Efficient Clustering Techniques for the Geometric Traveling Salesman Problem

B. Codenotti\textsuperscript{1}   L. Margara\textsuperscript{2}

TR-92-036

June 1992

Abstract
This paper presents some direct and iterative heuristic methods for the geometric Traveling Salesman Problem (TSP). All these methods are based on a particular notion of mass density, which can be used to construct a tour for the geometric TSP in an incremental fashion. In the iterative method, this technique is combined with the Lin-Kernighan method (LK), and this allows us to obtain better tours than those found by using LK itself. More precisely, the tour length we get is only 1.1\% off the optimum. The direct method finds a solution passing through a sequence of subsolutions over progressively larger sets of points. These points are the relative maxima of the mass density obtained by using different parameter settings. The method has $O(n^3)$ worst case running time and finds tours whose length is 9.2\% off the optimal one.

\textsuperscript{1}This work has been partly supported by the Italian National Research Council, under the “Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo”, subproject 2 “Processori dedicati”.
\textsuperscript{2}International Computer Science Institute, Berkeley, CA 94704 and Istituto di Elaborazione dell’Informazione, Consiglio Nazionale delle Ricerche, Pisa (Italy).
\textsuperscript{3}International Computer Science Institute, Berkeley, CA 94704 and Dipartimento di Informatica, Università di Pisa (Italy).
1 Introduction

In this paper, we present a general clustering technique which can be applied to sets of points distributed on a $k$-dimensional space. Based on this technique, we develop iterative and direct methods for the geometric Traveling Salesman Problem (TSP) on the plane.

The main idea is to partition the plane into clusters. These clusters are constructed according to information gathered from the location of the maxima of a function. Such function, which we call mass density, measures the "density" of the cities in the plane.

We analyze each problem instance in terms of the distribution of the cities. The density of mass allows us to view the TSP graph as a set of clusters instead of a set of cities (see [13] for a related approach). Each cluster is completely determined by a pair $(c, r)$, where $c$ and $r$ are the center and the radius of the cluster, respectively. The centers of the clusters are given by the maxima of the density function, while the radius are determined by using the technique described in section 3.

In the case of the iterative methods, we take advantage of the clustering to extend a subsolution by adding one cluster of cities at a time (rather than one city at a time) until a complete solution is found. More precisely, at each cluster insertion, we extend a subsolution $s_\alpha$ over a set of points $N_\alpha$, to a subsolution $s_n$ over a larger set of points $N_n$. Let $\text{opt}_N$ the length of the optimal tour over the set of points $N$. Then, in general, $\text{length}(s_\alpha) < \text{length}(s_n)$. We are able to bound the effect of this fact by using the Lin-Kernighan heuristic which keeps the cost of the subsolution as close as possible to the optimal one.

Moreover, iterative methods, once used inside a constructive method seem to loose their exponential nature. In particular, Lin-Kernighan heuristic becomes much faster once used inside our technique.

In the case of the direct method we propose, we use the density function with different parameter setting, in order to obtain "different level descriptions" of the TSP graph.

In fact, when the parameter $c$ of the density function assumes values close to 0, the set of maxima has the same cardinality as the set of the cities. Moreover, each maximum is close to a different city. In this case, the set of maxima provide an accurate description of the problem instance. When $c$ grows, the cardinality of the set of maxima decreases, and each maximum represents a progressively larger set of cities. Then we obtain high level descriptions of the TSP graph.

The algorithm first constructs a solution over a set of maxima with fixed cardinality (in our experiments the cardinality of the initial set of maxima is roughly 30). Then the subsolution over the maxima obtained with a certain value of the parameter $c$ is extended to another subsolution over a larger set of maxima, obtained modifying the value of $c$. At each step we obtain solutions over sets of cities given by the maxima of the density function with different parameter settings. When $c$ is small enough, the solution found over the maxima is also a solution for the TSP.
The literature on the TSP contains a variety of heuristic methods and approximation algorithms. A major distinction is between constructive (or direct) and iterative methods. The former class essentially identifies methods which construct a solution in polynomial time, while the latter class corresponds to methods which iteratively construct a solution from another one, e.g., by local search, and are not in general guaranteed to terminate in polynomial time (see also [12]). A popular approach consists of using the outcome of a direct method as the starting point of an iterative one. Here we adopt a completely different strategy: we use an iterative method to perform one step of the tour construction process. We thus combine the power of iterative methods with the low cost of direct ones. Intuitively, this is possible because we use an iterative method on a problem of substantially smaller size and with special properties.

Single runs of the Lin-Kernighan heuristic find solutions with cost within 2.2% of the optimal one (estimated by using the Held-Karp lower bound [9]). Our iterative method finds, on average, solutions whose costs are 1.1% lower than those obtained by Lin-Kernighan heuristic.

Among direct methods, Christofides heuristic is probably the most accurate, since it provides solutions which are 10% off the optimum. Our direct method provides, on the average, solutions which are only 9.2% off the Held-Karp lower bound.

The rest of this paper is organized as follows.

Section 2 recalls the basic features of some known methods for solving the TSP. Section 3 describes a general clustering technique which can be applied to any set of points distributed on a \(k\)-dimensional space. Section 4 and 5 contain the description of our heuristics. Section 6 shows how different outcomes of the iterative method presented in section 4 can be combined to obtain improved solutions. Section 7 discusses some implementation issues, and gives an evaluation of the computational cost of our heuristics; section 8 reports on the experimental results we have performed.

2 Existing methods

In this section we recall the basic features of some popular constructive and iterative methods for computing approximate solutions for the TSP.

2.1 Direct methods

**Nearest Neighbor (NN).** This method generates a solution in the following way. It starts with an arbitrary city, and then adds to the path the city not yet visited which is closest to the city last added. In the geometric case, the length of the tour provided by the NN heuristic is always less or equal than \([\frac{3n-1}{2}]\) times the length of the optimal tour (see [23]), where \(n\) is the number of the cities. In the general case, the cost of the NN heuristic is \(O(n^2)\).
Insertion methods (IM). The insertion methods build solutions over larger and larger sets of cities until a solution that visits all the cities is found. The cheapest insertion method (CI) [23], for example, starts from a subsolution $s$ over a subset of the cities (in the geometric case, $s$ can be found by computing the convex hull over all the cities) and subsequently the next city $c$ to be included in the tour is chosen as to minimize the difference between the cost of the subsolution obtained after the insertion of $c$ and the cost of the subsolution before this insertion. In the geometric case, CI provides a tour of length less or equal than twice the optimal one. The running time of CI is $O(n^2 \log n)$.

Christofides heuristic (CH). This heuristic method ([6]) is based on minimum spanning trees. It constructs a minimum spanning tree of the set of points, then computes the optimum matching of all the vertices that have odd degree. At this point, it computes an eulerian tour in the graph by combining the edges of the matching with the edges of the minimal spanning tree. In the geometric case, this method finds a solution which has a cost less or equal than $\frac{3}{2}$ times the cost of the optimal solution. In the worst case, CH takes $O(n^2)$ operations. Among the known constructive techniques, CH heuristic provides, on the average, the shortest tours.

Multiple fragment heuristic (MF). This heuristic ([24]) builds the solution adding, at each step, the edge of minimum length $l$ which makes possible to complete the solution. This means that $l$ does not make early cycles or nodes with degree three. The worst case ratio between the MF solution and the optimum one is $O(\log n)$. The computational cost of MF heuristic is $O(n^2 \log n)$ operations.

2.2 Iterative methods

k-change local search (KLS). Local search is based on the notion of neighborhood. The neighborhood of a solution $s$ is given by a subset $I^1$ of all the feasible solutions. In the case of KLS, the neighborhood of a solution $s$ contains all the feasible solutions which differ from $s$ for at most $k$ edges. KLS transforms a solution $s$ into a better one replacing $k$ arcs in $s$ with $k$ arcs not in $s$. At the end of this procedure, KLS provides a solution $s$ which has cost less or equal than the costs of the solutions which belong to its neighborhood.

The cardinality of the neighborhood in which KLS performs its search is $O(n^k)$. Thus, the local optimality of a solution is verifiable in polynomial time. In practice, local search converges quickly, but it has been shown ([20]) that there exist instances of TSP for which local search requires an exponential number of iterations to reach the local optimum. No algorithm is known for finding a k-change local optimum in polynomial time.

Lin-Kernighan heuristic (LK). LK heuristic [16] passes from a solution $s$ to a better one by changing a number $k$ of arcs not fixed in advance. The procedure tries to find $k$ and a sequence of arcs $x_1, y_1, x_2, y_2, \ldots, x_k, y_k$, such that, taking off the

\footnote{generally $I$ contains a polynomial number of solutions}
arcs $x_1, \ldots, x_k$ and replacing them by $y_1, \ldots, y_k$, the best improvement is obtained. As in the case of local search, LK algorithm may require an exponential number of iterations. One can readily verify that the tours found by LK heuristic have a length less or equal than the length of the tours provided by 3-change local search.

Many other methods for finding approximate solutions to the TSP can be found in the literature ([8, 13, 14, 18, 15, 23]). For simulated annealing heuristics see [14], for elastic net methods see [5] and [7], and for genetic algorithms see [4, 19].

3 A clustering technique

In this section we present a clustering technique which allows us to view an instance of a geometric TSP in terms of clusters of cities instead of single cities. We define a function, called density function, which maps each point $(x, y)$ in the plane onto a nonnegative real number. This number is obtained summing up the contributions of all the cities. The contribution of a city $c$ to the density function evaluated in $(x, y)$ depends on the distance between $c$ and $(x, y)$. The maxima of the density function play an important role. In fact, they localize the clusters (see Definition 6).

We concentrate our attention on the geometric TSP in two dimensions. Our analysis can be extended to the $k$ dimensional case.

We now give some basic definitions.

**Definition 1 (Geometric TSP)** Given a set of points on the plane and using the euclidean distance, find the path of minimum length that visits each point exactly once.

We associate to the geometric TSP a density function $\sigma_G(x, y)^2$ which gives some information about the concentration of cities in a particular region of the plane.

**Definition 2 (Density function)** Let $G$ be a geometric TSP and $(x_i, y_i)$ the coordinates of the $i$th city, $i = 1, \ldots, n$. Then density function $\sigma_G(x, y)$ is the following:

$$\sigma_G(x, y) = \sum_{i=1}^{n} f_i(x, y).$$

where

$$f_i(x, y) = e^{-[(x-x_i)^2+(y-y_i)^2]}.$$  

Note that each $f_i(x, y)$ reaches its maximum at the point $(x_i, y_i)$, where it takes the value 1; hence the maximum of the density function 1 is $n$. The density function depends only on the graph $G$. To make Definition 2 more suitable for our goals, we define a modified density function which contains an extra parameter.

---

2When no confusion can arise we will write $\sigma(x, y)$ instead of $\sigma_G(x, y)$. 

---
Figure 1: Density function $c\sigma_G(x, y)$, for $c = 0.4$ and $G = \{(1, 2), (2, 1)\}$.

Figure 2: Density function $c\sigma_G(x, y)$, for $c = 0.7$ and $G = \{(1, 2), (2, 1)\}$.

**Definition 3 (Modified density function)** Let $G$ be the set of cities of a geometric TSP and $(x_i, y_i)$ the coordinates of the $i^{th}$ city, $i = 1, \ldots, n$. Then the modified density function $c\sigma_G(x, y)$ is the following:

$$c\sigma_G(x, y) = \sum_{i=1}^{n} e^{-\left[\frac{x-x_i}{c}\right]^2 + \left[\frac{y-y_i}{c}\right]^2}.$$  \hfill (2)

We can modify the form of the density function by working on the value of $c$ as shown in figures 1, 2, and 3.

Informally, we can say that the value of $c$ gives an idea about the locality of the function $c\sigma_G(x, y)$, i.e. how much a point $p_1$ located at a certain distance from $p_2$ can influence the value assumed by the density function at $p_2$. 

5
Figure 3: Density function \( \sigma_G(x, y) \), for \( c = 2 \), and \( G = \{(1, 2), (2, 1)\} \).

It can be easily shown that, as \( c \) grows, each city contributes in an increased way to the value of \( \sigma_G(x, y) \), for \( x \) and \( y \) belonging to a progressively larger region of the plane.

More precisely, we can analyze how the density function behaves when \( c \) approaches 0 and \(+\infty\). We have

\[
\forall (x, y) \quad \lim_{c\to\infty} \sigma_G(x, y) = n,
\]

\[
\forall (x, y) \neq (x_i, y_i), \ 1 \leq i \leq n \quad \lim_{c\to0} \sigma_G(x, y) = 0,
\]

\[
\forall i, 1 \leq i \leq n \quad \lim_{c\to0} \sigma_G(x_i, y_i) = 1.
\]

In our analysis we are interested in the localization of the maxima of the function \( \sigma_G(x, y) \). The magnitude and the localization of these maxima depend on \( c \).

**Definition 4 (Maxima)** Given a geometric TSP \( G \),

- \( MN_c \) is the number of maxima of the function \( \sigma_G(x, y) \),
- \( x_{i,c} \) and \( y_{i,c} \) are the coordinates of the \( i^{th} \) maximum,
- \( V_{i,c} \) is the value of the \( i^{th} \) maximum.

**Claim 1** Let \( G \) be a geometric TSP and \( \sigma_G(x, y) \) be the corresponding density function. We have

\[ MN_c \geq MN_d, \quad d > c > 0. \]
Figure 4: Number of maxima as function of the parameter $c$ for a geometric TSP with 100 cities.

Figure 5: Number of maxima as function of the parameter $c$ for a geometric TSP with 316 cities.

**Claim 2** Let $\bar{c}$ be the value of $c$ such that, if $c > \bar{c}$, the maximum $V_c$ of $e^{\sigma_G(z,y)}$ exists and is unique, and let $(x_c, y_c)$ be the coordinates of $V_c$. Then

$$\lim_{c \to +\infty} (x_c, y_c) = (x_m, y_m),$$

where $x_m = \frac{\sum_{i=1}^{n} x_i}{n}$, $y_m = \frac{\sum_{i=1}^{n} y_i}{n}$. \hfill \blacksquare$

For $c$ large enough, the density function has a unique maximum.

Figures 4 and 5 show how the number of maxima depends on the parameter $c$ in the case of two geometric TSPs with 100 and 316 uniformly distributed cities, respectively.

Figures 6, 7, and 8 show a 200-cities geometric TSP, and the maxima evaluated for $c=100$, $c=50$, and $c=20$, respectively.
Figure 6: 200-cities graph and maxima with c=100, (dots represent cities and squares represent maxima)

Figure 7: 200-cities graph and maxima with c=50, (dots represent cities and squares represent maxima)
The definition of density function suggests a generalization of the notion of convex hull of a set of points.

**Definition 5 (Generalized convex hull)** Let $G = (N, E)$ be a geometric TSP and $\sigma_G(x, y)$ the density function associated to $G$. Consider the set of points $(x_{j,c}, y_{j,c})$, $1 \leq j \leq MN_c$, for which the density function attains its maxima, then the the generalized convex hull is the convex hull formed on $(x_{j,c}, y_{j,c})$, $1 \leq j \leq MN_c$.

In general, the extreme points determined by Definition 5 do not coincide with any city. Thus the generalized convex hull does not provide a subsolution to the TSP associated to $G$. One can overcome this complication by considering as extreme points of the generalized convex hull the cities which are closer to the maxima.

We can now define the notion of cluster of cities.

**Definition 6 (Cluster)** Given a geometric TSP $G$, a point $p$ in the plane, and a real number $r > 0$, we define a cluster of cities $C(p, r)$ as the set of cities in $G$ which are placed within distance $r$ from $p$.

From Definition 6, it follows that a set of clusters is determined by a set of pairs $\{(p_i, r_i), 1 \leq i \leq k\}$, where $p_i$ is a point in the plane (center of the clusters) and $r_i$ is the radius associated to $p_i$, $1 \leq i \leq k$.

In our analysis, the $p_i$’s are expressed by the relative maxima of the density function. For what concerns the radius of the clusters, we need the following definition.
Figure 9: 200-cities graph and maxima with c=50 and clusters for $p = 0.8$, (lines link each maximum with the cities which belong to the cluster)

**Definition 7 (Rebuilding coefficient)** Let $M$ be a maximum of the density function, and let $S_d$ be the subset of cities which lie within distance $d$ from $M$. We map $M$ onto a real number $r_M$ such that

$$r_M = \min\{d : \sum_{c \in S_d} e^{-\left(\frac{x_{M,c} - x_c}{e}\right)^2 + \left(\frac{y_{M,c} - y_c}{e}\right)^2} \geq pM\}, \quad 0 < p < 1.$$**

We call $p$ the rebuilding coefficient.

Given a rebuilding coefficient $p$, we are able to associate to each maximum $M$ the radius $r_M$.

Summarizing, given a geometric TSP $G$, a value for the parameter $c$ in the density function, and a rebuilding coefficient $p$, we uniquely determine a set of clusters

$$C = \{C((x_{i,c}, y_{i,c}), r_{(x_{i,c}, y_{i,c})}) \mid 1 \leq i \leq MN_c\}.$$**

Figure 9 shows the clusters for the graph and the maxima in figure 7 for $p = 0.8$. As we can see in Figure 9, there could be cities which do not belong to any cluster.

**4 Iterative methods based on clustering**

In this section we describe two iterative methods for the geometric TSP which take advantage of the density function to localize some regions of the plane containing clusters of cities. The two algorithms will be called CLR1 and CLR2, respectively.
4.1 CLR1

The first heuristic we present starts from an initial subsolution \( s \), and then, at each step, inserts in \( s \) all the cities belonging to a certain cluster. After the insertion, the new subsolution is improved by using an iterative method. This kind of approach allows us to keep the cost of the subsolutions as close as possible to the optimal one. After all the clusters have been considered, if necessary, CLR1 finds a new set of clusters over the cities not yet inserted in the subsolution. The procedure ends when all the cities are inserted in the tour.

CLR1 can be expressed as follows.

Algorithm CLR1.

1. Compute a subsolution \( s \) by using the generalized convex hull approach.
2. Repeat the following 5 steps until a complete solution \( S \) is obtained.
   a. Find all the maxima \( M_i \) of the density function \( c \sigma_G(x,y) \) restricted to the cities not yet inserted in the subsolution \( s \).
   b. Choose the maximum \( M \) closer to the subsolution we are building.
   c. Insert in the subsolution all the cities contained in the cluster \( C(M,r_M) \).
   d. Use local search to reduce the cost of the subsolution.
   e. If all the maxima have been considered then goto step a, else goto step b.

Step 1 produces an initial subsolution \( s \) by using a generalized convex hull procedure. The generalized convex hull procedure computes the convex hull over the set of the relative maxima of the density function. Modifying the value of the parameter \( c \) in the density function we obtain different initial subsolutions and then different solutions. On the average, the solutions obtained starting from a generalized convex hull are better than those obtained starting from the classical convex hull.

Step 2.a finds all the maxima of the density function \( c \sigma_G(x,y) \). The choice of the parameter \( c \) plays a crucial role. In general, different values of \( c \) lead to solutions that differs by 20 – 30% of the arcs. On the average, the cost of all these solutions is very close to the optimum. In section 6 we define an algorithm which combines different outcomes of CLR1, to obtain improved tours.
Figure 10: Tour found by CLR1 for a 100 cities instance. The cities are uniformly distributed on the unit square. The cost of the tour is 0.4% off the Held-Karp lower bound.

The choice of the cluster to insert (Step 2.b) is an attempt to minimize the difference between the cost of the subsolution obtained after the cluster insertion and the cost of the subsolution before the cluster insertion. To this extent, in Step 2.b we choose the maximum $M$ closest to the subsolution built so far. In Step 2.c we insert all the cities contained in the cluster determined in Step 2.b by using the CI method described in section 2. The local search method we have used in Step 2.d is LK algorithm. Experimental results show that LK heuristic, once applied inside CLR1, provides solution which have costs 1% lower than the costs of the solutions found by single runs of LK on the entire problem.

CLR1 depends on two parameters, namely the value of $c$ in the density function and the values of the rebuilding coefficient $p$. In the case of uniform distribution of the cities in the unit square, experimental results show that CLR1 finds the best solutions when $c$ and $p$ are chosen depending on the problem size. As an example, for 100 cities, the setting of the parameters which leads to the best solutions is the following: $50 \leq c \leq 55$ and $0.80 \leq p \leq 0.85$. In general, the best setting of the parameter for a fixed problem size can be determined observing some experimental results on random instances of the given size.

Figures 10 and 11 show two solutions for graphs with 100 and 200 cities, respectively. In particular, Figure 11 also shows a subsolution found in an intermediate stage of the computation.
Figure 11: 200 cities instance. On the left a subsolution found by CLR1 in an
intermediate stage of the computation is illustrated. Dots represent cities not yet
inserted. On the right the final tour is shown. The cost is only 1.02% off the Held-
Karp lower bound.

4.2 CLR2

The second heuristic we present can be split in two stages. In the first stage (bottom-
up), CLR2 computes the set of clusters for progressively larger values of \( c \). When \( c \)
assumes low values, according to the problem size the number of clusters is close to
the number of cities and a single cluster contains few cities (the radius being very
small). At this point, CLR2 replaces each cluster of cities with the center of the
cluster.

In general, the new TSP contains either cities or maxima of the density function.
The above described procedure is repeated until it provides a TSP with at most 3
cities. Replacing clusters with single cities, at each step of the bottom-up stage,
CLR2 gathers information on the structure of the TSP graph. This information will
be exploited in the second stage of the algorithm, when a solution is constructed.

In the second stage (top-down), CLR2 builds the subsolution over the last set
of cities computed during the bottom-up stage. Then it extends this subsolution
replacing each maximum with the cities it represents. The replacement is performed
by using the CI method described in section 2. The subsolution obtained so far is
improved by using the LK heuristic.

CLR2 can be expressed as follows.
Algorithm CLR2.

Bottom-up
1. c=initial_value, and Old_cities=the set of cities.
2. Find all the maxima of the density function.
3. Find the new set of cities New_cities in the following way.
   Take off all the cities which belong at least to one cluster
   from Old_cities, and add the maxima above found to it.
4. Increase c according to a fixed rule.
5. If the set of cities we got through the steps 2 and 3
   contains more than 3 cities goto step 2.

Top-down
6. Find the tour over the set of cities obtained so far.
7. For each maximum in the current set insert all the cities
   it represents in the subsolution we are building,
   i.e., by using an insertion method.
8. Use local search to improve the subsolution.
9. If in the new set of cities there are still maxima
    then goto step 7.
10. End.

Step 3 replaces each cluster $C(p, r)$ found in Step 2 with the single point $p$. At the end of Step 3, we obtain a new graph with less nodes. Step 4 increases the parameter $c$. Let $c_1, c_2, \ldots, c_k$ be the sequence of values taken by $c$ during the bottom-up stage. Then in the top-down stage CLR2 generates a sequence of solutions over a progressively larger set of points which have cardinality $MN_{c_1}, MN_{c_{k-1}}, \ldots, MN_c$. Experimental results show that the sequence $c_1, c_2, \ldots, c_k$ has to be chosen as a function of the problem size. Step 7 extends the subsolution over $c_{i+1}$ points to a subsolution over $c_i$ points by using the CI method. Each time Step 7 is performed, $c_{i+1} - c_i$ cities are inserted in the subsolution. Experimental results show that the length of the tours provided by LK heuristic also depends on the quality of the starting solutions. As an example, LK method applied to a randomly-generated starting solution, finds on the average local optima 1% bigger than those found by applying LK method to more accurate solutions. Each time we perform Step 8, we apply LK heuristic to a subsolution which is only locally perturbed (because of the cluster insertion). As a result, we obtain a sequence of subsolutions with length very close to the optimal one.
5 A tour construction method based on clustering

In this section we present a direct method (CLR3).

CLR3 finds a solution for a geometric TSP passing through a sequence of solutions $s_{c_1}, s_{c_2}, \ldots, s_{c_k}$, $k \leq n$, over sets of point $M_{c_1}, M_{c_2}, \ldots, M_{c_k}$, which have cardinality $n_{c_1}, n_{c_2}, \ldots, n_{c_k}$, respectively. Each set $M_{c_i}$, $1 \leq i \leq k$, contains the maxima of the density function obtained for $c = c_i$. Each solution $s_{c_i}$, $1 \leq i \leq k$, is not a subsolution for the TSP (as in CLR1) since the points in $M_{c_i}$ do not coincide with the cities.

For $c$ small enough in Definition 3, the number of maxima of the density function approaches the number of the cities, and each maximum $(x_{i,c}, y_{i,c}), 1 \leq i \leq n_c$, converges to a different city.

The following constructive algorithm takes advantage of this property to compute a tour for a geometric TSP.

Algorithm CLR3.

1. $c = c_0$.
2. Find all the maxima $M_c = \{(x_{i,c}, y_{i,c}), i = 1, \ldots, n_c\}$ of the density function.
3. Find a solution $s_c$ over $M_c$ by using a constructive method, e.g., convex hull + cheapest insertion method.
4. Let $f$ be a function such that $f(c) < c$. Compute $d = f(c)$.
5. Find all the maxima $M_d = \{(x_{i,c}, y_{i,c}), i = 1, \ldots, n_d\}$ of the density function.
6. Extend the solution $s_c$ over $M_c$ to a solution $s_d$ over $M_d$ by using the polynomial time algorithm described at the end of this section.
   Let $s_c = s_d$ and $c = d$.
7. If a stop criterion occurs, then extend $s_c$ to the set of the cities in order to obtain the solution.

The constant $c_0$ depends on the number of cities. We have implemented Step 1 choosing $c_0$ such that the number of maxima $n_{c_0}$ does not exceed $h$, where $h$ is a constant (in our experiments, we have set $h = 20$). Experimental results show that, for $n_{c_0} \approx 20$, CI method finds solutions whose cost is very close to the optimal one. When $n_{c_0}$ increases, CI method becomes less accurate.

Step 2 computes an initial tour $s_{c_0}$ over the set of points $M_{c_0}$ as follows.

(i) Compute the convex hull $s$ over the set of points $M_{c_0}$.

(ii) Use CI method to obtain a complete solution over $M_{c_0}$. 
Figure 12: Tour found by CLR3 for a 200 cities instance. The cities are uniformly distributed on the unit square. The cost of the tour is 6.1% off the Held-Karp lower bound.

Function $f$, in Step 4, allows us to decrease the value of $c$ so that the cardinality of the set of maxima $M_d$, found in Step 5, grows by a constant factor at each iteration.

The new set of maxima $M_d$ computed is Step 5 provides a more accurate description of the distribution of the cities in the plane than that provided by the previous set of maxima $M_c$.

Let $s_c = i_1, i_2, \ldots, i_{n_c}$ be the solution over $M_c$. Then Step 6 computes a solution over $M_d$ exploiting the structure of $s_c$ according to the following two phases:

(i) Find the points $j_l \in M_d, 1 \leq l \leq n_c$, such that $j_k$ is the closest point to $i_k, 1 \leq k \leq n_c$. Let $s = j_1, \ldots, j_{n_c}$ be a subsolution to $s_d$.

(ii) Insert the $n_d - n_c$ points not yet inserted in $s$ by using CI method (or another polynomial time insertion method) to obtain $s_d$.

Phase (i) finds a subsolution $s$ for $M_d$ which has a structure similar to $s_c$. Phase (ii) extends $s$ to $s_d$ inserting the points in $M_d$ not yet considered in Phase (i).

In general, when we pass from $M_c$ to $M_d$, some clusters contained in $M_c$ splits in two or more clusters contained in $M_d$, and some other clusters move towards the cities they contain.

In Step 7, we check if the cardinality of the set of the maxima is greater than a certain function of the number of cities. If this is the case, we extend the solution computed so far to the set of cities. This is performed by using the algorithm (Step 6) for extending all the intermediate solutions. Otherwise, we go to Step 4.

A tour obtained by using CLR3 is drawn in Figure 12.
A technique for combining different outcomes

In this section we discuss a technique that combines different outcomes of CLR1 in order to obtain improved solutions for the geometric TSP. This technique is called iterative refinement (IR).

As mentioned in section 4, the tours found by method CLR1 can be substantially different, if different values of \( c \) are used. When \( c \) assumes values in a suitable range, then the cost of the solutions are 2% within the optimum. These solutions share roughly 70 - 80% arcs (see Figure 13).

For this reason, one can think of these shared arcs as "desirable" in a very good solution. IR method can be expressed as follows.

Algorithm IR.

1. Let \( s_1, s_2, \ldots, s_k \) be \( k \) initial solutions obtained by running CLR1 for \( c \) equal to \( c_1, c_2, \ldots, c_k \), respectively. Let \( \mu_1 = 0 \).
2. Let \( E \) be the set of the arcs shared by \( s_1, s_2, \ldots, s_k \), and \( \mu_2 \) be its cardinality.
3. Construct a initial subsolution \( S \), with the constraint that it must contain all the arcs in \( E \).
4. Apply CLR1 to \( S \) for \( c \) equal to \( c_1, c_2, \ldots, c_k \), in order to obtain \( k \) new solutions. Let \( s_1, s_2, \ldots, s_k \) be these new solutions.
5. If \( \mu_2 = \mu_1 \) then End.
6. Let \( \mu_1 = \mu_2 \), goto Step 2.

Step 1 finds a certain number of initial solutions \( s_1, s_2, \ldots, s_k \) by using CLR1 with different parameter settings. Step 2 finds a set \( E \) of arcs such that each arc \( l \in E \) satisfies the following property: \( \forall i, 1 \leq i \leq k, l \in s_i \). Step 3 finds a subsolution \( S \) as follows. Let \( s_h = i_1, i_2, \ldots, i_n \) be the solution of minimum cost, over \( s_1, s_2, \ldots, s_k \). Then we construct a subsolution \( S \), deleting from \( s_h \) all the cities which are not extreme points of at least one arc in \( E \). One can readily verify that \( S \) contains all the arcs in \( E \). It is possible to improve \( S \) applying the KLS heuristic to the arcs in \( S \) not contained in \( E \).

CLR1 proceeds through subsolutions over progressively larger sets of cities until a complete solution is found. At each step it adds to the subsolution a cluster of cities and then uses local search to keep the cost as low as possible. As a consequence, every subsolution for a geometric TSP can be viewed as a "starting point" for CLR1.
Step 4 takes advantage of this property and applies CLR1 to the subsolution found in Step 3, with \( k \) different parameter settings.

Step 2, 3, and 4 are repeated until the cardinality of the set of the shared arcs does not increase anymore. Let \( \mu \) be the cardinality of \( E \). Then \( 1 \leq \mu \leq n \). In addition, \( \mu \) never decreases. Then IR method always terminates.

At the end of the procedure we choose the tour of minimal length.

The intuition behind the above approach is that the arcs shared by the outcomes of CLR1 with different \( c \)’s, are likely to satisfy global properties w.r.t. the optimal solution, and one may be willing to keep them when trying to improve the tour length.

Note that the above strategy can also be applied to combine solutions obtained by using other methods. As an example, a slightly modified version of IR can be used to combine local optima obtained by KLS heuristics applied to different initial solutions.

7 Computational cost and implementation issues

In this section we determine the computational cost of our heuristics and we discuss some implementation issues.

7.1 Algorithm CLR1.

In the evaluation of the computational cost of CLR1, it is crucial to consider the application of the Lin-Kernighan procedure after each cluster insertion (performed in Step 2.d), for which no polynomial time bound can be stated. In fact, even if the
search for a Lin-Kernighan improvement takes $O(n^4)$ time, no polynomial time bound is known on the number of improvements needed to achieve the local optimum ([22]).

In [16] it is shown that on the average Lin-Kernighan method runs in time proportional to $n^{2.2}$, and the number of improvements is usually between $\frac{2}{3}$ and $\frac{3}{4}$. In practice, inside our algorithm, Lin-Kernighan stops after a few iterations (from 1 to 4 iterations for instances of 300 cities) for almost 60% of the cases (after each cluster insertion), and, however, the total number of improvements computed in Step 2.d is, on the average, less than the number of improvement required for a single run of Lin-Kernighan method on the whole problem.

Sometimes, Lin-Kernighan algorithm has been implemented disregarding some of the details presented in the original paper [16]. This fact leads to experimental results which are much worse (1 - 3%) than those which can be obtained by a full implementation of the algorithm. Here, we have implemented the full version of Lin-Kernighan heuristic (see [22]). For a complete analysis of the complexity of the LK heuristic see [21].

We now analyze the computational cost of the rest of CLR1.

Let $k$ be the number of maxima of the density function computed in Step 1. Since $k = O(n)$, Step 1 costs $O(n \log n)$.

Step 2.a finds all the maxima $M_i$ of the density function restricted over a set of $h$ points. Our implementation of this procedure finds an approximation of all the $M_i$'s and takes $O(h)$ operations. The technique used is the following. We evaluate the density function over a set of $k^2$ points distributed on a grid, and we compare the value of the density function at those points, with that at the neighbor points. A maxima is any point on the grid for which the density function attains a bigger value than those attained in the neighbor points.

Let $l$ be the number of time we need to perform Step 2.a. One can easily verify that $l \leq n$. As a consequence, the cost of Step 2.a is $O(n^2)$. In Step 2.b, we choose the maximum $M$ closest to the subsolution built so far. This step can be performed at most $m$ times, $m \leq n$, where $m$ is the total number of maxima computed during the entire execution of CLR1. The worst case is obtained when the first set of maxima computed in Step 2.a has cardinality $n$. In this case Step 2.b globally takes $O(n^3)$ time. Step 2.c takes globally $O(n^2 \log n)$ operations (see [23]).

Summarizing, the computational cost of CLR1 is $O(n^3)$ plus the cost due to Lin-Kernighan.

### 7.2 Algorithm CLR2.

The computational cost of CLR2 can be split in two parts: the cost due to the bottom-up stage and the cost due to the top-down stage.

The cost of the bottom-up stage can be determined as follows. Let $h_n$ be the number of times Step 2 through Step 4 are performed. Step 2 takes $O(nh_n)$ time. Each time Step 3 is performed, a new set of points are determined. We can think of
these sets as different levels of a tree, where the leaves are given by the cities, and the internal nodes are given by the maxima of the density function. Each internal node represents a cluster. A node $a$ is a son of another node $b$ if $a$ belongs to the cluster represented by $b$. Building the tree costs $O(n)$.

Globally, bottom-up stage takes $O(nh_n)$ time.

In the top-down stage, we take advantage of the tree built in the bottom-up stage and we construct a sequence $s_1, s_2, \ldots, s_{h_n}$ of solutions over the sets of points $S_1, S_2, \ldots, S_{h_n}$ contained at each level of the tree, respectively. Since the number of nodes contained in the whole tree is $O(n)$, the cost of Step 7 is $O(n^2 \log n)$.

Step 8 is performed exactly $h_n$ times. Unfortunately, no polynomial time bound can be stated on the single application of local search in Step 8.

Preliminary experimental results show that $h_n$ is very small. As an example, for instances of 300 cities $h_n$ is always less than 4. In general $h_n$ is at most $O(\log n)$ ($h_n$ being the height of the tree built in the bottom-up stage). Assuming $h_n = O(\log n)$, the computational cost of CLR2 is $O(n^2 \log n)$ plus the cost due to local search.

### 7.3 Algorithm CLR3.

Let $n_i, 1 \leq i \leq k_n$ be the cardinality of the $i^{th}$ set of maxima found during the execution of CLR3. Step 2 computes the maxima of the density function for the first time and Step 5 repeats this task for $k_n - 1$ times. Then, the cost due to Step 2 and 5 is $O(nk_n)$.

Step 3 applies the convex hull with cheapest insertion method to $n_1$ points, then it takes $O(n_1^2 \log n_1)$ operations. Step 6 extends a solution over $n_i$ points to a solution over $n_{i+1}$ points, $1 \leq i < k_n - 1$, by using the algorithm described at the end of section 5. Steps (i) and (ii) of the above mentioned algorithm cost $\sum_{i=1}^{k_n - 1} n_i n_{i+1}$ and $n^2 \log n$, respectively. Since $\sum_{i=1}^{k_n} n_i n_{i+1} \leq \sum_{i=1}^{k_n} n_i^2$, and $k_n \leq n$ then $\sum_{i=1}^{k_n} n_i n_{i+1} \leq n^3$.

Summarizing, CLR3 has computational cost $O(n^3)$.

In practice, experimental results show that $k_n$ is very small (for example, $5 \leq k_n \leq 8$, $n = 3000$) so that the actual (observed) running time of CLR3 is proportional to $n^2 \log n$.

### 8 Experimental results

We have tested heuristics CLR1 and CLR3, and compared them with Convex Hull with Cheapest insertion method, Christofides heuristic, Nearest Neighbor method, 2-Change and 3-Change local search, and Lin-Kernighan heuristic.

We have performed several experiments (on a SPARC workstation) for instances with 100, 200, and 300 cities, for iterative methods, and up to 3,000 cities for direct methods. For each value of the problem size, we tested all the algorithms over about 20 different instances. The starting tours used for the iterative methods are found by using the convex hull plus CI method [23].
<table>
<thead>
<tr>
<th>Cities</th>
<th>Direct methods</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>5.1</td>
<td>8.8</td>
<td>13.2</td>
</tr>
<tr>
<td>200</td>
<td>6.1</td>
<td>8.8</td>
<td>14</td>
</tr>
<tr>
<td>300</td>
<td>7.1</td>
<td>8.9</td>
<td>14.2</td>
</tr>
<tr>
<td>500</td>
<td>7.5</td>
<td>9.2</td>
<td>15.2</td>
</tr>
<tr>
<td>1000</td>
<td>9</td>
<td>9.9</td>
<td>15.6</td>
</tr>
<tr>
<td>2000</td>
<td>9.2</td>
<td>9.9</td>
<td>16.7</td>
</tr>
<tr>
<td>3000</td>
<td>9.2</td>
<td>10.2</td>
<td>17.8</td>
</tr>
</tbody>
</table>

Table 1: Average percentage excess over the Held-Karp lower bound on optimal tour length for direct methods.

<table>
<thead>
<tr>
<th>Cities</th>
<th>Iterative methods</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CLR1</td>
<td>Lin-Kernighan</td>
<td>3-change</td>
</tr>
<tr>
<td>100</td>
<td>0.7</td>
<td>1.9</td>
<td>3</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>2</td>
<td>3.2</td>
</tr>
<tr>
<td>300</td>
<td>1.1</td>
<td>2.2</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Table 2: Average percentage excess over the Held-Karp lower bound on optimal length tour for iterative methods.

All the experimental results are expressed in term of the cost of the optimal solution (estimated by using the Held-Karp lower bound [9]). We performed our experiments with a fixed parameter setting. Working on the parameters we are able to improve the cost of the solutions in almost all instances. In all our experiments, the cities are uniformly distributed in the unit square. Some of the benchmark data have been provided by David S. Johnson (as an example see the instance with 100 cities in Figure 10).

The main evidence is that our iterative algorithms find better solutions than 2-change local search, 3-change local search, and Lin-Kernighan algorithm. For what concerns algorithm CLR3, it improves over Christofides heuristic which is considered the best constructive method for the TSP.

Table 1 summarizes how CLR3 behaves with respect to Christofides heuristic, convex hull + cheapest insertion method, and nearest neighbor method.

The cost of the solutions provided by CLR3 is lower than the cost of the solutions provided by all the other direct methods we tested. We want to underline that, up
<table>
<thead>
<tr>
<th>parameter $c$</th>
<th>cost of the solution</th>
<th>excess over the HK lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>11355</td>
<td>1.14</td>
</tr>
<tr>
<td>50</td>
<td>11395</td>
<td>1.50</td>
</tr>
<tr>
<td>55</td>
<td>11323</td>
<td>0.86</td>
</tr>
<tr>
<td>60</td>
<td>11373</td>
<td>1.30</td>
</tr>
<tr>
<td>65</td>
<td>11336</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Table 3: Cost and excess over the Held-Karp lower bound for solutions found by CLR1 for different values of $c$.

to 300 cities, CLR3 performs as well as 2-change local search.

Table 2 shows how CLR1 behaves with respect to Lin-Kernighan heuristic, 2-change and 3-change local search, respectively.

Experiments reported in Table 2 indicate that CLR1 provides solutions having a cost very close to the optimum. Lin-Kernighan heuristic, which is considered the best known algorithm for finding approximate solutions to the TSP, provides solutions which have cost 1.1% bigger than the cost of the solutions obtained by using CLR1.

Table 3 summarizes the behaviour of CLR1 for a graph with 200 cities, when $c$ assumes values between 45 and 65.

As we mentioned in Section 6, when $c$ assumes suitable values, the costs of the solutions provided by CLR1 are very close to the optimum. The average percentage excess over the Held-Karp lower bound for the solutions shown in Table 3 is 1.154, while the best solution, found for $c = 55$, is only 0.86% off the lower bound.

The running time of our methods, depends on two main factors:

- the applications of Lin-Kernighan heuristic,
- and the computation of the sets of maxima.

The Lin-Kernighan heuristic can be implemented so that there is a good trade-off between running time and accuracy (see [11]), while we have not been able to design a fast procedure for the computation of the maxima of the density function. This fact is the main limitation of our methods.

9 Conclusions and further work

In this paper we have presented a new approach, based on a clustering technique, for finding nearly-optimal solutions for the geometric TSP on the plane. We have developed direct and iterative methods which are more accurate than Christofides
heuristic and Lin-Kernighan algorithm, respectively, at least for the problem sizes we were able to test.

Future work includes the implementation of CLR2, and the optimization of the code. The goal is to run experiments on much larger instances.

Acknowledgement. We would like to thank David S. Johnson for several helpful comments and for providing some benchmark data.

References


