Learning What Is Relevant to the Effects of Actions for a Mobile Robot

Matthew D. Schmill, Michael T. Rosenstein, Paul R. Cohen, and Paul Utgoff
Experimental Knowledge Systems Laboratory
Department of Computer Science, LGRC, Box 34610
{schmill,mtr,cohen,utgoff}@cs.umass.edu

Abstract

We have developed a learning mechanism that allows robots to discover the conditional effects of their actions. Based on sensorimotor experience, this mechanism permits a robot to explore its environment and observe effects of its actions. These observations are used to learn a *context operator difference table*, a structure that relates circumstances (context) and actions (operators) to effects on the environment. From the context operator difference table, one can extract a relatively small set of state variables, which simplifies the problem of learning policies for complex activities. We demonstrate results with the Pioneer 1 mobile robot.

1 Introduction

This paper describes a method by which a robot learns context-dependent effects of its actions. The robot is a Pioneer 1, manufactured by Real World Interfaces, Inc., equipped with sonars and a gripper. By interacting with its environment, the robot learns a *context operator difference table* (CODT) that relates primitive actions to changes in sensor readings. In general, the effects of actions will depend on context. For instance, if the robot has run into an obstacle, then attempted forward movement will have little or no effect on the values returned by forward-pointing sonars, whereas moving backward, away from the obstacle, will affect these sonar readings. Each cell in a CODT specifies the effects of a single action on readings from a single sensor in context. For example, when the robot is jammed against an obstacle, the `left-wheel-stalled` and `right-wheel-stalled` sensors return "true", and in this context, attempted forward movement will have no effect on the robot's frontmost sonar, but attempted backward movement will be accompanied by an increase in the readings from that same sonar. Conditional knowledge of this sort is stored in decision trees, and the CODT contains one such tree learned for every action-sensor pair.

Although CODTs represent the effects of actions, they are not used to control the robot. Control policies are learned by another algorithm, described below. In fact, the valuable contribution of CODTs is not the mappings they provide from actions to effects but, rather, the contexts for these mappings. CODTs tell us which of many variables are *relevant* to the conditional effects of actions. Said differently, they specify upon which variables effects are conditioned. Because they specify which (relatively few) variables are relevant to the effects of actions, CODTs can significantly reduce the dimensionality of search spaces for control policies. Thus CODTs are part of a divide-and-conquer strategy for learning robot control strategies. We outline the entire strategy in order to show the role of CODTs.

Influenced by the interactionist philosophy of Mark Johnson and George Lakoff [Lak84, Joh87], the developmental psychology of Jean Mandler [Man92], and our previous work in the Baby project [COAB96], we are using Pioneer 1 robots to implement a theory of the origins of knowledge. The theory says that conceptual systems (or ontologies) are grounded in activity and that classes and concepts are essentially *interactionist*. Through interaction an infant learns that some objects are furry and soft, others are inflexible and hard. If this distinction is salient, the infant may identify one or more extensional categories of objects. Similarly, a robot may distinguish graspable objects from those it cannot grasp. But why should it? The distinction between graspable and non-graspable objects must be salient—it must matter *to the robot*—otherwise it should not be made and the corresponding categories should not be formed. In our theory, distinctions are salient if they help the robot predict whether it will achieve its self-determined goals. For example, the robot might establish the goal of moving an object to a location. The distinction between graspable and non-graspable objects is salient in this case because it affects how (or whether) the robot will achieve its goal. In sum, categories and concepts are based on distinctions (such as graspability) which are salient to the robot as it learns activities. Much of our research has focused on making a robot learn increasingly complex activities in an unsupervised way—without us specifying *which* activities it should learn.

We are developing two approaches to learning activities in which increasingly complex activities are constructed from subactivities and primitive actions. For instance, carrying an object to a location (`carry-to`) consists of subactivities `grasp-object` and `move-to`, which are themselves constructed from parameterized primitives including `move` and `close-gripper`. In the first approach, *hierarchical reinforcement learning*, each activity has its own relevant state variables, its own available subactivities or actions, its own goal, and its own evaluation function for representing a policy to achieve the goal. The second approach is also constructive in

To appear in the *Proceedings of the Second International Conference on Autonomous Agents.*

the sense of building larger activities from smaller ones, and it also reinforces some activities over others, but it does not rely on goals, nor does it back up reward, as reinforcement learning usually does.

Context operator difference tables facilitate activity learning by reducing the state space that the learning algorithm must search. This is especially important for hierarchical reinforcement learning. To avoid the curse of dimensionality, most applications of reinforcement learning involve careful design, by an experienced researcher, of the state space and reinforcement function. However, our research program precludes this: We want robots to learn complex activities and concepts without ourselves having to specify individual reward functions and state spaces for each activity. How can the learning algorithm construct its own state spaces (one for each activity) from subsets of state variables? Very few state variables are actually relevant to any given activity, so the value function for an activity can often be learned efficiently in a small state space, if only the learning algorithm knows which state variables to ignore. This is exactly the knowledge provided by CODTs, which identify the context variables that are relevant to the effects of actions. Only those variables need be included in the state space for learning an activity.

2 Context Operator Difference Tables

Operator difference tables, familiar to students of means-ends analysis [NS63], have actions on one axis and state variables on the other, and the cells contain symbols, such as '+', '−', or functions, which represent the unconditional effects of the actions on the variables. Context operator difference tables (CODTs) represent the *conditional* effects of actions. The rows and columns of CODTs list actions and state variables, respectively, but the cells contain decision trees that represent the effects of actions on state variables in different contexts. At the leaves of these trees are the symbols '+', '−', and '0', which represent effects. The interior nodes are tests on state variables. In this study, each state variable corresponds to one sensor on the Pioneer 1 robot.

Consider a table entry relating the action `move` to the state variable `moving`. The tree in this cell may split on the variable `left-wheel-stalled` and terminate in two leaves, + and 0. Its interpretation is, "If the robot's left wheel is not stalled, a move command produces movement, otherwise it produces no movement." A more surprising example relates the action `open-gripper` to the state variable `stalled`: The tree in this CODT cell says "If the robot is stalled, and the sonars indicate that an object is very nearby, then opening the gripper will make the robot unstalled." The reason for this (which is not encoded in the CODT tree) is that when the robot bumps into a wall and stalls, opening the gripper will sometimes push the robot away from the wall, freeing it.

The process of learning context operator difference tables from sensiomotor experience involves repeated observation and learning. During observation, the agent perceives states and classifies the changes that are observed in its sensor readings. Once an observation becomes available, the agent learns from it by updating the structures of the CODT to reflect the new information. We now describe the mechanisms that classify changes in sensor readings and generate CODT entries.

2.1 Classifying Changes

To learn the context-dependent effects of its actions, the robot must interact with its environment and accumulate observations. Each observation is represented as a data structure consisting of an action description, a list of sensor readings (the *raw context*), and a list of class labels that describes the changes in sensor readings (the *raw effects*).

Observations are acquired with the help of the *sensor monitor*, which classifies changes in sensor readings. We use the Saphira multitasking system (SRI International, Menlo Park, CA), which handles the low-level details of communication with the Pioneer robot. Saphira supports customized *microtasks*, or synchronous C functions. The sensor monitor is such a microtask. Every 100 ms the monitor polls each of the robot's 19 sensors, and buffers the most recent readings.

To classify the effects of an action on sensor readings, the sensor monitor looks for a difference in the values of the readings before and after the action. More specifically, for each sensor, the *slope* of the readings before the action is compared with the slope after the action. For instance, if the robot is sitting still with its gripper open, then the slope of the readings from a forward-looking sonar will be zero. If the robot then closes its gripper, this action will not affect the sonar readings, in the sense that the slope will continue to be flat. But if the robot is moving toward an obstacle (producing a negative slope on forward-looking sonar readings) and then moves away from it, the slopes of the sonar readings will shift from negative to positive, and the robot will have the opportunity to learn that moving backwards changes the sign of the rate of change of sonar readings.

These slope values are computed by an incremental algorithm that fits a least-squares line to the most recent ten data points. (More or fewer points could be used, but ten points, representing one second of activity, yields sufficient data for the statistical test described below, given the noisiness of the data.) Let $before_i$ be the slope for sensor i before the action, and $after_i$ be the slope of ten values from sensor i after the action and a short delay. (The delay gives the action time to have an effect. Presently, the delay is fixed at 1000 ms, although in future work, the robot may learn an appropriate delay for each sensor.) To classify the effects of an action, the sensor monitor calculates $before_i - after_i$ for each sensor i, and converts the result to a class label, '+', '−', or '0', representing an increase, a decrease, or no change in slope, respectively. More precisely, the monitor tests the hypothesis that $before_i = after_i$ with the following statistical comparison of the slopes of two regression lines:

$$t = \frac{before_i - after_i}{\hat{\sigma}_{b_1 - b_2}}, \tag{1}$$

where $\hat{\sigma}_{b_1-b_2}$ is the estimate of the standard error for the difference of slopes from two regression lines. This estimate is computed from the pooled variance, or sums of squares of residuals from each least-squares line:

$$\hat{\sigma}_{b_1-b_2} = \sqrt{\hat{s}^2_{pooled}\left(\frac{1}{SS_{X_1}} + \frac{1}{SS_{X_2}}\right)} \tag{2}$$

$$\hat{s}^2_{pooled} = \frac{SS_{residual_1} + SS_{residual_2}}{df} \tag{3}$$

$$SS_{residual} = \left(1 - r^2\right) SS_Y \tag{4}$$

In Eqs. (2)-(4), SS_X and SS_Y are the sums of squares for the independent and dependent variables (respectively, time

Figure 1: A Pioneer 1 robot exploring the laboratory environment.

and sensor i), and r^2 is the goodness of fit for the regression line. Finally, the t statistic is compared to the t distribution with $n_1 + n_2 - 4$ degrees of freedom to see whether the hypothesis of equal slope should be rejected [KJ89]. In our experiments, $n_1 = n_2 = 10$, because each slope is based on ten observations.

2.2 Generating CODT Entries

The sensor monitor provides a fast, incremental way to classify the observed effects of its actions. The next step is to generate CODT entries.

The CODT is a two dimensional table, one dimension indexed by an action, the other by a sensor, that is, a state variable. Each entry (a, s) of the CODT contains a decision tree that represents the effects of action a on sensor s, induced on whatever observations were made of the pair (a, s). The decision trees encode two types of information about the Pioneer's experience. The leaves express how an action is expected to change a sensor reading. A leaf that predicts the class '+' reflects the experience that action a has been shown to cause the slope of sensor s to increase. Decision nodes within the trees express context dependencies among the observations. A decision node that splits on the value of `left-wheel-stall` indicates that the effect of action a on sensor s varies and may be predicted more accurately when the stall state of the left wheel is taken into account. By a simple tree traversal of a column of the CODT, the Pioneer robot can quickly generate a set of actions and context variables relevant to achieving a desired sensory state.

The trees in CODTs are built by an algorithm called TBA that uses bonferroni adjustments to guard against overfitting [JS97]. We developed TBA originally to test a statistical theory of overfitting, but the algorithm is especially important for the current application. The whole point of CODTs is to find small subsets of context variables that are sufficient to predict context dependent effects of actions. Decision tree algorithms that overfit (and all but TBA do) will produce subsets of context variables that are larger than they need to be. Overfitting-avoidance schemes other than TBA's do not in general fix this problem [JS97]; thus TBA is necessary to fulfill the promise of CODTs.

Unfortunately, TBA is not an incremental algorithm, so observations are processed in batches. Currently, TBA is employed on a fixed schedule to rebuild the CODT trees whenever 20 observations are waiting for processing. We hope to develop an incremental version of TBA in the near future. In section 4, we examine some trees built by TBA and their meaning.

3 Experiments

To evaluate the CODT learning algorithm, we conducted four trials in which the Pioneer mobile robot explored its environment. During each trial the robot performed 150 actions selected randomly from four primitive types: `move(`x`)`, `turn(`x`)`, `open-gripper`, and `close-gripper`. The parameter x denotes the desired translation or rotation of the robot, in millimeters or degrees, respectively. Values of x for `move(`x`)` were selected from the uniform distribution $[-1000, +2000]$; similarly, values of x for `turn(`x`)` were selected from the uniform distribution $[-360, +360]$. Actions were performed sequentially, so that the robot's state after one action served as the initial state for the next. Occasionally, an action would begin before the previous action finished; for example, `move(`x`)` might not finish because the robot runs into an obstacle.

In these experiments the goal was to learn a CODT relating four primitive actions to nineteen native sensors: seven sonars, four velocity transducers (translational, rotational, left and right wheels individually), four gripper state switches (vertical position, two break beams, one bumper),

three stall encoders, and heading.

The experiments were performed in a messy laboratory environment with some irregularly-shaped open floor space and common obstacles including chairs, desks, waste baskets, partitions, and people (see Figure 1). During a typical trial, the Pioneer wandered about the room, occasionally opening or closing its gripper. Sometimes, the robot approached a small object (a plastic bottle) with its gripper open and, if it was lucky, closed the gripper at the right moment to lift the object and carry it until `open-gripper` was selected. While exploring the environment, the robot inevitably ran into obstacles or got wedged under a desk or chair, and stalled. As no human intervention was allowed during these experiments, the robot remained stalled until one of its randomly chosen actions set it free. Usually the robot freed itself by turning its wheels in the opposite direction, although in one interesting case, the Pioneer discovered that `open-gripper` (which opens the gripper paddles) sometimes affects the heading and stall sensors by pushing the robot away from a wall.

4 Results

The experiments described in the previous section yielded 600 observations, roughly 150 per action, on which to evaluate the CODT generation algorithm. We begin with qualitative evaluation and then present some quantitative summaries of the algorithm's performance. These analyses address several questions. Recall that the primary purpose of CODTs is not to predict the effects of actions but, rather, to specify which contextual information is *relevant* to these effects. Thus, the following analyses will focus primarily on how well CODTs identify relevant factors and secondarily on how well they predict the effects of actions. Of particular interest is the amount of training required to learn relevant factors, and whether CODTs identify the same relevant factors consistently in different trials despite the highly variable interactions between the robot and its environment.

4.1 Qualitative Analysis

It is instructive to look at some of the decision trees constructed by the CODT generation algorithm. For instance, Figure 2 shows the conditional effects of the `close-gripper` action on `heading`, based on 115 observations. One might expect this tree to be empty, for it is hard to imagine why closing the gripper would have any effect on the robot's heading. But imagine you are standing with your hand on a wall, bearing some of your weight. If you relax your arm and don't compensate, you will fall toward the wall. This is what happens when the robot is pushed against an obstacle and then it closes its gripper. It is one of four scenarios described in Figure 2. The root node of the tree tests rotational velocity. The middle branch, [-0.97...0.97] corresponds to very little rotational velocity. Below it is a test on whether the right wheel is stalled. If so, then closing the gripper will affect the robot's heading. This is the case we just described: the robot is hardly moving, in fact its right wheel is stalled, which happens when the robot is jammed against an obstacle, and in this situation, closing the gripper induces a small rotation and change in heading. On the other hand, if the robot isn't moving and isn't stalled then closing the gripper has no effect on heading; this is the leaf labelled '0' in Figure 2. The other branches of the tree represent the effects of residual rotational velocity. When the robot decides to close its gripper, it may still be turning. The '+' and '−' labels on the far left and far right of the tree represent these effects. In fact, "effects" is not the right word, because closing the gripper did not cause the robot to change heading in these cases: the robot was already changing heading when it closed its gripper.

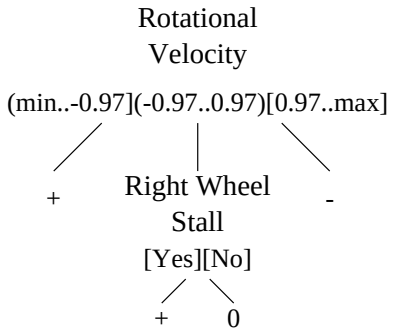

Figure 2: A CODT entry that represents the factors relevant to the effects of the `close-gripper` action on `heading`.

4.2 Quantitative Analyses

The tree in Figure 2 was based on 115 observations, and one wonders whether it is "stable," whether another set of observations would produce essentially the same tree or a different one. For our purposes (namely, providing knowledge about relevant factors), CODT trees for an action-sensor pair are stable if they contain the same factors, or decision nodes. For instance, suppose we collected other batches of observations and built trees for the conditional effects of `gripper` on `heading` (of which Fig. 2 is one). We would say these trees are stable if they all split on rotational velocity and right wheel stall. In other words, stability means that a set of factors is identified in the CODT as relevant to an effect of an action, and the elements of this set do not vary much. (Note that we are not asking whether trees for successive batches of observations have the same structure, only whether they have the same elements, because we are emphasizing not classification performance but the ability to identify relevant factors).

To test whether trees built by the CODT generation algorithm are stable, we ran an incremental cross-validation procedure on the observations [Coh95]. We have roughly 150 observations for each of the four primitive actions—`move(`x`)`, `turn(`x`)`, `gripper-up`, `gripper-down`. It is a simple matter to select a subset of these observations at random and build a CODT for that subset. The subset can be of any size, but for this analysis we used subsets of 50 and 115 observations. Repeating the process K times yields K CODTs, each generated from a different batch of observations. Now we can look at the K versions of a tree generated for a single cell of the CODT, say the tree for the effects of `gripper` on `heading`, and ask whether all K trees split on rotational velocity and right wheel stall as the tree in Figure 2 does. In fact, we calculate a slightly more informative statistic: The probability that a factor (e.g., right wheel stall) is included in a tree. This probability is just the number of trees in which the factor is included, divided by K. Sometimes, the CODT generation algorithm did not build a tree for an action-sensor pair. This non-tree still contributes to K. Thus, if only two trees are built given $K = 10$ opportunities, and a factor is included in just one of them, then its probability of occurring in a tree is 0.1. Finally, we can rank

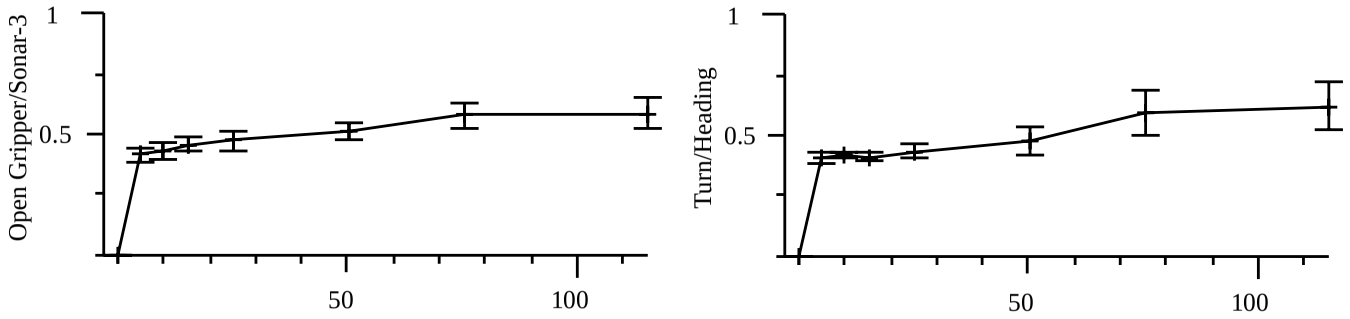

Figure 3: Two learning curves generated from the Pioneer CODT data. The curves plot the accuracy against training set size measured with 10-fold cross validation and 95% confidence intervals. The curve on the left represents the CODT entry for the effects of `open-gripper` on `sonar-3`, while the right hand curve is for `turn` on `heading`.

the factors by their probabilities. Trees will be considered stable if all their factors have high probability, because this means that each factor is likely to be included in each tree.

Table 1 shows the three highest-ranking factors for each action-sensor pair. The rows of the tables represent sensors (i.e., effects of actions) and the columns represent the actions. Every tree is build from a batch of either $N = 50$ or $N = 115$ observations, randomly sampled from the roughly 150 observations pertinent to each action. When $N = 50$, 20 batches were sampled, and the CODT algorithm had $K = 20$ opportunities to build a tree. When $N = 115$ only ten batches were sampled ($K = 10$). Consider, for instance, the effects of `turn`(x) on `heading` (roughly halfway down the first two columns of Table 1). For batches of 50 observations, the relevant factors are x, the magnitude of the turn, `left-wheel-stall` and `sonar-4`. For larger batches (115 observations) x and `left-wheel-stall` remain important factors (they show up in 100% and 90% of the trees, respectively) and `sonar-4` is no longer a relevant factor. Clearly, given enough training data, x and `left-wheel-stall` emerge as *stable relevant factors,* meaning that most of the trees for this action-sensor combination will include them.

In general, few factors are stable when trees are built from 50 observations. Notable exceptions are `status`, and `left-` and `right-wheel-stall`, all of which measure whether the robot is moving. Often, factors have low probabilities in the $N = 50$ cases because few trees were actually built in those cases. Thus, one often sees factors with low probabilities in the $N = 50$ conditions and the same factors wih higher probabilities in the $N = 115$ conditions. This tells us that these factors are identified as relevant, if not particularly stable, even with relatively few observations.

As one might expect, given the low quality of sonar data, these Factors rarely appeared in trees with any stability. For example, `sonar-1` is a relevant factor in just 20% of the trees built for the effects of `close-gripper` on `translational-velocity` ($N = 115$). On the other hand, several factors are relevant to the effects of actions on sonar readings. For instance, the state of the rear gripper beam (`gr-beam`) is relevant to the effect of `close-gripper` on the sensor `sonar-2` in 60% of the trees. This is probably because `gr-beam` is one way to assess whether the robot is jammed against an object that breaks the gripper beam, such as a table leg. Closing one's gripper around a table leg virtually assures that the sonar value will not change, because the robot cannot move.

In sum, factors become more stable as the batch size for observations increases, and few of the factors are *very* stable, excepting those that measure movement. We conclude that the CODT generation algorithm probably can build stable trees, but it needs more training data.

4.2.1 Learning Curves

While our concern with accuracy is only secondary, we do not pretend that classification accuracy in the CODT trees is unimportant. Consistent misclassification of effects must be attributed to noise in the environment or to trees that do not adequately capture relevant factors. We must live with noise, and strive for accurate trees.

Again using the incremental cross-validation procedure described in [Coh95], we measured the predictive accuracy of the CODT entries for each action-state pair as the size of the training set increased from 5 observations to 115. Two typical learning curves are shown in Figure 3. On the left is the curve for the effects of `open-gripper` on `sonar-3`, and on the right is the effects of `turn` on `heading`.

The CODT entries easily capture the majority class label (e.g., `turn` does not cause a change in translational velocity, thus the majority label is '0'), generally achieving baseline performance with only 5 observations. The curves are in general fairly flat thereafter, climbing typically no more than 10 or 20 percentage points, as with the curves of Figure 3. This, though, is as we would expect. The CODTs quickly detect the primary effect of the action, using the balance of the data to explain pathological situations, such as using the gripper to become unstuck from an obstacle.

5 Conclusions

This work is motivated by a theory of conceptual development that we are implementing on Pioneer 1 robots. Activities are central to the theory, as conceptual distinctions are thought to be functional—pertaining to what we do. Thus, a major challenge for the theory is to explain how robots can learn complex activities. Our approach is to divide this difficult problem into smaller, tractable learning tasks. Unfortunately, many attractive state-based planning and learning algorithms, such as reinforcement learning, suffer from the curse of dimensionality. Algorithms of this type are all but useless if state spaces are very large. If another learning mechanism could learn which factors are relevant to the effects of actions, then these algorithms could work in state spaces consisting of only the relevant factors instead of all

of the factors. The CODT algorithm effectively finds small subsets of factors that are relevant to the effects of actions.

Acknowledgements

This research is supported by DARPA contract N66001-96-C-8504, and by AFOSR contract F49620-97-1-0485. U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory or the U.S. Government. We wish to thank Agustin Schapira for his help running the experiments.

References

[COAB96] Paul R. Cohen, Tim Oates, Marc S. Atkin, and Carole R. Beal. Building a baby. In *Proceedings of the Eighteenth Annual Conference of the Cognitive Science Society*, pages 518–522, 1996.

[Coh95] Paul R. Cohen. *Empirical Methods for Artificial Intelligence.* The MIT Press, Cambridge, 1995.

[Joh87] Mark Johnson. *The Body in the Mind.* University of Chicago Press, 1987.

[JS97] David Jensen and Matthew D. Schmill. Adjusting for multiple comparisons in decision tree pruning. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 195–198, 1997.

[KJ89] S. Kotz and N.L. Johnson, editors. *Encyclopedia of Statistical Sciences.* Wiley, New York, 1982-1989.

[Lak84] George Lakoff. *Women, Fire, and Dangerous Things.* University of Chicago Press, 1984.

[Man92] Jean M. Mandler. How to build a baby: II. Conceptual primitives. *Psychological Review*, 99(4):587–604, 1992.

[NS63] Allen Newell and H.A. Simon. GPS: A program that simulates human thought. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw-Hill, New York, 1963.

	Turn		Move		Close Gripper		Open Gripper	
	50	115	50	115	50	115	50	115
SONAR-0	GRIP-STATE(5%) SONAR-0(5%) GF-BEAM(5%)	SONAR-0(50%) GRIP-STATE(20%) L-STALL(10%)	GRIP-STATE(5%) R-STALL(5%)		R-STALL(10%) STATUS(10%) TIME-SECS(5%)	R-STALL(80%) STATUS(80%) GF-BEAM(10%)	R-STALL(5%) GF-BEAM(5%)	R-STALL(20%) GF-BEAM(10%)
SONAR-1	STATUS(10%) L-STALL(5%) SONAR-0(5%)	SONAR-1(20%) TRANS-VEL(20%)	Y-POS(10%) SONAR-1(5%) SONAR-2(5%)	VIS-A-X(10%) SONAR-2(10%) VIS-C-X(10%)	SONAR-1(5%) Y-POS(5%) SONAR-2(5%)	SONAR-1(90%)		R-STALL(10%)
SONAR-2	SONAR-5(5%)	SONAR-2(40%) GRIP-STATE(10%)	X(15%) STATUS(10%) VIS-A-X(5%)	X(50%) L-STALL(10%) SONAR-2(10%)	SONAR-0(5%) SONAR-5(5%) SONAR-2(5%)	GR-BEAM(60%) HEADING(20%) SONAR-3(10%)	GRIP-STATE(5%) SONAR-0(5%) STATUS(5%)	R-STALL(40%) GRIP-STATE(30%) L-STALL(20%)
SONAR-3	SONAR-3(25%) R-STALL(5%) SONAR-6(5%)	SONAR-3(90%) VIS-A-X(10%) SONAR-4(10%)	GRIP-STATE(10%) Y-POS(5%) SONAR-2(5%)	VIS-C-X(20%) SONAR-3(20%) VIS-A-X(10%)	Y-POS(10%) VIS-A-X(5%) SONAR-5(5%)	SONAR-5(20%) ROT-VEL(20%) SONAR-1(10%)	SONAR-3(35%) R-STALL(10%) VIS-C-X(5%)	SONAR-3(100%) R-STALL(50%) GRIP-STATE(10%)
SONAR-4	SONAR-4(15%) TIME-SECS(5%) R-STALL(5%)	SONAR-4(100%)	GRIP-STATE(10%) R-STALL(5%) GR-BEAM(5%)	SONAR-4(20%) GRIP-STATE(10%) SONAR-0(10%)	SONAR-4(15%) TIME-SECS(5%) GRIP-STATE(5%)	SONAR-4(80%) Y-POS(10%) SONAR-5(10%)	R-STALL(20%) SONAR-4(5%) Y-POS(5%)	SONAR-4(70%) R-STALL(70%)
SONAR-5	SONAR-5(10%) L-STALL(5%)	SONAR-5(90%) L-STALL(30%)	SONAR-6(5%) HEADING(5%)			SONAR-4(10%) SONAR-5(10%)	SONAR-5(5%)	SONAR-5(20%)
SONAR-6	SONAR-6(5%)	SONAR-2(10%) R-STALL(10%)	Y-POS(5%) SONAR-5(5%) GF-BEAM(5%)	SONAR-6(70%) SONAR-3(10%)	GF-BEAM(5%)	SONAR-6(20%) SONAR-5(10%) GF-BEAM(10%)		VIS-A-X(10%) R-STALL(10%)
HEADING	X(30%) L-STALL(20%) SONAR-4(5%)	X(100%) L-STALL(90%) R-STALL(30%)	L-STALL(10%) TIME(10%) X(5%)	L-STALL(10%) VIS-C-X(10%) ROT-VEL(10%)	STATUS(55%) R-STALL(20%) L-STALL(10%)	R-STALL(60%) ROT-VEL(60%) L-STALL(30%)	STATUS(40%) R-STALL(10%) RW-VEL(10%)	R-STALL(60%) STATUS(40%) LW-VEL(30%)
RW-VEL	L-STALL(50%) GR-BEAM(30%) GF-BEAM(15%)	L-STALL(90%) X(80%) GF-BEAM(80%)	R-STALL(10%) VIS-A-X(5%) SONAR-6(5%)	R-STALL(50%) L-STALL(20%) GR-BEAM(20%)	STATUS(85%) TRANS-VEL(15%)	RW-VEL(60%) STATUS(40%)	TRANS-VEL(15%) STATUS(15%) RW-VEL(5%)	TRANS-VEL(80%) RW-VEL(10%) STATUS(10%)
LW-VEL	GF-BEAM(15%) GR-BEAM(15%) R-STALL(10%)	L-STALL(70%) GF-BEAM(70%) GR-BEAM(30%)	R-STALL(10%) GRIP-STATE(5%) VIS-C-X(5%)	GR-BEAM(80%)	STATUS(85%) RW-VEL(10%) R-STALL(5%)	RW-VEL(70%) STATUS(30%) X-POS(20%)	TRANS-VEL(10%) SONAR-2(5%) SONAR-3(5%)	TRANS-VEL(80%) SONAR-0(20%) RW-VEL(10%)
TRANS-VEL	SONAR-3(15%) STATUS(15%) Y-POS(5%)	STATUS(70%) TIME(30%) X-POS(20%)	L-STALL(20%) GR-BEAM(10%)	R-STALL(30%) L-STALL(20%) SONAR-0(20%)	STATUS(80%) Y-POS(5%) RW-VEL(5%)	RW-VEL(60%) STATUS(40%) SONAR-1(20%)	STATUS(20%) TRANS-VEL(10%) GRIP-STATE(5%)	TRANS-VEL(70%) L-STALL(10%) LW-VEL(10%)
ROT-VEL	R-STALL(20%) GF-BEAM(15%) STATUS(15%)	GF-BEAM(70%) L-STALL(60%) R-STALL(40%)		X-POS(20%) HEADING(10%)	Y-POS(5%) VIS-C-X(5%)	L-STALL(10%) X-POS(10%)	STATUS(35%) SONAR-4(5%) BATTERY-LEVEL(5%)	STATUS(80%) RW-VEL(20%)
R-STALL			GR-BEAM(10%)	X(30%) SONAR-1(10%) VIS-C-X(10%)	R-STALL(25%) GR-BEAM(15%) STATUS(5%)	R-STALL(100%) GRIP-STATE(10%) STATUS(10%)	R-STALL(10%) GR-BEAM(10%)	GR-BEAM(80%) R-STALL(70%) STATUS(10%)
L-STALL			GR-BEAM(10%) R-STALL(5%) STATUS(5%)	GR-BEAM(20%)	GR-BEAM(15%)	L-STALL(100%) TIME(30%) TIME-SECS(10%)	GR-BEAM(15%) L-STALL(10%)	L-STALL(80%) GR-BEAM(80%) GRIP-STATE(10%)
STATUS	L-STALL(15%) TIME-SECS(5%) TIME(5%)	GRIP-STATE(30%) L-STALL(30%) STATUS(20%)	SONAR-6(10%)	SONAR-6(10%)	R-STALL(45%) L-STALL(5%) VIS-B-X(5%)	R-STALL(100%) STATUS(60%)	STATUS(20%) R-STALL(10%) SONAR-2(5%)	STATUS(90%) R-STALL(60%) SONAR-1(10%)
GRIP-STATE	GRIP-STATE(5%)	GRIP-STATE(30%)		GRIP-STATE(10%)	GRIP-STATE(10%) TIME-SECS(5%) VIS-B-X(5%)	GRIP-STATE(40%)	GF-BEAM(5%)	
GF-BEAM			BUMPER(10%)	R-STALL(10%)				
GR-BEAM								
BUMPER			R-STALL(5%)		BUMPER(25%) GF-BEAM(10%)	BUMPER(100%) GF-BEAM(10%)	GRIP-STATE(30%) BUMPER(30%) GF-BEAM(15%)	GF-BEAM(60%) BUMPER(60%) GRIP-STATE(20%)

Table 1: The probabilities of seeing context variables in CODT entries, for trees built on 50 and 115 observations.