A Hybrid Heuristic Approach for Solving the Generalized Traveling Salesman Problem

Petrică C. Pop
Dept. of Mathematics and Computer Science
North University of Baia Mare
Str. Victoriei, 430122, Baia Mare, Romania
petrica.pop@ubm.ro

Serban Iordache
Scoop Software GmbH
Am Kielshof 29
51105, Cologne, Germany
siordache@acm.org

ABSTRACT
The generalized traveling salesman problem (GTSP) is an NP-hard problem that extends the classical traveling salesman problem by partitioning the nodes into clusters and looking for a minimum Hamiltonian tour visiting exactly one node from each cluster. In this paper, we combine the consultant-guided search technique with a local-global approach in order to solve efficiently the generalized traveling salesman problem. We use candidate lists in order to reduce the search space and we introduce efficient variants of 2-opt and 3-opt local search in order to improve the solutions. The resulting algorithm is applied to Euclidean GTSP instances derived from the TSPLIB library. The experimental results show that our algorithm is able to compete with the best existing algorithms in terms of solution quality and running time.

Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *heuristic methods.*

General Terms
Algorithms.

Keywords
Generalized traveling salesman problem, hybrid algorithms, consultant-guided search.

1. INTRODUCTION
The generalized traveling salesman problem (GTSP) is an NP-hard problem that extends the classical traveling salesman problem by considering a related problem given a partition of the nodes of a graph into clusters. The problem consists in finding the shortest closed tour visiting exactly one node from each cluster. The existence of several applications of the GTSP and the difficulty of obtaining optimum solutions for the problem has led to the development of several heuristics and metaheuristics, see for example [4], [13], [14], [15].
Consultant-guided search (CGS) is a recent metaheuristic for combinatorial optimization problems, inspired by the way real people make decisions based on advice received from consultants. The CGS metaheuristic can be used to solve hard combinatorial optimization problems and it has been successfully applied to the classical Traveling Salesman Problem (TSP) [8] and to the Quadratic Assignment Problem [9].

The aim of this paper is to propose a hybrid algorithm that combines the consultant-guided search technique with a local-global approach for solving the GTSP. Our algorithm constructs Hamiltonian tours in the global graph obtained by replacing all nodes of a cluster with a supernode. Then, each global solution is improved using an efficient variant of 2-opt or 3-opt local search that takes advantage of the structure imposed by the global graph (i.e., the graph obtained by replacing all nodes of each of the clusters with a supernode representing it). Finally, a cluster optimization procedure is applied in order to find the best generalized tour corresponding to the given sequence of clusters.

We report the computational results obtained by applying our algorithm to symmetric Euclidean GTSP instances derived from the TSPLIB benchmark library. The experimental results show that our algorithm can compete with the best existing algorithms for the GTSP in terms of both solution quality and running time.

2. THE LOCAL-GLOBAL APPROACH TO THE GENERALIZED TRAVELING SALESMAN PROBLEM
Let $G = (V, E)$ be an n-node undirected complete graph whose edges are associated with non-negative costs and let $V_1, \dots, V_m$ be a partitioning of V into m subsets called *clusters* (i.e. $V = V_1 \cup V_2 \cup \dots \cup V_m$ and $V_l \cap V_k = \emptyset$ for all $l, k \in \{1, \dots, m\}$).

Then the *generalized traveling salesman problem* asks for finding a minimum-cost tour H spanning a subset of nodes such that H contains *exactly* one node from each cluster V_i, $i \in \{1, \dots, m\}$. We call such a cycle a *generalized Hamiltonian tour*.

Based on the way the generalized combinatorial optimization problems are defined as extensions of the classical variants, a natural approach that takes advantage of the similarities between them is the *local-global approach* introduced by Pop [11] in the case of the generalized minimum spanning tree problem.

In the case of the GTSP, the local-global approach aims at distinguishing between *global connections* (connections between clusters) and *local connections* (connections between nodes from different clusters). This approach was already pointed out and exploited by Hu *et al.* in [5] and by Bontoux *et al.* [1].

GECCO'11, July 12-16, 2011, Dublin, Ireland.

Given a sequence in which the clusters are visited (i.e. a global Hamiltonian tour), there are several generalized Hamiltonian tours corresponding to it. The best corresponding (with respect to cost minimization) generalized Hamiltonian tour can be determined either by using a layered network as we will describe next or by using integer programming.

We denote by G' the graph obtained from G after replacing all nodes of a cluster V_i with a supernode representing V_i. We will call the graph G' the *global graph*. For convenience, we identify V_i with the supernode representing it. Edges of the graph G' are defined between each pair of the graph vertices $V_i, \dots, V_m$.

Given a sequence $V_{k_1}, \dots, V_{k_m}$ in which the clusters are visited, we want to find the best feasible Hamiltonian tour H^* (with respect to cost minimization), visiting the clusters according to the given sequence. This can be done in polynomial time by solving $|V_{k_1}|$ shortest path problems, as we describe below.

We construct a layered network, denoted by LN, having $m+1$ layers corresponding to the clusters $V_{k_1}, \dots, V_{k_m}$ and in addition we duplicate the cluster V_{k_1}. The layered network contains all the nodes of G plus some extra nodes v' for each $v \in V_{k_1}$. There is an arc (i,j) for each $i \in V_{k_l}$ and $j \in V_{k_{l+1}}$ $(l = 1, \dots, m-1)$, having the cost c_{ij}. Moreover, there is an arc (i,j') for each $i \in V_{k_m}$ and $j' \in V_{k_1}$ having the cost $c_{ij'}$.

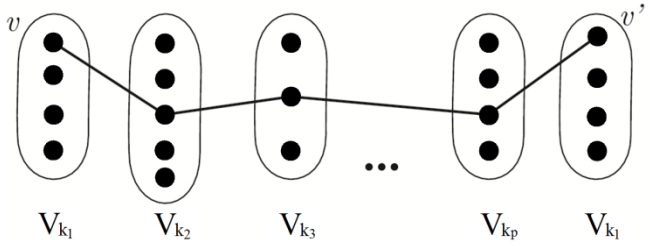

Figure 1. Example showing a Hamiltonian tour in the constructed layered network LN

For any given $v \in V_{k_1}$, we consider paths from v to v', $v' \in V_{k_1}$, that visits exactly one node from each cluster $V_{k_2}, \dots, V_{k_m}$, hence it gives a feasible Hamiltonian tour.

Conversely, every Hamiltonian tour visiting the clusters according to the sequence $(V_{k_1}, \dots, V_{k_m})$ corresponds to a path in the layered network from a certain node $v \in V_{k_1}$ to $v' \in V_{k_1}$.

Therefore, it follows that the best (with respect to cost minimization) Hamiltonian tour H^* visiting the clusters in a given sequence can be found by determining all the shortest paths from each $v \in V_{k_1}$ to the corresponding $v' \in V_{k_1}$ with the property that it visits exactly one node from each of the clusters $V_{k_2}, \dots, V_{k_m}$.

The overall time complexity is then $|V_{k_1}| O(|E| + \log n)$, i.e. $O(n|E| + \text{n} \log n)$, in the worst case, where by $|E|$ we denote the number of edges. We can reduce the time by choosing V_{k_1} as the cluster with minimum cardinality.

Notice that the above procedure leads to an $O((m-1)!\,(n|E| + \text{n} \log n))$ time exact algorithm for the GTSP, obtained by trying all the $(m-1)!$ possible cluster sequences.

Clearly, the algorithm presented is an exponential time algorithm, unless the number of clusters m is fixed.

3. THE CGS METAHEURISTIC

In this section, we briefly describe the Consultant-Guided Search (CGS) metaheuristic. We refer the reader to [7] for a detailed presentation.

CGS is a swarm intelligence technique for solving hard combinatorial optimization problems, which takes inspiration from the way real people make decisions based on advice received from consultants.

CGS is a population-based method. An individual of the CGS population is a virtual person, which can simultaneously act both as a client and as a consultant. As a client, a virtual person constructs at each iteration a solution to the problem. As a consultant, a virtual person provides advice to clients, in accordance with its *strategy*. Usually, at each step of the solution construction, there are several variants a client can choose from. The variant recommended by the consultant has a higher probability to be chosen, but the client may opt for one of the other variants, which will be selected based on some heuristic.

At the beginning of each iteration, a client chooses a consultant based on its *personal preference* and on the consultant's *reputation*. The reputation of a consultant increases with the number of successes achieved by its clients. A client achieves a *success*, if it constructs a solution better than all solutions found until that point by any client guided by the same consultant. Each time a client achieves a success, the consultant adjusts its strategy in order to reflect the sequence of decisions taken by the client.

Because the reputation fades over time, a consultant needs that its clients constantly achieve successes, in order to keep its reputation. If the consultant's reputation sinks below a minimum value, it will take a *sabbatical leave*, during which it will stop offering advice to clients and it will instead start searching for a new strategy to use in the future.

4. THE HYBRID ALGORITHM FOR THE GTSP

We propose in this section an algorithm for the GTSP that combines the consultant-guided search technique with a local-global approach and improves the solutions using a local search procedure. Most GTSP instances of practical importance are symmetric problems with Euclidean distances, where the clusters are composed of nodes that are spatially close one to the other. We design our algorithm to take advantage of the structure of these instances.

4.1 The Algorithm

At each iteration, a client constructs a global tour, that is, a Hamiltonian cycle in the global graph. The strategy of a consultant is also represented by a global tour, which the consultant advertises to its clients. The algorithm applies a local search procedure in order to improve the global tour representing either the global solution of a client or the strategy of a consultant in sabbatical mode. Then, using the cluster optimization procedure described in section 2, the algorithm finds the best generalized tour corresponding to the global tour returned by the local search procedure.

In order to compare the strategies constructed during the sabbatical leave, a consultant uses the cost of the generalized tour corresponding to each strategy. Similarly, the success of a client is

evaluated based on the cost of the generalized solution. The pseudocode of our algorithm is shown in Figure 2.

```
 1 procedure CGS-GTSP()
 2    create the set 𝒫 of virtual persons
 3    foreach p ∈ 𝒫 do
 4       setSabbaticalMode(p)
 5    end foreach
 6    while (termination condition not met) do
 7       foreach p ∈ 𝒫 do
 8          if actionMode[p] = sabbatical then
 9             currStrategy[p] ← constructStrategy(p)
10             applyLocalSearch(currStrategy[p])
11             genStrategy ← clusterOptimization(currStrategy[p])
12             if cost(genStrategy) < bestStrategyCost then
13                bestStrategy[p] ← currStrategy[p]
14                bestStrategyCost[p] ← cost(genStrategy)
15             end if
16          else
17             c ← chooseConsultant(p)
18             if c ≠ null then
19                currSol[p] ← constructSolution(p, c)
20                applyLocalSearch(currSol[p])
21                currGenSol[p] ← clusterOptimization(currSol[p])
22                if currGenSol[p] is better than all solutions found
23                         by a client of c since last sabbatical then
24                   successCount[c] ← successCount[c] + 1
25                   strategy[c] ← currSol[p]
26                end if
27             end if
28          end if
29       end foreach
30       updateReputations()
31       updateActionModes()
32    end while
33 end procedure
```

Figure 2. The CGS-GTSP algorithm

A virtual person may be in one of the following modes: *normal* and *sabbatical*. During the initialization phase (lines 2-5), virtual people are created and placed in sabbatical mode. Based on its mode, a virtual person constructs at each iteration of the algorithm (lines 7-31) either a global solution to the problem (line 19) or a global consultant strategy (line 9). In subsection 4.2, we describe the operations involved by the construction of a global solution or strategy, as well as the method used by a client in order to choose a consultant for the current iteration (line 17).

Global strategies and global solutions are improved by applying a local search procedure (lines 10 and 20). The *clusterOptimization* procedure described in section 2 is then used to find the best generalized strategy (line 11) corresponding to the current global strategy or to find the best generalized solution (line 21) corresponding to the current global solution.

After constructing a global strategy, a virtual person in sabbatical mode checks if the corresponding generalized strategy is the best generalized strategy found since the beginning of the sabbatical (lines 12-15). Similarly, after constructing a global solution, a client checks the corresponding generalized solution in order to decide if it has achieved a success and, if this is the case, it updates the strategy of its consultant (lines 22-26).

At the end of each iteration, the reputation and action mode of each virtual person are updated (lines 30-31).

Figure 3 details how consultants' reputations are updated based on the successes achieved by their clients.

```
 1 procedure updateReputations()
 2    foreach p ∈ 𝒫 do
 3       if actionMode[p] = normal then
 4          rep[p] ← rep[p] * (1 - fadingRate)
 5          rep[p] ← rep[p] + successCount[p]
 6          if cost(currGenSol[p]) < cost(bestSoFarSol) then
 7             bestSoFarSol ← currGenSol[p]
 8             rep[p] ← rep[p] + 10          // reputation bonus
 9          end if
10          if rep[p] > 10 * initialReputation then
11             rep[p] ← 10 * initialReputation
12          end if
13          if p is the best consultant then
14             if rep[p] < initialReputation then
15                rep[p] ← initialReputation
16             end if
17          end if
18       end if
19    end foreach
20 end procedure
```

Figure 3. Procedure to update reputations

Reputations fade over time at a constant rate, given by the parameter *fadingRate* (line 4). The reputation of a consultant is incremented with each success achieved by one of its clients (line 5) and it receives an additional bonus of 10 for finding a best-so-far solution (lines 6-9). The reputation of a consultant cannot exceed a maximum value (lines 10-12) and the algorithm prevents the reputation of the best consultant, that is, the consultant that has found the best-so-far solution, from sinking below a given value (lines 13-17). The constant parameter *initialReputation* represents the reputation assigned to a consultant at the end of the sabbatical leave.

Figure 4 details how the action mode of each virtual person is updated: consultants whose reputations have sunk below the minimum level are placed in sabbatical mode, while consultants whose sabbatical leave has finished are placed in normal mode.

```
 1 procedure updateActionModes()
 2    foreach p ∈ 𝒫 do
 3       if actionMode[p] = normal then
 4          if rep[p] < 1 then
 5             setSabbaticalMode(p)
 6          end if
 7       else
 8          sabbaticalCountdown ← sabbaticalCountdown – 1
 9          if sabbaticalCountdown = 0 then
10             setNormalMode(p)
11          end if
12       end if
13 end procedure
```

Figure 4. Procedure to update action modes

Figure 5 shows the actions taken to place a virtual person in sabbatical or in normal action mode.

```
 1 procedure setSabbaticalMode(p)
 2    actionMode[p] ← sabbatical
 3    bestStrategy[p] ← null
 4    bestStrategyCost[p] ← ∞
 5    sabbaticalCountdown ← 20
 6 end procedure

 7 procedure setNormalMode(p)
 8    actionMode[p] ← normal
 9    rep[p] ← initialReputation
10    strategy[p] ← bestStrategy[p]
11 end procedure
```

Figure 5. Procedures to set the sabbatical and normal mode

4.2 Strategy and Solution Construction

The heuristic used during the sabbatical leave in order to build a new strategy is based on virtual distances between the supernodes in the global graph. We compute the virtual distance between two supernodes as the distance between the centers of mass of the two corresponding clusters. The choice of this heuristic is justified by the class of problems for which our algorithm is designed: symmetric instances with Euclidean distances, where the nodes of a cluster are spatially close one to the other.

By introducing virtual distances between clusters, we have the possibility to use candidate lists in order to restrict the number of choices available at each construction step. For each cluster i, we consider a candidate list that contains the closest $cand$ clusters, where $cand$ is a parameter. This way, the feasible neighborhood of a person k when being at cluster i represents the set of clusters in the candidate list of cluster i that person k has not visited yet. Several heuristic algorithms for the TSP use candidate lists during the solution construction phase (see [1] for examples of their use with Ant Colony Optimization algorithms), but candidate lists have not been widely used to construct solutions for the GTSP. Our algorithm uses candidate lists during both strategy construction and solution construction.

The use of candidate lists may significantly improve the time required by an algorithm, but it could also lead to missing good solutions. Therefore, the choice of appropriate sizes and elements of the candidate lists is critical for the working of an algorithm. In the case of TSP, candidate lists with size 20 are frequently used, but other values between 10 and 40 are also usual [10]. For GTSP instances with clusters composed of nodes spatially close to each other, appropriate sizes for the candidate lists are considerably smaller. Our experiments show that values of 4 or 5 are adequate in this case.

During the sabbatical leave, a consultant uses a random proportional rule to decide which cluster to visit next. For a consultant k, currently at cluster i, the probability to choose cluster j is given by formula (1):

$$p_{ij}^k = \frac{1/d_{ij}}{\sum_{l \in \mathcal{N}_i^k}(1/d_{il})} \quad (1)$$

where:

- $\mathcal{N}_i^k$ is the feasible neighborhood of person k when being at cluster i.
- d_{il} is the virtual distance between clusters i and l.

As mentioned before, the feasible neighborhood $\mathcal{N}_i^k$ contains the set of clusters in the candidate list of cluster i that person k has not visited yet. If all the clusters in the candidate list have already been visited, the consultant can choose one of the clusters not in the candidate list, using a random proportional rule similar to that given by formula (1).

Using virtual distances between clusters as a heuristic during the sabbatical leave, leads to reasonably good initial strategies. In general, however, a global tour that is optimum with respect to the virtual distances between clusters does not produce the optimum generalized tour after applying the cluster optimization procedure. Therefore, during the solution construction phase, the algorithm does not rely on the distances between clusters, although it still uses candidate lists in order to determine the feasible neighborhood of a cluster.

At each step, a client receives a recommendation regarding the next cluster to be visited. This recommendation is based on the global tour advertised by the consultant. Let i be the cluster visited by the client k at a construction step of the current iteration. To decide which cluster to recommend for the next step, the consultant finds the position at which the cluster i appears in its advertised global tour and identifies the cluster that precedes i and the cluster that succeeds i in this tour. If neither of these two clusters is already visited by the client, the consultant randomly recommends one of these two clusters. If only one of these two clusters is unvisited, this one is chosen to be recommended. Finally, if both clusters are already visited, the consultant is not able to make a recommendation for the next step.

The client does not always follow the consultant's recommendation. The rule used to choose the next cluster j to move to is given by formula (2):

$$j = \begin{cases} v & , if\ v \neq null \wedge q \leq q_0 \\ random(\mathcal{N}_i^k) & , otherwise \end{cases} \quad (2)$$

where:

- v is the cluster recommended by the consultant for the next step.
- q is a random variable uniformly distributed in [0,1] and q_0 ($0 \leq q_0 \leq 1$) is a parameter.
- $\mathcal{N}_i^k$ is the feasible neighborhood of person k when being at cluster i.
- $random$ is a function that randomly chooses one element from the set given as argument.

Again, if all the clusters in the candidate list have already been visited, the feasible neighborhood $\mathcal{N}_i^k$ is empty. In this case, a client that ignores the recommendation of its consultant can choose one of the clusters not in the candidate list, using a random proportional rule similar to that given by formula (1).

The personal preference of a client for a given consultant is computed as the inverse of the cost of the generalized tour corresponding to the global tour advertised by the consultant. In conjunction with the reputation, the personal preference is used by clients in order to compute the probability to choose a given consultant k:

$$p_k = \frac{(reputation_k \cdot preference_k)^2}{\sum_{c \in \mathcal{C}}(reputation_c \cdot preference_c)^2} \quad (3)$$

where $\mathcal{C}$ is the set of all available consultants.

4.3 An Algorithm Variant Using Confidence

In this subsection, we propose a variant of our algorithm based on the approach introduced in [8], which correlates the recommendation of a consultant with a level of confidence. Each arc in the global tour advertised by a consultant has an associated strength. Strengths are updated each time the consultant adjusts its strategy. If an arc in the new advertised tour was also present in the old advertised tour, its strength will be incremented; otherwise, its strength is set to 0. The strength of an arc could be interpreted as the consultant's confidence in recommending this arc to a client. A client is more likely to accept recommendations made with greater confidence. This idea is expressed in this algorithm variant by allowing the value of the parameter q_0 from formula (2) to vary in a given range, at each construction step:

$$q_0 = \begin{cases} q_{min} + s \cdot \dfrac{q_{max} - q_{min}}{s_{max}} & ,if\ s < s_{max} \\ q_{max} & ,otherwise \end{cases} \quad (4)$$

where s is the strength of the recommended arc and q_{min}, q_{max} and s_{max} are constant parameters. The use of confidence compensates somewhat for the absence of a heuristic during the solution construction phase.

4.4 Local Search

The global tours built during the strategy construction and solution construction phase are improved using a local search procedure generically described in Figure 6.

```
1  procedure applyLocalSearch(H_G)
2      H ← clusterOptimization(H_G)
3      foreach H'_G ∈ tourNeighborhood(H_G) do
4          if quickCheck(H'_G) then
5              H' ← partialClusterOptimization(H'_G, H)
6              if cost(H') < cost(H) then
7                  H_G ← H'_G
8                  H ← H'
9              end if
10         end if
11     end foreach
12 end procedure
```

Figure 6. The local search procedure

H_G and H'_G denote global Hamiltonian tours, that is, tours in the graph of clusters, while H and H' denote generalized Hamiltonian tours. Our algorithm can be combined with any local search procedure conforming to the above algorithmic structure. The working of the *clusterOptimization* function (line 2) is explained in section 2. The *cost* function (line 6) computes the cost of a generalized Hamiltonian tour. The other functions referred in Figure 6 are only generically specified and they must be implemented by each concrete instantiation of the local search procedure.

The *tourNeighborhood* function (line 3) should return a set of global tours representing the neighborhood of the global tour H_G provided as argument. The *quickCheck* function (line 4) is intended to speed up the local search by quickly rejecting a candidate global tour from the partial cluster optimization, if this tour is not likely to lead to an improvement.

The *partialClusterOptimization* function (line 5) starts with the generalized tour obtained by traversing the nodes of H in accordance with the ordering of clusters in the global tour H'_G. Then, it reallocates some vertices in the resulting generalized tour, trying to improve its cost. Typically, this function considers only a limited number of vertices for reallocation and it usually has a lower complexity than the *clusterOptimization* function.

The generalized tour constructed by the function *partialClusterOptimization* is accepted only if its cost is better than the cost of the current generalized tour (lines 6-9).

We provide two instantiations of the generic local search procedure shown in Figure 6: one based on a 2-opt local search and one based on a 3-opt local search. We describe here only the 2-opt based variant. Except from the fact that it considers exchanges between 3 arcs, the 3-opt based local search is very similar to the 2-opt based variant.

In the 2-opt based local search, the *tourNeighborhood* function returns a set of global tours obtained by replacing a pair of arcs (C_α, C_β) and (C_γ, C_δ) in the original global tour with the pair of arcs (C_α, C_γ) and (C_β, C_δ). In order to reduce the number of exchanges taken into consideration, the set returned by our *tourNeighborhood* function includes only tours for which γ is in the candidate list of α. In other word, a pair of arcs is considered for exchange only if the center of mass of the cluster γ is close to the center of mass of the cluster α.

The *partialClusterOptimization* function used in this case is similar to the RP1 procedure introduced in [3]. Let (C_α, C_β) and (C_γ, C_δ) be the two arcs from the original global tour H_G that have been replaced with (C_α, C_γ) and (C_β, C_δ) in the neighbor global tour H'_G, as shown in Figure 7:

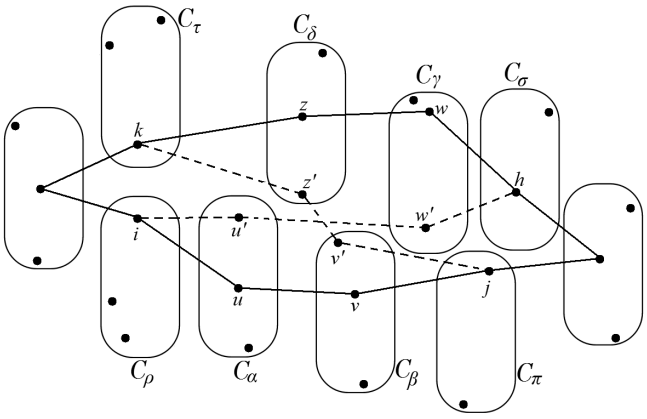

Figure 7. The 2-opt partial cluster optimization

The vertices in clusters C_α, C_β, C_γ and C_δ can be reallocated, in order to minimize the cost of the generalized tour. For this purpose, we have to determine the two node pairs (u', w') and (v', z') such that:

$$d_{iu'} + d_{u'w'} + d_{w'h} = min\{d_{ia} + d_{ab} + d_{bh} \mid a \in C_\alpha, b \in C_\gamma\}$$
$$d_{jv'} + d_{v'z'} + d_{z'k} = \min\{d_{ja} + d_{ab} + d_{bk} \mid a \in C_\beta, b \in C_\delta\} \quad (6)$$

This computation requires $|C_\alpha||C_\gamma| + |C_\beta||C_\delta|$ comparisons.

The *quickCheck* function permits the application of the partial cluster optimization only if the following inequality holds:

$$d_{min}(C_\rho, C_\alpha) + d_{min}(C_\alpha, C_\gamma) + d_{min}(C_\gamma, C_\sigma) + d_{min}(C_\pi, C_\beta) + d_{min}(C_\beta, C_\delta) + d_{min}(C_\delta, C_\tau) < d_{iu} + d_{uv} + d_{vj} + d_{hw} + d_{wz} + d_{zk} \quad (7)$$

where $d_{min}(C_\alpha, C_\beta)$ is the minimum distance between each pair of vertices from clusters C_α and C_β. These minimum distances are computed only once, at algorithm startup.

5. EXPERIMENTAL SETUP

We have implemented our algorithm as part of a software package written in Java, which is available online at http://swarmtsp.sourceforge.net/. At this address we provide all information necessary to reproduce our experiments.

The parameters of the algorithm have been tuned using the paramILS configuration framework [6]. ParamILS executes an iterated local search in the parameter configuration space and it is appropriate for algorithms with many parameters, where a full factorial design becomes intractable. We have generated a set of 100 Euclidean TSP instances with the number n of cities uniformly distributed in the interval [200, 500] and with coordinates uniformly distributed in a square of dimension 10000 x 10000. These instances have been then converted to GTSP by applying the CLUSTERING procedure introduced in [3]. This procedure sets the number of clusters $s = \lceil n/5 \rceil$, identifies the s farthest nodes from each other and assigns each remaining node to its nearest center. We have used the resulting GTSP instances as training data for paramILS.

Before starting the tuning procedure, we have run our algorithm 10 times on each instance in the training set, using a default configuration. Each run has been terminated after n/10 seconds and we have stored the best result obtained for each GTSP instance. During the tuning procedure, these best known results are used as termination condition for our algorithm. Each time paramILS evaluates a parameter configuration with respect to a given instance, we determine the mean time (averaged over 10 trials) needed by our algorithm in order to obtain a result at least as good as the best known result for this instance, using the given parameter configuration.

The best parameter configuration found after 10 iterations of paramILS is given in Table 1:

Table 1. Parameter configuration for the standard algorithm

Parameter	Value	Description
m	8	number of virtual persons.
q_0	0.8	*see formula (2).*
initialReputation	6	reputation after sabbatical; see Figure 3 and Figure 5.
reputationFadingRate	0.003	reputation fading rate; *see Figure 3.*
candidateListSize	5	number of clusters in the candidate list.

For the algorithm variant using confidence, we have used the same procedure as for the standard algorithm, but we have tuned only the values of the parameters q_{min}, q_{max} and s_{max}. For the parameters m, initialReputation, reputationFadingRate and candidateListSize we have used the values from Table 1. The best parameter configuration found for the algorithm variant with confidence after 10 iterations of paramILS is given in Table 2:

Table 2. Parameter configuration for the algorithm variant with confidence

Parameter	Value	Description
q_{min}	0.7	parameters used to compute the value of q_0; *see formulas (2) and (4).*
q_{max}	0.98	
s_{max}	3	

6. COMPUTATIONAL RESULTS

The performance of the proposed algorithm has been tested on 18 GTSP problems generated from symmetric Euclidean TSP instances. These TSP instances, containing between 198 and 442 nodes, are drawn from the TSPLIB [12] benchmark library. The corresponding GTSP problems are obtained by applying the CLUSTERING procedure introduced in [3]. For 16 of the considered GTSP instances, the optimum objective values have been determined by Fischetti *et al.* [3]. For the remaining 2 instances (45tsp225 and 56a280), the best known results from the literature are conjectured to be optimal.

Currently, the memetic algorithm of Gutin and Karapetyan [4] clearly outperforms all published GTSP heuristics. Therefore, we use this algorithm as a yardstick to evaluate the performance of the different variants of our algorithm. We use the following acronyms to identify the algorithms used in our experiments:

- GK: the memetic algorithm of Gutin and Karapetyan [4].
- CGS-2: the standard variant of our algorithm combined with 2-opt local search.
- CGS-3: the standard variant of our algorithm combined with 3-opt local search.
- CGS-C-2: the variant of our algorithm using confidence combined with 2-opt local search.
- CGS-C-3: the variant of our algorithm using confidence combined with 3-opt local search.

For each GTSP instance, we run each algorithm 25 times and we report the average time needed to obtain the optimal solution. For the GK algorithm, we use the C++ implementation offered by its authors. The running times for GK differ from the values reported in [4], because we run our experiments on a 32-bit platform using an Intel Core2 Duo 2.2 GHz processor, while the results presented in [4] have been obtained on a 64-bit platform and using a faster processor (AMD Athlon 64 X2 3.0 GHz).

The computational results are shown in Table 3. The name of each problem is prefixed by the number of clusters and it is suffixed by the number of nodes. Average times that are better than those obtained by the GK algorithm are in boldface. For each problem and for each CGS algorithm variant, we also report the p-values of the one-sided Wilcoxon rank sum tests for the null hypothesis (H_0) that for the given problem there is no difference between the running times of the considered algorithm variant and the running times of the GK algorithm, and for the alternative hypothesis (H_1) that the considered algorithm outperforms the GK algorithm for the given problem. Applying the Bonferroni correction for multiple comparisons, we obtain the adjusted α-level: 0.05 / 18 = 0.00278. The p-values in boldface indicate the cases where the null hypothesis is rejected at this significance level.

Table 3. Times (in seconds) needed to find the optimal solutions, averaged over 25 trials.

Problem instance	Optimal cost	GK	CGS-C-3		CGS-C-2		CGS-3		CGS-2	
		time	time	p-value	time	p-value	time	p-value	time	p-value
40d198	10557	0.46	**0.36**	**0.0004**	**0.33**	**0.0012**	0.47	0.0034	**0.45**	0.0050
40kroA200	13406	0.38	**0.33**	**0.0000**	**0.25**	**0.0000**	**0.37**	0.5711	**0.30**	**0.0001**
40kroB200	13111	0.48	0.60	0.9460	**0.37**	**0.0008**	0.59	0.9689	0.60	0.6156
41gr202	23301	0.71	**0.64**	0.0141	0.91	0.4674	1.35	1.0000	1.10	0.9101
45ts225	68340	0.61	3.32	1.0000	4.06	0.9957	1.92	0.9999	2.67	1.0000
45tsp225	1612	0.51	4.83	1.0000	3.25	0.9994	4.07	1.0000	2.28	0.9967
46pr226	64007	0.28	**0.13**	**0.0000**	**0.07**	**0.0000**	**0.13**	**0.0000**	**0.09**	**0.0000**
46gr229	71972	0.81	**0.36**	**0.0000**	**0.33**	**0.0000**	**0.39**	**0.0000**	**0.37**	**0.0000**
53gil262	1013	0.83	1.22	0.1071	2.63	0.9999	1.63	1.0000	3.49	1.0000
53pr264	29549	0.67	**0.57**	0.0070	**0.49**	**0.0005**	0.94	0.9482	1.08	0.9406
56a280	1079	0.94	1.79	0.8215	3.71	0.9999	2.02	0.9998	4.46	1.0000
60pr299	22615	1.10	3.54	0.9992	2.91	0.9992	3.23	1.0000	4.74	0.9999
64lin318	20765	1.16	**0.85**	**0.0000**	2.68	0.9929	1.28	0.8946	3.81	1.0000
80rd400	6361	2.57	10.30	0.9996	13.27	1.0000	87.96	1.0000	270.04	1.0000
84fl417	9651	1.91	**1.10**	**0.0000**	**1.59**	**0.0001**	**1.51**	**0.0012**	2.27	0.0512
87gr431	101946	6.01	8.16	0.8361	12.86	0.9916	477.38	1.0000	866.53	1.0000
88pr439	60099	4.07	**1.56**	**0.0000**	**1.32**	**0.0000**	**3.68**	0.0104	10.71	0.9999
89pcb442	21657	4.24	11.11	0.9980	13.53	1.0000	395.93	1.0000	1430.13	1.0000

It can be observed that CGS-C-3 outperformed GK for 9 of the 18 instances and in 7 cases these results are significantly better. CGS-C-2 outperformed GK for 8 of the 18 instances and in all these 8 cases the results are significantly better. The variants without confidence perform poorer and for a few instances they need considerably more time to find the optimal solution.

For several pairs of algorithms, we use the one-sided Wilcoxon signed rank test to compute the p-values for the null hypothesis (H_0) that there is no difference between the running times of the first and the running times of the second algorithm, and the alternative hypothesis (H_1) that the running times of the first algorithm are better than the running times of the second algorithm. The p-values are given in Table 4, where the significant values ($p < 0.05$) are in boldface.

Table 4. Performance comparison using the one-sided Wilcoxon signed rank test

First algorithm	Second algorithm	p-value
GK	CGS-C-3	0.1061
GK	CGS-C-2	**0.0368**
GK	CGS-3	**0.0069**
GK	CGS-2	**0.0005**
CGS-C-3	CGS-C-2	0.0708
CGS-C-3	CGS-3	**0.0152**
CGS-C-2	CGS-2	**0.0028**

It can be observed that GK outperforms our algorithms, but in the case of CGS-C-3, the differences are not statistically significant. Similarly, CGS-C-3 outperforms CGS-C-2, but not statistically significant. The fact that 3-opt local search does not significantly improve the results obtained with 2-opt local search could be a consequence of the greater complexity of 3-opt. There are, however, significant differences between the running times of CGS variants with confidence and those without confidence. Due to the very poor results obtained in some cases by the algorithm variants without confidence, these differences are not only statistically, but also practically significant, thus indicating the importance of the *confidence* component.

Figure 8 shows how the candidate list size affects the time needed by CGS-C-3 to find the optimal solution of the problem instance 64lin318. The results are averaged over 25 trials.

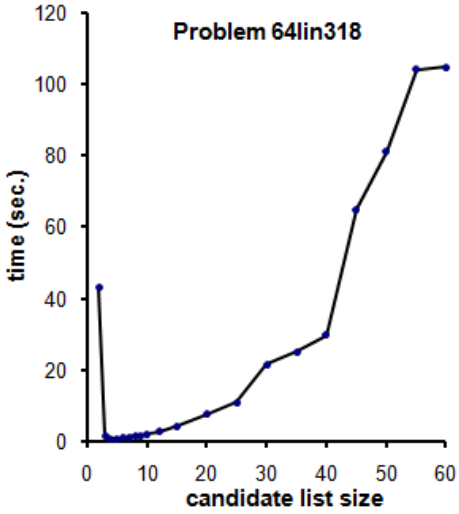

Figure 8. The influence of the candidate list size on the time needed to find the optimal solution for problem 64lin318

It can be observed that the size of the candidate list has a huge influence on the time needed by the algorithm to find the optimal solution. Therefore, the use of candidate lists is a key component contributing to the success of our algorithm.

The best results are obtained for candidate lists of size 4 or 5, but we should note that the algorithm is able to find the optimum even for candidate lists with only 2 elements. However, in this case the time needed increases considerably. This is due to the fact that the probability to find the next cluster of the optimal tour in the

candidate list of the current cluster is significantly smaller when using a candidate list with only 2 elements. For the 64lin318 instance, only 44 of the 64 clusters are present in the candidate list of their precedent cluster when using candidate lists with 2 elements. In contrast, 59 of the 64 clusters are present when using candidate lists with 5 elements. The algorithm is able to find the optimal solution even for very small sized candidate lists, because during the construction phase a client may visit clusters not contained in the current candidate list, if all clusters in this candidate list are already visited or when the consultant recommends it.

For candidate lists with a large number of elements, the algorithm performance in terms of running time worsens, due to the increase in the number of exchanges considered during the local search.

7. Acknowledgements

This work was cofinanced from the European Social Fund through Sectoral Operational Programme Human Resources Development 2007-2013, project number POSDRU/89/1.5/S/56287 "Postdoctoral research programs at the forefront of excellence in Information Society technologies and developing products and innovative processes", partner University of Oradea.

8. CONCLUSIONS AND FUTURE WORK

We have described an efficient algorithm that combines the consultant-guided search heuristic with a local-global approach in order to solve the GTSP. The local-global approach distinguishes between global connections (connections between clusters) and local connections (connections between nodes from different clusters). Our algorithm constructs Hamiltonian tours in the global graph obtained by replacing all nodes of a cluster with a supernode representing it.

The algorithm takes advantage of the fact that most GTSP instances of practical importance are symmetric problems with Euclidean distances, where the clusters are composed of nodes that are spatially close one to the other. For this class of problems, a useful measure is the virtual distance between two supernodes, computed as the distance between the centers of mass of the two corresponding clusters. We use virtual distances as a heuristic during the strategy construction phase. Additionally, based on the virtual distances between clusters, the algorithm creates candidate lists used for strategy and solution construction, as well as for local search. The use of candidate lists significantly reduces the search space and contributes to the efficiency of our algorithm.

A variant of our algorithm, which uses the concept of confidence in relation to the recommendations made by consultants, improves the algorithm performance, especially for large instances.

Computational results show that there are no statistically significant differences between our algorithm variant with confidence and the memetic algorithm of Gutin and Karapetyan (GK), which is currently the best published heuristic for the GTSP. The GK algorithm uses a sophisticated local improvement strategy that combines many local search heuristics. One goal of our future research is to adopt a similar approach for the local improvement part of our algorithm, but still using candidate lists for each local search heuristic considered.

9. REFERENCES

[1] Bontoux, B., Artigues, C. and Feillet, D. *A Memetic Algorithm with a large neighborhood crossover operator for the Generalized Traveling Salesman Problem*. Computers & Operations Research, 37 (11), 2010, 1844-1852.

[2] Dorigo, M. and Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge, 2004.

[3] Fischetti, M. *A Branch-and-Cut Algorithm for the Symmetric Generalized Travelling Salesman Problem*. Operations Research , 45 (3), 1997, 378-394.

[4] Gutin, G. and Karapetyan, D. *A memetic algorithm for the generalized traveling salesman problem*. Natural Computing, vol. 9, 2010, 47-60.

[5] Hu, B. and Raidl, G. *Effective neighborhood structures for the generalized traveling salesman problem*. In: Proc. of Evolutionary Computation in Combinatorial Optimisation - EvoCOP 2008, LNCS, Vol. 4972, Naples, Italy, 2008, 36-47.

[6] Hutter, F., Hoos, H.H., Leyton-Brown, K. and Stützle, T. *ParamILS: An Automatic Algorithm Configuration Framework*. Journal of Artificial Intelligence Research (JAIR), vol. 36, October 2009, 267-306.

[7] Iordache, S. *Consultant-Guided Search - A New Metaheuristic for Combinatorial Optimization Problems*. In: GECCO 2010: Proceedings of the 12th Genetic and Evolutionary Computation Conference. ACM Press, 2010.

[8] Iordache, S. *Consultant-Guided Search Algorithms with Local Search for the Traveling Salesman Problem*. In: PPSN XI - International Conference Parallel Problem Solving from Nature. LNCS 6239, Krakow, Poland, Springer, 2010, 81-90.

[9] Iordache, S. *Consultant-Guided Search Algorithms for the Quadratic Assignment*. In: Hybrid Metaheuristics - 7th International Workshop, HM 2010. LNCS 6373, Vienna, Austria. Springer, 2010, 148-159.

[10] Johnson, D.S. and McGeoch, L. *Experimental Analysis of Heuristics for STSP*. In: Gutin, G., Punnen, A. (eds) The Traveling Salesman Problem and its Variations. Kluwer, Dordrecht, 2002.

[11] Pop, P.C. *The generalized minimum spanning tree problem*. Twente University Press, The Netherlands, 2002.

[12] Reinelt, G. *TSPLIB - A Traveling Salesman Problem Library*, ORSA Journal on Computing, vol. 3, no. 4, 1991, 376-384.

[13] Silberholz, J. and Golden, B. *The Generalized Traveling Salesman Problem: a new Genetic Algorithm approach*. Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies, 2007, 165-181.

[14] Snyder, L.V. and Daskin, M.S. *A random-key genetic algorithm for the generalized traveling salesman problem*. European Journal of Operational Research 174, 2006, 38-53.

[15] Tasgetiren, M.F., Suganthan, P.N. and Pan, Q.-K. *A discrete particle swarm optimization algorithm for the generalized traveling salesman problem*. GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, 2007, 158-167.