Feature-Sensitive Motion Planning

Terra Horton
Author

Marco Morales
Graduate Student Mentor

Lydia Tapia
Graduate Student Mentor

Roger Pearce
Graduate Student Mentor

Nancy M. Amato
Faculty Mentor

terrahorton@hotmail.com, {marcom,ltapia,rap2317,amato}@cs.tamu.edu

Distributed Mentoring Project
Parasol Lab
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112

August 8, 2005

Abstract

Motion planning consists of finding a sequence of motions for a robot (movable object) to move from a start configuration to a goal configuration without colliding with any obstacle. PRMs create a roadmap that paths can be extracted from. To create the roadmap, nodes are generated in the environment at random. Each node represents a possible position the robot can move to. Only feasible nodes are stored in the roadmap. Feasibility of nodes is usually determined through a collision detection test. The nodes in the roadmap are connected with local planners. One local planner that is currently used is Rotate_At_S. This planner rotates at a given value between configurations before connecting them. Rotating between connections helps to avoid collision. The original implementation of Rotate_At_S only allows one rotation between the pair of nodes. We implemented a method that will allow several rotations between connections. Several rotations between connections may increase the chance of finding connections. Feature-Sensitive Motion Planning is a method of motion planning that divides a given environment into sections and uses a planner that will create the best roadmap for that section. Once a roadmap is generated in each section they are combined to create one map. The methods currently available to connect roadmaps are the most expensive step in Feature-Sensitive Motion Planning. They can be optimized. One of these methods makes a list of nodes from each section. Each node on the list tries to make a connection with every node on the other list. We will create a method that will be faster and more efficient. It will generate a list of nodes for each section and try to connect it to the closest nodes from the other list.

1 Introduction

The *motion planning* (MP) problem is that of finding a sequence of valid states for a movable object, usually called a robot, to get from an initial to a goal state. A robot's state or *configuration* is a set of parameters that describe the position, orientation, and any linkages associated with the robot. A configuration is said to be valid if it satisfies all constraints given in the MP problem (e.g., in many cases the configuration is valid if it is collision free). Many applications address the problem of path planning from robotics and CAD to computational biology.

There is strong evidence that any complete planner will require exponential time in the number of *degrees of freedom* (DOF) of the robot [5, 13, 14]. Thus, the motion planning problem is thought to be intractable except for robots with few DOFs. Initially, heuristic methods based on cell decomposition [4] and potential fields [9] were explored to address this complexity but after the introduction of the Randomized Path Planner (RPP) [2] randomized methods have been the focus of extensive research. The roadmap-based probabilistic roadmap methods (PRMs) [7] and several tree-based methods that explore the planning space starting from one or two points [3, 6, 10] are notable examples. Remarkable results, including solutions for previously unsolved MP problems, have been obtained with randomized methods.

Although there are many randomized planners, their performance varies depending on their individual strengths and weaknesses and on the construction of the problem instance to solve. Some planners are best suited for problems with few obstacles while others show their real strength in cluttered areas. In addition, many environments have vastly different regions, so there may not be any planner that can deal efficiently with all the pieces of the problem by itself.

In our previous work [12], we proposed a meta-planner for motion planning that would oversee the coordinated application of multiple planners. We used a machine learning approach to characterize and partition C-space into regions that were suited to one of the methods in a library of roadmap-based motion planners. After the best-suited method was applied in each region, the resulting regional roadmaps were combined to form a roadmap of the entire planning space.

Our prototype meta-planner for feature-sensitive motion planning demonstrated the promise of this approach by outperforming any of the individual planners on a variety of problem instances [12]. However, our results also illustrated some performance bottlenecks in the prototype system. For example, our rather naive subdivision strategy was not actually feature sensitive and hence did not always result in the most natural subdivisions. Of greatest impact, however, was our straight forward brute force strategy for integrating regional roadmaps – in many cases this step accounted for the majority of the computation costs!

In this paper, we discuss a new method for connecting maps. This connection method will help reduse the computation cost. We also discuss a variation in one of our local planners. The new implementation will help increase node connection.

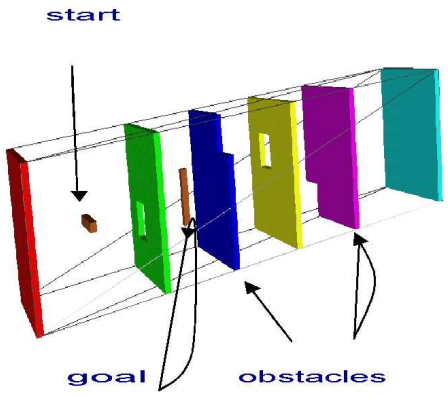


Figure 1: Motion Planning Problem

2 Preliminaries

A robot is a movable object that can be controlled through n parameters or *degrees of freedom*, each corresponding to a unique component (e.g., object positions, object orientations, link angles, or link displacements). A robot's placement, or configuration, can be uniquely described by a point $(x_1, x_2, ..., x_n)$ in an n dimensional space (x_i being the ith DOF). This space, consisting of all robot configurations (feasible or not) is called *configuration space* (C-space) [11]. The subset of all feasible configurations is the *free C-space* (C-free), while the union of the unfeasible configurations is the *blocked C-space* (*C-obstacle*). Thus, the MP problem becomes that of finding a continuous trajectory for a point in C-space connecting the start and the goal configurations that completely lies in C-free. And, although it is intractable to compute C-space, we can often determine whether a configuration is feasible or not quite efficiently, e.g., by performing a collision detection test in the *workspace*, the robot's natural space.

3 Related Work

Probabilistic roadmap planning methods have been shown to perform well in a number of practical situations. The idea behind these methods is to create a graph of randomly generated collision-free configurations with connections between these nodes made by a simple and fast local planning method. These methods run quickly and are easy to implement; unfortunately there are simple situations in which they perform poorly. When required to pass through narrow passages in C-free their performance degrades [8]. The Medial Axis Probablistic Roadmap Method (MAPRM) samples the configuration space in which randomly generated configurations, free or not, are retracted onto the medial axis of the free space. This improves performance on problems requiring traversal of narrow passages. The Obstacle Based Probablistic Roadmap Method (OBPRM) samples points on or near obstacles in C-space (C-obstacle) [1]. This node generation strategy will have more chances of generating some configurations in the narrow passages since those passages are near C-obstacles.

Although there are many variants of randomized planners, no single variant performs well for every problem. The characteristics of the problem often dictate the performance of the method applied. For example, OBPRM [1], a PRM where configurations generated on or near C-obstacle surfaces, performs better in cluttered regions. On the other hand, Basic PRM [7], a PRM where configurations are generated through uniform random sampling, performs best in free regions.

4 Feature-Sensitive Motion Planning Framework

In [12], we proposed a machine learning based characterization and partitioning approach to motion planning problems. The C-space of the instance is recursively partitioned, so that in each level a region may be subdivided into different overlapping subregions. Subdivisions stop when a region is classified as homogeneous (based on a set of features measured for each region). Then, the best suited planner available is applied to find a regional solution. When all the subregions have been mapped, their roadmaps are combined to produce a regional roadmap. At the end of this process we have a global roadmap for the environment.

The classification of a region was performed with the use of a trained decision tree. This method requires the collection of descriptive features of each region. We use 21 different features that are easy to collect during and after the creation of a small PRM roadmap. Based on the values of these features, the trained decision tree classifies a region as homogeneous (free, cluttered, or narrow passageway) or non-homogeneous.

The feature-sensitive framework depends on the effectiveness of the partitioning into homogeneous regions. Previously, we applied a technique that selects, at random, both the positional dimension to subdivide and the point to split the selected dimension. Regions were partitioned so that they overlap in 10% to 15% of their neighboring area. While this method eventually created homogeneous regions, it created numerous regions. Also, while the classification was

performed using descriptive features collected for the space, this information, although available, was not considered when creating and selecting the partition. This framework is skecthed in Fig.[2].

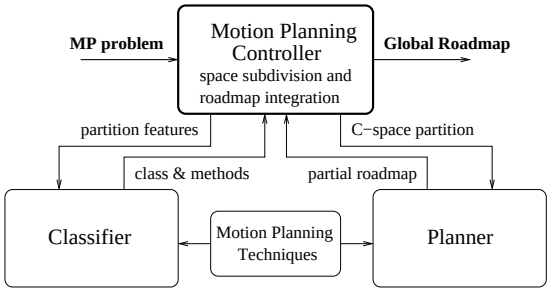


Figure 2: Block diagram of framework for feature-sensitive motion planning

The *feature-sensitive meta-planner* recursively subdivides the environment. In the process, it creates a tree representing the subdivision where each node in the tree represents a C-space region that is the union of the regions of the leaves of the subtree rooted at that node. This process is sketched in the divide-and-conquer Algorithm 1: FeatureSensitiveMap.

Algorithm 1 receives as input a C-space region and outputs a region mapped. Initially, *FeatureSensitiveMap* is called for the C-space of the whole problem. First, the characterizer analyzes region features to try to identify a good planner for the corresponding region (Line 1). Next, the partitioner tests whether it should subdivide the region (Line 2), and if so, it places boundaries based on region features to define subregions (Line 3). Then, each subregion is mapped with a recursive call to *FeatureSensitiveMap* (Lines 4-6). Finally, the planner combines the subregion roadmaps to form the roadmap of the parent (Line 7). At the bottom of the recursion, when the region is not subdivided, it maps the leaf with the planning strategy determined by the characterizer (Lines 8-9). The bottom-up merging process produces a roadmap for the region covering the problem's C-space.

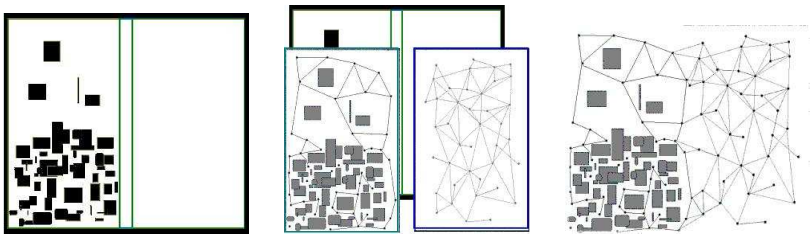
Algorithm 1 FeatureSensitiveMap

Input: C-space *region*[boundaries, features(roadmap, characterization), planner strategy, roadmap],
C-space *parent*, *constraints*[maximum tree height, subdivision limit], node *height*
Output: C-space region R with [roadmap]

1: Characterizer.MatchPlanner(*region*)
2: **if** Partitioner.Subdivide(*region*, *parent*, *constraints*, *height*) **then**
3: *subregions* = Partitioner.PlaceBoundaries(*region*, *constraints*)
4: **for all** $subregion_i$ in *subregions* **do**
5: $subregion_i$ = FeatureSensitiveMap($subregion_i$, *region*, *constraints*, *height* + 1)
6: **end for**
7: *region*.[roadmap] = Planner.CombineRoadmaps(*subregions*)
8: **else**
9: *region*.[roadmap] = Planner.MapRegion(*region*)
10: **end if**
11: **Return** *region*

5 Map Connection for Feature-Sensitive Motion Planning

C-Space roadmap combination consists of constructing one roadmap for the parent region from roadmaps of the child regions (subregions). These subregions are divided by a (possibly overlapping) boundary in n-dimensional C-space. The input of this problem is the two roadmaps and the regions to which they correspond, the output is a roadmap combining the two original roadmaps (See Figure 3).



(a) Split regions (b) Roadmaps to combine (c) Regions combined

Figure 3: Roadmaps of neighboring subregions are combined

There are two main issues in combining roadmaps: selecting source nodes from each source roadmap to connect to the target roadmap, and mechanisms to perform the connection. In our initial prototype we used a brute-force connection strategy where for each node in either of the roadmaps we selected the k-closest nodes and we tried connections with a straight-line local planner [12]. In this system, roadmap combination used from 50% to 90% of the total validity tests. This shows that better strategies for combining roadmaps is a key requirement to make our feature-sensitive framework feasible. Our initial naive strategy was poor because it considers connecting all the nodes in the two roadmaps without taking into account the fact that the nodes are already organized into roadmaps of the subregions.

5.1 Map Connection Methods

The **Brute Force** method of map connection is the most expensive step in Feature-Sensitive Motion Planning. This approach performs k-closest connection over all the nodes in the regional roadmaps without utilizing any proximity information. It will retest all the edge connections including the edges that have been declared unfeasible. This step took from 50 to 90 percent of the total collision-detection calls. The brute force method is expensive, but it has the best connecttivity and is used as a base method to help test the other map connection methods.

The **Naive** method of map connection also uses the k-closest connection. The naive method recongnizes the different regions as seperate entities and only tries to connect the k-closest of one region to the k-closest of another region. Unlike the brute force method, the naive method does not retest edges in the same region. It only tries to make connections with nodes in the other region. This method

The brute force and naive methods ignore the fact that nodes are already in two roadmaps of neighboring regions. We need a method that exploits this fact to make more efficient connections. The **Regional Overlap** method of map connection attempts connections between nodes in or close to the overlapping regions of neighboring sections. Only the k-closest nodes will be attempted. The brute force and naive method are expensive becuse of their attempt to connect many useless nodes. Regional overlap only connects the closest nodes in an overlapping region, this reduces the cost of merging regional maps. The first step in Regional overlap is to make a list of nodes for each section. Once a list of the nodes is created they are tested to see which nodes are in the overlap. A vector of nodes in the ovelap is created for each section. A connection method is applied to the nodes in the overlap.

Algorithm 2 receives as input an environment divided into regions with roadmaps in each region and outputs one

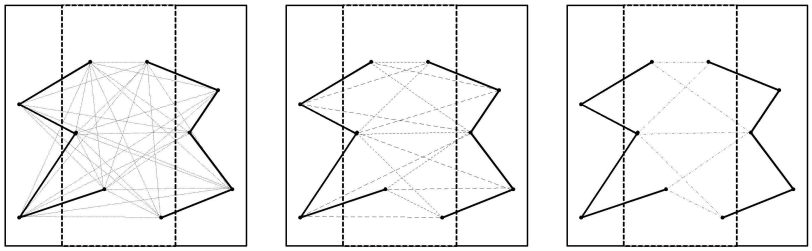
roadmap for the environment. First, a list of all the nodes in each region is made (Line 1 and 2). Next, the nodes are compared to see which nodes are in the overlap (Line 3). A vector of the nodes in the overlap is created for each region (Line 4 and 5). Finally, a connection method is called to connect the nodes in the overlap(Line 6 and 7). Once the nodes are connected a roadmap for the entire environment is returned.

Algorithm 2 Regional Overlap Map Connection

Input: An environment divided into sections of left region and right region. Each section has a roadmap
Output: An environment with one roadmap

Make l_subregions a list of nodes in the left region
2: Make r_subregions a list of nodes in the right region
Calculate which nodes are in overlaping region
4: Make l_subregion a list of nodes in l_subregions that are on the overlap
Make r_subregion a list of nodes in r_subregions that are on the overlap
6: Connet nodes in l_overlap to right map
Connet nodes in r_overlap to left map
8: **Return** One roadmap

Figure [4] shows the three methods of map connections that are listed above. Each method is shown in a basic environment with two sections. The roadmap in each section contains five connected nodes. The brute force method(figure[4a]) connects every node to evry other node. It tried 35 edges. The naive method (figure[4b])connects the k-closest (k=2) nodes in one section to the k-closest nodes in the other section. It tries 15 edges. The regional overlap method (figure[4c]) connects the k-closest nodes in or near the overlapping area. This method generates 7 edges. The results of this map connection test show that regional overlap method is the best method of connection.



(a) Brute Force (b) Naive (c) Regional Overlap

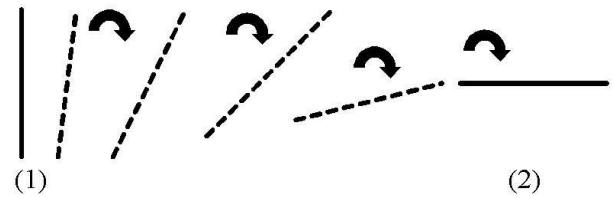
Figure 4: Map connection methods

Method	**Nodes**	**K**	**#Edges**
Brute Force	10	2	35
Naive	10	2	15
Regional Overlap	10	2	7

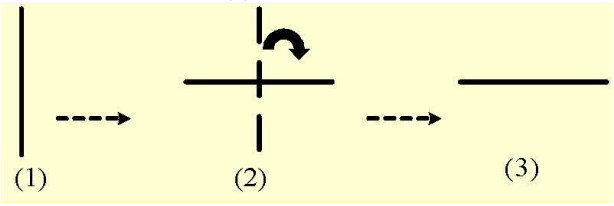
Table 1: Map connection method results

6 Local Planning

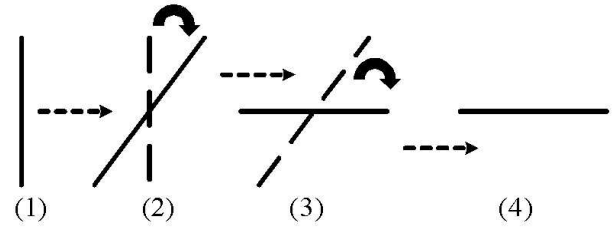
Local planners are used to make connections between roadmap nodes when building the roadmap, and also between the start and goal configurations and the roadmap when processing queries. There is a trade-off between the ability of a local planner to make connections and its running time. The general strategy of PRMs is to use a "dumb" local planner during initial roadmap construction, that is, a planner that is fast, perhaps not very powerful, and deterministic. The reason for this is that the faster the planner, the more connections can be attempted and most connection attempts will fail when environments are cluttered. The planner should be deterministic so that the paths don't have to be saved (they can be re-generated when needed). A full PRM will also need a "smart" local planner, that is, a planner that is slower, but is more likely to make connections. Smarter local planners csn be used during enhancement to connect different connected components of the roadmap, or during query processing to connect the start and goal to the roadmap; these planners could be randomized since they will be invoked fewer times and their paths can be saved.



(a) Straight-Line



(b) Original Rotate_At_S



(c) New Rotate_At_S

Figure 5: Local Planners Straight-Line and Rotate_At_S

Two local planners that are used in Motion Planning are Straight-Line and Rotate_At_S. The straight-line local planner slowly rotates to change orientation before connecting configurations. In Figure[5(a)] the starting configuration (1) slowly rotates to connect to the ending position (2). The Rotate_At_S local planner will rotate at a given value before connecting two nodes.The value represents a precentage between two nodes. In Figure[5(b)] the starting configuration(1) moves to the given point and makes a rotation to change orientation (2). After making the rotation it continues to make the connection with the ending configuration(3). To improve the Rotate_At_S planner we added the ability to perform several rotations between connections. When multiple values are given the robot will divide the

rotation between each point. In Figure [5(c)] the starting configuration(1) moves until it gets to its first point. At the first point it makes a partial rotation(2) and continues to the next point. At the next point (3) it completes the rotation and moves to the ending configuration(4). Several rotations between connections may increase the chance of finding connections.

Algorithm 3 receives as input multiple values of 's' in a vector and the configurations c_1 and c_2 to connect. It outputs true if c_1 and c_2 can be connected. First, sequence is created as a vector of configurations(Line 1). The configuration c_1 is added to sequence vector(Line 2). A temporary variable is created with the weighted sum of c_1, c_2, and the value of s_i(Line 4). The configuration partial_1 is created with the same position as the temporary variable and the same orientation of the last configuration in the sequence vector(Line 5), then partial_1 is added to the sequence vector. The partial_1 configuration marks the starting rotation point between connecting the nodes. RotWeight is a value that is inversly porportional to the product of 's_i' and '1'(Line 7). A temporary variable is created with the weighted sum of c_1, c_2, and rotWeight(Line 8). The configuration partial_2 is created with the same orientation as the temporary variable and the same position as partial_1(Line 9), then partial_2 is added to the sequence vector. The partial_2 configuration marks the last rotation point between connecting nodes. Afterwards the c_2 configuration is added to the sequence vector and an attempt to connect every configuration in the sequence vector is made(Line 13). If the configurations can be made, true is returned.

Algorithm 3 Rotate_At_S: is connected

Input: Multiple values of 's' in vector s_values, configurations to connect c_1 and c_2
Output: Return true if c_1 and c_2 can be connected

1: let *sequence* be an empty vector of configurations
2: add c_1 to *sequence*
3: **for all** s_i in the vector s_values **do**
4: let *tmp* be the weighted sum of $c_1 * s_i$ and $(1 - s_i) * c_2$
5: make $partial_1$ be a configuration with same position as *tmp* and same orientation as *sequences*.last()
6: add $partial_1$ to *sequences*
7: let $rotWeight$ be inversely proportional to s_i times i
8: let *tmp* be the weighted sum of $c_1 * rotWeight$ and $(1 - rotWeight) * c_2$
9: let $partial_2$ be a configuration with same position as $partial_1$ and same orientation as *tmp*
10: add $partial_2$ to *sequences*
11: **end for**
12: add c_2 to *sequences*
13: **if** can connect every configuration $sequence_i$ to $sequence_{i+1}$ with straight_line local planner **then**
14: **Return** True
15: **else**
16: **Return** False
17: **end if**

7 Conclusions and Future Work

We introduced techniques to improve feature-sensitive motion planning. Our early work on this topic produced a framework based on machine learning that subdivides a problem into regions which are then mapped with a randomized planner determined by the features of the region. Here, we proposed more efficient integration methods that do not waste effort in mapping areas that specialized planners have already mapped.

To get a better idea of how the new implementation of Rotate_At_S compares to the old version and the straight-line method, several test will be conducted. Thes test will show how well each method works and for which environment

they work best in. In this paper a simple test to show the differences in each method of map connection was shown. As part of our future work we would like to run several test to compare the different map connection methods.

References

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Robotics: The Algorithmic Perspective*, pages 155–168, Natick, MA, 1998. A.K. Peters. Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.

[2] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, 1991.

[3] P. Bessiere, J. M. Ahuactzin, E. G. Talbi, and E. Mazer. The Ariadne's clew algorithm: Global planning with local methods. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, volume 2, pages 1373–1380, 1993.

[4] R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. In *Proc. Int. Conf. Artif. Intel.*, pages 799–806, 1983.

[5] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[6] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages SA1–SA18, 2000.

[7] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[8] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Trans. Robot. Autom.*, 12:566–580, 1996.

[9] O. Khatib. Real–time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.*, 5(1):90–98, 1986.

[10] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 473–479, 1999.

[11] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[12] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, Utrecht/Zeist, The Netherlands, July 2004.

[13] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 421–427, San Juan, Puerto Rico, October 1979.

[14] J. T. Schwartz and M. Sharir. On the "piano movers" problem II: General techniques for computing topological properties of real algebraic manifolds. *Adv. Appl. Math.*, 4:298–351, 1983.