A clustering ant colony algorithm for the long-term car pooling problem

Yuhan GUO¹, Gilles Goncalves¹, Tienté Hsu¹

¹ Univ Lille Nord de France, F-59000 Lille, France
UArtois, LGI2A, F-62400, Béthune, France
jonguo.fr@gmail.com

Abstract

The increased use of private vehicles has caused significant traffic congestion, noise and air pollution. Public transport is often incapable of effectively servicing non-urban areas. Car pooling, where sets of car owners having the same travel destination share their vehicles, has emerged to be a viable possibility for reducing private vehicle usage around the world. This paper describes a clustering ant colony algorithm for solving the long-term car pooling problem. Computational results are given to show the superiority of our approach compared with other meta-heuristics.

Key words

Swarm intelligence, Ant colony optimization, Transportation problem, Car pooling problem

1 Introduction

During the past decade, the increased human mobility combined with high use of private cars increases the load on the environment and raises issues about the quality of life. The usage of private cars lends to high levels of air pollution, congestion, parking problem and low transfer velocity. Although a lot of effort is being spent on improving the public transport service, there are still numerous non-urban areas where cost-effective transportation systems cannot be set up.

Car pooling is a mobility service normally proposed and controlled by large organizations, such as large companies and public administrations, which encourage their employees to pick up colleagues while driving to or from the workplace in order to minimize the number of private cars traveling on the road. It differs from other sharing schemes where private vehicles are used by several people at different times. Car pooling saves travel costs by sharing journey expenses and reduces carbon emissions, traffic congestion and requirements for parking space. It also decreases driving stress as each driver gets a break from being at the wheel. Moreover, the car pools create social interaction between friends, neighbors and co-workers. As a matter of fact, it can enhance the sense of connectedness within the community as a social network.

Nowadays car pooling has already been considered as an important alternative transportation service throughout the world. Some countries have introduced high-occupancy vehicle lanes to encourage car pooling. Successful car pooling development has tended to be associated mainly with densely populated areas such as city centers and more recently universities and other campuses.

There are two different forms of car pooling problem: the Daily Car Pooling Problem (DCPP) and the Long-term Car Pooling Problem (LTCPP). In DCPP, a few users declare their availability for picking up or bringing back colleagues on one particular day. These users are defined as servers, and then other users are assigned to servers and to the routes to be driven by the servers are identified. Based on this view, the DCPP can be considered as a special case of Dial-a-Ride Problem (DARP). In LTCPP, each user acts as both a server and a client. The objective is to define car pools where each user on different days in turn picks up the remaining pool members. The aim is to minimize the amount of ve-
vehicles used and the total distance traveled by all users. Considering the various successful implemented approaches for DARP and the similarity between DCPP and DARP, the DCPP can be solved simply by adapting the approaches from DARP [1]. But the LTCPP so far has received little attention from the optimization community. A few researches have been carried out on this problem; however, these studies are either time consuming or lacking of solution quality when dealing with large scale instances. Thus, a more efficient and powerful meta-heuristic is still required in the real-world application.

Our Clustering Ant Colony algorithm (CAC) is based on the Ant Colony Optimization (ACO) [2] paradigm, the objective of this work is to cluster the users into car pools by the ant making its tour. Our ultimate goal is to solve the LTCPP efficiently and obtain a good solution with limited exploration to the search space. In order to achieve these objectives, we introduce a preference concept into the ACO system to replace the traditional pheromone information, which converts the classic ACO into a clustering methodology. In our approach, the preference information is used to guide the movement direction and the car pool construction behavior of the ants. In addition, a variable neighborhood search procedure is defined to further optimize the best solutions obtained during iteration. Computational results are reported to illustrate the effectiveness of the CAC comparing with other meta-heuristics for LTCPP.

This paper proceeds as follows. Section 2 describes the LTCPP and its mathematical formulation. Section 3 presents related works and the CAC algorithm for LTCPP. Then, Section 4 contains obtained computational results. The last section gives some conclusions and perspectives.

2 Problem definition and formulation

In this section, we will provide the definition and the mathematical formulation which are necessary for the understanding of LTCPP. The mathematical model is based on the research [3] with some modification in accordance with the real-world situation.

2.1 The mathematical model

The LTCPP can be modeled by means of a directed graph $G = (U \cup \{0\}, A)$, where $U$ is the set of users, and $A=\{arc(i,j) / i \in U, j \in U \cup \{0\}\}$ is the set of arcs. Each user $u \in U$ is associated with a home and node 0 represents the workplace, respectively. $A$ is a set of directed weighted arcs where each $arc(i,j) \in A$ is associated with a positive travel cost $cost_{ij}$ and a travel time $t_{ij}$. Each user enlisted in the long-term car pooling specifies: the maximal extra driving time $T$ the user is willing to accept for picking up colleagues, in addition to the time needed to drive directly from his home to workplace; the acceptable time $e$ for leaving home; the acceptable time $r$ for arriving at work and the capacity $Q$ of his car. Note that pools will not change frequently, which entails the number of members in a pool will be at most equal to the smallest car capacity of this pool, since each member will eventually pick up all other ones.

The LTCPP is a multi-objective problem, requiring minimizing the total amount of vehicles traveling to or from the workplace and the total travel cost of all users. However, it is possible to combine these two objectives in a single objective function by using a penalty concept. The LTCPP then can be formulated as an integrated program presented as follows.

Define a pool $k$ of users and let $|k|$ be the size of this pool. Each user of pool $k$, on different days, will use his car to pick up the other pool members and then go to the workplace. The driver thus has to find a Hamiltonian path starts at the node associated to his home, then passes through all other nodes corresponding to his pool members’ homes exactly once and ends at the workplace. Let $\text{min}_{-}\text{path}(i,k)$ be the shortest abovementioned path, starting from $i \in k$, connecting all $j \in k \setminus \{i\}$ and ending in 0. Suppose $|k| \leq Q_i$, where $Q_i$ being the minimum capacity of all the cars in pool $k$ and all users’ time
The cost for a user driving to the workplace directly from his home is denoted by $cost_{i0}$, while $p_i$ is a penalty value incurred when the user travels alone. Then, the cost of pool $k$ is defined to be:

$$cost(k) = \begin{cases} 
\sum_{i \notin k} cost(\text{min}\_\text{path}(i,k)), & \text{if } |k| > 1, \\
\sum_{i \notin k} cost_{i0} + p_i, & \text{otherwise}.
\end{cases}$$

(1)

The total cost of a complete solution to the LTCPP is then defined to be the sum of the costs of the pools in it. This view optimizes at the same time both objective functions. In our experiments, the penalty of a user driving alone is set to be several times of the cost when he drives directly from his home to the destination, so it is always more convenient to pool users together than to leave them alone.

### 2.2 The objective function

The problem can be defined in a four indices formulation considering the variables:

- $x_{ij}^{hk}$: Binary variable equals to 1 if arc $(i,j)$ is traveled by a server $h$ of a pool $k$;
- $y_{ik}$: Binary variable equals to 1 if user $i$ is in pool $k$;
- $\xi_{ij}$: Binary variable equals to 1 if user $i$ is not pooled with any other user;
- $S_i^h$: Positive variable denoting the pick-up time of user $i$ by server $h$;
- $F_i^h$: Positive variable denoting the arrival time of user $i$ at work when traveling with server $h$;
- $Cost_{ij}$: Positive variable denoting the travel cost between users $i$ and $j$;
- $t_{ij}$: Positive variable denoting the travel time between users $i$ and $j$;
- $Q_k$: Positive variable denoting the minimum car capacity of pool $k$;
- $T_i$: Positive variable indicating the extra driving time specified by user $i$;
- $e_i$: Positive variable indicating the acceptable time for leaving home of user $i$;
- $r_i$: Positive variable indicating the acceptable time for arriving at work of user $i$;
- $p_i$: Positive variable indicating the penalty for user $i$ when he travels alone;
- $K$: Index set of all pools;
- $U$: Index set of all users;
- $A$: Index set of all arcs.

Objective function:

$$f_{\text{LTCPP}} = \min \sum_{k \in K} \sum_{i \in U} \sum_{j \in A} \cos t_{ij} x_{ij}^{hk} + \sum_{i \in U} p_i \xi_i$$

(2)

$$\sum_{j \in U \setminus \{h\}} x_{ij}^{hk} = y_{ik} \quad i, h \in U, k \in K$$

(3)

$$\sum_{i \in U} x_{ij}^{hk} = y_{ik} \quad i, h \in U, k \in K$$

(4)

$$\sum_{i \in U} x_{ij}^{hk} = \sum_{j \in U} x_{ji}^{hk} \quad i, h \in U, k \in K$$

(5)

$$\sum_{k \in K} y_{ik} + \xi_i = 1 \quad i \in U$$

(6)
Several different new automatic and heuristic data processing routines are in operation with sub-offspring. A solution approach offers average savings of more than 50 percent. Equation (3) and (4) forces a user i to be declared to be in pool k, if there is a path originated in h going from i to j or j to i; equation (5) is continuity constraint. Equation (6) forces each user to be assigned to a pool or to be penalized, while (7) and (8) are car capacity and extra driving time constraints, respectively. Equation (9) and (10), where $M$ is a big constant, collectively set feasible pick-up times, while (11) and (12) set minimum and maximum values of feasible arrival times, respectively. Constraints (13) to (15) are binary constraints while (16) and (17) are the positivity constraints.

3 Clustering ant colony algorithm for LTCPP

3.1 Related works

Attempts to resolve the LTCPP have resulted in the development of an ANTS Algorithm [4], a Saving Functions Based Algorithm [5] and a Multi-Matching System [6].

The general structure of the ANTS approach is closely akin to that of a standard tree-search procedure. At each stage the algorithm has a partial solution which is expanded by branching on all possible offsprings; a bound is then computed for each offspring, possibly fathoming dominated ones, and the current partial solution is selected among that associated to the surviving offsprings on the basis of lower bound considerations. The approach provides good solution quality, but the tree-search structure results in less efficiency when dealing with large scale instances.

In the second approach, several different new automatic and heuristic data processing routines are designed to support efficient matching in car pool schemes. These are based on savings functions and belong to two distinct macro classes of algorithms to give two different modeling of this problem. The approach offers average savings of more than 50 percent in traveled distances demonstrating the effectiveness of a trivial matching scheme for real applications.

In the last Multi-Matching System, the authors develop a system-optimized matching model which is proven to be an effective tool to aid the matching between the car pool members. A solution algorithm based on lagrangian relaxation with sub-gradient methods is provided to solve the large scale instances.

Among all the current literatures found, none of the methods developed is cost-effective enough for solving large scale instances. Grounding on the characteristic of LTCPP, which is a combination of clustering and routing, we believe ACO is a suitable paradigm for solving this problem on the basis of Cergy, France, June 14-15, 2011
its good exploration ability and flexible pheromone representation. So during our following research, we first tried using a “route first, cluster second” approach where the ACO is applied to search the shortest path connecting all users, then the path is divided into car pools according to car capacity and time window constraints; we also implemented a “cluster first, route second” approach where we use K-means algorithm to cluster the users into car pools, then construct route for each car pool. But the research reveals that the separation of routing and clustering leads to lots of difficulty in finding a good solution. No matter which mechanism is applied first, it limits the search space, to such an extent that it is very hard for the second applied mechanism to find good solutions. So in order to provide an effective and efficient method for LTCPP, in our CAC algorithm, the ant is vested the ability of clustering during its tour and memorizes its clustering experience to direct the search of the future ants. Thus, the classic ACO algorithm has been transformed into a clustering method.

3.2 Main principles

In CAC, to direct the clustering activity of the ant, we introduce a preference concept into the ant system to replace the traditional pheromone information. The preference information is considered as the preference of one user willing to be pooled in the same cluster with another. When an ant starts a tour from a user, it starts to build a car pool in the meantime. The ant then behaves according to a roulette wheel selection based on the preference values; it can either visit and insert a new user into its current car pool or end the current car pool and select a new user to start a new car pool. When all the users are visited by the ant, the tour of the ant is considered finished. After all ants finish their tour in the current iteration, several solutions with the least cost are selected to be applied a variable neighborhood search. At last, in the end of iteration, the preference information between the users in the same car pool of each solution will be increased. By this mechanism, the clustering experience is always memorized and updated to direct the ant search of future iterations.

The general structure of the CAC is specified as following Algorithm 1.

Algorithm 1
1. Initial Preference information and attractiveness.
2. Repeat until the stop criteria are met
   For $k = 1, k \leq$ the number of ants do
   Select a new user and build a new pool (become current pool);
   While the tour isn’t completed do
   If the current pool is ended;
   Select a new user and build a new pool (become current pool);
   Else
   Find an unserved user who satisfies time window constraints by roulette wheel;
   If found, insert user into the current car pool and the tabu list of ant $k$;
   If car capacity is reached, end current pool;
   Else end current pool;
   End while;
   End for;
3. Select the best $m$ solutions;
4. Apply variable neighborhood search;
5. Update the preference information based on the composition of selected solutions.

3.3 Preference information

The preference information can be considered as variant pheromone information. It is stored in an $n \times n$ matrix where $n$ is the number of users in an instance. The weight values of the matrix indicate the preference level between each two users to be pooled together, as shown in Figure 1.
In the initialization of the preference information, the time window constraints are pre-checked. Equation (18) and (19) examines if user $i$ and $j$ are able to arrive at work in time when pooled together. Equation (20) checks whether user $i$ will be late for work if picking up user $j$, while (21) is used to confirm the time cost for picking up user $j$ is in the range of user $i$'s extra driving time. If pooling user $i$ and $j$ together cannot satisfy these constraints, the preference $w_{ij}$ is set to zero, which means there is no probability that user $i$ and $j$ will be pooled together by ants. By this procedure, we are able to remove some car pool combinations which do not belong to any feasible solution, so the complexity for the ants to search for pool members is significantly decreased.

$$\begin{align*}
e_i + t_{ij} + t_{i0} & \leq r_i & (18) \\
e_i + t_{ij} + t_{j0} & \leq r_j & (19) \\
e_j + t_{i0} & \leq r_j & (20) \\
t_{ij} + t_{j0} & \leq T_i + t_{i0} & (21)
\end{align*}$$

Then, if the constraints are well satisfied, the weight values between two different users are initialized by the geographic distance and the departure time difference between each two users as (22), and the weight values between the same users are computed as (23).

$$w_{ij} = \alpha \times \frac{1}{d_{ij}} + \beta \times \frac{1}{|e_i + t_{ij} - e_j|}$$

$$w_i = \theta \times \frac{1}{T_i}$$

Where $d_{ij}$ and $t_{ij}$ are the geographical distance and travel time between users $i$ and $j$, $T_i$ is the extra driving time of user $i$, respectively. $\alpha$, $\beta$ and $\theta$ are the factors to balance the weight of all information, the values are set according to the unit of distance and time of the instances.

1. Select a user and start a car pool
2. Visit and insert a new user into the current car pool
5. Start a new car pool
7. End the car pool since the car capacity is reached
3. End the current car pool before car capacity is reached
4. Select a new user to start a new car pool
6. Visit and insert new users into the current car pool

Figure 2: Activities of ants in CAC

Note that the weight $w_{ij}$ is also initialized in CAC. Since $w_{ij}$ indicates preference between one user and himself, it is used to inform the ant to end the current car pool. So when the ant chooses the next user to visit from user $i$, it also has a probability to still select user $i$ as the next visitor. In this case, the ant will end the current car pool, as shown in Figure 2. The purpose of this mechanism is to provide a reference for the ant to compare, in order to select a relatively cost-effective opportunity to end the current car pool before reaching the car capacity. For instance, the ant has high probability to end the cluster, when the existing users in current car pool have relatively low preference to other available users compared with the preference to themselves. This means that all possible users are not suitable to Cergy, France, June 14-15, 2011.
be pooled into the current car pool, so the better behavior is to end the cluster rather than pool a user into it. This mechanism is essential in CAC as it is the key point how we end a cluster except the car capacity and time window constraints.

As abovementioned, the two functions of the preference information, which are directing the ants to insert a new user into current car pool and forcing the ant to end current car pool, is controlled by a roulette wheel selection procedure. The probability for the ant to select a new user \(j\) to visit and insert into current car pool \(k\) is based on the preference and attractiveness between user \(j\) and car pool \(k\). In CAC, the preference between a user \(j\) to a car pool \(k\) is calculated as the sum of the preference between user \(j\) and every existing user in car pool \(k\), as shown in (24). In like manner, the probability for the ant to end the current car pool is computed as the sum of preference values of the existing users in car pool \(k\) to themselves, as shown in (25).

\[
Probability_{\text{select }, j} = \frac{\left( \sum \text{attractiveness}_j \right)^{a} \left( \sum \text{preference}_j \right)^{b}}{\sum_{i \in C} \left( \sum \text{attractiveness}_j \right)^{a} \left( \sum \text{preference}_j \right)^{b} + \sum_{i \in H} \left( \sum \text{attractiveness}_j \right)^{a} \left( \sum \text{preference}_j \right)^{b}} \quad (24)
\]

\[
Probability_{\text{end, cluster}} = \frac{\left( \sum \text{attractiveness}_j \right)^{a} \left( \sum \text{preference}_j \right)^{b}}{\sum_{i \in C} \left( \sum \text{attractiveness}_j \right)^{a} \left( \sum \text{preference}_j \right)^{b} + \sum_{i \in H} \left( \sum \text{attractiveness}_j \right)^{a} \left( \sum \text{preference}_j \right)^{b}} \quad (25)
\]

Where \(w_{ij}\) and \(w_{ji}\) are the preference values; \(C\) is the set of existing users in current pool; \(H\) is the set of users who have positive preference value with car pool \(k\); while \(a\) and \(b\) are weight parameters. The attractiveness between users will be introduced in the following part.

If the ant chose to end the cluster, then it will stochastically select another user \(j\) to start another car pool based on the attractiveness between user \(j\) and the current visiting user \(i\). The preference information is ignored in this procedure, since the zero values in the preference matrix will disable the probability to select some relatively far users and affect the diversity of our approach.

When all ants finish their tours, \(n\) solutions with the least travel cost are selected to be applied a variable neighborhood search, and then for each selected solution \(s\), the preference values between the users in the same cluster consist in an augmentation by a factor \(\phi_s\), computed as (26).

\[
\phi_s = \lambda \times \frac{\text{total cost}_s - \text{total cost}_{\text{average}}}{\text{total cost}_{\text{average}}} \quad (26)
\]

\[
w_{ij} = \mu \times w_{ij} \quad (27)
\]

Where \(\text{total cost}_{\text{average}}\) is the average cost of the solutions obtained by each ant in the colony with equation (2); \(\text{total cost}\) is the cost of current selected solution \(s\), calculated also by equation (2). Factor \(\lambda\) is a weight factor used to keep a low value for factor \(\phi\) in anterior iterations and a high value in posterior iterations, so that the ants are more freely to explore the solution space in the beginning iterations.

<table>
<thead>
<tr>
<th>Preference Matrix before</th>
<th>Preference Matrix after</th>
<th>Composition of selected solution (C: {01 02 05}) (04 05 06)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
<tr>
<td>(u_i) (u_j) (u_k)</td>
<td>(u_i) (u_j) (u_k)</td>
<td>(\phi = 0.1)</td>
</tr>
</tbody>
</table>

Figure 3: Updating the preference matrix
Before updating the preference values with the new solutions, all weight values $w_{ij}$ in the preference matrix will decrease with an evaporate rate $\mu$, as in (27), in order to enlarge the influence of the new preference information obtained in current iteration. Figure 3 shows an example of preference matrix updating.

3.4 Attractiveness

The basic paradigm of ACO involves the movement of a colony of ants through the different states influenced by two local decision policies, pheromone and attractiveness. In CAC, we simply use the equations (22) and (23) for initializing the preference information to define the attractiveness. The difference is the attractiveness is computed for every two users, with no exception.

3.5 Variable neighborhood search

The variable neighborhood search procedure in CAC is composed of four different operators. The main structure consists in a loop considering sequentially each neighborhood. The variable neighborhood search stops when it reaches its maximum loop number or no improvement made during the current loop. The first improvement policy is applied to each operator, that is, if an improvement is obtained, the operator will confirm the new solution and terminate.

- Divide
The divide operator consists in divide a selected car pool into smaller car pools with respects to the total travel cost. The operator selects $m$ pools with relatively high travel cost; the selection is performed by a roulette wheel selection based on the travel cost of each car pool. Then, for each selected car pool $i$, the operator tries to pool any members of pool $i$ into a new pool, if the total cost of the new pools is less than the current pool, confirm the partition.

- Merge
The merge operator tries to merge any two non-full car pools. First, a non-full pool $j$ is randomly selected and other non-full car pools which are able to satisfy the car capacity constraints after merging are put into a set. Then the operator tries to merge car pool $i$ with every car pool in the set, if the total cost decreases, confirm the merging.

- Swap
The swap operator tries to swap any two users in two car pools. It first stochastically selects $n$ car pools with high travel costs. For each selected car pool $i$, the operator selects its nearest car pool $j$ according to their gravity center. Then, it tries to swap every member in pool $i$ with every member in pool $j$, if the total cost of the two car pools decreases, confirm the swap.

- Move
The move operator tries to move a user from a selected car pool into a non-full car pool. It first stochastically selects $k$ non-full pools with high travel costs. For each selected pool $i$, the operator selects its nearest pool $j$ according to the gravity center. Then, it moves every member in pool $j$ into pool $i$, if the total cost of the two car pools decreases, confirm the move.

4 Computational results

Computational experiments have been conducted to compare the performance of the proposed approach with some other meta-heuristics. The testing was carried out on a set of structurally different problem instances. The instances were originally derived from VRP hard instances presented in [1]. We added a few constraints in order to transfer them into LTCPP instances. Customer locations for the instance are either generated randomly (named with R), or clustered (named with C) or mixed (named with M). The experiments consisted in performing 20 simulation runs for each problem instance on Windows operating system with Intel Core2 Duo 3.2 GHz CPU and 4 GB RAM.

Cergy, France, June 14-15, 2011
4.1 Configuration

Parameter setting and simulation configuration for the investigated algorithm are specified as follows. Number of ants: 30; Initialization parameters: $\alpha = 0.8$, $\beta = 0.2$, $\theta = 4$; Probability parameters: $a = 2$, $b = 1$; Preference updating parameters: $\lambda = 0.3$, if iteration number $< 300$; $\lambda = 1$, if iteration number $\geq 300$; Evaporate parameter: $\mu = 0.6$; Variable neighborhood search parameters: $m = n = k = 10\%$ of total car pool amount. 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.

4.2 Results

Three sets of experiments have been performed to evaluate the CAC algorithm. In the first set, the variable neighborhood search procedure is removed, in order to evaluate the performance of a pure CAC. The approach then was compared with a classic ACO approach and a K-means algorithm. In the second set, the variable neighborhood search became to function in CAC, and the other two approaches were also added with the same variable neighborhood search to maintain the fairness. The last set focused on the comparison between CAC, a classic genetic algorithm (GA) and a preference guided genetic algorithm (PGA) we designed earlier for solving the LTCPP [7].

In the result tables, $C_{\text{avg}}$ is the average total cost of 20 best solutions obtained in 20 random runs; $T_{\text{cpu}}$ shows the CPU time cost by each algorithm; $Ipv$ indicates the percentage CAC improves the $C_{\text{avg}}$ compared with other algorithms; and $T\text{-test}$ reveals the t values (20 samples, threshold: 2.845) obtained by paired t-tests between CAC and other algorithms, respectively. In the first and the second sets of experiments, the CAC and the classic ACO were given 1000 iterations and the K-means algorithm received 30000 iterations. According to the number of ants we previously set, the three approaches can generate same number of solutions during the experiments. In the last set of experiments, the populations of the two genetic algorithms were set to 100. The CAC was given 3000 iterations to run and the classic genetic algorithm and PGA obtained 1000 iterations.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Size</th>
<th>CAC</th>
<th>Classic ACO</th>
<th>K-means</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$T_{\text{cpu}}$</td>
<td>$C_{\text{avg}}$</td>
<td>$T_{\text{cpu}}$</td>
</tr>
<tr>
<td>1C</td>
<td>100</td>
<td>6.8</td>
<td>3834.17</td>
<td>6.1</td>
</tr>
<tr>
<td>1R</td>
<td>100</td>
<td>6.7</td>
<td>5432.37</td>
<td>6.3</td>
</tr>
<tr>
<td>1M</td>
<td>100</td>
<td>6.9</td>
<td>6257.33</td>
<td>5.8</td>
</tr>
<tr>
<td>2C</td>
<td>200</td>
<td>18.7</td>
<td>7037.34</td>
<td>16.4</td>
</tr>
<tr>
<td>2R</td>
<td>200</td>
<td>18.9</td>
<td>10555.36</td>
<td>17.1</td>
</tr>
<tr>
<td>2M</td>
<td>200</td>
<td>19.5</td>
<td>12552.21</td>
<td>16.8</td>
</tr>
<tr>
<td>3C</td>
<td>400</td>
<td>86.4</td>
<td>13026.74</td>
<td>67.3</td>
</tr>
<tr>
<td>3R</td>
<td>400</td>
<td>92.7</td>
<td>17192.21</td>
<td>69.4</td>
</tr>
<tr>
<td>3M</td>
<td>400</td>
<td>88.1</td>
<td>17246.29</td>
<td>71.5</td>
</tr>
</tbody>
</table>

Table 1: Results obtained by CAC, ACO and K-means, all without VNS

<table>
<thead>
<tr>
<th>Instance</th>
<th>Size</th>
<th>CAC (VNS)</th>
<th>Classic ACO (VNS)</th>
<th>K-means (VNS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$T_{\text{cpu}}$</td>
<td>$C_{\text{avg}}$</td>
<td>$Ipv$</td>
</tr>
<tr>
<td>1C</td>
<td>100</td>
<td>7.4</td>
<td>3664.37</td>
<td>6.7</td>
</tr>
<tr>
<td>1R</td>
<td>100</td>
<td>7.6</td>
<td>5338.47</td>
<td>6.9</td>
</tr>
<tr>
<td>1M</td>
<td>100</td>
<td>7.9</td>
<td>6154.39</td>
<td>6.8</td>
</tr>
<tr>
<td>2C</td>
<td>200</td>
<td>20.7</td>
<td>6937.36</td>
<td>19.4</td>
</tr>
<tr>
<td>2R</td>
<td>200</td>
<td>19.9</td>
<td>10174.25</td>
<td>17.8</td>
</tr>
<tr>
<td>2M</td>
<td>200</td>
<td>20.5</td>
<td>11675.47</td>
<td>17.6</td>
</tr>
<tr>
<td>3C</td>
<td>400</td>
<td>89.1</td>
<td>12856.61</td>
<td>68.3</td>
</tr>
<tr>
<td>3R</td>
<td>400</td>
<td>98.2</td>
<td>16744.43</td>
<td>70.1</td>
</tr>
<tr>
<td>3M</td>
<td>400</td>
<td>98.9</td>
<td>16972.52</td>
<td>72.8</td>
</tr>
</tbody>
</table>

Table 2: Results obtained by CAC, ACO and K-means, all with VNS
Table 3: Results obtained by CAC, GA and PGA

<table>
<thead>
<tr>
<th>Instance</th>
<th>Size</th>
<th>CAC (VNS)</th>
<th>Classic GA</th>
<th>PGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>T_cpu</td>
<td>C_avg</td>
<td>T_cpu</td>
</tr>
<tr>
<td>1C</td>
<td>100</td>
<td>13.1</td>
<td>3634.32</td>
<td>15.4</td>
</tr>
<tr>
<td>1R</td>
<td>15.2</td>
<td>5312.49</td>
<td>15.2</td>
<td>5864.37</td>
</tr>
<tr>
<td>1M</td>
<td>14.3</td>
<td>6092.23</td>
<td>15.1</td>
<td>6432.51</td>
</tr>
<tr>
<td>2C</td>
<td>39.1</td>
<td>6684.44</td>
<td>54.5</td>
<td>7247.44</td>
</tr>
<tr>
<td>2R</td>
<td>200</td>
<td>42.5</td>
<td>10022.45</td>
<td>55.2</td>
</tr>
<tr>
<td>2M</td>
<td>48.2</td>
<td>11411.29</td>
<td>53.6</td>
<td>12898.04</td>
</tr>
<tr>
<td>3C</td>
<td>400</td>
<td>182.5</td>
<td>12031.53</td>
<td>196.7</td>
</tr>
<tr>
<td>3R</td>
<td>186.9</td>
<td>16023.57</td>
<td>203.5</td>
<td>18583.11</td>
</tr>
<tr>
<td>3M</td>
<td>400</td>
<td>188.3</td>
<td>16054.22</td>
<td>203.9</td>
</tr>
</tbody>
</table>

Table 1 shows that, for all instances, without variable neighborhood search, the CAC algorithm can provide better solution quality than classic ACO approach and K-means algorithm. The improvement reveals to be significant according to the T-test results.

In table 2, the variable neighborhood search is applied to all three approaches. The CAC algorithm can still provide better solution quality than K-means algorithm, but compared with classic ACO approach, it can only provide an insignificant improvement. However, when dealing with large instances, e.g. 400 users, the CAC can provide remarkable solution quality compared with classic ACO.

Table 3 demonstrates that the CAC approach can always provide better solution quality than classic GA. But comparing with PGA, the CAC shows its superiority in processing the cluster distributed instances, and the PGA is more powerful in dealing with random or mixed distributed instances.

5 Conclusion

The straightforward CAC that has been described above works well. Experiments have been performed to confirm the efficiency and the effectiveness of the preference mechanism. For most of the instances, the CAC can provide good solutions quality compared with other meta-heuristics. Thus, it has been demonstrated that the CAC algorithm is an effective approach for solving the long-term car pooling problem. Our future research will involve real-world benchmarks in order to perform further evaluation to this approach and explore other algorithms for solving this problem.

6 References


Cergy, France, June 14-15, 2011