A HYBRID HEURISTIC ALGORITHM FOR THE CLUSTERED TRAVELING SALESMAN PROBLEM

Mário Mestria

Received July 11, 2015 / Accepted March 21, 2016

ABSTRACT. This paper proposes a hybrid heuristic algorithm, based on the metaheuristics Greedy Randomized Adaptive Search Procedure, Iterated Local Search and Variable Neighborhood Descent, to solve the Clustered Traveling Salesman Problem (CTSP). Hybrid Heuristic algorithm uses several variable neighborhood structures combining the intensification (using local search operators) and diversification (constructive heuristic and perturbation routine). In the CTSP, the vertices are partitioned into clusters and all vertices of each cluster have to be visited contiguously. The CTSP is \( \mathcal{NP} \)-hard since it includes the well-known Traveling Salesman Problem (TSP) as a special case. Our hybrid heuristic is compared with three heuristics from the literature and an exact method. Computational experiments are reported for different classes of instances. Experimental results show that the proposed hybrid heuristic obtains competitive results within reasonable computational time.

Keywords: Combinatorial Problems, Clustered Traveling Salesman Problem, Hybrid Heuristic, GRASP, Iterated Local Search, Variable Neighborhood Structures.

1 INTRODUCTION

The Clustered Traveling Salesman Problem (CTSP) can be defined as follows. Let \( G = (V, E) \) be a symmetric complete graph with vertex set \( V = \{v_1, v_2, \ldots, v_n\} \) and edge set \( E = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\} \). The vertex set \( V \) is partitioned into disjoint clusters \( V_1, V_2, \ldots, V_m \) and each edge \( (v_i, v_j) \in E \) is associated with a non-negative cost \( c_{ij} \). The objective of the CTSP is to find a minimum cost Hamiltonian tour on \( G \), where all vertices of each cluster must be visited consecutively. Note that when there is only a subset \( V_1 = V \) the CTSP becomes a Traveling Salesman Problem (TSP). Therefore, the CTSP is \( \mathcal{NP} \)-hard since it includes the TSP as a special case.

CTSP applications arise in automated warehouse routing Chisman [6], emergency vehicle dispatching Weintraub, Aboud, Fernandez, Laporte & Ramirez [42], production planning Lokin...
Heuristic techniques are widely used to solve many TSP variants [8]. There are also heuristic procedures based on different approaches [9], [10], [11]. There are also heuristic procedures based on different approaches [12], [13], [14]. There are also heuristic procedures based on different approaches [15], [16], [17]. There are also heuristic procedures based on different approaches [18], [19], [20]. There are also heuristic procedures based on different approaches [21], [22], [23]. There are also heuristic procedures based on different approaches [24], [25], [26]. There are also heuristic procedures based on different approaches [27], [28], [29]. There are also heuristic procedures based on different approaches [30], [31], [32]. There are also heuristic procedures based on different approaches [33], [34], [35]. There are also heuristic procedures based on different approaches [36], [37], [38]. There are also heuristic procedures based on different approaches [39], [40], [41]. There are also heuristic procedures based on different approaches [42], [43], [44]. There are also heuristic procedures based on different approaches [45], [46], [47]. There are also heuristic procedures based on different approaches [48], [49], [50]. There are also heuristic procedures based on different approaches [51], [52], [53]. There are also heuristic procedures based on different approaches [54], [55], [56]. There are also heuristic procedures based on different approaches [57], [58], [59]. There are also heuristic procedures based on different approaches [60], [61], [62]. There are also heuristic procedures based on different approaches [63], [64], [65]. There are also heuristic procedures based on different approaches [66], [67], [68]. There are also heuristic procedures based on different approaches [69], [70], [71]. There are also heuristic procedures based on different approaches [72], [73], [74]. There are also heuristic procedures based on different approaches [75], [76], [77]. There are also heuristic procedures based on different approaches [78], [79], [80]. There are also heuristic procedures based on different approaches [81], [82], [83]. There are also heuristic procedures based on different approaches [84], [85], [86]. There are also heuristic procedures based on different approaches [87], [88], [89]. There are also heuristic procedures based on different approaches [90], [91], [92]. There are also heuristic procedures based on different approaches [93], [94], [95]. There are also heuristic procedures based on different approaches [96], [97], [98]. There are also heuristic procedures based on different approaches [99], [100], [101]. There are also heuristic procedures based on different approaches [102], [103], [104]. There are also heuristic procedures based on different approaches [105], [106], [107]. There are also heuristic procedures based on different approaches [108], [109], [110]. There are also heuristic procedures based on different approaches [111], [112], [113]. There are also heuristic procedures based on different approaches [114], [115], [116]. There are also heuristic procedures based on different approaches [117], [118], [119]. There are also heuristic procedures based on different approaches [120], [121], [122]. There are also heuristic procedures based on different approaches [123], [124], [125]. There are also heuristic procedures based on different approaches [126], [127], [128]. There are also heuristic procedures based on different approaches [129], [130], [131]. There are also heuristic procedures based on different approaches [132], [133], [134]. There are also heuristic procedures based on different approaches [135], [136], [137]. There are also heuristic procedures based on different approaches [138], [139], [140]. There are also heuristic procedures based on different approaches [141], [142], [143]. There are also heuristic procedures based on different approaches [144], [145], [146]. There are also heuristic procedures based on different approaches [147], [148], [149]. There are also heuristic procedures based on different approaches [150], [151], [152]. There are also heuristic procedures based on different approaches [153], [154], [155]. There are also heuristic procedures based on different approaches [156], [157], [158]. There are also heuristic procedures based on different approaches [159], [160], [161]. There are also heuristic procedures based on different appro...
adding a large constant \( M \) to the cost of each inter-cluster edge. A branch-and-bound algorithm was used to solve the problem.

Several \( \alpha \)-approximation algorithms were developed for the CTSP Anily, Bramel & Hertz [1], Arkin, Hassin & Klein [3], Gendreau, Laporte & Potvin [14], Gendreau, Laporte & Hertz [15], and Guttmann-Beck, Hassin, Khuller & Raghavachari [18]. These algorithms extend those developed for the TSP Christofides [7] and Hoogeveen [21] and provide solutions for the CTSP within a given approximation factor. Different values for such factors were found in the literature, namely: \( 5/3 \) Anily, Bramel & Hertz [1], \( 3\sqrt{2} \) Arkin, Hassin & Klein [3], \( 2 \) Gendreau, Laporte & Potvin [14], and \( 3/2 \) Gendreau, Laporte & Hertz [15].

It is worth mentioning that the \( \alpha \)-approximation algorithms require as input either the starting \( s_i \) and ending \( t_i \) vertices in each cluster \( V_i \) or a prespecified order of visiting the clusters in the tour, i.e., \( V_1, V_2, \ldots, V_m \), with \( V_j < V_{j+1} \) for \( i = 1, 2, \ldots, m-1 \). Note that these algorithms solve the inter-cluster and intra-cluster problems independently. One drawback of these \( \alpha \)-approximation algorithms is the fact that it does not allow the construction of cycles when the order of visiting the clusters is not given \emph{a priori}.

In Guttmann-Beck, Hassin, Khuller & Raghavachari [18] some \( \alpha \)-approximation algorithms with bounded performance ratios were presented for four CTSP variants. In the first one, a \((21/11)\)-approximation algorithm was proposed for the case in which the starting and ending vertices of each cluster are specified. In the second one, a \((9/5)\)-approximation algorithm was suggested for the variant where for each cluster the two end vertices are given, but with freedom of choosing the starting and ending vertices. The third approximation scheme consists of a \((37/14)\)-approximation algorithm for the case where the starting vertex of each cluster is given. In the last one, a \((11/4)\)-approximation algorithm was devised for the case where no specific starting and ending vertices are provided. More recently, in Bao & Liu [4], a new approximation algorithm with a factor of \( 13/6 \) was proposed for the CTSP, also for the case where no starting and ending vertices are specified.

A heuristic algorithm that combines the concepts of Tabu Search and Evolutionary Algorithms was proposed in Laporte, Potvin & Quilleret [25] to solve a particular case of the CTSP in which the visiting order within the clusters is prespecified. It was verified that the developed heuristic outperformed the Genetic Algorithm (GA) described in Potvin & Guertin [36], which exploits order-based crossover operators and local search heuristics. The CTSP with a prespecified visiting order within the clusters was also considered in Gendreau, Laporte & Potvin [14]. An approximation algorithm with guaranteed performance ratio and two methods with satisfactory empirical performance were developed. Three algorithms were compared with a lower bound.

An algorithm which provides lower bounds on the optimal tour lengths for the CTSP using a 1-tree based Lagrangian Relaxation was developed in Jongens & Volgenant [23]. The computational results were reported for a set of instances ranging from 80 to 150 vertices and different sizes of clusters.
A hybrid heuristic algorithm for the clustered traveling salesman problem

A GA for the CTSP which finds inter-cluster paths and then intra-cluster paths was developed in Potvin & Guertin [37]. Comparisons were performed with the heuristic proposed in Gendreau, Laporte & Potvin [14] and with lower bounds obtained using the procedure suggested in Jon-gens & Volgenant [23]. This GA solved problems with up to 500 vertices and with 4 and 10 clusters. A Two-Level Genetic Algorithm (TLGA) was proposed for the CTSP in Ding, Cheng & He [8]. In the lower level, the algorithm finds Hamiltonian cycles in each cluster, whereas in the higher level, it randomly chooses an edge to be deleted on the cycle in each cluster and simultaneously determines routing among clusters. Computational results demonstrated that the TLGA outperformed the classical GA developed by the authors Ding, Cheng & He [8].

Several GRASP based heuristics for the CTSP were recently proposed in Mestria, Ochi & Martins [33]. One heuristic corresponds to the traditional GRASP while another five include alternative Path Relinking (Glover, Laguna & Martí [17]) procedures. The computational results showed that the proposed heuristics outperformed the GA presented in Ding, Cheng & He [8]. Two particular versions, GPR1R2 and GPR4, had the best performance among the different methods put forward by the authors. We compare the results obtained by our hybrid heuristic with those found by these two approaches and also with the traditional GRASP.

3 A MATHEMATICAL FORMULATION OF CTSP

A mathematical formulation of CTSP using integer programming is described in Chisman [6], Miller, Tucker & Zemlin [34]. The salesman leaves an origin city \( v_1 \), without loss of generality, and returns to \( v_1 \). The cost \( c_{ij} \) represents the distance between city \( v_i \) and city \( v_j \). The salesman proceeds from city \( v_i \) to city \( v_j \) if and only if \( x_{ij} = 1 \).

The mathematical formulation is as follows:

\[
\begin{align*}
\text{Minimize } z &= \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{subject to } \sum_{j=1}^{n} x_{ij} &= 1, \quad \forall i \in V \\
\sum_{i=1}^{n} x_{ij} &= 1, \quad \forall j \in V \\
u_i - u_j + (n - 1)x_{ij} &\leq (n - 2), \quad 2 \leq i \neq j \leq n \\
\sum_{i \in V_k} \sum_{j \in V_k} x_{ij} &= |V_k| - 1, \quad \forall V_k \subset V, \ |V_k| \geq 1, \ k = 1, \ldots, m \\
u_i &\geq 0, \quad 2 \leq i \leq n \\
x_{ij} &\in \{0, 1\}, \quad \forall i, j \in V
\end{align*}
\]

The objective function (1) minimizes the total distance traveled by the salesman and constraints (2) and (3) ensure that each city is visited once. Constraints (4) serve to eliminate tours that do

Pesquisa Operacional, Vol. 36(1), 2016
not begin and end at city $v_1$ and tours that visit more than $(n - 1)$ cities. Constraints (5) state that a Hamiltonian path of length $|V_k| - 1$ must go through the $|V_k|$ points of cluster $k$. Constraints (6) and (7) define the domain of variables.

The Figure 1 shows a feasible solution for an example of an instance of the CTSP composed of four clusters ($V_1$, $V_2$, $V_3$, and $V_4$) and 11 vertices, where $V_1$ = {1, 4, 6}, $V_2$ = {2, 3, 7}, $V_3$ = {5, 8}, and $V_4$ = {9, 10, 11}. The dotted line edge between the vertices 4 and 9 show a connection inter-cluster and the full line edge between the vertices 9 and 11 a intra-cluster connection. Note that the vertices of each cluster are visited contiguously.

4 THE PROPOSED HYBRID HEURISTIC

As already mentioned, the developed hybrid heuristic, called HHGILS, brings together components of the metaheuristics GRASP, ILS, and VND. The pseudocode of the proposed hybrid heuristic is illustrated in Algorithm 1. In step 2, the probability distribution of the parameter $\alpha$, which is related to the constructive procedure, is initialized. The role of this parameter (reactive strategy) will be explained later.
Algorithm 1 Pseudocode of the HHGILS algorithm.

1: \texttt{iterprob} $\leftarrow 1$;
2: \texttt{initializealphas}(); \{initialize the probability distribution of the alphas\}
3: \texttt{for} $i = 1$ to $\text{maxext}$ \texttt{do}
4: $S^0 \leftarrow \text{ConstructiveHeuristic}()$;
5: $S \leftarrow \text{VND}(S^0)$;
6: \texttt{for} $j = 1$ to $\text{maxint}$ \texttt{do}
7: $S' \leftarrow \text{Pert}(S)$;
8: $S'' \leftarrow \text{VND}(S')$;
9: $S \leftarrow \text{AcceptCrit}(S, S', S^*)$;
10: if $S$ is better than $S^*$ then
11: $S^* \leftarrow S$;
12: \texttt{end if}
13: \texttt{end for}
14: if \texttt{iterprob} = $\text{maxprob}$ then
15: \texttt{updatealphas}(); \{update the probability distribution of the alphas\}
16: \texttt{iterprob} $\leftarrow 0$;
17: \texttt{end if}
18: \texttt{iterprob} $\leftarrow \text{iterprob} + 1$;
19: \texttt{end for}
20: \texttt{return} $S^*$;

Other three important parameters of the algorithm are: $\text{maxext}$, $\text{maxint}$ and $\text{maxprob}$. They correspond, respectively, to the maximum number of restarts of the method, ILS iterations, and restarts to update the probability distribution of $\alpha$.

The value of $\text{maxint}$ is typically larger than $\text{maxprob}$ because more computational effort is usually required to improve an initial solution than to improve another one that has been slightly modified (perturbed) from a local optimal solution. Restarting the method from a different initial solution and perturbing local optimal solutions are both promising ways of diversifying the search, especially when put together. Since perturbations may lead to cycles, we build new solutions so as to avoid this kind of cycling. Thus at each external iteration a solution is built. For each of the $\text{maxext}$ iterations (step 3 to step 19), a solution is generated by a constructive procedure based on GRASP and, in step 5, a local search is performed by means of a VND procedure. For each of the $\text{maxint}$ iterations (step 6 to step 13), a perturbation is applied in step 7 and a local search is performed again using VND (step 8). In steps 9-11, the incumbent solution is updated.

The acceptance criterion (step 9) is responsible for deciding the solution from which the search will continue. In our case, it chooses the solution, from $S$, $S'$ and $S^*$, associated with the minimum cost. In traditional ILS, the acceptance criterion does not consider $S^*$ (the best solution in the current iteration). It only consider $S$ e $S'$ e thus if $S$ passes an acceptance test, it becomes the next element of the walk in $S$, otherwise, one returns to $S'$. But, our algorithm consists...
in choosing the solution which presents the minimum cost among the three solutions \( S, S^* \) and \( S'' \). In step 15, the probability distribution of the \( \alpha \) values is updated using a reactive strategy. Finally, in step 20 the best solution is returned.

The constructive heuristic extends the well-known Nearest Insertion heuristic by introducing concepts of GRASP. At each iteration of the constructive heuristic, a Restricted Candidate List (RCL) is built using a parameter \( \alpha \) to restrict the size of the list of candidates to be inserted in the partial solution. The \( \alpha \) values are defined using a reactive strategy, leading to a better performance when compared with fixed values. The reactive strategy usually helps generating better quality solutions, also avoiding parameter tuning (Festa & Resende [12]). The reactive strategy is presented as follows. Let \( \Psi = \{a_1, \ldots, a_m\} \) be the finite set of \( m \) possible values for \( \alpha \) and let \( p_i \) be the corresponding probability of selecting \( a_i, i = 1, \ldots, m \). Initially, \( p_i \) is uniformly distributed:

\[
p_i = 1/m, \quad i = 1, \ldots, m.
\]  

After \( K \) iterations, the \( p_i \) values are reevaluated as follows. Let \( f^* \) be the best cost solution found in \( K \) previous iterations and let \( \overline{f} \) be the average cost solutions obtained using \( \alpha = a_i, i = 1, \ldots, m \) during \( K \) iterations. The probabilities are updated after \( K \) iterations according to:

\[
p_i = q_i/\sum_{j=1}^m q_j, \quad i = 1, \ldots, m.
\]  

where \( q_i = f^*/\overline{f} \).

**Algorithm 2** Pseudocode of the nearest insertion heuristic.

1. Select three vertices \( v_l, v_{l+1} \) and \( v_{l+2} \).
2. \( S \leftarrow \{v_l, v_{l+1}, v_{l+2}\} \);
3. Initialize the candidate set \( C \leftarrow V \setminus\{v_l, v_{l+1}, v_{l+2}\} \);
4. while \( C \neq \emptyset \) do
5. Find the \( k \) nearest vertices of \( C \) with respect to the vertices of the constructed tour (solution \( S \)) and evaluate the incremental costs \( c(v_k) \) for \( v_k \in C \);
6. \( c_{min} \leftarrow \min\{c(v_k)/v_k \in C\} \);
7. \( c_{max} \leftarrow \max\{c(v_k)/v_k \in C\} \);
8. \( RCL \leftarrow \{v_k \in C/c(v_k) \leq c_{min} + \alpha \leq c_{max}\} \);
9. Select an element \( v_k \) from the \( RCL \) at random;
10. \( S \leftarrow S \cup\{v_k\} \); \{connect \( v_k \) to the vertices \( v_l \) and \( v_{l+1} \) of the tour with minimum cost (price) \( p(v_k) = p_l + p_{l+1} + p_{l+1} \) and update the tour\}
11. Update the candidate set \( C \leftarrow C \setminus\{v_k\} \);
12. end while
13. return \( S \)

The pseudocode of the constructive heuristic is presented in Algorithm 2. In step 1, the vertex \( v_l \) is randomly selected and the two nearest neighborhood vertices from the same cluster as \( v_l \) (\( v_{l+1} \) and \( v_{l+2} \)) are chosen as the vertices that will form the initial partial solution \( S \) (step 2).
The candidate set $C$ is initialized in step 3. In step 5, the $k$ nearest vertices of $C$, with respect to the vertices already in the partial tour, are determined and the incremental costs $c(v_k)$ are evaluated. Only feasible insertions are considered, meaning that the vertices chosen belong to the same cluster of existing vertices in the partial solution. Note that the vertices of a same cluster must be visited consecutively in a feasible CTSP tour. In steps 6 and 7, the minimum $c_{\text{min}}$ and maximum $c_{\text{max}}$ incremental costs are determined. In step 9, a vertex $v_k$ is randomly selected from RCL. In step 10, the partial tour (solution $S$) is updated by inserting a vertex $v_k$ between two adjacent vertices ($v_i, v_{i+1}$) which leads to the minimum cost, denoted by $p(v_k)$. In step 11, the candidate set $C$ is updated and finally, in step 13, the initial feasible solution $S$ is returned.

Table 1 – Data for an example of the Algorithm 2.

<table>
<thead>
<tr>
<th>vertex $v_k$</th>
<th>x-coordinate</th>
<th>y-coordinate</th>
<th>cluster $V_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>177</td>
<td>311</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>924</td>
<td>320</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>265</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>45</td>
<td>500</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>233</td>
<td>300</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>167</td>
<td>104</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>117</td>
<td>275</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>190</td>
<td>78</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>61</td>
<td>269</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>684</td>
<td>539</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>380</td>
<td>526</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>550</td>
<td>549</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1 presents the data of a small size instance containing 12 vertices and 3 clusters. In the first column shows a vertex $v_k$, in the second the $x$-coordinate of vertex $v_k$, in the third the $y$-coordinate of vertex $v_k$, and in the last column the cluster $V_i$ of vertex $v_k$. This instance is considered to illustrate how Algorithm 2 works, as shown in Table 2. It shows the some steps of the Algorithm 2 with data of Table 1. In the first column shows the steps, in the second the inserting the initial vertices ($v_1, v_2, v_3$) or vertex $v_k$, in the third the cluster $V_i$ of initial vertices or of vertex $v_k$, and in the last column the initial, partial or final solution. In this case, the initial partial tour is composed of vertices 1, 5 and 7, all belonging to cluster 1. Next, vertex 3 is inserted between vertices 5 and 7. The procedure is repeated until vertex 9 is inserted and a feasible initial solution is built.

As pointed before, the local search procedure is based on VND and it is composed of four types of neighborhood structures. The first neighborhood structure, $N_1$, consists of moving vertex $v_k$ from position $i$ to position $j \neq i$ (1-Shift) of the same cluster so as to ensure feasibility. We describe two types of procedures (Drop and Cheap) in order to define the second neighborhood structure $N_2$. The procedure Drop removes the vertex $v_k$ which leads to the best saving. The
Table 2 – Steps of the Algorithm 2 with data of Table 1.

<table>
<thead>
<tr>
<th>Steps</th>
<th>vertex $v_k$</th>
<th>cluster $V_i$</th>
<th>partial solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 and 2</td>
<td>1, 5, 7</td>
<td>1</td>
<td>[157]</td>
</tr>
<tr>
<td>9 and 10</td>
<td>3</td>
<td>1</td>
<td>[1537]</td>
</tr>
<tr>
<td>9 and 10</td>
<td>11</td>
<td>1</td>
<td>[11137]</td>
</tr>
<tr>
<td>9 and 10</td>
<td>4</td>
<td>3</td>
<td>[1541137]</td>
</tr>
<tr>
<td>9 and 10</td>
<td>8</td>
<td>3</td>
<td>[15841137]</td>
</tr>
<tr>
<td>9 and 10</td>
<td>6</td>
<td>3</td>
<td>[158641137]</td>
</tr>
<tr>
<td>9 and 10</td>
<td>2</td>
<td>3</td>
<td>[15864211137]</td>
</tr>
<tr>
<td>9 and 10</td>
<td>12</td>
<td>2</td>
<td>[1586421137]</td>
</tr>
<tr>
<td>9 and 10</td>
<td>10</td>
<td>2</td>
<td>[15864210121137]</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>2</td>
<td>[158642101291137]</td>
</tr>
</tbody>
</table>

$a$ vertices ($v_l, v_{l+1}, v_{l+2}$) 
$b$ initial solution 
$c$ final solution

procedure *Cheap* inserts a vertex $v_k$ between two adjacent vertices ($v_i, v_{i+1}$) using the cheapest insertion criterion. $N_2$ thus consists in applying the *Drop* procedure followed by the *Cheap* procedure. The moves of $N_2$ are applied until no further improvement of the procedures (Drop and Cheap) is found. The third neighborhood structure, $N_3$, is based on vertex-exchange movements in the same cluster (2-Swap). The fourth neighborhood structure, $N_4$, uses the well-known 2-opt procedure both inside and between the clusters. In this work, we apply the first improvement strategy for $N_1$ and $N_3$, with a view of reducing the computational time, and the the best improvement strategy for $N_2$ and $N_4$.

The double-bridge Martin, Otto & Felten [31] was adopted as one of the perturbation mechanisms. It consists of deleting four arcs and inserting another four in such a way that a new feasible solution is built. This perturbation, called *Pert*, is only applied inside a cluster, with more than eight vertices, which is selected at random. For the remaining cases, i.e. when the clusters have less or equal than eight vertices, we introduce another perturbation mechanism, called *Pertt*, which randomly chooses two clusters and modify their visiting order. In this case, the procedure does modify the visiting order of vertices inside the clusters. Such perturbations are called individually at a time, that is, they are never combined.

5 COMPUTATIONAL RESULTS

The set of instances used in Mestria, Ochi & Martins [33] were used to evaluate the performance of the proposed hybrid heuristic. These instances are available on-line at http://labic.ic.uff.br/Instance/index.php. Six different classes of instances were considered.

Class 1: instances adapted from the TSPLIB Reinelt [38] using the $k$-means clustering algorithm to generate the clusters.
Class 2: instances adapted from those found in Johnson & McGeoch [22] where the clusters were created by grouping the vertices in geometric centers.

Class 3: instances generated using the Concorde interface available in Applegate, Bixby, Chvátal & Cook [2].

Class 4: instances generated using the layout proposed in Laporte, Potvin & Quilleret [25].

Class 5: instances similar to Class 2, but generated with different parameters.

Class 6: instances adapted from the TSPLIB Reinelt [38], where the rectangular floor plan is divided into several quadrilaterals and each quadrilateral corresponds to a cluster.

We chose a set of small, medium and large size instances. The set of small instances is composed of 27 test-problems from Class 1. The set of medium instances is composed of 3 instances from each of the six classes, which leads to a total of 18 instances. Finally, the last set is composed of 15 instances, more specifically, 5 from classes 4, 5 and 6, respectively. In this way, we have a variety of different classes of instances with varying sizes. Each class has one different granularity (the vertices are positioned and dispersed in various ways).

The empirical performance of HHGILS was compared with the optimal solution or the lower bound obtained by a Mixed Integer Programming (MIP) formulation proposed in Chisman [6]. The Parallel ILOG CPLEX was used a MIP solver, with four threads. The classes of cuts used in the CPLEX were clique cuts, cover cuts, implied bound cuts, flow cuts, mixed integer rounding cuts, and Gomory fractional cuts. The tests were executed in a 2.83 GHz Intel Core 2 Quad with 4 cores and 8 GB of RAM running Ubuntu Linux (kernel version 4.3.2-1). The HHGILS algorithm was coded in C programming language and it was executed in the same environment mentioned above, but in this case only a single thread was used.

The main parameters used in our testing are described as follows. Eleven values were used for $\alpha_k$ and they were initially chosen with uniform probability from the interval $[0.0, 0.1, 0.2, \ldots, 1.0]$. The parameter $\text{maxint}$ was set to 35, $\text{maxext}$ (maximum number of external iterations) set to 40, and $\text{maxprob}$ was set to 10. HHGILS was executed ten times for each instance. These parameters were calibrated after preliminary experiments.

5.1 Comparison with CPLEX

Table 3 shows the results found in the set of instances of Class 1, where the first column contains the instances. The instances $XX$-$name$-$YYY$ is composed of $XX$ clusters, one name $name$, and $YYY$ vertices, for example, (a) the instance 5-eil51 is composed of five clusters and 51 vertices (first row of Table 3 with the results), (b) the instance 10-eil51 is composed of ten clusters and 51 vertices (second row), (c) the instance 75-lin105 is composed of 75 clusters and 105 vertices (last row), and so on. The second column contains the optimal values obtained by CPLEX and the third column the time (in seconds) required by CPLEX to find the optimal solution, i.e., the CPLEX was carry out until to find the optimal solution with gap equal a 0 (zero). The gap is calculated by (10). The fourth column presents the gap between the solution obtained by...
HHGILS and the optimal solution. The last column shows the average computing time (in seconds) of HHGILS. The stopping criterion of HHGILS in this case was the number of iterations. The boldface values indicate that the hybrid heuristic found the optimal solution. From Table 3, it can be seen that HHGILS found 17 optimal solutions out of 27 instances. The average gap between the solutions obtained by HHGILS and the optimal ones was 0.27%, whereas the average computing time was 12.05 seconds. In this case, the perturbation $Pertc$ was used in 11 of the 27 instances.

$$gap = 100 \times \left( \frac{best - lb}{best + \epsilon} \right),$$  \hspace{1cm} (10)$$

where, $best$ is the best value found by CPLEX, $lb$ the lower bound, and $\epsilon$ is equal to $10^{-10}$.

Table 3 – Comparison between HHGILS and CPLEX for the small size instances of Class 1.

<table>
<thead>
<tr>
<th>Instances</th>
<th>CPLEX Opt</th>
<th>CPLEX Time (s)</th>
<th>HHGILS Gap (%)</th>
<th>HHGILS Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-eil51</td>
<td>437</td>
<td>12.31</td>
<td>0.00</td>
<td>3.80</td>
</tr>
<tr>
<td>10-eil51</td>
<td>440</td>
<td>74.38</td>
<td>0.00</td>
<td>5.50</td>
</tr>
<tr>
<td>15-eil51</td>
<td>437</td>
<td>2.04</td>
<td>0.00</td>
<td>5.00</td>
</tr>
<tr>
<td>5-berlin52</td>
<td>7991</td>
<td>201.80</td>
<td>0.11</td>
<td>4.50</td>
</tr>
<tr>
<td>10-berlin52</td>
<td>7896</td>
<td>89.17</td>
<td>0.52</td>
<td>5.10</td>
</tr>
<tr>
<td>15-berlin52</td>
<td>8049</td>
<td>75.93</td>
<td>0.00</td>
<td>4.00</td>
</tr>
<tr>
<td>5-st70</td>
<td>695</td>
<td>13790.11</td>
<td>0.00</td>
<td>6.90</td>
</tr>
<tr>
<td>10-st70</td>
<td>691</td>
<td>4581.00</td>
<td>0.43</td>
<td>3.70</td>
</tr>
<tr>
<td>15-st70</td>
<td>692</td>
<td>883.50</td>
<td>0.87</td>
<td>6.40</td>
</tr>
<tr>
<td>5-eil76</td>
<td>559</td>
<td>83.70</td>
<td>0.00</td>
<td>5.10</td>
</tr>
<tr>
<td>10-eil76</td>
<td>561</td>
<td>254.30</td>
<td>1.07</td>
<td>8.60</td>
</tr>
<tr>
<td>15-eil76</td>
<td>565</td>
<td>49.66</td>
<td>0.88</td>
<td>6.80</td>
</tr>
<tr>
<td>5-pr76</td>
<td>108590</td>
<td>99.29</td>
<td>0.00</td>
<td>6.30</td>
</tr>
<tr>
<td>10-pr76</td>
<td>109538</td>
<td>238.13</td>
<td>0.00</td>
<td>8.30</td>
</tr>
<tr>
<td>15-pr76</td>
<td>110678</td>
<td>261.94</td>
<td>0.00</td>
<td>9.70</td>
</tr>
<tr>
<td>10-rat99</td>
<td>1238</td>
<td>650.67</td>
<td>0.00</td>
<td>12.60</td>
</tr>
<tr>
<td>25-rat99</td>
<td>1269</td>
<td>351.15</td>
<td>0.00</td>
<td>21.40</td>
</tr>
<tr>
<td>50-rat99</td>
<td>1249</td>
<td>2797.58</td>
<td>0.00</td>
<td>18.60</td>
</tr>
<tr>
<td>25-kroA100</td>
<td>21917</td>
<td>3513.57</td>
<td>0.00</td>
<td>21.70</td>
</tr>
<tr>
<td>50-kroA100</td>
<td>21453</td>
<td>947.55</td>
<td>0.00</td>
<td>21.70</td>
</tr>
<tr>
<td>10-kroB100</td>
<td>22440</td>
<td>4991.44</td>
<td>0.91</td>
<td>12.40</td>
</tr>
<tr>
<td>50-kroB100</td>
<td>22355</td>
<td>2579.22</td>
<td>0.00</td>
<td>22.40</td>
</tr>
<tr>
<td>25-eil101</td>
<td>663</td>
<td>709.45</td>
<td>1.06</td>
<td>21.10</td>
</tr>
<tr>
<td>50-eil101</td>
<td>644</td>
<td>275.33</td>
<td>0.16</td>
<td>20.80</td>
</tr>
<tr>
<td>25-lin105</td>
<td>14438</td>
<td>6224.55</td>
<td>1.38</td>
<td>16.70</td>
</tr>
<tr>
<td>50-lin105</td>
<td>14379</td>
<td>1577.21</td>
<td>0.00</td>
<td>22.60</td>
</tr>
<tr>
<td>75-lin105</td>
<td>14521</td>
<td>15886.77</td>
<td>0.00</td>
<td>23.70</td>
</tr>
<tr>
<td><strong>Average values</strong></td>
<td><strong>2266.73</strong></td>
<td><strong>0.27</strong></td>
<td><strong>12.05</strong></td>
<td></td>
</tr>
</tbody>
</table>
In Table 4, we present the computational results obtained by CPLEX and HHGILS for medium size instances. The first column contains the names of the instances and in the second their identifiers, followed by the number of vertices, number of clusters and the Class of the instances. The next three columns show the upper bound, the lower bound and the gap found by CPLEX. The last two columns show the best and average gaps found by HHGILS with respect to the lower bound obtained by CPLEX. We set a time limit of 7200 seconds for all CPLEX executions. The stopping criterion for HHGILS was set to 720 seconds. As already mentioned, the hybrid heuristic was executed ten times for each instance, leading to a total time of 7200 seconds.

On average, the best and average solutions found by HHGILS are less than 1% away from the lower bounds obtained by CPLEX, which suggests that the proposed hybrid heuristic had a robust performance, in terms of solution quality, on this set of instances.

It should be noticed that, for the set of small instances, the worst average gap found by HHGILS, with respect to the optimal solutions, was 1.38%, whereas the worst average gap for the medium instances, with respect to the lower bound, was 1.45%. To our knowledge, the best CTSP α-approximation algorithm proposed in the literature is the one presented in Bao & Liu [4], with an approximation factor of $13/6 \approx 2.17$. Therefore, it is possible to observe that the largest gaps found by HHGILS are much smaller than the approximation factor obtained in Bao & Liu [4].

<table>
<thead>
<tr>
<th>Instances</th>
<th>Id.</th>
<th>#vertices</th>
<th>#Vi</th>
<th>Class</th>
<th>Cost</th>
<th>Lower Bound</th>
<th>Gap (%)</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-50-gil262</td>
<td>J1</td>
<td>262</td>
<td>50</td>
<td>1</td>
<td>135529</td>
<td>135374.68</td>
<td>0.11</td>
<td>0.10</td>
<td>0.15</td>
</tr>
<tr>
<td>10-lin318</td>
<td>J2</td>
<td>318</td>
<td>10</td>
<td>1</td>
<td>534640</td>
<td>526412.07</td>
<td>1.54</td>
<td>0.73</td>
<td>0.83</td>
</tr>
<tr>
<td>10-pcb442</td>
<td>J3</td>
<td>442</td>
<td>10</td>
<td>1</td>
<td>547152</td>
<td>536478.33</td>
<td>1.95</td>
<td>0.46</td>
<td>0.65</td>
</tr>
<tr>
<td>C1k.0</td>
<td>J4</td>
<td>1000</td>
<td>10</td>
<td>2</td>
<td>134025123</td>
<td>131354925.50</td>
<td>1.99</td>
<td>1.45</td>
<td>1.81</td>
</tr>
<tr>
<td>C1k.1</td>
<td>J5</td>
<td>1000</td>
<td>10</td>
<td>2</td>
<td>13050874</td>
<td>128540131.50</td>
<td>1.69</td>
<td>0.99</td>
<td>1.28</td>
</tr>
<tr>
<td>C1k.2</td>
<td>J6</td>
<td>1000</td>
<td>10</td>
<td>2</td>
<td>14434185</td>
<td>141501445</td>
<td>1.97</td>
<td>1.24</td>
<td>1.41</td>
</tr>
<tr>
<td>300-6</td>
<td>J7</td>
<td>300</td>
<td>6</td>
<td>3</td>
<td>8969</td>
<td>8915.18</td>
<td>0.60</td>
<td>0.21</td>
<td>0.31</td>
</tr>
<tr>
<td>400-6</td>
<td>J8</td>
<td>400</td>
<td>6</td>
<td>3</td>
<td>9117</td>
<td>9021.51</td>
<td>1.05</td>
<td>0.33</td>
<td>0.46</td>
</tr>
<tr>
<td>700-20</td>
<td>J9</td>
<td>700</td>
<td>20</td>
<td>3</td>
<td>41638</td>
<td>41274.00</td>
<td>0.87</td>
<td>0.43</td>
<td>0.51</td>
</tr>
<tr>
<td>200-4-h</td>
<td>J10</td>
<td>200</td>
<td>4</td>
<td>4</td>
<td>63429</td>
<td>62244.84</td>
<td>1.87</td>
<td>0.89</td>
<td>1.29</td>
</tr>
<tr>
<td>200-4-x1</td>
<td>J11</td>
<td>200</td>
<td>4</td>
<td>4</td>
<td>60797</td>
<td>60242.96</td>
<td>0.91</td>
<td>1.13</td>
<td>1.85</td>
</tr>
<tr>
<td>600-8-z</td>
<td>J12</td>
<td>600</td>
<td>8</td>
<td>4</td>
<td>132897</td>
<td>127901.75</td>
<td>3.76</td>
<td>1.17</td>
<td>1.56</td>
</tr>
<tr>
<td>600-8-x2</td>
<td>J13</td>
<td>600</td>
<td>8</td>
<td>4</td>
<td>132228</td>
<td>127901.75</td>
<td>3.27</td>
<td>1.04</td>
<td>1.38</td>
</tr>
<tr>
<td>300-5-108</td>
<td>J14</td>
<td>300</td>
<td>5</td>
<td>5</td>
<td>68361</td>
<td>67128.93</td>
<td>1.80</td>
<td>1.01</td>
<td>1.18</td>
</tr>
<tr>
<td>300-20-111</td>
<td>J15</td>
<td>300</td>
<td>20</td>
<td>5</td>
<td>311286</td>
<td>308595.45</td>
<td>0.86</td>
<td>0.52</td>
<td>0.59</td>
</tr>
<tr>
<td>500-15-306</td>
<td>J16</td>
<td>500</td>
<td>15</td>
<td>5</td>
<td>196001</td>
<td>193522.8</td>
<td>1.26</td>
<td>0.86</td>
<td>0.98</td>
</tr>
<tr>
<td>500-25-308</td>
<td>J17</td>
<td>500</td>
<td>25</td>
<td>5</td>
<td>367586</td>
<td>364108.13</td>
<td>0.95</td>
<td>0.47</td>
<td>0.54</td>
</tr>
<tr>
<td>25-eil101</td>
<td>J18</td>
<td>101</td>
<td>25</td>
<td>6</td>
<td>23671</td>
<td>23668.63</td>
<td>0.01</td>
<td>0.04</td>
<td>0.09</td>
</tr>
<tr>
<td>42-a280</td>
<td>J19</td>
<td>280</td>
<td>42</td>
<td>6</td>
<td>130043</td>
<td>129560.53</td>
<td>0.37</td>
<td>0.12</td>
<td>0.21</td>
</tr>
<tr>
<td>144-rat783</td>
<td>J20</td>
<td>783</td>
<td>144</td>
<td>6</td>
<td>916147</td>
<td>913715.52</td>
<td>0.27</td>
<td>0.16</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Average 1.36 0.67 0.86
5.2 Results obtained by the hybrid heuristic HHGILS for large size instances

Table 5 illustrates the best and average results obtained by HHGILS for large size instances using a stopping criterion of 1080 seconds for each run. The first column contains the names of the instances and the second column their identifiers, followed by the number of vertices, number of clusters and the Class of the instances. The sixth and seventh columns show the values obtained by HHGILS. These results appear to be quite competitive when compared to other approaches from the literature as shown later in Table 8.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Id.</th>
<th>#vertices</th>
<th>#Vc</th>
<th>Class</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>49-pcb1173</td>
<td>J21</td>
<td>1173</td>
<td>49</td>
<td>6</td>
<td>67043</td>
<td>68260.70</td>
</tr>
<tr>
<td>100-pcb1173</td>
<td>J22</td>
<td>1173</td>
<td>100</td>
<td>6</td>
<td>68786</td>
<td>70640.83</td>
</tr>
<tr>
<td>144-pcb1173</td>
<td>J23</td>
<td>1173</td>
<td>144</td>
<td>6</td>
<td>68830</td>
<td>69084.25</td>
</tr>
<tr>
<td>10-nrw1379</td>
<td>J24</td>
<td>1379</td>
<td>10</td>
<td>6</td>
<td>63620</td>
<td>64643.88</td>
</tr>
<tr>
<td>12-nrw1379</td>
<td>J25</td>
<td>1379</td>
<td>12</td>
<td>6</td>
<td>63558</td>
<td>64741.57</td>
</tr>
<tr>
<td>1500-10-503</td>
<td>J26</td>
<td>1500</td>
<td>10</td>
<td>5</td>
<td>11986</td>
<td>12109.45</td>
</tr>
<tr>
<td>1500-20-504</td>
<td>J27</td>
<td>1500</td>
<td>20</td>
<td>5</td>
<td>17107</td>
<td>17315.72</td>
</tr>
<tr>
<td>1500-50-505</td>
<td>J28</td>
<td>1500</td>
<td>50</td>
<td>5</td>
<td>25264</td>
<td>25558.90</td>
</tr>
<tr>
<td>1500-100-506</td>
<td>J29</td>
<td>1500</td>
<td>100</td>
<td>5</td>
<td>32260</td>
<td>33760.64</td>
</tr>
<tr>
<td>1500-150-507</td>
<td>J30</td>
<td>1500</td>
<td>150</td>
<td>5</td>
<td>37658</td>
<td>38433.09</td>
</tr>
<tr>
<td>2000-10-a</td>
<td>J31</td>
<td>2000</td>
<td>10</td>
<td>4</td>
<td>116254</td>
<td>116881.38</td>
</tr>
<tr>
<td>2000-10-b</td>
<td>J32</td>
<td>2000</td>
<td>10</td>
<td>4</td>
<td>36447</td>
<td>37305.14</td>
</tr>
<tr>
<td>2000-10-z</td>
<td>J33</td>
<td>2000</td>
<td>10</td>
<td>4</td>
<td>37059</td>
<td>37443.69</td>
</tr>
<tr>
<td>2000-10-x1</td>
<td>J34</td>
<td>2000</td>
<td>10</td>
<td>4</td>
<td>36752</td>
<td>37704.03</td>
</tr>
<tr>
<td>2000-10-x2</td>
<td>J35</td>
<td>2000</td>
<td>10</td>
<td>4</td>
<td>36660</td>
<td>37117.11</td>
</tr>
</tbody>
</table>

5.3 Comparison with other heuristic algorithms and an exact method

We now compare our results with those reported in Mestria, Ochi & Martins [33]. More precisely, we compare HHGILS with the following methods: GPR1R2, GPR4 and G. The first two consist of a combination between GRASP and Path Relinking while the third one is the traditional GRASP. According to the experiments performed in Mestria, Ochi & Martins [33], GPR1R2 outperforms all other methods proposed in that paper. The other two methods also obtained satisfactory results but only in a particular class of instances. Therefore, GPR1R2 will be considered in all comparisons, while GPR4 and G will only be considered in some particular cases. It is important to mention that GPR1R2, GPR4 and G were executed in the same computational environment of HHGILS.

Table 6 compares the results obtained by HHGILS and GPR1R2 for the small size instances of Class 1. The stopping criterion for both methods in this case was the number of iterations. It can be observed that GPR1R2 appears to have a better performance both in terms of solution quality and computational time. Nevertheless, HHGILS found more optimal solutions than GPR1R2.
Table 6 – Comparison between HHGILS and GPR1R2 Mestria, Ochi & Martins [33] for the small instances of Class 1.

<table>
<thead>
<tr>
<th>Instances</th>
<th>HHGILS</th>
<th>GPR1R2</th>
<th>GPR1R2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gap (%)</td>
<td>Time (s)</td>
<td>Gap (%)</td>
</tr>
<tr>
<td>5-eil51</td>
<td>0.00</td>
<td>3.80</td>
<td>0.00</td>
</tr>
<tr>
<td>10-eil51</td>
<td>0.00</td>
<td>5.50</td>
<td>0.00</td>
</tr>
<tr>
<td>15-eil51</td>
<td>0.00</td>
<td>5.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5-berlin52</td>
<td>0.11</td>
<td>4.50</td>
<td>0.00</td>
</tr>
<tr>
<td>10-berlin52</td>
<td>0.52</td>
<td>5.10</td>
<td>0.00</td>
</tr>
<tr>
<td>15-berlin52</td>
<td>0.00</td>
<td>4.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5-st70</td>
<td>0.00</td>
<td>6.90</td>
<td>0.00</td>
</tr>
<tr>
<td>10-st70</td>
<td>0.43</td>
<td>3.70</td>
<td>0.00</td>
</tr>
<tr>
<td>15-st70</td>
<td>0.87</td>
<td>6.40</td>
<td>0.00</td>
</tr>
<tr>
<td>5-eil76</td>
<td>0.00</td>
<td>5.10</td>
<td>0.36</td>
</tr>
<tr>
<td>10-eil76</td>
<td>1.07</td>
<td>8.60</td>
<td>0.53</td>
</tr>
<tr>
<td>15-eil76</td>
<td>0.88</td>
<td>6.80</td>
<td>0.35</td>
</tr>
<tr>
<td>5-pr76</td>
<td>0.00</td>
<td>6.30</td>
<td>0.00</td>
</tr>
<tr>
<td>10-pr76</td>
<td>0.00</td>
<td>8.30</td>
<td>0.00</td>
</tr>
<tr>
<td>15-pr76</td>
<td>0.00</td>
<td>9.70</td>
<td>0.15</td>
</tr>
<tr>
<td>10-rat99</td>
<td>0.00</td>
<td>12.60</td>
<td>0.00</td>
</tr>
<tr>
<td>25-rat99</td>
<td>0.00</td>
<td>21.40</td>
<td>0.63</td>
</tr>
<tr>
<td>50-rat99</td>
<td>0.00</td>
<td>18.60</td>
<td>0.72</td>
</tr>
<tr>
<td>25-kroA100</td>
<td>0.00</td>
<td>21.70</td>
<td>0.00</td>
</tr>
<tr>
<td>50-kroA100</td>
<td>0.00</td>
<td>21.70</td>
<td>0.00</td>
</tr>
<tr>
<td>10-kroB100</td>
<td>0.91</td>
<td>12.40</td>
<td>0.16</td>
</tr>
<tr>
<td>50-kroB100</td>
<td>0.00</td>
<td>22.40</td>
<td>1.33</td>
</tr>
<tr>
<td>25-eil101</td>
<td>1.06</td>
<td>21.10</td>
<td>1.36</td>
</tr>
<tr>
<td>50-eil101</td>
<td>0.16</td>
<td>20.80</td>
<td>1.09</td>
</tr>
<tr>
<td>25-lin105</td>
<td>1.38</td>
<td>16.70</td>
<td>0.00</td>
</tr>
<tr>
<td>50-lin105</td>
<td>0.00</td>
<td>22.60</td>
<td>1.08</td>
</tr>
<tr>
<td>75-lin105</td>
<td>0.00</td>
<td>23.70</td>
<td>0.59</td>
</tr>
<tr>
<td>Average values</td>
<td>0.27</td>
<td>12.05</td>
<td>0.25</td>
</tr>
</tbody>
</table>

In Table 7 we compare the gaps between the best solutions obtained by HHGILS, GPR1R2 and GPR4 and the lower bounds found by CPLEX on instances of Mestria, Ochi & Martins [33]. The stopping criterion for all methods is 720 seconds for each of the 10 executions.

The last three columns of Table 7 show the gap of the average solutions for all methods with respect to the lower bound found by CPLEX. From Table 7, one can verify that HHGILS yields the best solutions in most cases. On average, HHGILS also produced superior results when compared to the other two methods.
Table 7 – Comparison between HHGILS, GPR1R2 Mestria, Ochi & Martins [33] and GPR4 Mestria, Ochi & Martins [33] for medium size instances.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Best gap (%)</th>
<th>Average gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HHGILS</td>
<td>GPR1R2</td>
</tr>
<tr>
<td>10-lin318</td>
<td>0.73</td>
<td>0.76</td>
</tr>
<tr>
<td>10-pcb442</td>
<td>0.46</td>
<td>0.66</td>
</tr>
<tr>
<td>C1k.0</td>
<td>1.45</td>
<td>1.60</td>
</tr>
<tr>
<td>C1k.1</td>
<td>0.99</td>
<td>1.27</td>
</tr>
<tr>
<td>300-6</td>
<td>0.21</td>
<td>0.49</td>
</tr>
<tr>
<td>700-20</td>
<td>0.43</td>
<td>0.64</td>
</tr>
<tr>
<td>200-4-h</td>
<td>0.89</td>
<td>1.19</td>
</tr>
<tr>
<td>600-8-z</td>
<td>1.17</td>
<td>1.96</td>
</tr>
<tr>
<td>300-20-111</td>
<td>0.52</td>
<td>0.43</td>
</tr>
<tr>
<td>500-25-308</td>
<td>0.47</td>
<td>0.58</td>
</tr>
<tr>
<td>25-eil101</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>144-rat783</td>
<td>0.16</td>
<td>0.20</td>
</tr>
<tr>
<td>Average values</td>
<td>0.67</td>
<td>0.82</td>
</tr>
</tbody>
</table>

Finally, Table 8 presents a comparison between the best and average results found by HHGILS, G, and GPR1R2 on large size instances. In this case we report the gaps between the solution found by the respective method and the best solution values found by GPR1R2.

The stopping criterion for the hybrid heuristics was set to 1080 seconds for each run. The first column contains the identifiers of the instances, the second and third columns the results obtained by the GPR1R2 heuristic, followed by the gaps of the best solution and the average solution of the methods.

From Table 8, we can observe that HHGILS found better solutions when compared to G and GPR1R2, except for the instance J28. With respect to the best solution values, HHGILS obtained an average gap of $-2.93\%$, whereas G obtained 0.35%. As for the average solutions, HHGILS obtained an average gap of $-3.83\%$, while G obtained $1.57\%$.

We also compare our results with an exact method by the Concorde solver, reported in Applegate, Bixby, Chvátal & Cook [2] through of instances of Class 3 (instances generated using its own user interface). Table 9 shows the results found in the set of instances of Class 3, where the first column contains the instances. The instances $XXXX$-$YY$ is composed of $XXXX$ vertices and $YYY$ clusters, for example, (a) the instance 300-6 is composed of 300 vertices and 6 clusters (first row of Table 3 with the results), (b) the instance 350-6 is composed of 350 vertices and 6 clusters (second row), and (c) the instance 1000-30 is composed of 1000 vertices and 30 clusters (last row). The second column contains the best values obtained by Concorde and the third column the time (in seconds) required by Concorde to find the solution. The fourth column presents the gap between the solution obtained by HHGILS and the best solution. The last column shows
the computing time (in seconds) of HHGILS. For the specific case, we set a time limit for all HHGILS executions, as the stopping criterion.

From Table 9, it can be seen that HHGILS found good solutions. The boldface values indicate that the algorithms carry out in less time. Table 9 shows strong robustness of our hybrid heuristic. The average gap between the solutions obtained by HHGILS and the best ones was 6.46%, whereas the average computing time was 43.33 seconds. The results from Table 9 show that HHGILS reached good results in a short period of time. Thus, the HHGILS allows for greater freedom of decision-making in operational planning in automated warehouse routing, emergency vehicle dispatching, production planning, vehicle routing, among others.

6 CONCLUSIONS

In this paper we proposed a hybrid heuristic algorithm, called HHGILS, based on GRASP, ILS, and VND for the Clustered Traveling Salesman Problem (CTSP). To our knowledge, this is the first time that ILS was applied to solve the CTSP. The results obtained for small and medium size instances were compared with optimal solutions or lower bounds found by a MIP formulation using CPLEX. For the set of instances of small size, HHGILS was capable of finding several optimal solutions. Moreover, the average gap between the average solutions and the optimal solutions, for the small size instances, was 0.27%, whereas the average gap, with respect to the lower bounds, for the medium size instances was 0.67%.

Pesquisa Operacional, Vol. 36(1), 2016
The CTSP has many applications in real life, for example, automated warehouse routing, emergency vehicle dispatching, production planning, vehicle routing, commercial transactions with supermarkets and shops and grocery suppliers and so on. This applications, set of small size instances, need to be solved by heuristic algorithm for rapid response to the operational planning process. In this paper we proposed the hybrid heuristic algorithm (HHGILS) for this applications with the best performance and high-quality solutions in a reasonable computational time.

We compare the results found by HHGILS with other GRASP based methods from the literature for small, medium and large size instances. It was observed that the method which combines GRASP with Path Relinking (GPR1R2) Mestria, Ochi & Martins [33] found, on average, better solutions for small size instances, but HHGILS outperformed such method when considering medium and large size instances. Our hybrid heuristic also was compared with an exact method and it presented robustness.

We observe that the largest gap (0.0376) found by HHGILS was much smaller than the approximation factor obtained in Bao & Liu [4], a new approximation algorithm with a factor of 13/6.

Because the running times of the algorithm presented here in this paper does not grow rather rapidly as the problem size grows, we can generalize that many problems of combinatorial optimization can be solved by using HHGILS, adapting it to the problem.
ACKNOWLEDGEMENTS.

This paper has been partially supported by the Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo (IFES). This acknowledgment extends to Prof. Anand Subramanian for your comments.

References


Pesquisa Operacional, Vol. 36(1), 2016
132


Pesquisa Operacional, Vol. 36(1), 2016