Computing Optimal Embeddings for Planar Graphs

Petra Mutzel* and René Weiskircher

Technische Universität Wien, Karlsplatz 13, 1040 Wien
{mutzel, weiskircher}@apm.tuwien.ac.at

Abstract. We study the problem of optimizing over the set of all combinatorial embeddings of a given planar graph. At IPCO' 99 we presented a first characterization of the set of all possible embeddings of a given biconnected planar graph G by a system of linear inequalities. This system of linear inequalities can be constructed recursively using SPQR-trees and a new splitting operation. In general, this approach may not be practical in the presence of high degree vertices.
In this paper, we present an improvement of the characterization which allows us to deal efficiently with high degree vertices using a separation procedure. The new characterization exposes the connection with the asymmetric traveling salesman problem thus giving an easy proof that it is NP-hard to optimize arbitrary objective functions over the set of combinatorial embeddings.
Computational experiments on a set of over 11000 benchmark graphs show that we are able to solve the problem for graphs with 100 vertices in less than one second and that the necessary data structures for the optimization can be build in less than 12 seconds.

1 Introduction

A graph is called *planar* if it admits a drawing into the plane without edge-crossings (*planar drawing*). We call two planar drawings of the same graph *equivalent* when the circular sequence of the edges around each vertex is the same in both drawings. The equivalence classes of planar drawings are called *combinatorial embeddings*. A combinatorial embedding also defines the set of cycles in the graph that bound faces in a planar drawing.

The complexity of embedding planar graphs has been studied by various authors in the literature [5, 4, 6]. In this paper we deal with the following optimization problem concerning embeddings: Given a planar biconnected graph and a cost function on the cycles of the graph, find an embedding Π such that the sum of the cost of the cycles that appear as face cycles in Π is minimized. The objective function is chosen only to demonstrate the feasibility of the approach in computational experiments. However, our description of the set of combinatorial

* Partially supported by DFG-Grant Mu 1129/3-1, Forschungsschwerpunkt "Effiziente Algorithmen für diskrete Probleme und ihre Anwendungen"

embeddings of a planar graph as an Integer Linear Program (ILP) makes some important NP-hard problems arising in graph drawing accessible to ILP-based optimization methods.

One example is the minimization of the number of bends in an orthogonal planar drawing of a graph. This number highly depends on the chosen planar embedding. Whereas bend minimization of a planar graph is NP-hard ([9], a branch and bound algorithm for the problem is given in [3]), it can be solved in polynomial time for a fixed embedding ([13]). Figure 1 shows two different orthogonal drawings of the same graph that were generated using the bend minimization algorithm by Tamassia ([13]). The algorithm used different combinatorial embeddings as input. Drawing 1(a) has 13 bends while drawing 1(b) has only 7 bends.

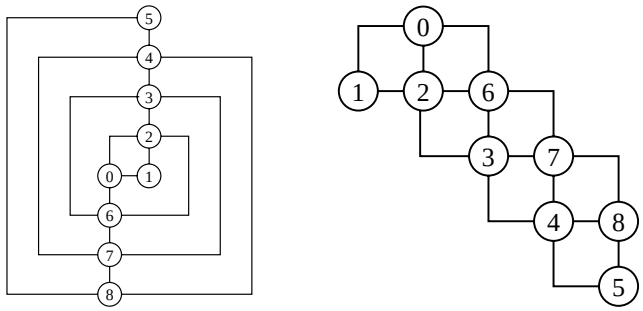


Fig. 1. The impact of the chosen planar embedding on the drawing

For a fixed embedding, the bend minimization problem can be formulated as a flow problem in the geometric dual graph. This is the point where we plan to use our description: Once we have characterized the set of all embeddings via an integer linear formulation on the variables associated with cycles of the graph, we can combine this formulation with the flow problem to solve the bend minimization problem over all embeddings.

In [12] we introduced an integer linear program (ILP) whose set of feasible solutions corresponds to the set of all possible combinatorial embeddings of a given biconnected planar graph. The program is constructed recursively with the advantage that we only introduce variables for those simple cycles in the graph that form the boundary of a face in at least one combinatorial embedding of the graph, thus reducing the number of variables tremendously. The constraints are derived using the structure of the graph. We use a data structure called SPQR-tree suggested by Di Battista and Tamassia ([2]) for the on-line maintenance of triconnected components. SPQR-trees can be used to code and enumerate all possible combinatorial embeddings of a biconnected planar graph.

The problem of our original formulation was that it may be inefficient for graphs with high-degree vertices, because it requires to compute the convex hull of a number of points that may be exponential in the degree of a vertex. When looking for a solution to this problem, we discovered a connection of the embedding problem with the asymmetric traveling salesman problem (ATSP) enabling us to apply the same machinery used for solving the ATSP-problem to the problem of finding an optimal embedding for a planar biconnected graph.

Our computational results on a set of more than 11000 benchmark graphs show that our new approach preserves the positive properties of our first approach while making it possible to deal efficiently with high degree vertices. As in our first version, the size of the integer linear system computed for the benchmark graphs grows only linearly with the size of the problem and the computed systems can be solved fast using a mixed integer program solver. There are only a few graphs with high degree vertices for which we need to apply our separation procedure and we never need more than three separation steps.

Section 2 gives a brief overview of SPQR-trees. In Section 3 we sketch the recursive construction of the linear constraint system using a splitting operation on the SPQR-tree. Section 4 introduces the connection of our problem with the ATSP problem and describes the separation procedure. Our computational results are described in Section 5.

2 SPQR-Trees

In this section, we give a brief overview of the SPQR-tree data structure for biconnected graphs. SPQR-trees have been suggested by Di Battista and Tamassia ([2]). They represent a decomposition of a biconnected graph into triconnected components. A connected graph is triconnected, if there is no pair of vertices in the graph whose removal splits the graph into two or more components.

An SPQR-tree has four types of nodes and with each node is associated a biconnected graph which is called the *skeleton* of that node. This graph can be seen as a simplified version of the original graph and its vertices are also contained in the original graph. The edges in a skeleton represent subgraphs of the original graph. The node types and their skeletons are as follows:

1. Q**-node:** The skeleton consists of two vertices that are connected by two edges. One of the edges represents an edge e of the original graph and the other one the rest of the graph.
2. S**-node:** The skeleton is a simple cycle with at least 3 vertices.
3. P**-node:** The skeleton consists of two vertices connected by at least three edge.
4. R**-node:** The skeleton is a triconnected graph with at least four vertices.

All leaves of the SPQR-tree are Q-nodes and all inner nodes S-,P or R-nodes. When we see the SPQR-tree as an unrooted tree, then it is unique for each biconnected planar graph. Another important property of these trees is

that their size (including the skeletons) is linear in the size of the original graph and that they can be constructed in linear time [2].

As described in [2], SPQR-trees can be used to represent all combinatorial embeddings of a biconnected planar graph. This is done by choosing embeddings for the skeletons of the nodes in the tree. The skeletons of S- and Q-nodes are simple cycles, so they have only one embedding. Therefore, we only have to look at the skeletons of R- and P-nodes. The skeletons of R-nodes are triconnected graphs. Our definition of combinatorial embeddings distinguishes between two combinatorial embeddings of a triconnected graph, which are mirror-images of each other (the circular order of the edges around each vertex in clockwise order is reversed in the second embedding). The number of different embeddings of a P-node skeleton is $(k-1)!$ where k is the number of edges in the skeleton.

Every combinatorial embedding of the original graph defines a unique combinatorial embedding for each skeleton of a node in the SPQR-tree. Conversely, when we define an embedding for each skeleton of a node in the SPQR-tree, we define a unique embedding for the original graph. Thus, if the SPQR-tree of G has r R-nodes and the P-nodes P_1 to P_k where the skeleton of P_i has L_i edges, then the number of combinatorial embeddings of G is exactly

$$2^r \prod_{i=1}^{k} (L_i - 1)! \quad .$$

Because the embeddings of the R- and P-nodes determine the embedding of the graph, we call these nodes the *decision nodes* of the SPQR-tree.

3 Recursive Construction of the Integer Linear Program

3.1 The Variables of the Integer Linear Program

A *face cycle* in a combinatorial embedding of a planar graph is a directed cycle of the graph with the following property: In any planar drawing realizing the embedding, the left side of the cycle is empty. Note that the number of face cycles of a planar biconnected graph with m edges and n vertices is $m - n + 2$.

We construct an integer linear program (ILP) where the feasible solutions correspond to the combinatorial embeddings of a graph. The variables of the program are the same as in [12]. They are binary and represent directed cycles in the graph. In a feasible solution of the ILP, a variable x_c has value 1 if the associated cycle c is a face cycle in the represented embedding and 0 otherwise. To keep the number of variables as small as possible, we only introduce variables for those cycles that are indeed face cycles in a combinatorial embedding of the graph.

3.2 Splitting an SPQR-Tree

We construct the variables and constraints of the ILP recursively. Therefore, we need an operation that constructs a number of smaller problems from our

original problem such that we can use the variables and constraints computed for the smaller problems to compute the ILP for the original problem. This is done by splitting the SPQR-tree at some decision-node v.

The splitting operation deletes all edges in the SPQR-tree incident to v whose other endpoint is not a Q-node, thus producing smaller trees. Then we attach new Q-nodes to all nodes that were incident to deleted edges to make sure that the trees we produce are again SPQR-trees. The new edges we use to attach the Q-nodes are called *split-edges* and the trees we produce *split-trees*. The graphs represented by the split-trees are called *split-graphs*. The new tree containing v is the *center split-tree* and the associated graph the *center split-graph*. The split-trees either have only one decision node (like the center split-tree) or at least one less than the original tree. The splitting process is depicted in Fig. 2.

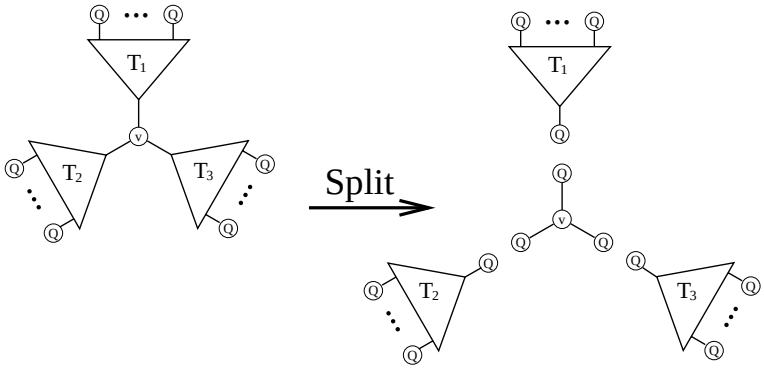


Fig. 2. Splitting an SPQR-tree at an inner node

3.3 Construction of the ILP for SPQR-Trees

Since this topic is treated in detail in [12] and [11], we only give a short sketch of our approach. Let T be the SPQR-tree of a biconnected planar graph G, v the node used for splitting the tree and $T_1, \ldots, T_k$ the split-trees of v. We assume that T_1 is the center split-tree and that the graph G_i is the split-graph belonging to split-tree T_i. We can distinguish two types of directed cycles in G:

1. *Local cycles* are contained in one of the graphs $G_1, \ldots, G_k$.
2. *Global cycles* of are not contained in any of the G_i.

We assume that we have already computed the ILP I_i for each T_i. The variables in I_i that represent local cycles will also be variables in the ILP for T. We compute the global cycles of G for which we need variables by combining cycles in the split-graphs that are represented by variables in the I_i.

The set C of all constraints of the ILP of T is given by $C = C_l \cup C_c \cup C_g$. C_l is the set of *lifted constraints*. For each constraint contained in I_i, we compute

a constraint that is valid for T by replacing each variable x_c by the sum of the variables in $R(x_c)$ (The set of variables for T whose associated cycles have been constructed using c).

The set C_c is the set of *choice constraints*. They state that for each variable x_c computed for T_i, the sum of the variables in the set $R(x_c)$ can be at most one. This is true because all of the cycles in $R(x_c)$ either pass an edge or one of the split-graphs in the same direction and therefore at most one of the cycles can be a face cycle in any embedding. The proof is omitted but can be found in [11].

The only element of C_g is the *center graph constraint*, which states that the number of global face-cycles in any feasible solution plus the number of local face-cycles contained in G_1 is equal to the number of faces of G_1. This is true, because any drawing of G can be generated from a drawing of G_1 by replacing some edges by subgraphs. In this process, the face cycles of G_1 may be replaced by global cycles or are preserved (if they are local cycles of G).

We observe that a graph G whose SPQR-tree has only one inner node is isomorphic to the skeleton of this node. Therefore, the ILP for an SPQR-tree with only one inner node is defined as follows:

- S-node: When the only inner node of the SPQR-tree is an S-node, G is a simple cycle. Thus it has two directed cycles and both are face-cycles in the only combinatorial embedding of G. So the ILP consists of two variables, both of which must be equal to one.
- R-node: G is triconnected and according to our definition of combinatorial embeddings, every triconnected graph has exactly two embeddings, which are mirror-images of each other. When G has m edges and n vertices, we have $k = 2(m - n + 2)$ variables and two feasible solutions. The constraints are given by the convex hull of the two points in k-dimensional space that correspond to the solutions.
- P-node: The ILP for graphs whose only inner node in the SPQR-tree is a P-node is described in detail in Section 4 because this is where the connection to the asymmetric travelings salesman problem (ATSP) is used.

The proof of correctness differs from the proof given in [12] and more detailed in [11] only in the treatment of the skeletons of P-nodes, so we only look at the treatment of P-node skeletons in more detail in the next section.

4 *P*-Node Skeletons and the ATSP

A P-node skeleton P consists of two vertices v_a and v_b connected by $k \geq 3$ edges and has therefore $(k-1)!$ different combinatorial embeddings. Every directed cycle in P is a face cycle in at least one embedding of P, so we need $k^2 - k$ variables in an ILP description of all combinatorial embeddings.

Let C be the set of variables in the ILP and $c : C \to \mathbb{R}$ a weight function on C. We consider the problem of finding the embedding of P that minimizes the sum of the weights of the cycles that are face cycles. We will show that this problem

can be stated as the problem of finding a Hamiltonian cycle with minimum weight in the complete directed graph $B = (V, E)$ with k vertices or simply as the asymmetric traveling salesman problem (ATSP). The transformation we will show here also works in the opposite direction thus showing NP-hardness of optimizing over all embeddings. But we show here the reduction to ATSP because this is what our algorithm does when it computes the ILP.

The complete directed graph B has one vertex for every edge of P. Let e_1 and e_2 be two edges in P and v_1 and v_2 the corresponding vertices in B. Then the edge (v_1, v_2) corresponds to the cycle in P that traverses edge e_1 from v_a to v_b and edge e_2 from v_b to v_a. The edge (v_2, v_1) corresponds to the same cycle in the opposite direction. In this way, we define a bijection $b : C \to E$ from the cycles in P to the edges in B. This bijection defines a linear function $c' : E \to \mathbb{R}$ on the edges of B such that $c'(e) = c(b^{-1}(e))$.

Now it is not hard to see that each embedding of P corresponds to a Hamiltonian cycle in B and vice versa. If the Hamiltonian cycle is given by the sequence $(v_1, v_2, \ldots, v_k)$ of vertices, then the sequence of the edges in P in the corresponding embedding in counter-clockwise order around v_a is $(e_1, e_2, \ldots, e_k)$. Figure 3 shows an example of a P-node skeleton with four edges and the corresponding ATSP-graph. The embedding of the P-node skeleton on the left corresponds to the Hamiltonian cycle marked in the ATSP-graph. The marked edges correspond to the face cycles in the embedding of the P-node skeleton. The sequence of the edges in P in counter-clockwise order around vertex v_a corresponds to the sequence of the vertices in the Hamiltonian cycle of the ATSP-graph.

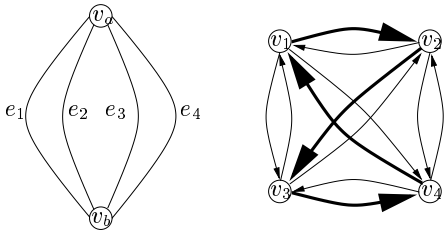


Fig. 3. A P-node skeleton and its corresponding ATSP-graph

The sum of the weights of all edges on the Hamiltonian cycle is equal to the sum of the weights of the cycles of P that are face cycles in this embedding. So finding an embedding of P that minimizes the sum of the weights of the face cycles is equivalent to finding a traveling salesman tour with minimum weight in B. Since we can easily construct a corresponding P-node embedding problem for any ATSP-problem, we have a simple proof that optimizing over all embeddings of a graph is in general NP-hard.

It also enables us to use the same ILP used for ATSP for the ILP that describes all combinatorial embeddings of a P-node skeleton. The formulation for the ATSP ILP found in [7] has two types of constraints.

1. The *degree constraints* state that each vertex must have exactly one incoming edge and one outgoing edge in every solution.
2. The *subtour elimination constraints* state that the number of edges with both endpoints in a nonempty subset S of the set of all vertices can be at most $|S| - 1$.

The number of degree constraints is linear in the number of edges in a P-node skeleton, while the number of subtour elimination constraints is exponential. Therefore, we define the ILP for a graph whose SPQR-tree has a P-node as the only inner node just as the set of degree constraints for the corresponding ATSP-problem.

To cope with the subtour elimination constraints, we store for each P-node skeleton the corresponding ATSP-graph. For each edge in the ATSP-graph we store the corresponding cycle in the P-node skeleton. During the recursive construction, we update the set of corresponding cycles for each edge in the ATSP-graph, so that we always know the list of cycles represented by an edge in the ATSP-graph. This is done in the same way as the construction of the lifted constraints in subsection 3.3.

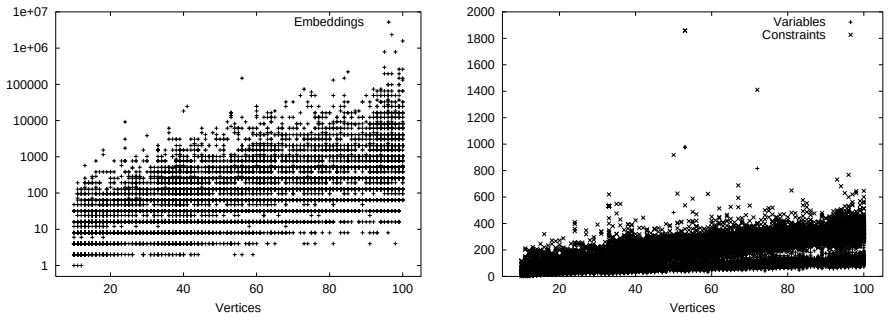
When the construction of the recursive ILP is finished, we use a mixed integer programming solver to find an integer solution. Then we check if any subtour elimination constraint is violated. We do this by finding a minimum cut in each ATSP-graph. The weight of each edge in the ATSP graph is defined as the sum of the values of the variables representing the cycles associated with the edge. If the value of this minimum cut is smaller than one, we have found a violated subtour elimination constraint and add it to the ILP. As we will show in the next section, separation of the subtour elimination constraint is rarely necessary.

5 Computational Results

In our computational experiments, we used a benchmark set of 11491 graphs collected by the group around G. Di Battista in Rome ([1]). Since some of these graphs are not planar, we first computed a planar subgraph using the algorithm from [10]. Then we made the resulting graphs biconnected while preserving planarity using the algorithm from [8]. For the resulting graphs, we first computed the recursive part of our ILP and then used a branch and cut algorithm with CPLEX as integer program solver to optimize randomly chosen linear functions over the set of all combinatorial embeddings of the graph. After each optimization phase, we checked if there was a violated subtour elimination constraint and if this was the case, we added the found constraint and re-optimized the ILP. Our experiments ran on a Sun Enterprise 10000.

Figure 4(a) shows that the number of embeddings can vary greatly for graphs with similar size. One graph with 97 vertices and 135 edges had 2,359,296 combinatorial embeddings while another one with 100 vertices and 131 edges had only 64 embeddings (note that the y-axis in the figure is logarithmic).

Figure 4(b) shows that the number of constraints and variables of our formulation grows roughly linear with the size of the graphs which is surprising when



(a) (b)

Fig. 4. The number of embeddings of the tested graphs and the number of variables and constraints

we consider the growth of the number of embeddings. Our ILP always has more constraints than variables. Figure 5(a) shows the time needed for building the recursive ILP and the time needed for optimization including separation. Optimization is very fast and the longest time we needed was 0.75 seconds. The time needed for building the ILP grows sub-exponential with the number of vertices and never exceeded 11 seconds.

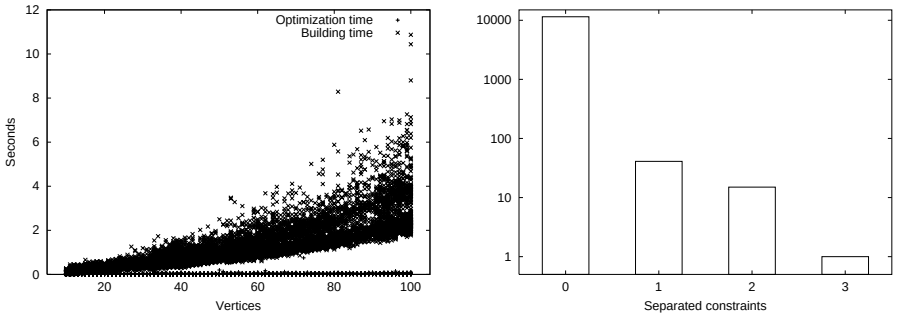
Figure 5(b) shows that separation was rarely necessary and in the cases where we separated constraints, the number of the separation steps was small. The boxes show the number of graphs that needed $0, 1, 2$ or 3 separation steps, e.g. $1, 2, 3$ or 4 optimization rounds (note that the y-axis is logarithmic). We needed at most three separation steps and this was only the case for one of the 11,491 graphs. For 11,472 of the graphs, no re-optimization was necessary.

Our future goal will be to extend our formulation such that each solution will correspond to an orthogonal representation of the graph. This will enable us to find drawings with the minimum number of bends over all embeddings. Of course, this will make the solution of the ILP much more difficult.

Acknowledgments We thank the group of G. Di Battista in Rome for giving us the opportunity to use their implementation of SPQR-trees in *GDToolkit*, a software library that is part of the ESPRIT ALCOM-IT project (work package 1.2), and to use their graph generator.

References

1. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7:303–326, 1997.



(a) (b)

Fig. 5. The time needed for building the ILP and for optimization and the benchmark set divided up by the number of separation steps

2. G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996.
3. P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *LNCS*, 1272:331–344, 1998.
4. D. Bienstock and C. L. Monma. Optimal enclosing regions in planar graphs. *Networks*, 19(1):79–94, 1989.
5. D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990.
6. J. Cai. Counting embeddings of planar graphs using DFS trees. *SIAM Journal on Discrete Mathematics*, 6(3):335–352, 1993.
7. G. Carpaneto, M. Dell'Amico, and P. Toth. Exact solution of large scale asymmetric travelling salesman problems. *ACM Transactions on Mathematical Software*, 21(4):394–409, 1995.
8. S. Fialko and P. Mutzel. A new approximation algorithm for the planar augmentation problem. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 260–269, San Francisco, California, 1998.
9. A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. In R. Tamassia and I. G. Tollis, editors, *Proceedings Graph Drawing '94*, volume 894 of *LNCS*, pages 286–297. Springer-Verlag, 1994.
10. M. Jünger, S. Leipert, and P. Mutzel. A note on computing a maximal planar subgraph using PQ-trees. *IEEE Transactions on Computer-Aided Design*, 17(7):609–612, 1998.
11. P. Mutzel and R. Weiskircher. Optimizing over all combinatorial embeddings of a planar graph. Technical report, Max-Planck-Institut für Informatik, Saarbrücken, 1998.
12. P. Mutzel and R. Weiskircher. Optimizing over all combinatorial embeddings of a planar graph. In G. Cornuéjols, R. Burkard, and G. Wöginger, editors, *Proceedings IPCO '99*, volume 1610 of *LNCS*, pages 361–376. Springer Verlag, 1999.
13. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.