Interactive learning of Bayesian Networks using OpenMarkov

Iñigo Bermejo
UNED, Spain
ibermejo@dia.uned.es

Jesús Oliva
CSIC, Spain
jesus.oliva@car.upm-csic.es

Francisco Javier Díez
UNED, Spain
fjdiez@dia.uned.es

Manuel Arias
UNED, Spain
marias@dia.uned.es

Abstract

Algorithms for learning Bayesian networks (BNs) behave as a black box that takes a database as an input and returns a network as the output. In contrast, OpenMarkov, our tool for probabilistic graphical models, includes the option to run the algorithms in a step-by-step fashion, presenting a ranked list of operations (such as adding, removing, or inverting links) the user can select, while allowing live edition of the BN throughout the learning process. The application offers some data preprocessing options and the possibility to use a model network to guide the learning process. This functionality in OpenMarkov can be employed to learn BNs with partial expert knowledge, to debug new algorithms, and as a pedagogical tool.

1 Introduction

A probabilistic graphical model (PGM) consists of a joint probability distribution defined on a set of variables $\mathbf{V}$ and a graph containing a node for each variable X in $\mathbf{V}$; the structure of the graph imposes some relations of conditional independence on the structure of the network, which depend mainly on the type of graph. Some types of PGMs are Bayesian networks (BNs), Markov networks, influence diagrams, hidden Markov models, factored MDPs and POMDPs, etc.[1]

In many cases PGMs are built from expert knowledge: causal relations are used to draw the arcs of the graph, and the conditional probabilities are obtained from the literature (for example, medical journals), from databases, or from experts' estimations. The difficulty and tediousness of this approach has led to an increasing interest for learning methods that can generate PGMs from databases automatically. This is the preferred approach when there is a large database with few or none missing values, and no causal knowledge. However, in many cases the size of the database does not allow to learn a PGM that accurately represents the conditional independencies existing in the domain of application. Practitioners of these methods often encounter that the PGM obtained contains some links that the expert considers as obviously spurious, but given that in general the algorithms perform as black boxes, it is diffi-

[1] `www.cisiad.uned.es`.

cult to determine to what degree those links are really supported by the data.

Another problem is that most learning algorithms do not return causal models. In the last decade the number of studies aimed at obtaining causal models from databases has grown exponentially (the UAI Conference held in Barcelona in July 2011 was a clear illustration of this phenomenon). However in many cases the problem does not lie in the algorithm but on the lack of information in the database: the only conclusion that can be drawn reliably from a set of data—provided that it is big enough and not biased—is the set of correlations that exist in the real world. These correlations rule out some causal models, but the number of models compatible with the data is usually very large. Many of those models clash with common knowledge of the experts, but it is not easy to feed that knowledge into automatic causal learning algorithms. For this reason, it would be useful to have interactive learning algorithms that propose a list of changes to improve the accuracy of the network but allow an expert to select only those that do not contradict his/her knowledge.

Secondly, interactive learning could be also of great interest for researchers and developers of new algorithms. An interactive learning tool able to show on a graphical interface the different actions that the algorithm is considering at each step and the scores assigned to them may be very useful to debug the algorithm, for example, by observing how a shift in some of the parameters leads to a different selection of actions.

Thirdly, interactive learning programs may have a high pedagogical value by allowing the students to know the actions that the algorithm has evaluated at each step and why it has selected each action. Then it is possible to run a different algorithm, for example with a different search strategy or a different metric, and observe why it selects different actions at each step.

For these reasons we decided to implement an interactive learning module on OpenMarkov, an open-source software tool for editing and evaluating PGMs. The interactive learning module includes a user-friendly graphical interface that allows to overcome all the above-pointed problems, with the corresponding benefit for experts, researchers, developers and students.

The rest of this paper is structured as follows. In Section 2 we give a brief overview of Bayesian network learning and of the OpenMarkov tool. Section 3 describes the different options available to the user for learning BNs interactively in OpenMarkov. Section 4 presents a case study: how to learn interactively the BN *Alarm*, a model frequently used in the literature as a benchmark for learning algorithms. In Section 5 we discuss the advantages of our approach and similar approaches, and Section 6 contains the conclusions and proposals for future work.

2 Background

2.1 Learning Bayesian networks

Learning Bayesian networks is one of the most important research areas in the field of BNs. Every year, around one third of the total publications in that area are related to automatic learning. Just like in manual construction, automatic learning of BNs presents two aspects: parametric learning and structural learning. Parametric learning consists of computing the conditional probabilities given by the structure of the network using the observed frequencies on the database. Structural learning tries to find the graph that best represents the probability distribution based on the frequencies in the database.

Structural learning methods There are two main methods for building the graph of a BN from a database. The first one consist of detecting the probabilistic conditional independencies present in the database. The most famous algorithm of this type is the *PC algorithm* (Spirtes and Glymour, 1991; Spirtes et al., 2000). The second method, called *search and score*, consists of performing a heuristic search through the space of possible structures, using a metric that measures how well each structure can represent the probability distribution of the variables in the database. Several

metrics have been proposed in the literature: Bayesian (which include K2 and BDe as particular cases), cross-entropy, AIC, and MDL—see (Bouckaert, 2004) for references. K2, the first algorithm of this type, performed a search by departing from a network without links and adding at each step the link leading to the highest score, provided that the score was positive (Cooper and Herskovits, 1991). The method that proceeds by examining one operation at each step (adding, removing, or inverting a link) is called *hill climbing*.

2.2 OpenMarkov

This project started in 2002 at the Department of Artificial Intelligence of the Universidad Nacional de Educación a Distancia (UNED), in Madrid, Spain. Its original name was Carmen (Arias and Díez, 2008), but in 2010 it was renamed as OpenMarkov.[2] We departed from our experience in the construction of Elvira (Elvira Consortium, 2002),[3] an open-source tool begun in 1997 as a joint project of several Spanish universities, but everything in the new program was redesigned and the code of OpenMarkov was built from scratch. The language chosen to develop OpenMarkov was Java, mainly to make it multi-platform.

OpenMarkov is able to represent several types of networks, such as Bayesian networks, Markov networks, influence diagrams, LIMIDs, and decision analysis networks (DANs), as well as several types of temporal models: dynamic Bayesian networks, Markov processes with atemporal decisions (MPADs), MDPs, POMDPs, Dec-POMDPs, and dynamic LIMIDs—see (Arias et al., 2011) for definitions and references. Currently it can only evaluate Bayesian networks, influence diagrams, and MPADs. Each network type is defined by a set of constraints (Arias et al., 2011, Appendix A), which leads to the possibility of defining new types of networks easily by combining the existing constraints and, if necessary, by adding new ones. Constraints play also an important role in the learning of BNs, as we will discuss below.

There are three types of variables in OpenMarkov: finite-states, numerical, and discretized. A discretized variable has a finite set of states, each one having an associated numeric interval.

The graphical user interface (GUI) is very similar to those of other software tools for PGMs, especially to that of Elvira. It has two working modes: edition and inference. It has been designed for internationalization; currently messages can be displayed in English and Spanish. For further details, see OpenMarkov's web pages and wiki.[4]

3 Options for learning BNs in OpenMarkov

In this section we describe the main options that OpenMarkov offers for learning BNs interactively.

3.1 Using a model network

OpenMarkov gives the user the option to use an existing network as a model for the one that will be learned. There are four options. The first is to use the model only to determine the positions of the nodes. When we learn a network from a database we can place the nodes on the screen by dragging them with the mouse, trying to minimize the number of links crossing one another; but if we learn another network from the same database (for example, using different options for the algorithm), we should drag replace the nodes again. OpenMarkov facilitates this task by placing the nodes in the same positions as in a network built previously.

The other three options are whether the algorithm can add, remove, or invert the links present in the model network. They are useful, for example, when we wish that the algorithm preserves all the links in the model network. The first option (i.e., using the model only to place the nodes) is incompatible with the other three, which are compatible with one another.

[2] `www.openmarkov.org`.

[3] `www.ia.uned.es/~elvira`.

[4] `www.openmarkov.org`, `wiki.openmarkov.org`.

There are other uses of the model network, that we describe below.

3.2 Data preprocessing

Unfortunately data in the databases is usually not suitable to be directly fed to the learning algorithm and has to be preprocessed. OpenMarkov offers the following options.

Selection of variables Usually raw databases contain information that is irrelevant for the model (e.g. the patient's name). OpenMarkov can learn a network that contains all the variables in the database, but it is also possible to tell it to use only those present in the model network. The third possibility is to select the variables one by one from a list.

Discretization of numeric variables Currently OpenMarkov can only learn BNs with variables having finite states. Therefore, the numeric variables in the database must be discretized before feeding the data to the learning algorithm. OpenMarkov can discretize a variable in different ways. First, the user can indicate a number of intervals and then indicate whether the intervals must have the same width (considering the maximum and the minimum for that variable in the database) or the same frequency (i.e., the number of database registers for every interval will be the same). Second, if the variable is discretized in the model network, its intervals can be used to assign each number in the database to a state. For example, if a variable has three states, "negative", "null", and "positive", with three associated intervals, $(-\infty, 0)$, $[0, 0]$, and $(0, +\infty)$, respectively, these intervals can be used to discretize the values in the database. This way, creating a model network is a way of specifying how numeric variables should be discretized.

Imputation of missing values Currently OpenMarkov offers only two ways to fill in the gaps in the database: either to ignore every register that contains at least one missing value, or to write the value "missing" in every empty cell, which is then treated as if it were an ordinary value.

3.3 List of suggested edits

In OpenMarkov an *edit* is an atomic modification of a data structure. There are three edits that an interactive learning algorithm can propose: adding, removing, or inverting a link. The list is composed by sorting the edits according to their scores. The hill climbing algorithm computes the scores using the metric selected by the user. The PC algorithm performs many statistical test in which the null hypothesis is that two variables are not correlated given other variables. Roughly speaking, a high p resulting from the test suggests that two variables are conditionally independent, i.e., that a link can be removed. Therefore, the p value can be used as a score to rank the edits, each edit being the removal of a link.

Interactive learning is performed by having two windows: one showing the graph of the network and another one showing the proposed edits. The user can select any edit from the list, not necessarily the one having the highest score, and the change will be immediately displayed on the network window. Alternatively, the user can add or remove any link from the graph. In both cases, the scores will be recalculated and a new list will be proposed. Figure 1 shows the lists of suggested edits shown during the interactive learning process.

Additional options There are additional options to control the flow of the algorithm. One of them is tell OpenMarkov to show only the edits having a positive score. Another option is to show only the edits allowed by the constraints associated to the network. For example, a constraint stemming from the definition of the BN is that the graph cannot contain cycles. A constraint that the user can impose is that a node cannot have more than n parents. If the user selects the "Show only allowed edits" option, those incompatible with the constraints will not be shown in the list, even if they have a high score.

Finally, the user has the possibility of blocking a certain edit to prevent the system from offering it again and again. Blocked edits can be later unblocked at any moment.

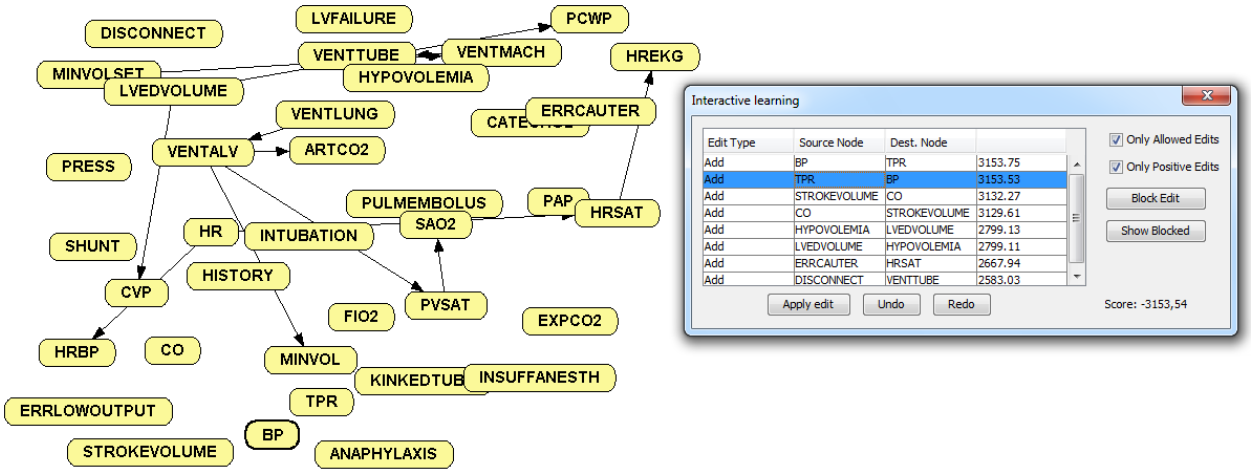

Figure 1: A moment of the interactive learning process: list of edits proposed by OpenMarkov and the network being learned.

4 Case study

In order to explore the benefits of interactive learning, we study the case of learning the well-known ALARM network (Beinlich et al., 1989), which has 37 nodes and 46 links. The nodes are classified into three levels. The first level contains *diagnostic nodes*, which have no predecessors. The second level contains *intermediate variables*, representing pathophysiological anomalies that cannot be observed directly. The third contains *measurement nodes*, which represent clinical variables that can be observed or measured, and do not have children. The network contains no link from a lower level to an upper level.

Using this network we generated a database containing 10,000 samples and applied the hill-climbing algorithm with the K2 metric. We applied the learning algorithm automatically in OpenMarkov, resulting in a model with 50 links, 13 of which were not in the original network, even though 6 of them were inverted links of the original network. On the other hand, 9 of the original links were missing in the network learned.[5]

Then we learned the network interactively using elementary causal knowledge, according to which we did not accept the addition of any link from a measurement node to an intermediate or a diagnostic node, nor from an intermediate node to a diagnostic node. Figure 2 shows an example of a moment in the interactive learning process where the edit with the highest score contravenes the causal knowledge. In this case, we chose to apply the second edit, which produces the same link but in the opposite direction.

The resulting net contained 47 links: only 2 of them were not in the original network and only one of the original links was missing. The two links "invented" by the learning algorithm were the last to be added and had such a low score that they might have been detected as spurious by the expert. We also observed that the missing link, from INSUFFANESTH to CATECHOL, has a very weak influence in the original network.

This experiment shows that even a portion of very rudimentary knowledge about the domain may lead to a significant improvement in the network built by our interactive learning algorithm.

5 Discussion

5.1 Advantages of our approach

As mentioned in the introduction, a problem of learning algorithms is that they often create spurious links due to small correlations exist-

[5]The files used in this study are available at `www.openmarkov.org/learning` so that the results can be reproduced by other researchers.

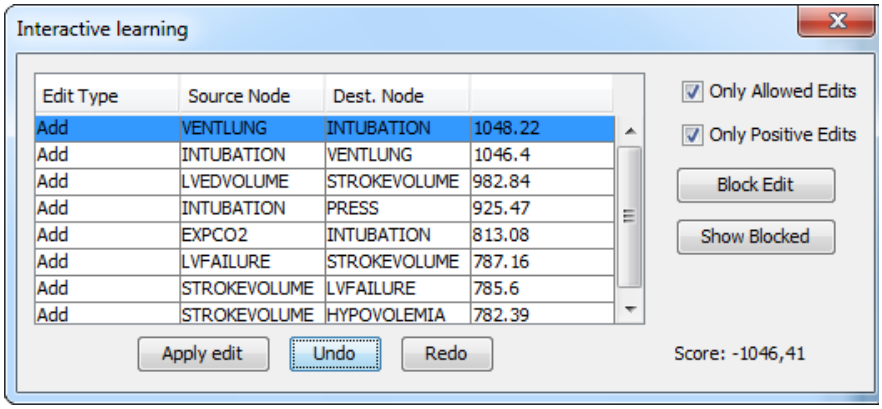

Figure 2: The edit suggested at the top of the list contravenes our causal knowledge because VENTLUNG is an intermediate variable and INTUBATION is a diagnostic variable.

ing in the database. Another problem is that in general the models obtained are not causal, not only because of the inherent limitation of most algorithms, but mainly because the information contained in the database does not permit to distinguish whether X is a direct cause of Y, or X is a cause of Y, or if there is a directed causal path between them involving other variables, or they have a common cause, or there is a selection bias in the database (Glymour and Cooper, 1999; Druzdzel and Díez, 2003).

OpenMarkov allows human users, who may be experts in their respective areas but novices in the field of probabilistic modelling, to supervise the execution of learning algorithms. The algorithm proposes some incremental modifications of the network, based on the information contained in the database, and the user has the opportunity to apply some of the changes proposed by the tool or impose others at any moment of the learning process, based on their expertise. Even if this might lead to a lower quality of the network according to the metric, the result might be better from the point of view of users' acceptability, because human experts are reluctant to accept the advice of a machine if they cannot follow its reasoning (Teach and Shortliffe, 1984).

An interactive learning tool might as well be useful for researchers that have developed a new algorithm and wish to trace its execution in order to debug or fine-tune it. This process can be done by inserting in the algorithm a few lines of code that print a trace on the standard output or in a file, but it is much nicer to observe graphically the operations performed by the algorithm, step by step, together with the qualitative information associated with the next modifications that the algorithm has evaluated. Obviously, this requires that the researchers implement the new functionality (such as a new metric, a new search technique, or a completely novel learning method). OpenMarkov's architecture has been carefully to permit these extensions: each new method can be implemented as a Maven subproject, that OpenMarkov will detect at run time as a plug-in.[6] This way, researchers can extend OpenMarkov dynamically without modifying the "official" source code.

Finally, an interactive learning program may be useful as a pedagogical tool to explain the performance of different algorithms: rather than observing the input and the output, students may follow the progression of the algorithm step by step, understanding why each change was selected, seeing the effects of taking different actions to those proposed by the system and comparing different algorithms.

5.2 Related work

Interactive structural learning was proposed by Sucar and Martínez-Arroyo (1998) as a means

[6] See `https://bitbucket.org/cisiad/org.openmarkov/wiki/OpenMarkov-organization.pdf`.

for human-computer collaboration in the search for the optimal network structure. Their system initially builds a tree automatically and then allows the user to modify it by adding, removing or inverting arcs. The strength of the correlation between the variables connected by a link is denoted graphically by the width of the link. It also shows a score representing the quality of the model, which is inversely related with the complexity of the network and the distance between the probability distributions specified by the network and the data.

Our work, developed independently, is also based on the idea of combining a learning algorithm and expert knowledge to build a model interactively. The main difference is that our system guides the expert through the creation process step-by-step instead of providing the final result of the learning algorithm and asking the expert to fine tune it. We think that our approach contributes to a better understanding of the behavior of the learning algorithm and therefore makes collaboration with the expert easier.

The idea of showing a score that represents the quality of the model is also present somehow in our approach, because the scores associated to each edit in the score and search algorithm represent the increment in the quality of the model, given by the complexity of the network and the distance between the probability distribution of the network and that of the data.

In the future, we will implement in OpenMarkov the possibility of representing the type of correlation (positive, negative, or null) by a color and the strength of the correlation by the thickness of the link, as we did in Elvira (Lacave et al., 2006; Lacave et al., 2007), thus making our approach more similar to that of Sucar and Martínez-Arroyo.

Later de Campos and Castellano (2007) encoded expert knowledge in the form of a set of restrictions that the learning algorithm had to satisfy. They used three types of restrictions: presence of a link, absence of a link, and partial ordering of the variables. They proved that, for several data sets, this approach improved the quality of the networks learnt. Our work is similar in that, by means of the model network, we can impose or prevent the presence of some links in the network learnt, but we do not have yet partial order restrictions. However, it would be easy to implement any restriction—not only those used by de Campos and Castellano—as an OpenMarkov constraint, as explained in Section 3.3. Another difference is that in OpenMarkov it is possible to specify whether the model network imposes the presence or absence of a link, or it only "suggests" them as a starting point, that can be overriden depending the scores computed by the algorithm.

Another similarity between their work and ours is that both of them have been combined with score-and-search and independence-based algorithms. However, the main difference is that their approach is not interactive: their restrictions must be declared before the execution of the algorithm, which then runs automatically, while in OpenMarkov every action proposed by the algorithm can be accepted or rejected by the user, who can also impose any action at any moment of the process, thus giving full control to the user.

6 Conclusions and Future Work

In this paper we have described an interactive learning approach for learning Bayesian networks from databases, which may be very useful for the experts in different application domains, as well as for researchers and students in the field of PGMs. We have shown with a case study that even very rudimentary causal knowledge about the domain may lead to a significant improvement of the network build interactively with a learning algorithm.

The main lines for future development would be to borrow some ideas from the work of Sucar and Martínez-Arroyo (1998) and de Campos and Castellano (2007), such as representing graphically the strength of the correlation between variables and having richer types of constraints. It would be also useful to show an absolute quality measure of the net rather than the incremental one we currently have, given by the complexity of the network and the

distance between the probability distribution of the network and that of the data. This quality measure could be used to compare the resulting nets of the interactive and non-interactive learning processes.

Another research line would be to adapt our approach to learning Bayesian classifiers, a somewhat different problem, as the objective is not to build the network that better represents the probability distribution of the data, but the network that better classifies new cases.

Acknowledgments

We thank Concha Bielza, Pedro Larrañaga and the reviewers for their useful comments. This work has been supported by grants TIN2006-11152 and TIN2009-09158, from the Spanish Ministry of Science and Technology. I.B. has received a predoctoral fellowship from the Universidad Nacional de Educación a Distancia (UNED), and J.O. from the Consejo Superior de Investigaciones Científicas (CSIC).

References

[Arias and Díez2008] M. Arias and F. J. Díez. 2008. Carmen: An open source project for probabilistic graphical models. In *Proceedings of the Fourth European Workshop on Probabilistic Graphical Models (PGM'08)*, pages 25–32, Hirtshals, Denmark.

[Arias et al.2011] M. Arias, F. J. Díez, and M. P. Palacios. 2011. ProbModelXML. A format for encoding probabilistic graphical models. Technical Report CISIAD-11-02, UNED, Madrid, Spain.

[Beinlich et al.1989] I. A. Beinlich, H. J. Suermondt, R. M. Chávez, and G. F. Cooper. 1989. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the 2nd European Conference on AI and Medicine*, pages 247–256, London. Springer-Verlag, Berlin.

[Bouckaert2004] R. R. Bouckaert. 2004. Bayesian networks in Weka. Technical Report 14/2004, Computer Science Department, University of Waikato, New Zealand.

[Cooper and Herskovits1991] G. F. Cooper and E. Herskovits. 1991. A Bayesian method for constructing Bayesian belief networks from databases. In *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence (UAI'91)*, pages 86–94, Los Angeles, CA. Morgan Kaufmann, San Mateo, CA.

[de Campos and Castellano2007] L. M. de Campos and J. G. Castellano. 2007. Bayesian network learning algorithms using structural restrictions. *International Journal of Approximate Reasoning*, 45:233–254.

[Druzdzel and Díez2003] M. J. Druzdzel and F. J. Díez. 2003. Combining knowledge from different sources in probabilistic models. *Journal of Machine Learning Research*, 4:295–316.

[Elvira Consortium2002] The Elvira Consortium. 2002. Elvira: An environment for creating and using probabilistic graphical models. In J. A. Gámez and A. Salmerón, editors, *Proceedings of the First European Workshop on Probabilistic Graphical Models (PGM'02)*, pages 1–11, Cuenca, Spain.

[Glymour and Cooper1999] C. Glymour and G. F. Cooper. 1999. *Computation, causation and discovery.* The MIT Press, Cambridge, Massachusetts.

[Lacave et al.2006] C. Lacave, A. Oniśko, and F. J. Díez. 2006. Use of Elvira's explanation facilities for debugging probabilistic expert systems. *Knowledge-Based Systems*, 19:730–738.

[Lacave et al.2007] C. Lacave, M. Luque, and F. J. Díez. 2007. Explanation of Bayesian networks and influence diagrams in Elvira. *IEEE Transactions on Systems, Man and Cybernetics—Part B: Cybernetics*, 37:952–965.

[Spirtes and Glymour1991] P. Spirtes and C. Glymour. 1991. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9:62–72.

[Spirtes et al.2000] P. Spirtes, C. Glymour, and R. Scheines. 2000. *Causation, Prediction and Search.* The MIT Press, Cambridge, Massachusetts, second edition.

[Sucar and Martínez-Arroyo1998] L. E. Sucar and M. Martínez-Arroyo. 1998. Interactive structural learning of bayesian networks. *Expert Systems with Applications*, 15:325–332.

[Teach and Shortliffe1984] R. L. Teach and E. H. Shortliffe. 1984. An analysis of physician's attitudes. In B. G. Buchanan and E. H. Shortliffe, editors, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, chapter 34, pages 635–652. Addison-Wesley, Reading, MA.