Solving Hybrid Markov Decision Processes

Alberto Reyes*[1], L. Enrique Sucar+, Eduardo F. Morales+ and Pablo H. Ibargüengoytia*

Instituto de Investigaciones Eléctricas*
Av. Reforma 113, Palmira
Cuernavaca, Mor., 62490, México
INAOE+
Luis Enrique Erro 1
Sta. Ma. Tonantzintla, Pue., México
{areyes,pibar@iie.org.mx}, {esucar,emorales}@inaoep.mx

Abstract. Markov decision processes (MDPs) have developed as a standard for representing uncertainty in decision-theoretic planning. However, MDPs require an explicit representation of the state space and the probabilistic transition model which, in continuous or hybrid continuous-discrete domains, are not always easy to define. Even when this representation is available, the size of the state space and the number of state variables to consider in the transition function may be such that the resulting MDP cannot be solved using traditional techniques. In this paper a reward-based abstraction for solving hybrid MDPs is presented. In the proposed method, we gather information about the rewards and the dynamics of the system by exploring the environment. This information is used to build a decision tree (C4.5) representing a small set of abstract states with equivalent rewards, and then is used to learn a probabilistic transition function using a Bayesian networks learning algorithm (K2). The system output is a problem specification ready for its solution with traditional dynamic programming algorithms. We have tested our abstract MDP model approximation in real-world problem domains. We present the results in terms of the models learned and their solutions for different configurations showing that our approach produces fast solutions with satisfying policies.

1 Introduction

A Markov Decision Process (MDP) [14] models a sequential decision problem, in which a system evolves in time and is controlled by an agent. The system dynamics is governed by a probabilistic transition function that maps states and actions to new states. At each time, an agent receives a reward that depends on the current state and the applied action. Thus, the main problem is to find a control strategy or *policy* that maximizes the expected reward over time. A common problem with the MDP formalism is that the state space grows exponentially

[1] Ph.D. Student at ITESM Campus Cuernavaca, Av. Reforma 182-A, Lomas, Cuernavaca, Mor., México

with the number of domain variables, and its inference methods iterate explicitly over the state and action spaces. Thus, in large problems, MDPs become impractical, inefficient and in many cases intractable.

Significant progress has been made on MDP problem specification through the use of factored representations [8]. These representations describe a system using only a small set of features (or factors) by exploiting the structure that many domains exhibit. In a factored MDP the transition model is represented as a dynamic Bayesian network (DBN) and the reward function as a decision tree. Factored MDPs have also been used in reinforcement learning contexts. For example, [10] presented an efficient and near-optimal algorithm for reinforcement learning in MDPs whose transition model was factored. In that work, they assumed that the graphical structure (but not the parameters) of the DBN was given by an expert. More recently in [17], a novel method for approximating the value function and selecting good actions for MDPs with large state and action spaces is described. In this method the model parameters can be learned efficiently because values and derivatives can be computed by a particular type of graphical model called *product of experts*.

Although factored representations can often be used to describe a problem compactly, they do not guarantee that a factored model can be solved effectively, particularly in continuous or highly dimensional domains. Abstraction and aggregation are techniques [2] that aid factored representations to avoid this problem. Several authors use these notions to find computationally feasible methods for the construction of (approximately) optimal and satisfying policies. For example, Dean and Givan [6], and Pineau et al [13] use the notions of abstraction and aggregation to group states that are similar with respect to certain problem characteristics to further reduce the complexity of the representation or the solution. Feng et al [9] proposes a state aggregation approach for exploiting structure in MDPs with continuous variables where the state space is dynamically partitioned into regions where the value function is the same throughout each region. The technique comes from POMDPs to represent and reason about linear surfaces effectively. Li and Littman [11] addresses hybrid state spaces including a comparison with the method of Feng et al.

Our approach is closely related to this work, however it differs on some aspects to offer simplicity in the abstraction construction, and an alternative to learn a complete MDP model from data. The proposed method is inspired on the qualitative change vectors used in [18, 16], which are particularly suitable for domains with continuous spaces. While other approaches [12, 3] start from a uniform grid over an exhaustive variable representation, we deduce an abstraction, called *qualitative states*, from the reward function structure. In our approach, a set of sampling data denoting the rewards and transitions in continuous terms are first collected to approximate the reward function with a tree learning algorithm (C4.5 [15]). Given a set of qualitative restrictions imposed by the reward function tree, the continuous information about the state transitions is transformed into abstract data that are processed by a Bayesian learning algorithm ($K2$ [4]) to produce a factored transition model. The resulting approximation

can be solved easily using standard dynamic programming algorithms. Since the abstraction is built over the factors related to the reward function, the method can work with both, pure continuous or hybrid continuous-discrete spaces. We have tested our abstract MDP model approximation with different configurations of a motion planning problem of different complexities. We present the results in terms of the models learned showing that our approach produces fast solutions with satisfying policies.

This paper is organized as follows: we start describing the standard MDP model and its factored representation. Section 3 develops the abstraction process. Then, we present our learning system and the experimental results. We finally conclude and give future directions of this work.

2 Markov Decision Processes

Formally, an MDP is a tuple $M =< S, A_s, \Phi, R >$, where S is a finite set of states $\{s_1, \ldots, s_n\}$. A_s is a finite set of actions for each state. $\Phi : A \times S \rightarrow \Pi(S)$ is the state transition function specified as a probability distribution. The probability of reaching state s' by performing action a in state s is written as $\Phi(a, s, s')$. $R : S \times A \rightarrow \Re$ is the reward function. $R(s, a)$ is the reward that the agent receives if it takes action a in state s.

A policy for an MDP is a mapping $\pi : S \rightarrow A$ that selects an action for each state. Given a policy, we can define its finite-horizon value function $V_n^\pi : S \rightarrow \Re$, where $V_n^\pi(s)$ is the expected value of applying the policy π for n steps starting in state s. The value function is defined inductively with $V_0^\pi(s) = R(s, \pi(s))$ and $V_m^\pi(s) = R(s, \pi(s)) + \Sigma_{s' \in S} \Phi(\pi(s), s, s') V_{m-1}^\pi(s')$. Over an infinite horizon, a discounted model is used to have a bounded expected value, where the parameter $0 \leq \gamma < 1$ is the *discount factor*, used to discount future rewards at a geometric rate. Thus, if $V^\pi(s)$ is the discounted expected value in state s following policy π forever, we must have $V^\pi(s) = R(s, \pi(s)) + \gamma \Sigma_{s' \in S} \Phi(\pi(s), s, s') V_{m-1}^\pi(s')$, which yields a set of linear equations in the values of $V^\pi()$.

A solution to an MDP is a policy that maximizes its expected value. For the discounted infinite–horizon case with any given discount factor $\gamma \in [0, 1)$, there is a policy V^* that is optimal regardless of the starting state that satisfies the *Bellman* equation [1]:

$$V^*(s) = max_a \{R(s, a) + \gamma \Sigma_{s' \in S} \Phi(a, s, s') V^*(s')\}$$

Two popular methods for solving this equation and finding an optimal policy for an MDP are: (a) value iteration and (b) policy iteration [14].

2.1 Factored MDPs

In a factored MDP, the set of states is described via a set of random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, where each X_i takes on values in some finite domain $Dom(X_i)$. A state $\mathbf{s}$ defines a value $x_i \in Dom(X_i)$ for each variable X_i. Thus, the set of states

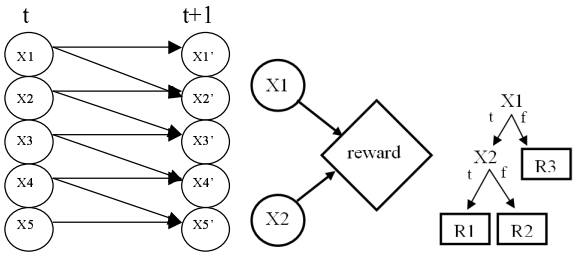

Fig. 1. A simple DBN with 5 state variables for one action (left). Influence Diagram denoting a reward function (center). Structured conditional reward (CR) represented as a binary decision tree (right)

$S = Dom(X_i)$ can be exponentially large, making it impractical to represent the transition model explicitly as matrices. Fortunately, the framework of dynamic Bayesian networks (DBN) [7, 5], and the decision trees gives us the tools to describe the transition model and the reward function concisely.

A Markovian transition model Φ defines a probability distribution over the next state given the current state and an action a. Let X_i denote the variable X_i at the current time and X_i' the variable at the next step. The *transition graph* of a DBN is a two–layer directed acyclic graph G_T whose nodes are $\{X_1, \ldots, X_n, X_1', \ldots, X_n'\}$. In the graph, the parents of X_i' are denoted as $Parents(X_i')$. Each node X_i' is associated with a *conditional probability distribution* (CPD) $P_\Phi(X_i' \mid Parents(X_i'))$, which is usually represented by a matrix (*conditional probability table*) or more compactly by a decision tree. The transition probability $\Phi(a, s_i, s_i')$ is then defined to be $\Pi_i P_\Phi(x_i' \mid \mathbf{u_i})$ where $\mathbf{u_i}$ is the value in $\mathbf{s}$ of the variables in $Parents(X_i')$.

Like an action's effect on a particular variable, the reward associated with a state often depends only on the values of certain features of the state. This reward or penalty is independent of other variables, and individual rewards can be associated with the groups of states that differ on the values of the relevant variables. The relationship between rewards and state variables is represented in value nodes in influence diagrams represented by the diamond in figure 1 (center). The conditional reward tables (CRT) for such a node is a table that associates a reward with every combination of values for its parents in the graph. This table is locally exponential in the number of relevant variables. Although in the worst case the CRT will take exponential space to store in many cases the reward function exhibits structure allowing it to be represented compactly using decision trees or graphs (as in figure 1 right).

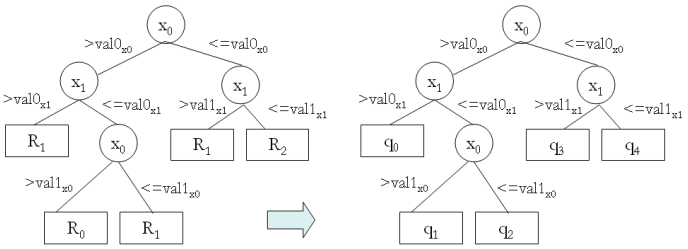

Fig. 2. Transformation of the reward decision tree (left) into a Q-tree (right). Nodes in the tree represent continuous variables and edges evaluate whether this variable is less or greater than a particular bound.

3 Hybrid MDPs

3.1 Qualitative states

Let us first define a qualitative state (or q–state) q_i as a set of continuous states that share similar immediate rewards. In consequence, a qualitative state space Q is a set of aggregated states $q_1, q_2, ..q_n$ that have this property in common. The qualitative state space can also be simply called as the *qualitative partition*.

Similarly to the reward function in a factored MDP, the qualitative partition Q is represented by a binary decision tree (Q–tree). The reward decision tree is then transformed into a Q-tree by simply renaming the reward values to q-state labels. Each leave in the Q–tree is labeled with a new qualitative state. Even for leaves with the same reward value, we assign a different qualitative state value. This produces more states but at the same time creates more guidance that helps to produce more adequate policies. States with similar reward are partitioned so each q–state is a continuous region. Figure 2 shows this tree transformation in a two dimensional domain.

Each branch in the Q–tree denotes a set of constraints for each q–state q_i that bounds a continuous region. The relational operators used in this approach split each continuous domain dimension in two portions. These operators are $<$ and $\geq$. For example, assuming that the immediate reward is a function of the linear position in a motion planning domain, a qualitative state could be a region in an $x - y$ coordinates system bounded by the constraints: $x \geq val(x_0)$ and $y \geq val(y_0)$, expressing that the current x coordinate is limited by the interval $[val(x_0), \infty]$, and the y coordinate by the interval $[val(y_0), \infty]$. Figure 3 illustrates the constraints associated to the example presented above, and its representation in a 2-dimensional space. It is evident that a qualitative state can cover a large number of states (if we consider a fine discretization) with similar properties.

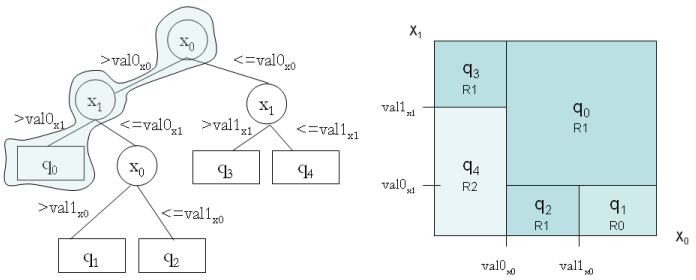

Fig. 3. In a Q-tree, branches are constraints and leaves are qualitative states (left). A graphical representation of the tree is also shown (right). Note that when an upper or lower variable bound is infinite, it must be understood as the upper or lower variable bound in the domain.

3.2 Hybrid MDP Model Specification

We can define a hybrid MDP as a factored MDP with a set of hybrid qualitative–discrete factors. The qualitative state space Q, is an additional factor that concentrates all the continuous variables. The idea is to substitute all these variables by this abstraction to reduce the dimensionality of the state space. Thus, a hybrid qualitative-discrete state is described in a factored form as $\mathbf{s_h} = \{X_1, \ldots, X_n, Q\}$, where $X_1, \ldots, X_n$ are the discrete factors, and Q is a factor the represents the relevant continuous dimensions in the reward function.

3.3 Learning Hybrid MDPs

The hybrid MDP model is learned from data based on a random exploration of a simulated environment with white Gaussian noise introduced on the actions outcomes of the step size. This noise was added to simulate probabilistic real effects of actions. We assume that the agent can explore the state space, and for each state–action can receive some immediate reward. Based on this random exploration, an initial partition, Q_0, of the continuous dimensions is obtained, and the reward function and transition functions are induced.

Given a set of state transition represented as a set of random variables, $O^j = \{\mathbf{X_t}, \mathbf{A}, \mathbf{X_{t+1}}\}$, for $j = 1, 2, ..., M$, for each state and action A executed by an agent, and a reward (or cost) R^j associated to each transition, we learn a qualitative factored MDP model:

1. From a set of examples $\{O, R\}$ obtain a reward decision tree, RDT, that predicts the reward function R in terms of continuous and discrete state variables, $X_1, \ldots, X_k, Q$.
2. Obtain from the decision tree, RDT, the set of constraints for the continuous variables relevant to determine the qualitative states (q–states) in the form of a Q-tree. In terms of the domain variables, we obtain a new variable Q

representing the reward-based qualitative state space whose values are the q–states.
3. Qualify data from the original sample in such a way that the new set of attributes are the Q variables, the remaining discrete state variables not included in the decision tree, and the action A. This transformed data set is called the qualified data set.
4. Format the qualified data set in such a way that the attributes follow a temporal causal ordering. For example variable Q_t must be set before Q_{t+1}, $X1_t$ before $X1_{t+1}$, and so on. The whole set of attributes should be the variable Q in time t, the remaining state variables in time t, the variable Q in time t+1, the remaining variables in time t+1, and the action A.
5. Prepare data for the induction of a 2-stage dynamic Bayesian net. According to the action space dimension, split the qualified data set into $|A|$ sets of samples for each action.
6. Induce the transition model for each action, A_j, using the K2 algorithm [4].

This initial model represents a high-level abstraction of the continuous state space and can be solved using a standard solution technique, such as value iteration. In some cases, our abstraction can miss some relevant details of the domain and consequently produced sub-optimal policies. This occurs particularly for domains in which the regions with rewards or punishments are very few or cover a low fraction of the state space. For these cases, we are currently developing a second phase which introduces additional partitions in abstract states with high variance with respect to their neighbors.

4 Experimental Results

We tested our approach in a robot navigation domain using a simulated environment. In this setting goals are represented as light-color square regions with positive immediate reward, and non-desirable regions as dark-color squares with negative reward. The remaining regions in the navigation area receive 0 reward (white). Experimentally, we express the size of a rewarded region (non zero reward) as a function of the navigation area. Rewarded regions are multivalued squares that can be distributed randomly over the navigation area. The number of these squares is also variable.

The robot sensor system included x-y position, angular orientation, and navigation bounds detection. In a set of experiments the possible noisy actions are discrete orthogonal movements to the right, left, up, and down. Figure 4 (left) shows an example of a navigation problem with 12 rewarded regions. The reward function in these cases have four possible values. The motion planning problem is to automatically obtain a satisfying policy for the robot to achieve its goals avoiding negative rewarded regions.

The abstraction was tested with several problems of different sizes and complexities, and compared to a fine discretization of the environment in terms of precision and complexity. The precision is evaluated by comparing the policies

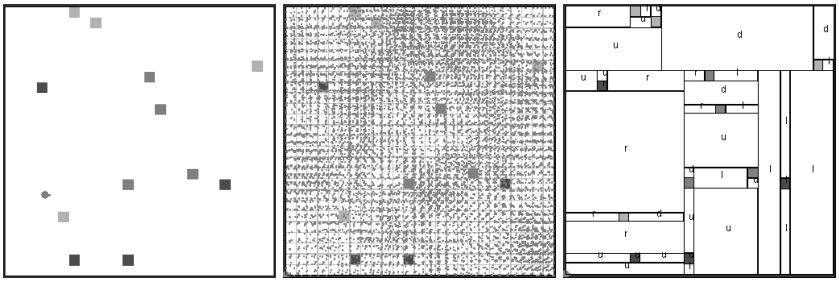

Fig. 4. Abstraction process. Left: Robot navigation area showing the distribution of goals (light color) and non-desirable zones (dark color). The simulated released robot is located at the left-bottom corner. Center: Exploration trace adding a 10% of Gaussian noise respecting to the action step. Right: initial qualitative states and their corresponding policies; u = up, d = down, r = right and l = left.

and values per state. The *policy precision* is obtained by comparing the policies generated with respect to the policy obtained from a fine discretization. In other words, we count the number of *fine* cells in which the policies are the same:

$$PP = (NEC/NTC) \times 100, \tag{1}$$

where PP is the policy precision in percentage, NEC is the number of fine cells with the same policy, and NTC is the total number of fine cells. This measure is pessimistic because in some states it is possible that more than one action have the same or similar value, and in this measure only one is considered correct. The utility error is calculated as follows: the utility values of all the states in each representation is first normalized. The sum of the absolute differences of the utility values of the corresponding states is evaluated and averaged over all the differences.

Figure 4 shows an example of one of the test cases. The left figure shows the motion planning problem. The center figure illustrates the exploration process. The right figure shows the qualitative states and their corresponding policies.

Table 1 presents a comparison between the behavior of seven problems solved with a simple discretization approach and our qualitative approach. Problems are identified with a number as shown in the first column. The first five columns describe the characteristics of each problem. For example, problem 1 (first row) has 2 reward cells with values different from zero that occupy 20% of the number of cells, the different number of reward values is 3 (e.g., -10, 0 and 10) and we generated 40,000 samples to build the MDP model. Table 2 presents a comparison between the qualitative and a fine representation. The columns describes the characteristics of the qualitative model in terms of utility error in % and policy precision.

As can be seen from Table 1, there is a significant reduction in the complexity of the problems using our abstraction approach. This can be clearly appreciated from the number of states and processing time required to solve the problems.

Table 1. Description of problems and comparison between a "normal" discretization and our qualitative discretization in terms of complexity and run time.

Problem					Discrete				Qualitative			
					Learning		Inference		Learning		Inference	
id	no. reward cells	reward size (% dim)	no. reward values	no. samples	no. states	time (ms)	no. itera-tions	time (ms)	no. states	time (ms)	no. itera-tions	time (ms)
1	2	20	3	40,000	25	7,671	120	20	8	2,634	120	20
2	4	20	5	40,000	25	1,763	123	20	13	2,423	122	20
3	10	10	3	40,000	100	4,026	120	80	26	2,503	120	20
4	6	5	3	40,000	400	5,418	120	1,602	24	4,527	120	40
5	10	5	5	28,868	400	3,595	128	2,774	29	2,203	127	60
6	12	5	4	29,250	400	7,351	124	7,921	46	2,163	124	30
7	14	3.3	9	50,000	900	9,223	117	16,784	60	4,296	117	241

Table 2. Comparative results between the abstraction and a fine discretization in terms of precision and errors.

	Qualitative	
id	Utility error (%)	Policy precision (%)
1	7.38	80
2	9.03	64
3	10.68	64
4	12.65	52
5	7.13	35
6	11.56	47.2
7	5.78	44.78

This is important since in complex domains where it can be difficult to define an adequate abstraction or solve the resulting MDP problem, one option is to create abstractions and hope for suboptimal policies. To evaluate the quality of the results Table 2 shows that the proposed abstraction produces on average only 9.17% error in the utility value when compared against the values obtained from the discretized problem.

5 Conclusions and Future Work

In this paper, a novel approach for solving continuous and hybrid MDPs is described. In the first phase we use an exploration strategy of the environment and a machine learning approach to induce an initial state abstraction. Our approach creates significant reductions in space and time allowing to solve quickly relatively large problems. The utility values on our abstracted representation are reasonably close (less than 13%) to those obtained using a fine discretization of the domain. Although tested on small solvable problems for comparison purposes, the approach can be applied to more complex domains where a simple discretization approach is not feasible. For space reasons, we did not include the partial models resulting of the learning process for the motion planning example. However they are available for clarifications.

As current research work we are including a refinement strategy of the abstraction to select a better segmentation of the abstract states and use alternative search strategies. We are also testing our approach in more sophisticated domains such as process control. The results of this research are oriented to built an intelligent assistant for training operators in power plants.

Acknowledgments This work was supported in part by IIE Project No. 12941 and CONACYT Project No. 47968.

References

1. R.E. Bellman. *Dynamic Programming*. Princeton U. Press, Princeton, N.J., 1957.
2. C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: structural assumptions and computational leverage. *Journal of AI Research*, 11:1–94, 1999.
3. Craig Boutilier, Moisés Goldszmidt, and Bikash Sabata. Continuous value function approximation for sequential bidding policies. In Kathryn Laskey and Henri Prade, editors, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*. Morgan Kaufmann Publishers, San Francisco, California, USA.
4. G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 1992.
5. A. Darwiche and Goldszmidt M. Action networks: A framework for reasoning about actions and change under understanding. In *Proceedings of the Tenth Conf. on Uncertainty in AI, UAI-94*, pages 136–144, Seattle, WA, USA, 1994.
6. T. Dean and R. Givan. Model minimization in Markov decision processes. In *Proc. of the 14th National Conf. on AI*, pages 106–111. AAAI, 1997.
7. T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.
8. R. Dearden and R. Boutillier. Abstraction and approximate decision-theoretic planning. *AI*, 89:219–283, 1997.
9. Z. Feng, R. Dearden, N. Meuleau, and R. Washington. Dynamic programming for structured continuous Markov decision problems. In *Proc. of the 20th Conf. on Uncertainty in AI (UAI-2004). Banff, Canada*, 2004.
10. M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. In *Proc. of the Sixteenth International Joint on Artificial Intelligence, IJCAI-99*, Stockolm, Sweden, 1999.
11. Lihong Li and Michael L. Littman. Lazy approximation for solving continuous finite-horizon MDPs. In *AAAI-05*, pages 1175–1180, Pittsburgh, PA, 2005.
12. Remi Munos and Andrew Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1348–1355. Morgan Kaufmann Publishers, San Francisco, California, USA, August 1999.
13. J. Pineau, G. Gordon, and S. Thrun. Policy-contingent abstraction for robust control. In *Proc. of the 19th Conf. on Uncertainty in AI, UAI-03*, pages 477–484, 2003.
14. M.L. Puterman. *Markov Decision Processes*. Wiley, New York, 1994.
15. J.R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Francisco, Calif., USA., 1993.

16. A. Reyes, L. E. Sucar, E. Morales, and P. H. Ibarguengoytia. Abstract MDPs using qualitative change predicates: An application in power generation. In *Planning under Uncertainty in Real-World Problems Workshop. Neural Information Processing Systems (NIPS-03)*, Vancouver CA, Winter 2003.
17. B. Sallans and G. E. Hinton. Reinforcement learning with factored states and actions. *Journal of Machine Learning and Research*, pages 1063–1088, 2004.
18. D. Suc and I. Bratko. Qualitative reverse engineering. In *Proc. of the 19th International Conf. on Machine Learning*, 2000.