RAIRO-Oper. Res. 47 (2013) 399–428
DOI: 10.1051/ro/2013049

RAIRO Operations Research
www.rairo-ro.org

A MULTI-DESTINATION DAILY CARPOOLING PROBLEM AND AN ANT COLONY BASED RESOLUTION METHOD

Yuhan Guo[1,2], Gilles Goncalves[1,2] and Tienté Hsu[1,2]

Abstract. The rising car usage deriving from growth in jobs and residential population causes air pollution, energy waste and consumption of people's time. Public transport cannot be the only answer to this increasing transport demand. Carpooling, which is based on the idea that sets of car owners pick up colleagues while driving to or from the workplace, has emerged to be a viable possibility for reducing private car usage in congested areas. Its actual practice requires a suitable information system support and, the most important, the capability of effectively solving the underlying combinatorial optimization problem. This paper describes an ant colony algorithm based hybrid approach (HAC) for solving the multi-destination carpooling problem. Experiments have been performed to confirm the efficiency and the effectiveness of the approach.

Keywords. Transportation, vehicle routing, carpooling problem, ant colony algorithm.

Mathematics Subject Classification. 90B06.

1. Introduction

Nowadays, along with the increase of population and the dispersion of habitation, public transport service is often incapable of effectively servicing the areas where cost-effective transportation systems cannot be set up. As a result, more and more people use private vehicles for their daily transportation. However, the

Received October 7, 2013. Accepted October 15, 2013.

[1] Univ Lille Nord de France, 59000 Lille, France.

[2] UArtois, LGI2A, 62400, Béthune, France. jonguo.fr@gmail.com

Article published by EDP Sciences

© EDP Sciences, ROADEF, SMAI 2013

high use of private vehicles combined with increased human mobility increases the load on the environment and raises transportation issues such as congestion, parking problem and low transfer velocity.

In order to ease these issues, different innovative mobility services are emerging. Carpooling is a mobility service proposed and organized by large organizations, such as large companies, public administrations and universities. These organizations encourage their employees or students to pick up or take back colleagues or schoolmates while driving to or from a common site. The service tries to decrease the number of private vehicles travel on the road by improving the average car occupancy. In fact, the carpooling has existed for more than 60 years. It first became prominent in the United States as a rationing tactic during World War II. It was popular in the 1970s due to the 1973 oil crisis and the 1979 energy crisis. At that time the first employee carpool programs were organized at Chrysler and 3M. However, since the 1970s carpooling has declined significantly all around the world, it peaked in the 1980 with a commute mode share of 19.7%. But since the 1990s, affected by the increasing cost of petrol and rising number of private vehicles, carpooling came back into the public eye. In the beginning of 20th century, the popularity of the Internet and mobile phones has greatly helped carpooling to expand by enabling people to find and contact carpool members more easily. With such background, the carpooling service now is experiencing the most prosperous time.

The reason why people join the carpooling system is that carpooling reduces travel costs by sharing journey expenses such as fuel, tolls and car rental between the travelers. It is also a more environmentally friendly and sustainable way to travel, as sharing journey reduces carbon emission, traffic congestion and requirement for parking space. Carpooling can also decrease driving stress since each driver has only to drive in one or two days during one week. It also creates increased social interaction between friends, neighbors and colleagues. As a matter of fact, it can enhance the sense of connectedness within the community as a small social network.

After several years of fast development, carpooling has already been considered as an important alternative transportation service throughout the world. As an effort to reduce traffic and encourage carpooling, some countries have introduced high occupancy vehicle (HOV) lanes where only vehicles with two or more passengers are allowed to drive. In some countries it is also common to find parking spaces that are reserved especially for carpoolers. Many companies and local authorities have introduced carpooling schemes, often as part of wider transport programs.

Successful carpooling development has tended to be associated mainly with non-urban areas such as suburbs and more recently universities and other campuses. Currently, most of the carpooling programs are operated on daily basis, where a number of users declare their availability for picking up or bringing back other users to a common destination on one particular day [2,13]. Hence, these users are considered as servers, and the other users being picked up or bringing back are considered as clients. Then the problem becomes to assign clients to servers and to

identify the routes to be driven by the servers. According to this definition, users in a problem are required to go to the same destination. Thus, in the real-world application, the organizer usually separates the users going to different destinations into different carpool instances. Thus, in order to schedule the users with the current daily carpooling system, it is necessary to divide users according to their destinations, and each set of users going to the same destination are considered as an individual instance.

Observed from the real-world application of the daily carpooling, we found that lots of servers travel through their neighbors' destinations before reaching their own ones with available car capacity. However, these servers are not allowed to pick up their neighbors because the neighbors go to different destinations. Dividing users into different carpooling instances based on the destinations results that some instances have redundant servers, while the other instances may lack of servers.

This situation greatly decreases the effectiveness of serving the clients and potentially increased the travel cost of all the users in the daily carpooling, since if a server can pick up and deliver the clients who go to the destinations other than the server's own, the total travel cost can be greatly saved. Thus, a daily carpool model which includes multiple destinations in one instance is required in the real-world application.

In order to respond to this need, a multi-destination daily carpooling model is defined in this paper. In this model, the server can pick up and deliver clients who go to different destinations as long as the server can accept the length of the detour he/she has to make. A concept called "transfer point" is also defined in this model, which means two carpool servers can meet at an intersection point, where the clients can change vehicles in order to decrease the length of the detour the servers have to make.

As abovementioned, in the Multi-destination Daily Carpooling Problem (MDCPP), a number of users declare their availability for picking up or bringing back other users on one particular day. These users are considered as servers, then the other users are assigned to servers as clients and the routes to be driven by the servers are identified. In each carpool group, the server and the clients can have different destinations, and each client can be served by two servers during the transition to the destination. Figure 1 shows the three situations of picking up and delivery considered in our model: (a) a server picks up clients, and then they go to a common destination; (b) a server picks up clients and then leads them to their destinations before going to the server's own destination; (c) a server picks up clients and delivers them to a transfer point, and then another server picks them up from this point and delivers them to their destination.

Based on the mathematical model defined for MDCPP, an efficient and effective metaheuristic is developed to solve this problem in the real-world application. Our Hybrid Ant Colony Algorithm (HAC) is based on the Ant Colony paradigm [4], but we introduce new definitions to the concepts of pheromone and attractiveness. Moreover, a Transfer Point Searching heuristic (TPS) and a local search procedure are integrated into the algorithm in order to identify the transfer points and

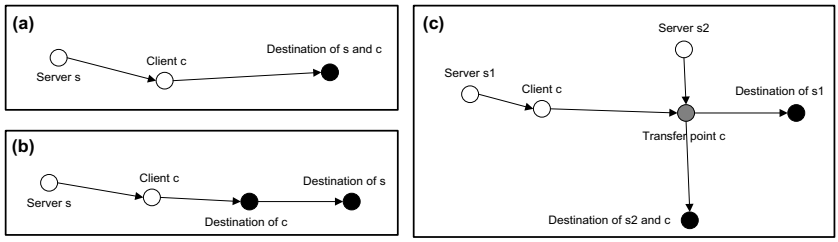

FIGURE 1. Situations of pickup and delivery in Multi-destination Daily Carpooling.

to further optimize the best solutions obtained during iterations. Computational results are reported to illustrate the effectiveness of our approach in solving MD-CPP.

This paper proceeds as follows. Section 2 describes the MDCPP and its mathematical model. Section 3 presents the HAC algorithm for MDCPP. Then, Section 4 illustrates the computational results obtained by our method. The last section gives conclusions and perspectives.

2. Problem definitions and formulation

The mathematical formulation of the MDCPP will be presented in this section. The objective function and constraints to describe the MDCPP will be introduced in detail manner.

2.1. Mathematical model

The MDCPP can be modeled by means of a directed graph $G = (U \cup D \cup O, A)$ with $n + o$ vertices, where set $U = \{1, \ldots, m\}$ is the set of vertices corresponding to the users' home, set $D = \{m+1, \ldots, n\}$ is the set of vertices associated with the destinations, and $A = \{\text{arc}(i, j)/i, j \in U \cup D\}$ is the set of direct links connecting nods i and j where each $\text{arc}(i, j) \in A$ is associated with a positive travel cost $cost_{ij}$ (equals to the distance d_{ij} in our model) and a travel time t_{ij}. The set U is furthermore partitioned as $U = U_s \cup U_c$, where $U_s = \{1, \ldots, m_s\}$ is the subset of vertices associated with servers and $U_c = \{m_s + 1, \ldots, m\}$ is the subset of vertices associated with the clients. Let set $O = \{o(i, j, m, n)/i, j, m, n \in U \cup D\}$ be the set of intersection points where *o(i, j, m, n)* is the intersection point of $\text{arc}(i, j)$ and $\text{arc}(m, n)$. These intersection points are considered as potential transfer points. Each server $s \in U_s$ enlisted in the multidestination daily carpooling specifies the car capacity Q_s and a maximal driving time T_s that the server is willing to accept when picking up clients. Each user $i \in U$ has to specify a destination $d_i \in D$, the earlist time e_i for leaving home and the latest time r_i for arriving at destination.

The MDCPP is a multi-objective problem, requiring minimizing the total travel cost of all servers and the amount of unserved clients. Note that the clients can be left on the transfer point and wait for the next server. So, the waiting time has

become a very important factor to maintain client's satisfaction. Therefore, minimizing the waiting time of all clients is also an objective in MDCPP. In spite of the multiple objectives, it is possible to combine them into a single objective function by using a penalty concept. We define a penalty p_i representing the contribution to the total cost in case a client is not picked up or a server does not pick up any client and a penalty q_i indicating the clients who have to wait in the transfer point. The objectives of MDCPP then can be transformed into an integrated formulation presented as follows.

Define a pool k of users and let user s be the server and a be the amount of members in this pool. Server s of pool k will use his/her car to pick up the other pool members and then deliver them before going to his/her own destination. The driver thus has to find a Hamiltonian path that starts at his/her home, and then passes through all his/her pool members' homes and destinations or transfer points exactly once then ends at his/her own destination. Let $\mathrm{ham}(s,k)$ be the shortest above mentioned path, starting from $s \in \mathrm{pool}\, k$, connecting all pool members' home $j \in \mathrm{pool}\, k \backslash \{s\}$ and transfer points or destinations of all pool members who do not go to the same destination as server s and ending in the destination of server s, with the constraint that destinations and transfer points must be visited after the corresponding clients. The cost for a server driving directly from his/her home to his/her destination is denoted by $cost_{sd}$ (equals to the distance d_{sd} in our model), while p_s is a penalty value incurred when the server travels alone, and q_j is the penalty for making the pool member j waiting on the transfer point after getting off the car, whose calculation is based on the length of the waiting time wt_j. Then, the cost of pool k is defined to be:

$$cost_{\mathrm{pool}}(k) = \begin{cases} cost\left(\mathrm{ham}\left(s,k\right)\right) + \sum\limits_{j \in k \backslash \{s\}} q_j wt_j, & \text{if } |\mathrm{a}| > 1, \\ cost_{sd} + p_s, & \text{otherwise.} \end{cases} \tag{2.1}$$

The cost for an unserved client c is defined as equation (2.2), where p_c is the penalty value for client c being not served.

$$cost_{\mathrm{unserved}}(c) = p_c. \tag{2.2}$$

The total cost of a complete solution of the MDCPP is then defined to be the sum of the costs of the pools in it plus the sum of the costs of the unserved clients. This view optimizes at the same time the three objectives. In our mathematical model, the penalty of a user driving alone is set to be much higher than the cost when he/she drives directly from his/her home to the destination, so it is always more convenient to pool users together than to leave them alone. And because of the penalty for waiting at transfer point, if the client has to wait too long time at the transfer point, it is better to deliver him by the current server instead of changing vehicle.

The transfer points are designed to decrease the travel distance of the servers. Although the transfer points can save the total travel costs, they also increase the inconvenience of the servers and the clients. Moreover, it is very hard to implement

the transfer point accurately in the real world application since the real world paths are always not direct lines. Based on the abovementioned reasons, in our model, we limit the amount of transfer point for each client to at most one.

MDCPP being NP can be easily proven since it is transformed from the DCPP and the DCPP is NP-hard since in a special case it contains the Vehicle Routing Problem with unit client demands [9], which is known to be NP-hard in the strong sense.

2.2. Objective function

The problem can be translated in a four indices formulation considering:

- Decision variables:
  x_{ij}^s: binary variable equals to 1 if and only if arc*(i,j)* is traveled by server s;
  y_i: binary variable equals to 1 if client i is not picked up by any server or server i does not pick up any client;
  ρ_i^s: binary variable equals to 1 if and only if client i is delivered to his/her destination by server s;
  ψ_{mn}^{is}: binary variable equals to 1 if and only if client i is left on transfer point of arc*(m,n)* by server s;
  σ_{mn}^{is}: binary variable equals to 1 if and only if client i is picked up on transfer point of arc*(m,n)* by server s;
  S_i: positive variable denoting the pickup time of client i or the departure time of server i;
  F_i^d: positive variable denoting the arrival time of user i at a destination d;
  L_i: positive variable denoting the arrival time of client i on his/her transfer point, equals to zero if the client does not visit any transfer point;
  H_i: positive variable denoting the pick-up time of client i on his/her transfer point;
  Q_g^k: a set denoting the clients that have been picked up until reaching each point g of server k's tour (including the clients being picked up at point g) where $g \in G_k$ is the set of all the points visited by server k.
- Constants:
  ϕ_{id}: binary value equals to 1 if and only if client i's destination is destination d;
  $\eta_{(m,n)(p,q)}$: binary value equals to 1 if and only if there is an intersection between arc(m, n) and arc(p, q);
  c_{ij}: positive value denoting the travel cost between users i and j;
  t_{ij}: positive value denoting the travel time between users i and j;
  Q_s: positive value denoting the car capacity of server s;
  T_s: positive value indicating the maximum driving time specified by server s;
  e_i: positive value indicating the earliest time for leaving home of user i;
  r_i: positive value indicating the latest time for arriving at the destination of user i;
  p_i: positive value indicating the penalty for drive alone server i or unserved client i;

q_i: positive value indicating the penalty for client i waiting on transfer point;
U_s: index set of all servers;
U_c: index set of all clients;
U: index set of all users;
A: index set of all arcs;
D: index set of all destinations;
O: index set of all intersections.

Objective function:

$$f_{MCPP} = \min \left(\sum_{s \in U_s} \sum_{(i,j) \in A} c_{ij} x_{ij}^s + \sum_{i \in U} p_i y_i + \sum_{i \in U_c} q_i (H_i - L_i) \right) \tag{2.3}$$

$$\sum_{j \in U_c} x_{ij}^s = \sum_{j \in U_c} x_{ji}^s \;\; i \in U_c, s \in U_s \tag{2.4}$$

$$\sum_{i \in U_c} x_{di}^s \leqslant 1 \;\; d \in D, s \in U_s \tag{2.5}$$

$$\sum_{i \in U_c} x_{id}^s \leqslant 1 \;\; d \in D, s \in U_s \tag{2.6}$$

$$\sum_{d \in D} \left(\phi_{sd} \sum_{i \in U_c} x_{di}^s \right) = 0 \;\; s \in U_s \tag{2.7}$$

$$\sum_{d \in D} \left(\phi_{sd} \sum_{i \in U_c} x_{id}^s \right) = 1 \;\; s \in U_s \tag{2.8}$$

$$\sum_{j \in U_c \cup D} x_{ij}^s = \rho_i^s + \sum_{m,n \in U_c \cup D} \psi_{mn}^{is} \;\; i \in U_c, s \in U_s \tag{2.9}$$

$$\sum_{m,n \in U_c \cup D} \sigma_{mn}^{is} \leqslant \rho_i^s \;\; i \in U_c, s \in U_s \tag{2.10}$$

$$\sum_{s \in U_s} \sum_{m,n \in U \cup D} \psi_{mn}^{is} = \sum_{k \in U_s} \sum_{p,q \in U \cup D} \sigma_{pq}^{ik} \;\; i \in U_c \tag{2.11}$$

$$\sum_{d \in D} \left(\phi_{id} \sum_{j \in U_c \cup D} x_{jd}^s \right) \geqslant \rho_i^s \;\; i \in U_c, s \in U_s \tag{2.12}$$

$$(1 - \phi_{sd}) \sum_{i \in U_c} x_{id}^s = (1 - \phi_{sd}) \sum_{i \in U_c} x_{di}^s \;\; d \in D, s \in U_s \tag{2.13}$$

$$x_{mn}^s \geqslant \psi_{mn}^{is} \;\; m, n \in U \cup D, s \in U_s, i \in U_c \tag{2.14}$$

$$x_{mn}^{s} \geqslant \sigma_{mn}^{is} \;\; m, n \in U \cup D, s \in U_s, i \in U_c \tag{2.15}$$

$$\sum_{s \in U_s} \sum_{m,n \in U \cup D} \psi_{mn}^{is} \leqslant 1 \;\; i \in U_c \tag{2.16}$$

$$\sum_{s \in U_s} \sum_{m,n \in U \cup D} \sigma_{mn}^{is} \leqslant 1 \;\; i \in U_c \tag{2.17}$$

$$\eta_{(m,n),(p,q)} \left(\psi_{mn}^{is}\right) = \sigma_{pq}^{ik} \; i \in U_c, m, n, p, q \in U \cup D, s, k \in U_s \tag{2.18}$$

$$\sum_{\mathrm{i} \in Q_{\mathrm{g}}^{s}} \sum_{j \in Q_{\mathrm{g}}^{s} \not\subset D} x_{ij}^{s} - \sum_{\mathrm{i} \in Q_{\mathrm{g}}^{s}} \sum_{d \in D} \left(x_{id}^{s} \sum_{\mathrm{x} \in Q_{\mathrm{g}}^{s}} \phi_{xd} \right) \leqslant Q_s \;\; s \in U_s, g \in G_s \tag{2.19}$$

$$\sum_{s \in U_s} \sum_{j \in U_c \cup D} x_{ij}^{s} + y_i = 1 \;\; i \in U_c \tag{2.20}$$

$$\sum_{(i,j) \in A} x_{ij}^{s} t_{ij} \leqslant T_s \;\; s \in U_s \tag{2.21}$$

$$S_i \geqslant e_i \;\; i \in U \tag{2.22}$$

$$S_j - S_i \geqslant t_{ij} - M \left(1 - \sum_{s \in U_s} x_{ij}^{s}\right) \;\; (i.j) \in A \tag{2.23}$$

$$S_i + t_{id} \leqslant F_i^{d} \leqslant r_i \;\; i \in U, d \in D \tag{2.24}$$

$$S_m - M(1 - \psi_{mn}^{is}) \leqslant L_i \leqslant S_m + t_{mn} + M(1 - \psi_{mn}^{is}) \;\; i \in U_c \, m \in U, n \in U \cup D, s \in U_s \tag{2.25}$$

$$F_s^{m} - M(1 - \psi_{mn}^{is}) \leqslant L_i \leqslant F_s^{m} + t_{mn} + M(1 - \psi_{mn}^{is}) \;\; i \in U_c \, m \in D, n \in U \cup D, s \in U_s \tag{2.26}$$

$$S_p - M(1 - \sigma_{pq}^{ik}) \leqslant H_i \leqslant S_p + t_{pq} + M(1 - \sigma_{pq}^{ik}) \;\; i \in U_c \, p \in U, q \in U \cup D, k \in U_s \tag{2.27}$$

$$F_k^{p} - M(1 - \sigma_{pq}^{ik}) \leqslant H_i \leqslant F_k^{p} + t_{pq} + M(1 - \sigma_{pq}^{ik}) \;\; i \in U_c \, p \in D, q \in U \cup D, k \in U_s \tag{2.28}$$

$$H_i \geqslant L_i \;\; i \in U_c \tag{2.29}$$

$$x_{ij}^{s} \in \{0,1\} \quad s \in U_s, (i,j) \in A \tag{2.30}$$

$$y_i \in \{0,1\} \quad i \in U_c \tag{2.31}$$

$$\rho_i^s \in \{0,1\} \quad i \in U_c\,, s \in U_s \tag{2.32}$$

$$\psi_{mn}^{is} \in \{0,1\} \quad i \in U_c\,, m,n \in U \cup D, s \in U_s \tag{2.33}$$

$$\sigma_{pq}^{ik} \in \{0,1\} \quad i \in U_c\,, p,q \in U \cup D, k \in U_s \tag{2.34}$$

$$S_i \geqslant 0 \quad i \in U_c \tag{2.35}$$

$$F_i \geqslant 0 \quad i \in U_c \tag{2.36}$$

$$L_i \geqslant 0 \quad i \in U_c \tag{2.37}$$

$$H_i \geqslant 0 \quad i \in U_c. \tag{2.38}$$

Equation (2.4) is continuity constraint for visiting clients. Equation (2.5) forces the server to go from each destination at most once, while equation (2.6) restricts the server to go to each destination at most once. Equation (2.7) forces the server's path ends at server's destination. Equation (2.8) makes sure each server's destination must be visited. Equations (2.9) and (2.10) force each picked up client must be delivered to the client's destination. Equation (2.11) makes sure a client left on a transfer point must be picked up eventually. Equation (2.12) restricts the client's destination must be visited if the client is served by a server. Equation (2.13) is continuity constraint for visiting the destinations. Equation (2.14) restricts the transfer point to leave a client must on the path of a server, while equation (2.15) confirms the transfer point to pick up a client must on the path of a server, respectively. Equation (2.16) forces a client can be left on the transfer point at most once, and equation (2.17) restricts a client can be pickup at the transfer point at most once, respectively. Equation (2.18) confirms there must be an intersection between the paths of leaving and picking up a client. Equation (2.19) is car capacity constraint. Equation (2.20) forces each user to be served by only one server or to be penalized, while equation (2.21) is maximum driving time constraints, respectively. Equations (2.22) and (2.23), where M is a big constant, collectively set feasible pick-up times, while equation (2.24) sets minimum and maximum values of feasible arrival times, respectively. Equations (2.25) and (2.26), where M is a big constant, collectively set feasible time to leave a client on a transfer point, while equations (2.27) to (2.29) set feasible time to pick up a client from a transfer point, respectively. Constraints (2.30) to (2.34) are the binary constraints, and constraints (2.35) to (2.38) are the positivity constraints.

3. Hybrid ant colony algorithm for the MDCPP

In this section, we explain in detail the concepts and the structure of our Hybrid Ant Colony algorithm (HAC) for the MDCPP. The adaptation of the different components for the MDCPP is described and examined.

3.1. Related works

The interesting methods for solving the classic DCPP are presented as follows. Some authors [7, 10, 12, 14] define a few simple matching rules to build carpools in order to obtain fast matching speed; however, a good solution quality cannot be guaranteed. For instance, in the work of Kothari [10], a multi-agent carpooling system called Genghis system is developed. The authors define a few fundamental constraints for choosing reasonable matches between clients and servers. An algorithm is developed to generate route proposals for the users. The algorithm picks a client only once and continues matching the client to proposed servers until all the proposed servers are examined. The proposed server is selected based on the distance between the client and the direct route from the server to the server's destination. For each client, the algorithm proposes a maximum 5 available drivers, then for each proposed driver, the algorithm matches him to the client in a hierarchy of 4 Levels. Each level consists in a matching constraint, such as user preference, distance constraint, time window constraint and cost constraint. The client may be rejected at any level of the matching. The Genghis system is designed for solving small size instances in an efficient way, but minimizing the total travel cost is not taken into consideration in this system.

To obtain a good solution quality, other authors use heuristics and exact algorithms to solve the DCPP. The interesting studies include a distributed geographic information system [3] and an exact algorithm based on Lagrangian Column Generation [1]. In the distributed geographic information system the authors present an integrated system for the organization of a carpooling service. The core of the system is an optimization module which solves heuristically the specific routing problem. The procedure includes a construction phase and a local optimization phase. The algorithm starts with no employee routed and an idle fleet of vehicles. A number of new routes equal to the number of servers are initialized as direct paths from each server to the destination, then, as long as possible, single clients are inserted into existing routes with a greedy algorithm. Then, from the initial solution produced by the procedure described above, better ones are obtained by means of a local search algorithm. In the study of the exact algorithm based on Lagrangian Column Generation, authors propose an exact method for the carpooling problem. This method is based on two integer programming formulations of the DCPP. The first formulation is a commodity flow formulation using three-index variables, while the second formulation models the DCPP as a set-partitioning problem whose variables correspond to feasible paths or to clients to be left unserved. A valid lower bound on the optimal DCPP cost is computed as the cost of

a feasible dual solution of the LP relaxation [8] of the set-partitioning problem; the solution is obtained by combining three different relaxations of the two formulations. The dual solution and a valid upper bound obtained by a heuristic algorithm based on the bounding procedure are then used to eliminate feasible paths that cannot belong to any optimal solution; thus the resulting reduced set-partitioning problem can be solved by a branch-and-bound algorithm. The main contributions of this research are the new bounding procedures for computing a feasible dual solution of the set-partitioning formulation and the method for generating a reduced set-partitioning problem that is used to find an optimal solution.

3.2. Main structure

Our HAC approach for solving MDCPP is based on the Ant Colony Optimization paradigm [5,6]; the approach tries to assign clients to servers during the ants making their tour. The HAC consists in four components, a pre-sorting process, an ant colony optimization based metaheuristic, a transfer point searching heuristic for determining the transfer points and a local search for further optimizing the solutions. The ultimate goal of this approach is to solve the MDCPP efficiently and obtain a good solution with limited exploration to the search space.

The pre-sorting procedure is designed to partition servers and clients according to several constraints defined based on the servers' convenience, in order to aid and facilitate the future approaches. In this procedure, the servers will be divided into two subsets based on their availability to pass other destinations before going to their own ones: servers who are able to visit other destinations; and servers who can only go to their own destinations. The two subsets are used as the start points to build carpools for the ants in the ant colony optimization based metaheuristic. In the same manner, the clients will also be grouped according to their destinations, in order to narrow down the candidates for the ants to assign to a server.

On the basis of the pre-sorting procedure, the ant colony based metaheuristic is then applied. The approach is based on the Ant Colony Algorithm, but its concepts of pheromone information and attractiveness are redefined. The pheromone is replaced by a preference mechanism which is designed as the tendencies of assigning a client to a server, while the attractiveness between a client and a server is defined according to the linear distance between the client and the straight line connecting the server and the destinations he/she has to visit.

The preference and attractiveness information are used to guide the client assignment behavior of the ants, in order to achieve the carpool construction. The first step of the ant colony based metaheuristic is to initialize the preference information. Then, when the ant starts to construct a solution, it will first select a server from the server subsets as the start point to start a new carpool, and then if the car capacity and time window constraints are satisfied, it will continue searching for clients, whose destinations are available to the server, to assign to the server. The probability for the ant to select a client to visit and assign him to a server is based on the preference information and the attractiveness value between the client and the server. During the construction of the carpool, if pooling

a client violates the time window constraints, the ant will try to select another client. If the maximum number of times of selection is exceeded, the ant will end the carpool, and stochastically select another server as a new start point of a new carpool.

After a solution has been constructed by an ant, the Transfer Point Searching heuristic (TPS) is applied to create transfer points for carpools. The heuristic proceeds as follows: suppose server s makes a relatively long detour to visit client i's destination, the heuristic will then try to locate another server k that is going to the same destination as client i, and create a transfer point for server s to transfer client i to server k, so the total travel cost can be decreased.

When all ants finish their tours, several best solutions are selected based on the objective function. A local search is then applied to further optimize these selected solutions. The main structure of the local search consists in a loop applying sequentially two operators, and it stops when no improvement made during x loops.

At the end of iteration, the composition of the best solutions will be used to update the preference information, and the ancient preference values will decrease with an evaporate rate, in order to enlarge the influence of the new preference information obtained in the current iteration.

The general structure of the HAC is specified as following Algorithm 3.1.

***Algorithm 3.1*:** *Hybrid ant colony algorithm*

```
/* Pre-sorting process */
Partition the servers according to their ability to pass to other destinations;
Partition the clients according to their destinations;

/* Ant colony optimization based metaheuristic */
Initialize preference and attractiveness.
 While the stop criteria are not met
   For k =1, k ⩽ the number of ants do
     Repeat
     Select start point for ants based on the partition of servers;
     Assign clients to servers based on preference attractiveness
     and the availability of the server for the clients' destinations;
     Until all servers are selected;
     Apply TPS to decide the transfer points;
   End for
   Select the best m solutions;
   Apply local search to the selected solutions;
   Update the preference based on the composition of selected solutions;
 End while
```

3.3. Solution representation

The aim to design a representation for the solution is to build a suitable mapping between our problem and the solution generated by the algorithm. Although both direct and indirect coding have been proven to be applicable for the representation of vehicle routing related problem, we favor to select the direct coding for the

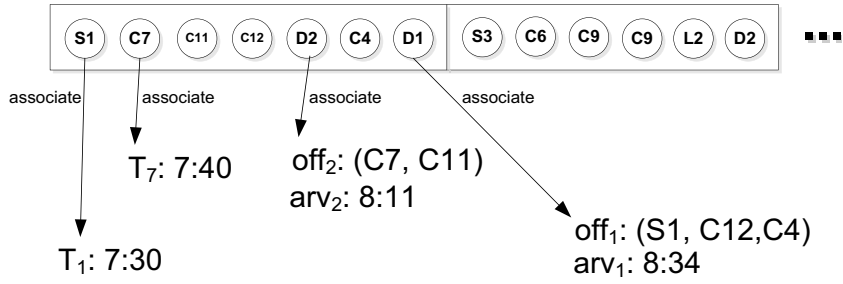

FIGURE 2. An example of the solution representation.

MDCPP, since the time-consuming encoding and decoding phase of the indirect coding can be avoided. The representation is designed with two layers. The first layer presents the match between servers and clients, as well as the pickup order of the clients and the visit orders of destinations and transfer points. The second layer records the departure time of the servers, the pickup time of clients and their arrival time at the destinations or the transfer points.

Therefore, the first layer of the proposed representation consists in a set of clusters $Rep = \{R_1, R_2, \ldots, R_m\}$, and each cluster $R_i = \{S_j, C_k, \ldots, C_p, L_i, D_s\}$ is a ordered sequence, started with the server of this cluster S_{j} and followed with clients C_k or transfer point (L_i indicates the transfer points to leave the clients on, P_i refers the transfer points to pick up clients from) or destinations D_s based on the order they are visited by this server. Note that the representation of each cluster may have different length, since the length is based on the number of clients and destinations visited by the server.

Then in the second layer, for each server S_j and client C_k in each cluster, the departure time T_j or pickup time T_k are associated. For each destination D_s, a value arv_s indicating the arrival time of the server at this destination and a set off_s including the clients who get off the vehicle at this destination are associated.

Note that some clients may correspond to a transfer point; the detail representation concerning this situation will be introduced in Section 3.7. This representation provides all the information of a user in the Multi-destination Daily Carpooling Problem. It offers and contributes in a clear manner to design multi-destination daily carpooling problem solutions. An example of the solution representation is shown in Figure 2.

3.4. Pre-sorting

Pre-sorting categorizes both servers and clients, it is designed as a preparation procedure for the ant colony optimization based metaheuristic. Two constraints are defined to aid the categorizations of servers and clients.

The servers are divided into two subsets, the M-server subset which contains the servers who are able to visit more than one destination within their maximum

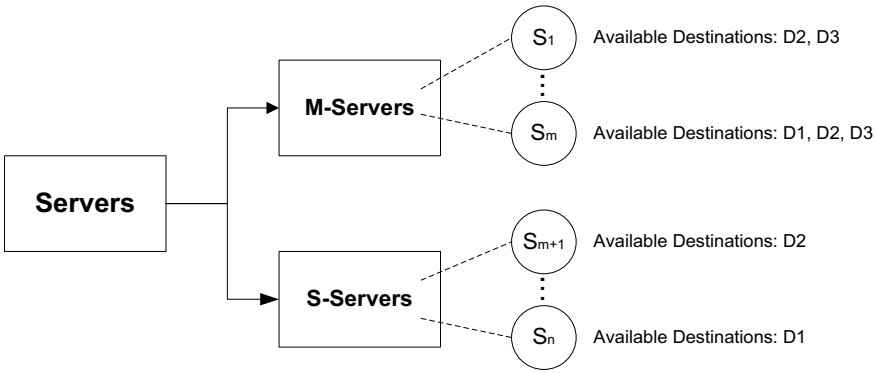

FIGURE 3. Categorization of servers in pre-sorting procedure.

driving time, and the S-server subset which consists in the servers who can only go to their own destinations. The selection of the members of each subset is performed by constraints (3.1) and (3.2). The evaluation takes place between each server and each destination except the server's own one

$$\mathrm{dis}_{sn} + \mathrm{dis}_{nm} < z \times \mathrm{dis}_{sm} \tag{3.1}$$

$$t_{sn} + t_{nm} < T_s \tag{3.2}$$

where dis_{sm} is the distance between server s and his/her own destination m, t_{sn} is the travel time between server s and another destination n, t_{nm} is the travel time between destination n and destination m, z is a parameter set to adjust the maximum detour length the server can afford, and T_s is the maximum driving time a server willing to accept respectively.

Constraint (3.1) restricts that the length of a server's detour cannot be longer than z times of the distance he/she travels to his/her own destination. Constraint (3.2) confirms that travel to destination n will not exceed the server s's maximum driving time. If server s is able to satisfy the two constraints when he/she travels to any destination other than his/her own, he/she will be categorized into the M-server subset. Otherwise, the server will be put in the S-server subset. For each processed server, his/her available destinations will be recorded with him, shown in Figure 3.

These two subsets will be both considered as the start points for building carpools for the ants in the ant colony optimization based metaheuristic, but they are given different priorities when ants process them. The servers in the M-server subset must be processed before the ones in the S-server subset, only after all servers in the M-server subset have been assigned with carpool members, the ants start to process the servers in the S-server subset. By defining this concept, we achieve the favor to the servers who can travel to multiple destinations, as they can serve more clients than the servers who go to only their own destinations.

Preference matrix in HAC

	C_1	C_2	C_3	C_4	...	C_n
S_1	0	0.5	1.1	0	0.5	0
S_2	0.5	0	1.4	0.7	0	0
⋮	1.0	0.2	0	0	0	0
S_m	0	0.7	0	0	0.8	1.2

FIGURE 4. The preference information matrix in HAC.

The clients are also divided into several subsets; the number of subsets is based on the number of destinations of an instance. The clients going to the same destinations are organized into the same subset. In the ant colony optimization based metaheuristic, when an ant searches clients to assign to a server, only the clients going to the available destinations of the server are visible to the ant. That is, only the clients with the destinations the server can reach without violating the constraints (3.1) and (3.2) are candidates for the ant to select. By applying this mechanism, the amount of candidates for an ant has been significantly decreased.

3.5. PREFERENCE INFORMATION

In our metaheuristic, the pheromone concept of the classic Ant Colony Algorithm is replaced by a preference concept, which is defined between each server and each client. The preference information is stored in an $m \times n$ matrix where m is the number of servers and n is the number of clients in an instance. The preference values of the matrix indicate the tendency level to assign a client to a server, as shown in Figure 4. These values are experiences gained from the best solutions obtained in each iteration, and are used to guide the ant for constructing carpools in future iterations.

In the initialization of the preference information, several time window constraints are pre-checked between the servers and the clients whose destinations are available for the servers. The preference values between the server and the clients whose destinations are not feasible for the servers are set to be zero. Let d_s be the destination of server s, d_i be the destination of client i, t_{ij} be the travel time between two locations, respectively. Constraints (3.3) and (3.4) examine whether server s and client i can both be able to arrive on time, if server s picks up client i before going to the destination. Constraint (3.5) checks if the pickup time of client i is too late for server s to arrive at the destination on time. Note that, if client i has a different destination from server s, the server must deliver the

client first, and then go to his/her own destination. If pooling server s and client i together cannot satisfy the abovementioned constraints, the preference value w_{si} is set to zero, which indicates there is no probability that client i will be assigned to server s by ants. By applying this procedure, we are able to remove some carpool combinations which do not belong to any feasible solution, so the computing time for the ants to search for carpool members is further decreased.

$$e_s + t_{si} + t_{id_i} + t_{d_i d_s} \leqslant r_s \tag{3.3}$$

$$e_s + t_{si} + t_{id_i} \leqslant r_i \tag{3.4}$$

$$e_i + t_{id_i} + t_{d_i d_s} \leqslant r_s. \tag{3.5}$$

Then, if the constraints are satisfied, the preference values between each server and each client are initialized by the geographic distance difference between server s going to his/her destination directly and server s picking up and delivering the client i before going to his/her own destination, and their earliest departure time difference, shown as equation (3.6).

$$\begin{aligned} w_{si} &= \alpha \times D + \beta \times E \\ D &= \begin{cases} \frac{1}{\mathrm{dis}_{si}+\mathrm{dis}_{in}+\mathrm{dis}_{nm}-\mathrm{dis}_{sm}} & \text{if } \frac{1}{\mathrm{dis}_{si}+\mathrm{dis}_{in}+\mathrm{dis}_{nm}-\mathrm{dis}_{sm}} \geqslant \sigma \\ \sigma & \text{otherwise} \end{cases} \\ E &= \begin{cases} \frac{1}{|e_s+t_{si}-e_i|} & \text{if } \frac{1}{|e_s+t_{si}-e_i|} \geqslant \sigma \\ \sigma & \text{otherwise} \end{cases} \end{aligned} \tag{3.6}$$

where dis_{si}, dis_{in}, dis_{nm} and dis_{sm} are the geographical distances between server s and client i, between client i and i's destination n, between client i's destination n and server s's destination m, and between server s and s's destination m, respectively. t_{si} are the travel time between server s and client i. e_s and e_i are the earliest departure time of server s and client i. α and β are weight factors. Equation (3.6) indicates that the shorter the detour the server s has to make to pick up client i, the higher the initial preference between them. If the arrival time of server s at client i's home is close to client i's earliest departure time, they will also obtain a higher preference value between them. Note that the constants in the abovementioned equation have both distance unit and time unit, so the factors α and β are designed to adjust these values. Moreover, the two denominators in equation (3.6) are limited to a minimum value σ, which replaces the denominators if their values are less than σ. This mechanism is designed to avoid generating too large preference values.

In each iteration, an ant starts its tour from a server s, selected stochastically from the M-server set obtained in pre-sorting procedure. Then the ant tries to assign clients to server s. The probability for the ant to select a client i to visit and assign to server s is based on the preference information and attractiveness between client i and server s. The attractiveness will be introduced in detail in next section. The probability for the ant to select a client i is performed by a roulette

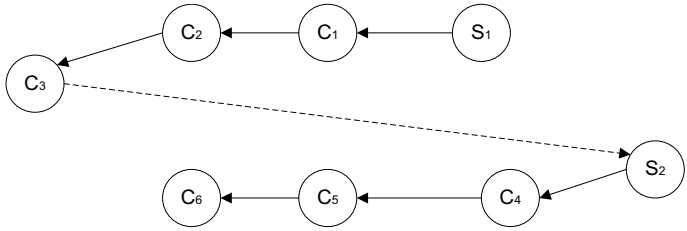

FIGURE 5. The procedure of ant searching for carpool members.

wheel selection procedure, as shown in equation (3.7). As abovementioned, only the clients with the destinations can be visited by server s without violating the constraints have positive preference values, which indicates they are visible for the ant to select.

$$\text{Probability}_{si} = \frac{(w_{si})^a (\eta_{si})^b}{\sum_{j \in H} (w_{sj})^a (\eta_{sj})^b} \tag{3.7}$$

where w_{si} is the preference value between server s and client i, while η_{si} is the attractiveness value, respectively. H is the set of visible candidates of server s, who have not been assigned to any carpool yet, while a and b are adjusting parameters, respectively.

If the client selected by the ant can satisfy the time window constraints of the server and the existing clients in the carpool, the selected client will be added into the carpool. Otherwise, the selection is considered as failed and the ant will try to select another client until the maximum number of times of failed selection y is exceeded. When a new client is assigned to the carpool, the new order of pickup and delivery of the clients will be updated. The update is based on the calculation of the attractiveness, which will be presented in Section 3.6.

When the current constructing carpool reaches its car capacity or exceeds the maximum number of times of failed selection, the carpool will be closed. And then the ant will select stochastically another server from the M-server, or S-server if the servers in M-server subset are all processed, then continue to assign clients to the server as shown in Figure 5.

After a solution has been constructed by an ant, the Transfer Point Searching heuristic (TPS) is applied to create the transfer points. The TPS will be presented in detail in Section 3.7. When the transfer point is built, the waiting time of client i on the transfer point is calculated and associated to him, in order to aid the evaluation at the end of iteration.

When all ants finish their tour, an evaluation is performed to all the solutions generated by the ants; the evaluation is given by equation (2.3), which is the objective function of the MDCPP. After the evaluation, the first m best-fit solutions are selected to be applied with a local search, in order to further optimize the solution.

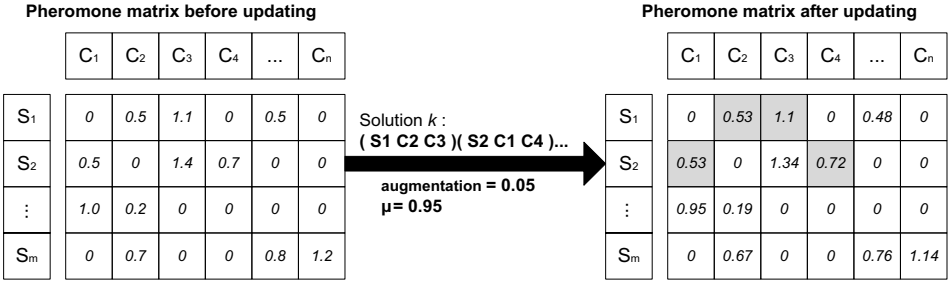

FIGURE 6. Updating the preference matrix in HAC.

Then, before updating the preference values with these solutions, all weight values w_{si} in the preference information matrix will decrease with an evaporate rate μ, as shown in equation (3.8), where $w_{si}^{''}$ is the preference value of the previous iteration, in order to enlarge the influence of the new preference information obtained in current iteration. Afterwards, for each selected solution s, the preference values between the server and the clients in the same carpool consist in an augmentation, as shown in equation (3.9), where $w_{si}^{'}$ is the preference after evaporation, f_{avg} is the average fitness of the whole ant colony, f_k is the fitness of current selected solution s, S is the set of all selected solutions, and factor λ is a weight factor, respectively. For the clients who have been transferred during the travel, only the preference value between the clients and their original servers are updated.

$$w_{si}' = \mu \times w_{si}'' \tag{3.8}$$

$$w_{si} = w_{si}' + \lambda \sum_{s \in S} \frac{f_{\text{avg}} - f_s}{f_{\text{avg}}}. \tag{3.9}$$

3.6. ATTRACTIVENESS

The paradigm of the ant colony optimization (ACO) involves the movement of a colony of ants through the different states influenced by two local decision policies, pheromone and attractiveness. In our ant colony optimization based metaheuristic, the attractiveness is considered as the level a client is attracted by the theoretical shortest path of the server. Therefore, the attractiveness between client i and server s is defined as the reciprocal of smallest linear distance between client i and the straight lines connecting server s and the destinations the server has to pass.

During the ant colony optimization based metaheuristic, when the ant is searching client i for server s, a "standard path" is designed for calculating the attractiveness. The standard path is a Hamilton path starts from server s and connects sequentially the destinations the server has to visit to deliver his/her current carpool members as well as client i, as shown in Figure 7. Thus the standard path is modified if client i goes to a different destination.

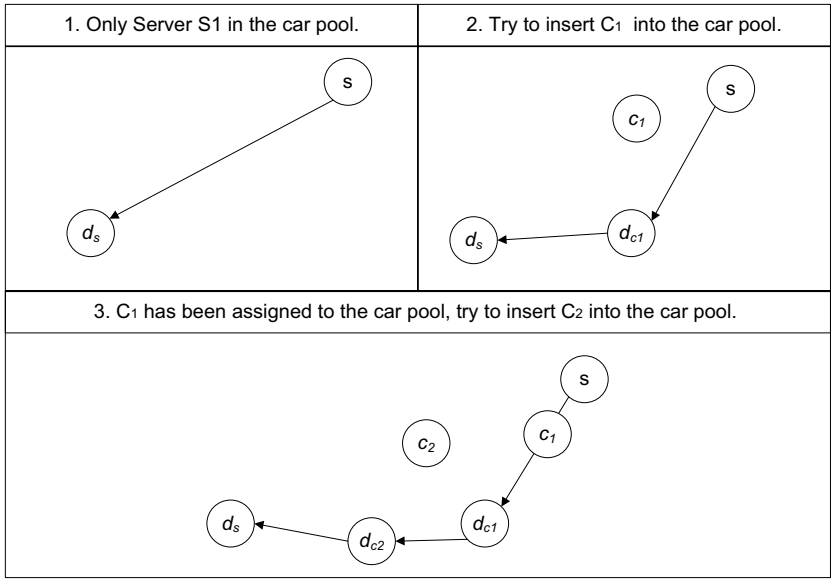

FIGURE 7. The modification of the standard path.

For instance, in Figure 7, server s's destination is d_s, and the two clients are with destination d_{c1} and d_{c2}. Then, in the beginning of a carpool construction, the standard path is the section of line connecting the server s and the server s's destination. When the ant tries to insert client c_1 with different destination into the carpool, the standard path is changed to the Hamilton path starting from the server, passing the destination of c_1 and ending at server s's destination. In the same manner, when the ant tries to insert client c_2 into the carpool the standard path of this carpool is the Hamilton path starts from server's home connecting d_{c1} and d_{c2}, ends at d_s.

As the standard path is the shortest path to go through the destinations for the server, if the server picks up the clients close to this path, he/she will make only relatively short detour. Thus, the standard path is used as the guide line to calculate the attractiveness for guiding the ants; the concept is designed to ensure that the distribution of selected clients vibrates around the standard path with a narrow width.

When ant makes its tour, the attractiveness is calculated according to the standard path of the server. We calculate the linear distance between the candidate client c and each section of the standard path before reaching c's destination, and then choose the reciprocal of shortest one as the attractiveness value of this client to the server, as shown in Figure 8. Note that if the projection point of the client is not on the corresponding section of the standard path, an extra distance will be added to the attractiveness distance. The extra distance is calculated as the linear

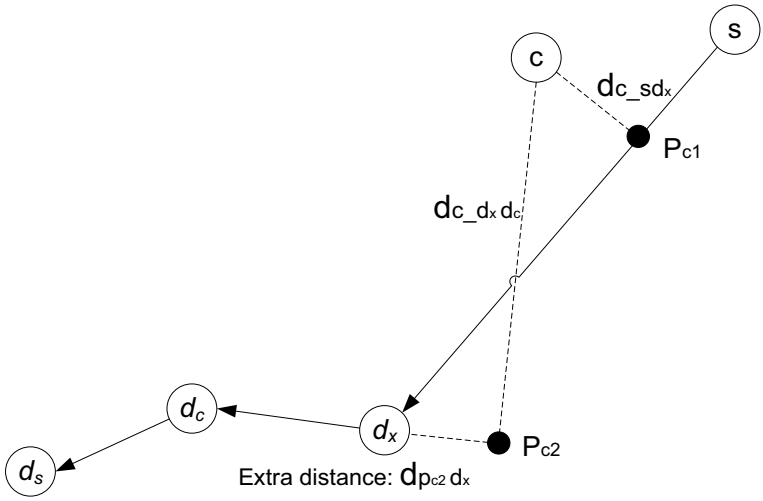

FIGURE 8. The distance between a client and the standard path.

distance between the projection point and the closest end of the section of the standard path. In Figure 8, server s's destination is d_s, candidate client c's destination is d_c, and d_x is the destination of an existing client in server s's carpool. d_{c_mn} indicates the linear distance from client c to the section between node m and node n, and P refers to the projection points of client c on each section of the standard path. In the example, the distances between c and the standard paths are d_{c_sdx} and $d_{c\_dxdc} + d_{pc2dx}$. Thus, the attractiveness value between c and s is calculated as the reciprocal of d_{c_sdx}, since d_{c_sdx} is shorter than $d_{c\_dxdc} + d_{pc2dx}$.

Note that only the distances between client c and the sections of the standard path before reaching c's destination can be calculated and used. In Figure 8 client c can only have the projection point on the section sd_x and d_xd_c of the standard path since his/her destination is d_c. The distance between client c and the sections d_cd_s of the standard path is not calculated because, even if the distance was the shortest, the server has to travel to an opposite direction to deliver the client. This is usually inconvenient and expensive for the server, thus it is better to leave the client to other servers.

At last, according to the section of the standard path selected to calculate the attractiveness and the location of the corresponding projection point, we are able to obtain the position for a client to be inserted in the server's pickup and delivery sequence. For instance, in Figure 8, client c will be inserted in the pickup and delivery sequence between s and d_x, since the attractiveness of c is calculated based on the distance to section sd_x.

Therefore, the distance dat_c for calculating the attractiveness is calculated as equation (3.10). By connecting client c to the two ends of section mn of the standard path, we can obtain a triangle. If $\hat{cmn}$ and $\hat{cnm}$ are acute-angles or

right-angles, dat_c simply equals to the distance d_{c_mn} between client c and section mn. Otherwise, dat_c is added with an extra distance $\min(d_{pm}, d_{pn})$.

$$dat_c = \begin{cases} d_{c\_mn} & \text{if } \hat{cmn} \text{ and } \hat{cnm} \leqslant 90^\circ \\ d_{c\_mn} + \min(d_{pm}, d_{pn}) & \text{otherwise} \end{cases} \tag{3.10}$$

where d_{c_mn} indicates the linear distance from client c to the section between m and n, while d_{pm} and d_{pn} indicates the linear distances from projection point p of client c to m and n.

And the attractiveness for client c to a server s will be:

$$\eta_{sc} = \frac{1}{\min(dat_c)} \tag{3.11}$$

where dat_c indicates all possible distances from client c to the sections of the standard path.

When a new client is assigned to a carpool, according to its projection point on the standard path, the pickup and delivery sequence will be updated. The client with the projection point closer to the server will be picked up first. The attractiveness is essential in our ant colony optimization based metaheuristic. Its role is not only a factor for ants to select clients to assign to servers, but also to indicate the position where the client should be located in the pickup and delivery sequence.

3.7. Transfer point searching heuristic

After a solution has been constructed by an ant, the Transfer Point Searching heuristic (TPS) is applied to create the transfer points. As mentioned in the mathematical model, the transfer point can increase the inconvenience of the servers and the clients. Thus, in our model, we limit the number of transfer points to at most one for each client.

In TPS, we define a parameter to decide whether a detour of a server is considered to be long. In a solution, if server s is confirmed to make a long detour to deliver client i, the heuristic will try to locate another server k that is going to the same destination as client i and has available car capacity, then try to create a transfer point for server s to transfer client i to server k, in order to decrease the total travel cost.

Two conditions are used to select servers: (1) the server must travel through multiple destinations; (2) the server's travel distance exceeds w times of the direct travel distance from his/her home to his/her destination. The servers satisfying both of these two conditions are put into a list. The list is organized in decrease order of the length of the servers' detours, and the server with longer detour will be processed first. Then, all the servers in the list proceed with the following procedures, as shown in Figure 9.

Suppose s is a server with destination d_s, then for every other destination d_i server s has to visit (in the case of Fig. 9, d_1 is the other destination server has to

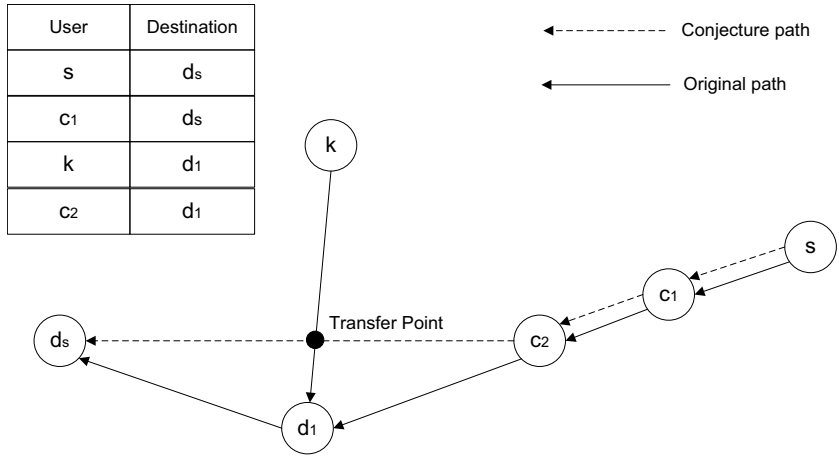

FIGURE 9. Construction of a transfer point.

visit), if there is no existing transfer point on the paths linking d_i (in Fig. 9, the paths are c_2d_1 and d_1d_s) and no client going to d_i has been transferred before, we remove d_i from the original path to construct a conjecture path, and the clients going to d_i are marked to be transferred in the next step. Otherwise, we skip d_i and process the next destination. Then, we select randomly a server who goes through destination d_i and examine the availability to construct a transfer point. The process for destination d_i continues until a transfer point is created or all possible servers who go through destination d_i are checked.

A server k must satisfy the following constraints to create a transfer point:

(1) there must be an intersection between server k's path and server s's conjecture path;
(2) the intersection must be on the part of the conjecture path which is visited after picking up the clients of server s who need to be transferred;
(3) server$k$ must have available car capacity to serve the clients transferred from server s;
(4) calculate the coordinates of the transfer point, and then check the arrival time of both servers at the transfer point. The time for server k to pick up the clients must be later than the time for server s to leave the clients.

The constraints are examined sequentially; the next constraint is examined only if the previous constraint is satisfied. Note that, the coordinates calculated in the fourth constraint are memorized, so the repetitive calculation can be avoided in the future iterations.

Since the verification of the constraints is time consuming, we apply the first improvement policy. That is, the transfer point is confirmed as soon as the total cost of the two carpools decreases. Note that, the path of server s will be modified to be same as the conjecture path after the creation of the transfer point.

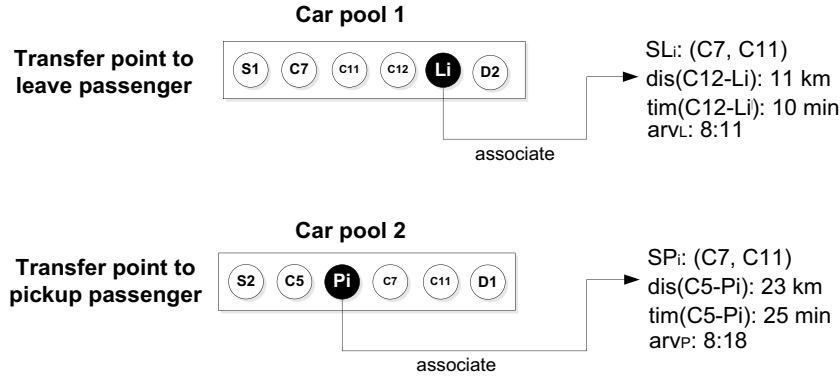

FIGURE 10. Representation of a transfer point.

The representation of the transfer point in a solution is introduced in Figure 10 The transfer point to leave clients is named with L_i, while the transfer point to pick up the corresponding clients is named with P_i. Each transfer point i is associated with set SL_i of clients being left or set SP_i of clients being picked up, the distance dis and travel time *tim* between the previous visited node and transfer point i, and the time *arv* of the server arriving at this transfer point.

3.8. Local search procedure

As abovementioned, when all ants have finished their tours and the TPS is applied, several best solutions are selected to be improved by a local search procedure. The main structure of local search consists in a loop applying sequentially two operators. The local search stops when no improvement made during x loops.

***Algorithm 3.2**: Local search in HAC*

```
Local_Search_Operators[] = {Swap, Move}
Create the forbidden list;
Repeat
  For each operator in Local_Search_Operators do
    Select carpools for operator;
      For selected carpools do
        Apply the operator;
          If solution is improved, update the solution;
      End for
  End for
Until the stop criteria of the local search is met.
Clear the forbidden list.
```

The two operators are designed specifically based on the characteristic of the MDCPP. The main structures of the operators are similar. For each operator, we

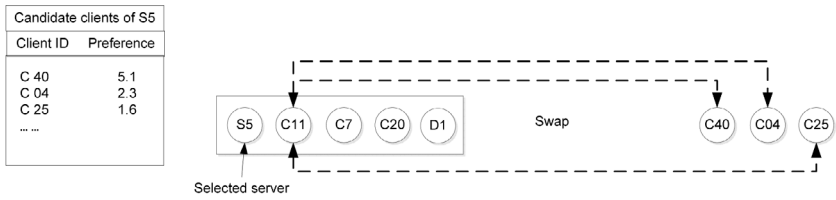

FIGURE 11. An example of the operation of swap operator.

first select several carpools according to the selection rules defined by each operator. Then, for each operation performed by an operator, we apply the operator and see whether there is an improvement obtained. If the solution is improved, we update the solution.

Swap operator

The operator first stochastically selects u servers who make long detours. The server satisfying both of the following two conditions: 1) the server must travel through multiple destinations; 2) the server's travel distance exceeds v times of the direct travel distance from his/her home to his/her destination, is considered making a long detour. The length of detour is calculated as the distance difference between the server's travel distance and the distance from server's home directly to his/her destination. The selection is performed by a roulette wheel selection based on the length of the detour made by the server. The probability of server s to be selected is calculated by equation (3.12).

$$\mathrm{Pd}_s = \frac{\mathrm{detour}_s}{\sum_{k \in K} \mathrm{detour}_k} \tag{3.12}$$

where detour_s is the length of detour of server s and K is the set of all servers.

For each selected server s, the operator tries to swap every existing carpool member of server s with any client j whose destination is visited by s and has positive preference value to s. The move is confirmed as soon as an improvement is obtained.

Note that, the clients being transferred cannot be swapped, in order to avoid affecting the schedule of the carpool where the clients being transferred from or to. An example of the operation of the swap operator is presented in Figure 11.

Move operator

The move operator tries to move a client from a selected carpool to a server with available car capacity. It first selects t servers who make long detours. The selection is performed by a roulette wheel selection still based on equation (3.12).

For each selected server s, the operator attempts to move every carpool member i to any server who passes i's destination, and has positive preference value to i and available car capacity, as shown in Figure 12. The move is confirmed as soon

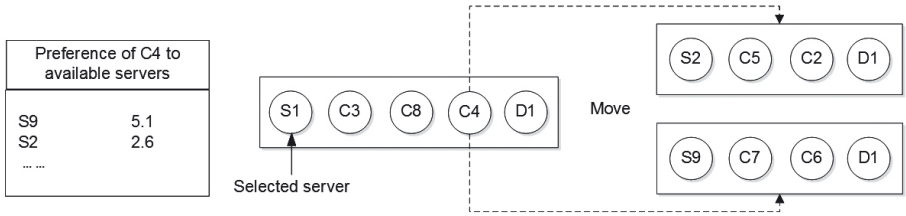

FIGURE 12. An example of the operation of move operator.

as an improvement is obtained. Following the same manner as the swap operator, the clients corresponding to the transfer points cannot be moved.

4. Experimental results and analysis

4.1. Benchmarks

Computational experiments have been conducted to examine the performance of the proposed approach. Since no literature can be found on MDCPP, we created three new sets of instance. The benchmark used in our experiments includes three sets of structurally different problem instances. Two of them are transformed from the benchmark of a similar problem and the other one is obtained based on the real-world case.

The first two sets of instances were originally derived from the Pickup and Delivery Problems with Time Windows (PDPTW) instances by Li and Lim [11]. We added and modified a few values in order to transfer them into MDCPP instances. Both of the two sets are composed of 9 instances with users from 100 to 400. The clients in the first set are distributed close to the center, so the set is named with C. The second set has the clients and servers allocated randomly, therefore is named with R. The last set of instances is obtained by real-world case, the data is collected from the car pooling program participants of the Artois University by using the car pooling platform Figure 13 shows a general idea of the user distributions in MDCPP instances.

4.2. Configuration

Parameter setting for the presented HAC algorithm is specified as follows:

pre-sorting: $z = 1.5$;
number of ants: 9;
initialization parameters: $\alpha = 0.9$, $\beta = 0.1$, $\sigma = 1$;
probability parameters: $a = 2$, $b = 1$, $y = 3$;
preference updating parameters: $\lambda = 0.1$, $\mu = 0.95$, $n = 10$;
penalty parameters: $p = 2\times$ user's direct travel cost to the destination, $q = 2$;
TPS parameters: $w = 1.3$;
local search: $u = t = 20\%$ of the amount of all carpools, $x = 2$, $v = 1.2$.

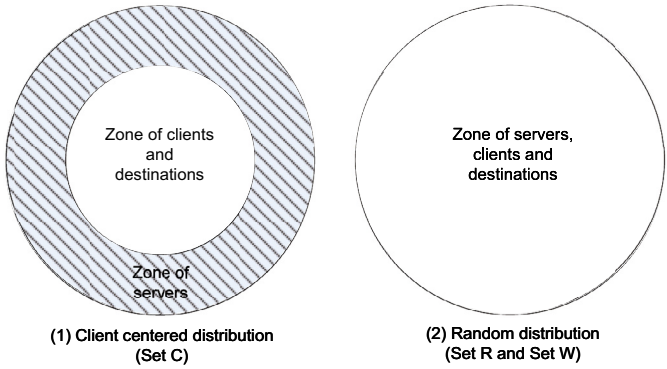

FIGURE 13. Different user distributions in MDCPP instances.

Given limited computational resources and combinatorial complexity, parameter values were determined empirically over a few intuitively selected combinations, choosing the one that yielded the best average output.

4.3. Results obtained with HAC

The HAC algorithm was implemented in JAVA, under Eclipse 6, and all results were obtained running the code on a Windows Operating System with Intel Core i7 740QM 2.9 GHz CPU and 4 GB of RAM. The algorithm has been executed 30 times for each instance, where each execution is given 1000 iterations.

In order to evaluate the gain of solving users with multiple destinations together in our first set of experiments, we compare the results obtained by our MDCPP model with the ones generated by classic daily carpooling problem (DCPP) model. We divide each instance into sub problems according to the destination of users. Therefore each sub problem includes only one destination, which transforms the instance into a classic DCPP one. Since the DCPP is less complex than the LTCPP and MDCPP, so it is possible to obtain a good solution to an instance of 100 users by using CPLEX with several hours of computing. Thus, we divide each DCPP instance again into smaller ones with around 100 users. At last, the total costs, as well as the traveled alone servers and unserved clients of each sub problem are summed up and compared with the results provided by HAC with the MDCPP model. By the comparison, we are able to examine the benefits of considering an instance as a MDCPP instead of several DCPPs.

In the result tables, *Size* with the format *[number of users (number of servers/number of clients)]*, refers to the amount of users, servers and clients in each instance, respectively. N indicates the number of destinations in an instance.

In Table 1, we compared the experimental results of the MDCPP model with the ones generated by the DCPP model. The MDCPP model with HAC outperforms the DCPP model with CPLEX on all instances.

TABLE 1. Experimental results of MDCPP model and DCPP model.

Size	N	Set C instances				Set R instances				Set W instances			
		MDCPP (HAC)			DCPPs	MDCPP (HAC)			DCPPs	MDCPP (HAC)			DCPPs
		Best	Avg	Time	Best	Best	Avg	Time	Best	Best	Avg	Time	Best
100(20/80)	2	1629.0	1641.3	11	2174.2	1963.1	1989.7	12	2372.1	878.7	886.0	10	1082.9
100(20/80)	2	1516.6	1542.7	10	1984.1	2091.4	2097.2	14	2495.2	791.4	798.2	10	1019.5
100(20/80)	2	1703.5	1715	10	2354.4	2139.1	2165.5	12	2763.6	1037.4	1040.0	12	1238.6
200(40/160)	2	2376.2	2413.1	35	3385.2	3408.7	3454.0	38	4735.9	1427.5	1464.0	41	2043.3
200(40/160)	2	2839.0	2883.5	41	4184.8	3997.5	4082.1	43	4822.8	1356.5	1358.2	33	1812.4
200(40/160)	2	3835.6	3879.5	37	5060.4	2781.3	2843.1	42	3950.5	1209.3	1229.5	34	1759.6
400(80/320)	3	4924.7	5049.6	297	7454.8	5829.0	5953.7	344	8077.7	2331.8	2404.5	255	3534.3
400(80/320)	3	3969.2	4091.4	332	6455.3	4722.5	4958.0	271	6395	2979.7	3097.4	321	4145.6
400(80/320)	3	5126.2	5267.2	281	7238.5	6187.0	6298.0	368	8831.5	4435.0	4538.8	318	5781
Total		27 920	28 483.3	1054	40 291.7	33 119.6	33 841.3	1144	44 444.3	16 447.3	16 816.6	1034	22 417.2

TABLE 2. Solution quality comparison.

Size	Set C instances	Set R instances	Set W instances
100	24.6%	17.9%	18.6%
200	27.7%	23.5%	27.8%
400	32.0%	25.8%	26.2%
Avg	28.1%	22.4%	24.2%

TABLE 3. Comparison with the results obtained without TPS.

Size	N	Set C instances		Set R instances		Set W instances	
		With TPS	No TPS	With TPS	No TPS	With TPS	No TPS
100(20/80)	2	1641.3	1835.4	1989.7	2157.2	886.0	915.0
100(20/80)	2	1542.7	1813.1	2097.2	2252.8	798.2	837.8
100(20/80)	2	1715.0	1905.0	2165.5	2495.0	1040.0	1138.8
200(40/160)	2	2413.1	2921.5	3454.0	4149.1	1464.0	1738.9
200(40/160)	2	2883.5	3386.3	4082.1	4311.7	1358.2	1635.3
200(40/160)	2	3879.5	4504.1	2843.1	3459.5	1229.5	1465.3
400(80/320)	3	5049.6	6051.3	5953.7	6846.3	2404.5	2715.9
400(80/320)	3	4091.4	4716.4	4958.0	5833.7	3097.4	3586.1
400(80/320)	3	5267.2	6406.9	6298.0	7160.9	4538.8	5064.8
Total		28 483.3	33 540.0	33 841.3	38 666.1	16 816.6	19 097.8

TABLE 4. Solution quality comparison with the results obtained without TPS.

Size	Set C instances	Set R instances	Set W instances
100	11.8%	9.3%	5.5%
200	15.4%	13.3%	16.3%
400	15.9%	13.4%	11.8%
Avg	14.4%	12.0%	11.2%

Table 2 shows the percentage MDCPP model solved by HAC outperforms the DCPP model solved by CPLEX on the three sets of instances, in the aspect of average solution quality. The computing time is not listed. Since the CPLEX approach costs several hours to generate the result, it is obvious that the HAC is significantly faster. For each instance, the outperforming percentage is calculated as *(CPLEX's value – HAC's value)/CPLEX's value*. Each value in Table 2 is obtained by averaging the outperforming percentages of the three same-size instances. According to our experiments, the MDCPP can improve the DCPP's results by 28.1%, 22.4% and 24.2% on three different sets of instances.

According to Tables 1 and 2, we believe that, solving the instances as a MDCPP can provide better solutions than solving the instance as several DCPPs. The total cost is decreased significantly. Therefore, the comparison reveals the effectiveness MDCPP model and the efficiency of the HAC in solving the MDCPP.

TABLE 5. Evaluation of the accuracy of the HAC.

Size	C set instances				R set instances				W set instances			
	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)	Best	Avg	Std	Diff(%)
100	1616.4	1633.0	8.5	1.0	2064.5	2084.1	11.3	0.9	902.5	908.1	3.6	0.6
200	3016.9	3058.7	17.2	1.4	3395.8	3459.7	27.9	1.8	1331.1	1350.6	11.7	1.4
400	4673.4	4802.7	65.8	2.7	5579.5	5736.6	93.4	2.7	3248.8	3346.9	76.4	2.9
Avg	3102.2	3164.8	31.8	1.7	3680.0	3760.1	44.2	1.8	1827.5	1868.5	30.6	1.7

Tables 3 and 4 show the improvement given by the Transfer Points Search heuristic (TPS). The average solution quality of HAC has been compared with the one obtained by disabling the TPS mechanism. According to the tables, all the solutions benefit from the TPS approach. The total cost of all instances has been decreased by this procedure. The improvements vary from the size of the instances. The large size instances, such as 200 and 400, obtain the most significant improvements.

The accuracy of the HAC is examined by calculating the standard error (column Std) of the solutions obtained in 30 runs of each instance. The solution quality difference (column Diff) between the best found solution quality and the average solution quality of *each* instance in the previous tables is also calculated. Table 5 shows the *average* of the abovementioned values of the three same-size instances. The average differences between the best found solution and the average solution of the three sets of instances are 1.7%, 1.8% and 1.7%, respectively, which indicates the HAC approach can be considered to be accurate for a metaheuristic.

5. Conclusion and perspectives

In this paper we defined a new carpooling problem model, Multi-destination Daily Carpooling Problem (MDCPP). The mathematical formulation has been introduced in detail manner. Then, we introduced the HAC algorithm, a Hybrid Ant Colony Algorithm to solve the MDCPP. We presented in detail the four components of HAC algorithm, a pre-sorting procedure, an ant colony optimization based metaheuristic, a heuristic process designed inside the ant colony structure for determining the transfer point and a local search for further optimize the solutions.

The presented approach has been applied successfully for solving the MDCPP. The experiments of the HAC have been performed on three sets of structurally different instances. Each set includes large scale instances. The experimental results are compared with a CPLEX based decomposition approach, and the comparison has proven that solving the instances as a MDCPP can provide better solutions than solving the instance as several DCPPs. Experiments also have been performed to confirm the effectiveness of the Transfer Point Searching heuristic. Thus, it has been demonstrated that the HAC algorithm is an effective approach for solving the MDCPP.

The MDCPP can be further studied as well as the HAC algorithm. The construction of the transfer points may be embedded into the behavior of the ants instead of using a separate heuristic. In this case, the solution quality of the algorithm could be further improved. However, the computational speed may decrease by this modification, which raises the issue of balancing the solution quality and processing time.

References

[1] R. Baldacci, V. Maniezzo and A. Mingozzi, An exact method for the car pooling problem based on Lagrangian column generation. *Oper. Res.* **52** (2004) 422–439.
[2] C. Blumenthal, S. Michalak, B. Goble, M. Garner, M. Haselkorn and J. Spyridakis, Bellevue smart traveler: Design, demonstration and assessment, Internet, USA (1997).
[3] R.W. Calvo, F. Luigib, P. Haastrupc and V. Maniezzod, A distributed geographic information system for the daily car pooling problem. *Comput. Oper. Res. Arch.* **31** (2004) 2263–2278.
[4] M. Dorigo and T. Stützle, Ant Colony Optimization MIT Press (2004).
[5] M. Dorigo, V. Maniezzo and A. Colorni, Ant system: optimization by a colony of cooperating agents. *IEEE Trans. Syst. Part B* **26** (1996) 29–41.
[6] M. Dorigo, Optimization, learning and natural algorithms, Ph.D. Thesis Politecnico di Milano, Italie (1992).
[7] E. Ferrari and R. Manzini, The car pooling problem: heuristic algorithms based on savings functions. *J. Adv. Transp.* **37** (2003) 243–272.
[8] M.L. Fisher, The Lagrangian relaxation method for solving integer programming problem. *Manag. Sci.* **27** (1981) 1–18.
[9] B. Kallehauge, J. Larsen, B.G. Madsen and M.M. Solomon, Vehicle routing problem with time windows Column Generation (2005) 67–98.
[10] A.B. Kothari, Genghis: a multiagent carpooling system. Dissertation, University of Bath, UK (2004).
[11] H. Li and A. Lim, A metaheuristic for the pickup and delivery problem with time windows. *Int. J. Artificial Intell. Tools* **12** (2003) 173–186.
[12] B. Maurizio, C. Diego, C. Alberto and L. Alessandro, PoliUniPool: a car pooling system for universities. *Soc. Behavior. Sci.* **20** (2011) 558–567.
[13] D. Meyers, D.J. Dailey and D. Lose, Seattle smart traveler: dynamic ride matching on the World Wide Web. *Transp. Res. C* **7** (1999) 17–32.
[14] M. Vargas, J. Sefair, J. Walteros, A.L. Medaglia and L. Rivera, Car pooling optimization: a case study in Strasbourg. Systems and Information Engineering Design Symposium, Virginia, USA (2008).