Finding Longest Cycles in Inner-Triangulated Graphs

Robert J. Cimikowski

Computer Science Department
Montana State University
Bozeman, Montana 59717-0388 U.S.A.

Abstract. We examine the problem of finding longest cycles in *inner triangulations*, that is, 2-connected planar graphs in which all interior faces are triangles. These include the important family of geometric graphs called *Delaunay triangulations*. In particular we present two efficient heuristics for finding a longest cycle in an inner triangulation. The heuristics operate by considering at each step a local set of faces adjacent to the current cycle as candidates for inclusion in the cycle.

1. Introduction.

An *inner triangulation* is a 2-connected planar graph in which every interior face is a triangle. This class of graphs includes Euclidean *Delaunay triangulations* [3], an important family of *proximity graphs*, that is, graphs in which points are *fixed* in n-dimensional space and adjacency is determined by the geometry of the point set. Delaunay graphs are of interest in computational geometry with regard to nearest neighbor problems [10] and in pattern recognition as a model of preceptual relevance amongst dot patterns [11].

We investigate the optimization problem of finding a longest cycle in an inner triangulation. For Delaunay graphs, this problem has applications in computer vision in determining the shape of an object represented by a cluster of points. Finding a hamiltonian or longest cycle through the points helps in identifying the object based on the resulting polygon [8]. For general planar graphs, the problem is *NP-complete* [4]; however, for the 4-connected case, a hamiltonian cycle can be found in linear time [1]. For inner triangulations the problem is also NP-complete [2]. We present two heuristics, with time complexities $O(n)$ and $O(n^2)$, for finding longest cycles in inner triangulations with n vertices. Both methods incorporate a *facial* representation of a plane graph to minimize search time during their operation. The heuristics operate by considering, at each step, the set of faces adjacent to the current cycle as candidates for merging into the cycle. The first heuristic, *triangular expansion*, lengthens a cycle by adding a new vertex adjacent to both endpoints of some edge of the cycle. The operation is repeated until either all vertices of the graph have been added to the cycle or no new vertices can be added. The second heuristic, *triangular reduction*, starts with a cycle consisting of all vertices on the exterior face of the graph. At each step, the cycle is augmented by deleting one of its edges, thereby adding a new vertex previously interior to the cycle. The method may succeed in adding all vertices to the cycle or may reach a "dead end" state in which no new vertices can be added.

JCMCC 9 (1991), pp. 65-77

For inner triangulations, no other methods for finding longest cycles are known which exploit the inner triangular structure. For general graphs, the method of Pósa [9] is best known and has a running time of $O(n^2)$; however, its success depends on the density of the input graph [6]. We sought a method for which success is independent of density and which exploits the inner triangularity property.

Experimentally, we show that the heuristics achieve remarkably good performance. Our results indicate that both methods find cycles at least 95% of optimum length when averaged over all test graphs. Furthermore, although we do not prove a worst-case performance bound for the heuristics, testing revealed that in all instances a cycle of at least 79% of the optimum length was found.

2. Preliminaries.

We follow the usual graph-theoretic notation such as that presented in [5]. Let $G = (V, E)$ be an undirected, unweighted graph without loops or multiple edges, and let $n = |V|$ and $e = |E|$.

A *hamiltonian cycle* is a simple cycle spanning all vertices of a graph. Similarly, a longest cycle is a cycle which has maximum length over all cycles in a graph.

An *inner-triangulated graph* or *inner triangulation* is a 2-connected planar graph in which every interior face is a triangle. Hence, maximal planar graphs are inner triangulations, as are maximal outerplanar graphs.

3. Heuristics for finding a longest cycle.

In this section, we present two heuristics for finding a longest cycle in an inner triangulation. Both heuristics start with an initial cycle which can be easily found and then expand it by merging in vertices which are adjacent to edges along the cycle. The heuristics differ both in their choice of an initial cycle and in the manner in which the cycle is lengthened.

3.1 Triangular expansion

In this method, we select any interior (triangular) face of an inner triangulation G as the initial cycle. We then expand the cycle by adding a vertex adjacent to the endpoints of an edge of the cycle. We continue the process until either all vertices of the graph have been added to the cycle or no other vertices can be added. Since some initial triangles may lead to longer cycles than others, we try all interior triangles as "seeds" and save the longest cycle generated.

The heuristic TEXPAND is outlined as follows:

(1) Select any interior face of G as the initial 3-cycle LC.

(2) Expand cycle LC by adding some new vertex u adjacent to the endpoints of an edge $\{v, w\}$ of LC, add the edges $\{u, v\}$ and $\{u, w\}$, and delete edge $\{v, w\}$ from LC.

(3) Repeat step (2) until either all vertices of the graph are included in LC (hamiltonian cycle), in which case halt, or a "dead end" is reached, that

is, no new vertices can be added to LC. In the latter case, compare the length of LC with the previous longest cycle obtained and save if longer.

(4) If all interior triangles have been tried as seeds, then halt and output the cycle saved as the longest; else select another seed triangle and go to step (2).

Since this method directly exploits the inner-triangular facial structure of the graphs, it has a distinct advantage over "path-extension" heuristics such as Pósa's [9] which proceed by constructing an initial path and attempt to extend the path by an additional edge at each step. The success of these methods is heavily dependent on the edge density of the graph. Our method has no dependencies other than the inner facial structure of the graph. The process of triangular expansion is illustrated in Figure 1. We show the stepwise construction of a longest (hamiltonian) cycle in an inner triangulation, starting with an initial triangle. For each snapshot in the figure, a dashed line indicates the edge deleted from the previous cycle in order to expand LC.

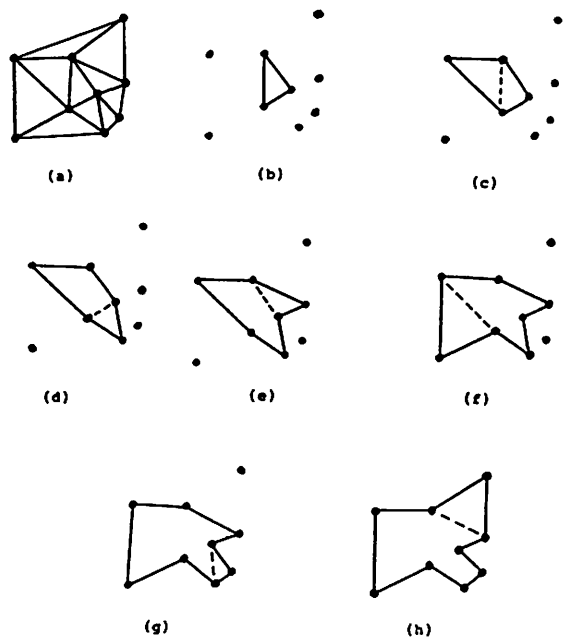

Figure 1. Finding a longest cycle by triangular expansion.

To implement the algorithm, we employ a queue Q of edges of G and a circular doubly-linked list LC, which is the cycle constructed. We use the standard queue operations *enqueue* and *dequeue* to insert and delete edges from Q.

A more detailed description of algorithm TEXPAND is as follows. Let LC_i, $0 \leq i < n-2$, denote the cycle after each iteration of the algorithm. We select an arbitrary interior triangle $\{u, v, w\}$ as a "seed" and form an initial 3-cycle $LC_0 = (u, v, w, u)$, enqueueing the interior edges, that is, edges not bounding the exterior face of G. Let $l = \{u, v\}$ be an edge of LC_0. We *expand* l by deleting it from LC_0 and adding new edges $\{u, x\}$ and $\{x, v\}$ to LC_0 for some vertex x not already in LC_0 and adjacent to u and v. Since only interior edges of G are expandable, we discard all exterior edges after they have been merged into LC_0. When a new interior edge is added to LC_0, we also insert the edge in Q for possible later expansion. For each cycle LC_i, we find a vertex x adjacent to both vertices of an edge, say $\{w, v\}$ of LC_i, such that $\{x, w, v\}$ forms an *elementary triangle*, that is, a triangle with no interior vertices. We call x an *elementary* vertex. By adding x to LC_0, we obtain the cycle $LC_1 = (u, v, x, w, u)$. Finding an elementary vertex is simple if we have a pair of arrays $f1[l]$ and $f2[l]$, called *edge faces*, which point to the faces on either side of edge l. Note that in a planar graph, no edge may bound more than two faces. Let $\{x, w, v\}$ and $\{y, w, v\}$ represent two faces containing a common edge $\{w, v\}$ on their boundaries. Then clearly x and y are both elementary.

During the early stages of execution, it is fairly easy to find elementary vertices to add to the cycle, since there is an abundance of *free* vertices, that is, vertices not yet in the cycle. However, at some later time it may not be possible to find a free vertex. We call this situation a *dead end*. More precisely, it is a state in which no new vertex u may be added to the cycle because the edge necessary to "find" u during the search for an elementary vertex is not present in Q. The situation is shown by the dead end configuration in Figure 2. The current cycle LC_i, which passes through vertices v, w and x, is shown by a solid line. Other adjacencies are indicated with dashed edges. Vertex v was last added to LC_i, and the current queue contents are as shown. In this case, the edge r_3, necessary for "finding" vertex u, is not present in the queue, and hence u cannot be added to the cycle. The success of the heuristic therefore depends partly upon the order in which edges are enqueued during the expansion process.

3.2 Implementation of TEXPAND

To implement the algorithm, we assume that a plane embedding of the graph is provided with adjacency lists $A[1..n]$, where $A[i]$ is a list of the neighbors of vertex i in counterclockwise order about i in the embedding. Also, F denotes the set of faces $f_1, \ldots, f_{|F|}$, where f_i is an ordered list of vertices on the boundary of face i and the unique exterior face f_{ext} is identified. As mentioned previously, edge faces $f1[l]$ and $f2[l]$ point to the faces on either side of edge l in the graph.

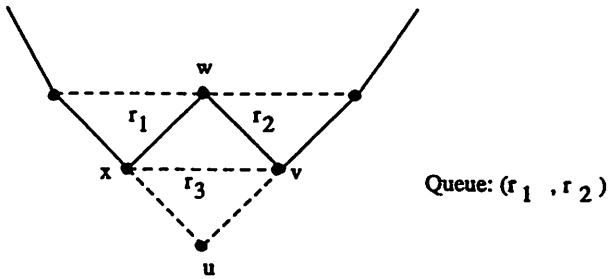

Figure 2. A "dead end" configuration for heuristic TEXPAND.

To make the updating of cycle LC efficient, an array of pointers $LP[1..n]$ is kept, where $LP[u]$ points to the location of vertex u in LC. Also, the variable *free* $[u]$ = *true* iff u is not in LC.

A listing of algorithm TEXPAND follows:

Algorithm TEXPAND.

input: inner triangulation $G = (V, E)$, $n = |V|$, $e = |E|$.
output: longest cycle of G in $MAXCY$.

```
begin

  [initialization]
  obtain set of faces F = {f_1, ..., f_|F|} and edge faces f1[1..e], f2[1..e];
  MAXCY := [];

  for each interior face f in F do
    begin
      Q := [];
      enqueue the interior edges of f;
      LC := f;
      S := V - f;
      free [1..n] := true;

      [triangular expansion phase]
      while S ≠ ∅ and Q ≠ [] do
          begin
            dequeue (r); (* let r = {u, v} *)
```

```
        (* find an elementary vertex and expand an edge of LC *)
        let x and y be the two elementary vertices w.r.t. edge r;
        if free [x] = true then (* x is a free elementary vertex *)
          begin
            insert x between u and v in LC using LP array;
            free [x]:= false;
            S:= S − {x};
            enqueue edges {u, x} and {v, x} if interior;
          end
        else if free [y] = true then (* y is a free elementary vertex *)
            begin
              insert y between u and v in LC using LP array;
              free [y]:= false;
              S:= S − {y};
              enqueue edges {u, y} and {v, y} if interior;
            end;
      end while;

    (* test if all vertices are in LC *)
    if S = ∅ then print("Hamiltonian cycle is", LC) and HALT;
    else if |LC| > |MAXCY| then MAXCY:= LC;
  end for;

print("Longest cycle is", MAXCY);

end {TEXPAND}.
```

3.3 Time complexity of TEXPAND

We show that the time complexity of TEXPAND is $O(n^2)$. All initialization steps can be performed in $O(n)$ time. Each iteration of the *while*-loop removes an edge from Q. Inserting or deleting an edge from the queue takes constant time. Since each edge of G enters and leaves Q at most once, the loop executes at most e times, which is $O(n)$. Finding an elementary vertex and checking if an edge is interior both take constant time, using edge faces $f1$ and $f2$. Thus, the time for all iterations of the *while*-loop is $O(n)$. There are $2n - 2 - k$ interior faces in any inner triangulation with k exterior vertices. Hence, the *for*-loop is performed at most $O(n)$ times. The conditional test at the end of the *for*-loop takes constant time. Therefore, the total running time of TEXPAND is $O(n^2)$.

3.4 Triangular Reduction

Heuristic TEXPAND starts with an *interior* face of the graph and expands it out-

wardly, adding a vertex exterior to the cycle at each step. In *triangular reduction*, we start with the vertices on the *exterior* face of the graph as the initial cycle LC, and lengthen the cycle inwardly by adding a vertex interior to the cycle at each step. We call a triangular face $f = \{u, v, w\}$ *reducible* if f has exactly two exterior vertices u and v, one exterior edge $l = \{u, v\}$, $degree[u] > 2$, and $degree[v] > 2$. By deleting edge l (and hence face f) from G, we obtain a cycle LC' of length one greater than LC. We proceed in this fashion, at each step finding a reducible triangle f with exterior edge l to reduce, and lengthening the previous cycle by one in the process. We only delete edges of *reducible* triangles in order to preserve 2-connectedness and hence cyclicity. As with TEXPAND, we may eventually add all vertices of G to the cycle (hamiltonian cycle) or reach a dead end if no further reducible triangles can be found.

The heuristic TREDUCE is outlined as follows:

(1) Let LC initially be the cycle of vertices on the exterior face of G.
(2) Expand cycle LC by finding a reducible triangle $f = \{u, v, w\}$ with exterior edge $l = \{u, v\}$ and deleting l.
(3) Insert the vertex w between u and v in LC.
(4) Repeat step (2) until either all vertices of the graph are included in LC (hamiltonian cycle) or a "dead end" is reached, that is, no more reducible triangles exist. Output LC as the longest cycle obtained.

As with TEXPAND, this method directly exploits the inner-triangularity of the graphs. The process of triangular reduction is illustrated in Figure 3. G is the same graph as in Figure 1, and we show the stepwise construction of a longest (hamiltonian) cycle. As before, a dashed line indicates the edge deleted from the previous cycle in order to expand LC.

3.5 Implementation of TREDUCE

The data structures used are similar to those for TEXPAND. We use a queue Q of edges of G and a circular doubly-linked list LC, which contains the longest cycle constructed. As before, $LP[u]$ points to the location of vertex u in LC, and *free* $[u]$ = *true* iff vertex u is not in LC. Array element *ext* $[u]$ is *true* iff u is an exterior vertex of the reduced graph. The degree of each vertex is stored in array *degree*.

The critical step in the operation of TEXPAND is in finding a reducible triangle. Initially, we place all exterior edges of reducible triangles in Q. We also store with an edge l a pointer $fp[l]$ to the reducible exterior face containing l. After each reduction step, an exterior edge is deleted and two previously interior edges become exterior edges in the reduced graph. We check each new exterior edge to see if it is part of a reducible triangle, and if so, enqueue the edge for later reduction. Hence, as long as there remain edges of reducible triangles in Q, the algorithm may proceed. A dead end state is reached if not all vertices of G are in LC and Q is empty. In this case, LC is either of optimum length or else cannot

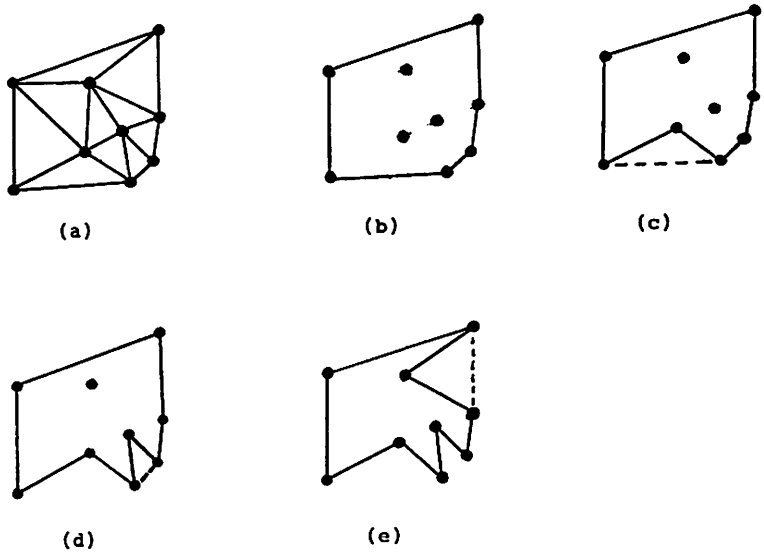

Figure 3. Finding a longest cycle by triangular reduction.

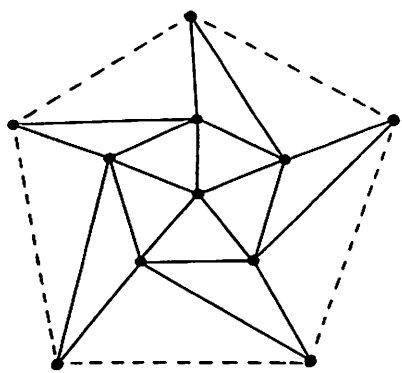

Figure 4. A "dead end" configuration for heuristic TREDUCE.

be further expanded. The situation is depicted in Figure 4. The exterior cycle LC cannot be expanded since none of the remaining exterior triangles are reducible. Dashed lines indicate edges already deleted.

A listing of algorithm TREDUCE follows:

Algorithm TREDUCE.

input: inner triangulation $G = (V, E), n = |V|, e = |E|$.
output: longest cycle of G in LC.

```
begin

  [initialization]
  obtain faces f_1, ..., f_|F|, f_ext;
  initialize arrays ext and degree;
  Q:= [];

  (* enqueue the exterior "reducible" edges of f_ext *)
  for each edge l = {u,v} ∈ f_ext do
    begin
      let {u,v,w} be the unique interior face containing l;
      if not ext [w] and degree [u] > 2 and degree [v] > 2 then enqueue (l);
    end;

  (* initialize longest cycle *)
  LC:= f_ext;

  [triangular reduction phase]
  while Q ≠ [] and |LC| ≠ n do
    begin
      dequeue (r); (* let r = {u,v} *)
      let fp[r] = {u,v,w};
      insert w between u and v in LC using LP array;
      decrement degree [u] and degree [v];
      let {u,x,w} be the unique interior face containing edge l = {u,w};
      if not ext [x] and degree [u] > 2 and degree [w] > 2 then
          begin
            enqueue (l);
            ext [x]:= true;
          end;
```

```
        let {v, y, w} be the unique interior face containing edge m = {v, w};
        if not ext [y] and degree [w] > 2 and degree [v] > 2 then
          begin
            enqueue (m);
            ext [y]: = true;
          end;
      end while;

    print("Longest cycle is ", LC);

  end {TREDUCE}.
```

3.6 Time complexity of TREDUCE

We show that the time complexity of TREDUCE is linear. All initialization steps can be performed in $O(n)$ time.

Each iteration of the *while*-loop removes an edge from Q. Since each edge of G enters and leaves Q at most once, the loop executes at most e times, which is $O(n)$. All operations within the loop take constant time, using arrays fp, *ext*, *degree*, and LP. Thus, the time for all iterations of the *while*-loop is $O(n)$. Therefore, the total running time of TREDUCE is linear.

4. Performance of the heuristics.

To measure the performance of the heuristics, we tested them on a set of 500 randomly-generated inner triangulations. So that we could easily verify the length of a longest cycle in each test graph, we generated only hamiltonian inner triangulations. Finding the length of a longest cycle in a nonhamiltonian graph by exhaustive search would have been prohibitive for graphs of reasonable size, that is, more than 30 vertices. However, the operation of either heuristic is governed by *local* information, that is, the set of faces bordering the current cycle. Hence, neither method considers hamiltonicity, a global property, in making decisions about which triangles to reduce or expand. Therefore, hamiltonicity is not a factor in performance. We elaborate on the test graph generation method in the next section.

4.1 Generating the test graphs

We refer to our method for generating random hamiltonian inner triangulations as *cycle triangulation*. Initially, we generate a cycle of n vertices. The interior of the cycle is then triangulated. Finally, an arbitrary number of chords are added to the exterior face, while preserving both the inner triangularity and planarity of the entire graph. It is straightforward to see that any hamiltonian inner triangulation can

be generated in this fashion. Consider any inner triangulation G with hamiltonian cycle C. Then C partitions $E(G)$ into subsets E_X, E_C, and E_I, denoting edges exterior to, part of, and interior to cycle C, respectively. Then, starting with E_C, we can form G by next adding E_I and then E_X. This corresponds exactly to our generation technique. To ensure randomness, pairs of vertices joined by chords were arbitrarily chosen from the cycle, and the number of exterior chords as well as the endpoints of each chord were also arbitrarily chosen.

4.2 Test results

Table 1 and Table 2 summarize the results of testing TEXPAND and TREDUCE on approximately 500 random hamiltonian inner triangulations with sizes ranging from 50 to 500 vertices. To study the behavior of the heuristics on various types of inner triangulations, we generated four types. Type I were random hamiltonian inner triangulations; Type II had relatively few exterior edges and thus were very edge dense; Type III were maximal planar; Type IV contained no K_4 subgraphs. In Table 1, $WC\%$ indicates the worst-case percentage of the optimum cycle length found by the heuristic over all test graphs, $LC\%$ is the average percentage of the optimum length, and $HC\%$ is the percentage of times a hamiltonian cycle was found. The bottom row of the table shows the averages of the heuristics for all test graphs.

Table 1 indicates that TREDUCE held a slight advantage in the overall testing, as it found cycles of length 2.1% longer than TEXPAND. However, TEXPAND performed better on the denser Type II and Type III graphs. For Type II graphs, TEXPAND found cycles which were always at least 98% of the optimum length and found hamiltonian cycles in over 90% of the graphs. For Type III graphs, TEXPAND had about the same degree of success. On the other hand, TREDUCE outperformed TEXPAND on approximately 90% of the time on Type I graphs, which were sparse on the average. TREDUCE also outperformed TEXPAND on Type IV graphs, which were also sparse. As a whole, the results suggest that the triangular reduction technique is more appropriate for inner triangulations containing a high number of exterior vertices, while triangular expansion is better suited for those with few exterior vertices. We suspect that because sparse inner triangulations have many exterior edges, they also tend to have many reducible triangles, at least initially. Dense inner triangulations, however, have fewer exterior edges initially and therefore fewer candidates for reduction. Factors in the performance of the expansion heuristic are less understandable. The choice of a starting interior face had no observable effect on performance; that is, selecting a face near the center of the graph rather than one on the exterior boundary did not always lead to a longer cycle. This, in fact, is why all faces of the graph must be tried as seeds in order to attain a high success rate for the heuristic.

The effect of problem size on the performance of the heuristics is shown in Table 2. The value of $LC\%$ is indicated for all graphs within the specified size

interval. The table shows that the performance of both methods degrades slightly for graphs with $n > 100$ vertices, but that the degree of degradation is not an increasing function of input size.

Clearly, any refinements to the heuristics should focus on methods of anticipating and avoiding dead end configurations. It is known that certain graphs create problems for the heuristics. For example, the graph shown in Figure 4 causes TREDUCE to "dead end" immediately, and additional instances can be constructed following the same scheme suggested in the figure. For TEXPAND, problem candidates are those containing dense clusters of K_4's and those with "separating triangles", that is, triangles not bounding a face. Methods of handling these configurations are left for future investigation.

Table 1. Performance of *TEXPAND* and *TREDUCE* on random hamiltonian inner triangulations.

	Heuristic					
	TEXPAND			*TREDUCE*		
Type	*WC%*	*LC%*	*HC%*	*WC%*	*LC%*	*HC%*
I	79.0	91.7	45.6	98.2	99.2	50.3
II	98.2	99.0	90.8	87.9	96.4	60.7
III	97.7	99.8	86.0	87.6	95.2	48.2
IV	86.9	96.8	70.8	97.6	99.7	68.7
Ave.	87.8	95.2	72.8	92.3	97.3	56.5

Note: *WC%* = worst-case % of optimum-length cycle found,
LC% = average % of optimum-length cycle found,
HC% = % of times a hamiltonian cycle was found. .

Table 2. Performance of *TEXPAND* and *TREDUCE* on graphs of specified size. *LC%* is indicated for each case.

	Number of vertices in graph				
	10-100	101-200	201-300	301-400	401-500
Heuristic					
TEXPAND	98.4	96.0	93.6	94.6	95.6
TREDUCE	99.4	98.4	94.4	97.4	95.8

5. Remarks.

One major open question concerns the worst-case performances of the heuristics. Although empirical testing revealed that TEXPAND and TREDUCE did no worse

than 79% and 87%, respectively, over all test graphs, can we obtain absolute or asymptotic worst-case approximation ratios for the heuristics? We suspect this to be quite difficult due to the random manner in which the heuristics operate.

Another interesting problem is to identify certain subclasses of inner triangulations for which the heuristics always generate exact solutions. We leave this for further study.

References

1. N. Chiba, and T. Nishizeki, *The Hamilton cycle problem is linear-time solvable for 4-connected planar graphs*, J. Algorithms **10** (1989), 187–211.
2. V. Chvátal, *Hamiltonian cycles*, in "The Traveling Salesman Problem", E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, eds., J. Wiley & Sons, New York, 1985, pp. 403–429.
3. B. Delaunay, *Sur la sphère vide, Izvestia Akademia Nauk SSSR, VII seria*, Otdelenie Matematicheskii i Estestvennyka Nauk **6** (1934), 793–800.
4. M.R. Garey and D.S. Johnson, "Computers and Intractability", W.H. Freeman & Co., New York, 1979.
5. F. Harary, "Graph Theory", Addison-Wesley, Reading, MA, 1969.
6. R. Karp, *The probabilistic analysis of some combinatorial search problems*, in "Algorithms and Complexity", J. Traub, ed., Academic Press, New York, 1976, pp. 1–19.
8. J. O'Callaghan, *Computing the perceptual boundaries of dot patterns*, Comp. Graphics and Image Proc. **3** (1974), 141–162.
9. L. Pósa, *Hamiltonian circuits in random graphs*, Disc. Math. **14** (1976), 359–364.
10. F. Preparata and M. Shamos, "Computational Geometry: An Introduction", Springer-Verlag, New York, 1985.
11. G. Toussaint, *Pattern recognition and geometrical complexity*, Fifth Int'l Conf. on Pattern Recognition (1980), 1324–1347, Miami Beach.