However, the average cpu time used by hSA is 83.5 times larger than the average cpu time used by GA. On the other hand, Cplex uses 4.65 times larger computation time than hSA uses. Hence all three methods can be said to be 100% accurate, but GA is the most efficient one. Performance of Cplex, GA and hSA on instances with 10 nodes and 13 edges is similar to their performances on small instances less than 10 nodes. There are three instances with 18 edges (Rnd_inst 26, Rnd_inst 28, Rnd_inst 30) for which Cplex is unable to prove the optimality, and for all of them GA is able to find better solutions than the others.

Different results are obtained with increasing the number of edges. There are 8 instances with 20 nodes for which Cplex is unable to produce the feasible solution, and for 6 of them, hSA achives better results. However, results of GA are better than those of hSA for 2 instances, too.

Performance of hSA gets better results than GA with the increasing problem size. This observation becomes clearer in the results for test instances with 30, 40 and 50 nodes. The average objective function value found by GA is 12.75% larger than the average objective function value found by hSA. The average time used by hSA is 1.57 times larger than the average time used by GA as can be observed for these instances in Table 4.

Generally speaking, hSA outperforms GA for most of the instances. GA results are better than hSA for 5 instances while hSA results are better than GA results for 30 instances implying that hSA clearly dominate GA. As a concluding remark, we can say that hSA is competitive with GA for small-and moderate-sized instances, and it becomes more and more successful as the problem size gets larger.

In GAs, as the size of the problem increases, the rate of improvement decreases. The reason for this might be that as the number of edges increase, the number of possible permutations quickly increases exponentially, but the number of valid permutations does not follow this growth rate. If the number of populations is increased this problem might be solved, although the computational time increases. In addition, GAs mainly use crossover and mutation operators. Crossover allows us to create new chromosomes from previously known good results. As described previously, the TD-HCPP is a complex problem and has too many constraints. Therefore, there is little difference in the child formed by the crossing of two individuals in generating sufficient feasible solutions to the problem. This is because the crossover operator of the GA cannot be applied effectively owing to the deterioration of the feasibility. For these reasons, GA cannot search the solution space adequately and it might be stuck in local optima. The obtained shorter computational times of GA are probably due to these reasons.

In hSA, we only modify our current solution, and do not crossover or mutate its subparts. This avoids using duplicates, and avoids using complex methods for

crossovers. The proposed neighbour solution strategy always generates feasible solutions. It ensures diversification so that to continue the search with a better solution without stuck in local optima. This leads to the conclusion that we can effectively generate new and feasible solutions in hSA for the TD-HCPP.

7 Conclusion and future works

This paper presents the time dependent hierarchical Chinese postman problem, an extension to the HCPP in which the travel time of priority edges depends on the different times of the day. Such problems can be demonstrated several service applications in urban areas with different traffic conditions. The model could help distribution, transportation and service firms to decide the shortest route with taking into consideration every edge traffic situation at each time. This model can be extended for several applications such as snow removal activities, patrolling security vehicles, garbage collection activities and road maintenance activities where the transition roads based on priorities are important. For these applications, time is a vital objective for ensuring safety transportation and prevent accidents. To the best knowledge of the authors, there is no published study concerning this problem. The continuous travel time function was obtained by applying the travel speed function which associated with the hierarchical roads. The model was called TD-HCPP and formulated as a mixed integer linear programme. Sixty-four test instances were randomly generated. Furthermore, in order to decrease the computational time, two metaheuristic algorithms were represented based on genetic algorithm and hybrid simulated annealing. Results of GA and hSA are compared with results of Cplex, and it is observed that the using the solver as the solution method for small sized instances is possible. hSA is equivalent to GA for instances with small and moderate sizes, and hSA outperforms GA for large instances. The main aim of TD-HCPP was to find the minimum total travel time, so considering the main objective of TD-HCPP, it has been concluded that hSA is considered as the more appropriate algorithm than GA for solving TD-HCPP.

The suggestions can be given for future studies as follows:

- New exact solution methods that are able to solve small instances within a reasonable amount of time can be developed. Advanced cuts may need to be developed and applied.
- Different parameter values (such as initial temperature, cooling rate, population size, crossover and mutation rate) of each of the compared algorithms can be used.
- In reproduction processes of the GA and hSA, different approaches that used in the process of generating new individuals can be tried.
- Different meta-heuristic approaches can be developed for the TD-HCPP.
- Important actual features of TD-HCPP, such as multi vehicles, non-constant travel time function and time windows can be added to the TD-HCPP.

References

Albiach J, Sanchis J, Soler D (2008) An asymetric TSP with time windows and with time-dependent travel times and costs: an exact solution through a graph transformation. Eur J Oper Res 189:789–802

Alfa AS, Liu DQ (1988) Postman routing problem in a hierarchical network. Eng Optim 14:127–138
Arakaki RK, Usberti FL (2018) Hybrid genetic algorithm for the open capacitated arc routing problem. Comput Oper Res 90:221–231
Black D, Eglese R, Wøhlk S (2013) The time-dependent prize-collection arc routing problem. Comput Oper Res 40:526–535
Cabral EA, Gendreau M, Ghiani G, Laporte G (2004) Solving the hierarchical Chinese postman problem as a rural postman problem. Eur J Oper Res 155:44–50
Cerny V (1985) Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. J Optim Theory Appl 45(1):41–51
Cordeau JF, Ghiani G, Guerriero E (2014) Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem. Transp Sci 48:46–58
Damodaran P (1997) A methodology for dynamic planning of road service during a snow fall. M.S. Thesis, Northern Illinois University, DeKalb, IL
Damodaran P, Krishnamurthi M, Srihari K (2008) Lower bounds for Hierarchical Chinese postman problem. Int J Ind Eng 15:36–44
Dhein G, Araújo OCB, Cardoso G (2018) Genetic local search algorithm for a new bi-objective arc routing problem with profit collection and dispersion of vehicles. Expert Syst Appl 92:276–288
Dror M, Stern H, Trudeau P (1987) Postman tour on a graph with precedence relation on arcs. Networks 17:283–294
Fan H, Wang WY (2002) Time-constrained Chinese postman problems. Comput Math Appl 44:375–387
Gaafar LK, Masoud SA (2005) Genetic algorithms and simulated annealing for scheduling in agile manufacturing. Int J Prod Res 43(14):3069–3085
Ghiani G, Improta G (2000) An algorithm for the hierarchical Chinese postman problem. Oper Res Lett 26:27–32
Glover F, Kochenberger GA (2003) Handbook of metaheuristics, chapter 10. Kluwer Academic Publishers, Dordrecht, pp 287–291
Hart E, Ross P, Corne D (2005) Evolutionary scheduling: a review. Genet Program Evolvable Mach 6:191–220
Harwood K, Mumford C, Eglese R (2013) Investigating the use of metaheuristics for solving single vehicle routing problems with time varying traversal costs. J Oper Res Soc 64:34–47
Helvig C, Robins G, Zelikovsky A (2003) The moving-target travelling salesman problem. J Algorithms 49:153–174
Hill AV, Benton WC (1992) Modelling intra-city time-dependent travel speeds for vehicle scheduling problems. J Oper Res Soc 43:343–351
Hua J, Li-shan K (2003) Genetic algorithm for Chinese postman problems. Wuhan Univ J Nat Sci 8(1):316–318
Ichoua S, Gendreau M, Potvin JY (2003) Vehicle dispatching with time-dependent travel times. Eur J Oper Res 144:379–396
Ishibuchi H, Misaki S, Tanaka H (1995) Modified simulated annealing algorithms for the flow shop sequencing problem. Eur J Oper Res 81:388–398
Jiang Q, Sarker R, Abbass H (2005) Tracking moving targets and the non-stationary traveling salesman problem. Complexity 11:171–179
Jiang H, Kang L, Zhang S, Zhu F (2010) Genetic algorithm for mixed Chinese postman problem. In: 5th International Symposium, ISICA 2010, Wuhan China, October 22–24, Proceedings, pp 193–199
Jung S, Haghani A (2001) Genetic algorithm for the time-dependent vehicle routing problem. Transp Res Rec J Transp Res Board 1771:164–171
Kirkpatrick S, Gelatt JRCD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
Korteweg P, Volgenant T (2006) On the hierarchical Chinese postman problem with linear ordered classes. Eur J Oper Res 169:41–52
Kumar SN, Panneerselvam R (2015) A time-dependent vehicle routing problem with time windows for e-commerce supplier site pickups using genetic algorithm. Intell Inf Manag 7:181–198
Lacomme P, Prins C, Sevaux M (2006) A genetic algorithm for a bi-objective capacitated arc routing problem. Comput Oper Res 33(12):3473–3493
Lemieux PF, Campagna L (1984) The snow ploughing problem solved by a graph theory algorithm. Civ Eng Syst 1:337–341

Ma YH, Tian GL, Li X (2015) Genetic algorithm for the capacitated Chinese postman problem on mixed networks. Appl Mech Mater 701:44–49
Maden W, Eglese R, Black D (2009) Vehicle routing and scheduling with time-varying data: a case study. J Oper Res Soc 61:515–522
Malandraki C (1989) Time dependent vehicle routing problems: formulations, solution algorithms and computational experiments. Ph.D. dissertation, Northwestern University, Evanston, IL
Malandraki C, Daskin M (1992) Time dependent vehicle routing problems: formulations, properties, and heuristic algorithms. Transp Sci 26:185–200
Mohammed MA, Ghani MKA, Hamed RI, Mostafa SA, Ahmad MS, Ibrahim DA (2017) Solving vehicle routing problem by using improved genetic algorithm for optimal solution. J Comput Sci 21:255–262
Moon C, Kim J, Choi G, Seo Y (2002) An efficient genetic algorithm for the travelling salesman problem with precedence constraints. Eur J Oper Res 140:606–617
Moreira MR, Ferreira JS (2010) A genetic algorithm for the undirected rural postman problem. https://www.semanticscholar.org
Morgan MJW, Mumford CL (2009) A weight-coded genetic algorithm for the capacitated arc routing problem. In: Proceedings of the 11th annual conference on Genetic and evolutionary computation, Montreal, Québec, Canada, 8–12 July, New York, pp 325–332
Mukhairez HHA, Maghari AYA (2015) Performance comparison of simulated annealing, GA and ACO applied to TSP. Int J Intell Comput Res 6(4):647–654
Rabbani M, Alamdar SF, Farrokhi-Asl H (2016) Capacitated windy rural postman problem with several vehicles: a hybrid multi-objective simulated annealing algorithm. Int J Supply Oper Manag 2(4):1003–1020
Razali NM (2015) An efficient genetic algorithm for large scale vehicle routing problem subject to precedence constraints. Proc Soc Behav Sci 195:1922–1931
Samadi-Dana S, Paydar MM, Jouzdani J (2017) A simulated annealing solution method for robust school bus routing. Int J Oper Res 28(3):307–326
Sayata UB, Desai NP (2015) An algorithm for Hierarchical Chinese postman problem using minimum spanning tree approach based on Kruskals's algorithm. In: Souvenir of the 2015 IEEE International Advance Computing Conference, IACC 7154702, pp 222–227
Schneider J (2002) The time-dependent travelling salesman problem. Physica A 314:151–155
Sun J, Tan G, Hou G (2011a) A new integer programming formulation for the Chinese postman problem with time dependent travel times. World Acad Sci Eng Technol Int J Comput Inf Eng 5(4):410–414
Sun J, Tan G, Qu H (2011b) Dynamic programming algorithm for the time dependent Chinese Postman Problem. J Inf Comput Sci 8(5):833–841
Sun J, Meng Y, Tan G (2015) An integer programming approach for the Chinese postman problem with time-dependent travel time. J Comb Optim 29(3):565–588
Tagmouti M, Gendreau M, Potvin JY (2007) Arc routing problems with time-dependent service costs. Eur J Oper Res 181:30–39
Tagmouti M, Gendreau M, Potvin JY (2010) A variable neighborhood descent heuristic for arc routing problems with time-dependent service costs. Comput Ind Eng 59:954–963
Tagmouti M, Gendreau M, Potvin JY (2011) A dynamic capacitated arc routing problems with time dependent service costs. Transp Res Part C Emerg Technol 19:20–28
Tirkolaee EB, Alinaghian M, Sasi MB, Esfahani MS (2016) Solving a robust capacitated arc routing problem using a hybrid simulated annealing algorithm: a waste collection application. J Ind Eng Manag Stud 3(1):61–76
Verbeeck C, Sörensen K, Aghezzaf EH, Vansteenwegen P (2014) A fast solution method for the time dependent orienteering problem. Eur J Oper Res 236:419–432
Wang L, Zheng DZ (2003) An effective hybrid heuristic for flow shop scheduling. Int J Adv Manuf Technol 21:38–44
Wøhlk S (2015) Simulated annealing for the capacitated arc routing problem, using an online formulation. https://www.researchgate.net/publication/242091823
Yu VF, Lin SW (2015) Iterated greedy heuristic for the time-dependent prize-collection arc routing problem. Comput Ind Eng 90:54–66