

factors and plotted (after cubic smoothing) the average solution time (shown in Figure 1d). An interesting observation about this plot is that the problem instances where the two discount factors are equal (or close) appear to be the hardest (notice the contours in the γ_1 - γ_2 plane). This is counterintuitive, because such MDPs are equivalent to standard MDPs with one discount factor. A possible explanation might be that when discount factors are far apart, one of the reward functions dominates the other and the problem becomes simpler, while when the discount factors are close, the tradeoffs become more complicated (with the equivalence to a standard MDP hidden in the MILP translation). However, this is speculation and the phenomenon deserves a more careful analysis.

6 Discussion and Conclusions

We have presented algorithms for finding optimal deterministic policies for two classes of constrained MDPs, and in both cases we were maximizing a measure of the total expected discounted reward subject to constraints on the total expected discounted costs. However, our technique of finding optimal stationary deterministic policies via mixed integer programming also applies to other classes of MDPs.

In particular, the same methodology applies to MDPs with average per-time rewards and constraints (e.g., [Puterman, 1994]). Similarly to the constrained total-reward discounted MDP model described in Section 3, the MDP with average rewards and constraints can also be formulated as an LP (similar to (6)) that yields optimal stationary randomized policies. The problem of finding optimal stationary deterministic policies for such MDPs is also known to be NP-complete [Filar and Krass, 1994]. Our MILP reduction of Section 3 carries through with almost no changes and can thus be used to find optimal stationary deterministic policies for such MDPs.

The techniques of Section 4 also apply more generally. For instance, consider an MDP with general utility functions in the optimization criteria and constraints on the *probability* that the total cost exceeds some upper bound. This class of MDPs was discussed in [Dolgov and Durfee, 2004a], and the problem of finding approximately-optimal stationary randomized policies was reduced to a non-convex quadratic program in the space of the higher-order moments of the state visits. The non-convex quadratic constraints resulted from the requirement that the moments of different orders had to correspond to the same policy, and were almost identical to the quadratic constraints in (14) that synchronized the occupation measures for different discount factors. In fact, the two are so similar that our MILP reduction from Section 4 can be used to approximate optimal stationary deterministic policies for MDPs in [Dolgov and Durfee, 2004a].

To summarize, we have presented a general integer programming method for finding optimal stationary deterministic policies in constrained MDPs. We have demonstrated the method on two classes of MDPs: (i) a constrained discounted MDP and (ii) a constrained MDP with multiple discount factors. For (i), our methodology is of most value for domains where randomized policies (which work better and are easier to compute) are undesirable or difficult to implement because of an agent's architectural limitations. However, even in the

absence of such limitations, the approach is useful in situations where it is desirable to compare the quality of randomized and deterministic policies, such as when an agent is being designed for a particular task and it is necessary to weigh the cost of implementing a more complex policy-execution mechanism against the gain in expected performance. For problem (ii), to the best of our knowledge, no feasible algorithms have been reported for finding optimal solutions in any interesting policy class, and thus our MILP approach for finding optimal stationary deterministic policies provides the first practical approach to dealing with constrained MDPs with multiple discount factors.

References

- [Bellman, 1957] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [D'Epenoux, 1963] D'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10:98–108, 1963.
- [Dolgov and Durfee, 2004a] Dmitri A. Dolgov and Edmund H. Durfee. Approximate probabilistic constraints and risk-sensitive optimization criteria in Markov decision processes. In *Proc. of the 8th Int. Symposiums on AI and Math (AI&M 7-2004)*, 2004.
- [Dolgov and Durfee, 2004b] Dmitri A. Dolgov and Edmund H. Durfee. Optimal resource allocation and policy formulation in loosely-coupled Markov decision processes. In *Int. Conf. on Automated Planning and Scheduling*, pages 315–324, 2004.
- [Feinberg and Schwartz, 1994] E. A. Feinberg and A. Schwartz. Markov decision processes with weighted discounted criteria. *Math. of OR*, 19:152–168, 1994.
- [Feinberg and Schwartz, 1995] E. A. Feinberg and A. Schwartz. Constrained Markov decision processes with weighted discounted criteria. *Math. of OR*, 20:302–320, 1995.
- [Feinberg and Schwartz, 1996] E. Feinberg and A. Schwartz. Constrained discounted dynamic programming. *Math. of OR*, 21:922–945, 1996.
- [Feinberg and Schwartz, 1999] E. Feinberg and A. Schwartz. Constrained dynamic programming with two discount factors: Applications and an algorithm. *IEEE Transactions on Automatic Control*, pages 628–630, 1999.
- [Feinberg, 2000] Eugene A. Feinberg. Constrained discounted Markov decision processes and hamiltonian cycles. *Math. of Operations Research*, 25(1):130–140, 2000.
- [Filar and Krass, 1994] J. A. Filar and D. Krass. Hamiltonian cycles and Markov chains. *Math. of OR*, 19:223–237, 1994.
- [Heyman and Sobel, 1984] D. P. Heyman and M. J. Sobel. *Volume II: Stochastic Models in Operations Research*. McGraw-Hill, New York, 1984.
- [Kallenberg, 1983] L.C.M. Kallenberg. *Linear Programming and Finite Markovian Control Problems*. Math. Centrum, Amsterdam, 1983.
- [Lazar, 1983] A. Lazar. Optimal flow control of a class of queueing networks in equilibrium. *IEEE Transactions on Automatic Control*, 28(11):1001–1007, 1983.
- [Paruchuri et al., 2004] Praveen Paruchuri, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Towards a formalization of teamwork with resource constraints. In *Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 596–603, 2004.
- [Puterman, 1994] M. L. Puterman. *Markov Decision Processes*. John Wiley & Sons, New York, 1994.
- [Wolsey, 1998] L.A. Wolsey. *Integer Programming*. John Wiley & Sons, 1998.

Solving POMDPs with Continuous or Large Discrete Observation Spaces

Jesse Hoey

Department of Computer Science
University of Toronto
Toronto, ON, M5S 3H5
jhoey@cs.toronto.edu

Pascal Poupart

School of Computer Science
University of Waterloo
Waterloo, ON, N2L 3G1
ppoupart@cs.uwaterloo.ca

Abstract

We describe methods to solve partially observable Markov decision processes (POMDPs) with continuous or large discrete observation spaces. Realistic problems often have rich observation spaces, posing significant problems for standard POMDP algorithms that require explicit enumeration of the observations. This problem is usually approached by imposing an *a priori* discretisation on the observation space, which can be sub-optimal for the decision making task. However, since only those observations that would change the policy need to be distinguished, the decision problem itself *induces* a lossless partitioning of the observation space. This paper demonstrates how to find this partition while computing a policy, and how the resulting discretisation of the observation space reveals the relevant features of the application domain. The algorithms are demonstrated on a toy example and on a realistic assisted living task.

1 Introduction

Partially observable Markov decision processes (POMDPs) [1] provide a rich framework for planning under uncertainty. In particular, POMDPs can be used to robustly optimize the course of action of complex systems despite incomplete state information due to poor or noisy sensors. For instance, in mobile robotics [15], spoken-dialog systems [21; 25] and vision-based systems [7], POMDPs can be used to optimize controllers that rely on the partial and noisy information provided by various sensors such as sonars, laser-range finders, video cameras and microphones. Unfortunately, to date, the use of POMDPs in such real-world systems has been limited by the lack of scalable algorithms capable of processing rich and continuous sensor observations.

While *model-free* approaches such as neuro-dynamic programming [3], Monte Carlo sampling [23] and stochastic gradient descent [13; 2; 16] can tackle POMDPs with arbitrary observation spaces, they tend to require a large amount of simulation or a priori knowledge to restrict the space of policies (which reduces the need for simulation). Significant progress has also been made in developing approx-

imate scalable algorithms for *model-based* POMDPs with large state spaces [20; 19] and complex policy spaces [17; 19; 24], however model-based algorithms cannot tackle problems with continuous nor large discrete observation spaces.

In this paper we study and propose new algorithms for model-based POMDPs with continuous or large discrete observation spaces. We first demonstrate that the complexity of observation spaces can often be significantly reduced without affecting decision quality. Intuitively, observations provide information to the decision maker for choosing a future course of action. When the same course of action is chosen for two different observations, these observations are indistinguishable from a decision making point of view, and can therefore be aggregated. Hence, when a policy is composed of a small set of *conditional plans* (conditional on the observations), it is possible to partition the observation space in a small number of regions corresponding to the relevant features of the observation space for decision making. Many systems tackle the feature detection problem separately from the decision making problem, first building a set of features, then computing a policy based on those observation features. In this paper, we demonstrate how the decision problem can be used to automatically define a set of relevant features that are sufficient to find an optimal policy.

The paper first provides some background on POMDPs in Sect. 2, followed by a discussion of the partitioning of the observation space in Sect. 3. Sects. 4 and 5 discuss methods for the one-dimensional and multi-dimensional cases, respectively. Sect. 6 reports experiments with an assisted living task.

2 Partially Observable MDPs

Formally, a POMDP is specified by a tuple $\langle S, \mathcal{A}, \mathcal{Z}, T, Z, R, \gamma, h \rangle$ which consists of a set S of states s capturing the relevant features of the world, a set $\mathcal{A}$ of actions a corresponding to the possible control decisions, a set $\mathcal{Z}$ of observations corresponding to sensor readings, a transition function $T(s, a, s') = \Pr(s'|s, a)$ encoding the stochastic dynamics of the world, an observation function $Z(a, s', z) = \Pr(z|a, s')$ indicating how sensor readings relate to physical states of the world, a reward function $R(s, a)$ encoding the objectives of the system, a discount factor γ (between 0 and 1) and a planning horizon h (assumed to be infinite in this paper). We assume that states and actions are discrete, but observations can be continuous or discrete.

While it is common for observations to be continuous because sensors often provide continuous readings, states are often abstract, unobservable quantities. In the case of user modeling problems and event recognition problems, states are often discrete. For continuous observations, $Z(a, s', z) = \text{pdf}(z|a, s')$ is a probability density function. Since states are not directly observable, the decision maker's belief about the current state is represented by a probability distribution $b(s) = \Pr(s)$ called *belief state*. After executing a and observing z , the belief state b is revised according to Bayes' theorem: $b_z^a(s') \propto \sum_s b(s)T(s, a, s')Z(a, s', z)$.

To illustrate the concepts we will present, we use the classic Tiger problem [5], in which the decision maker is faced with two doors. A small reward lies behind one door, but a large cost (a tiger) lies behind the other. The decision maker can either open a door, or listen for the tiger. Listening gives the decision maker information from which she can infer the location of the tiger. In the original, discrete, version of this problem, two possible observations result from listening (*left* or *right*), and these observations correspond with the true location of the tiger with probability 0.85. In the continuous version we will discuss here, the decision maker has access to the original microphone array measurement that (noisily) locates the tiger in the continuous horizontal dimension (from the far left to the far right).

At each time step, the decision maker picks an action to execute based on the information gathered in past actions and observations. We can represent the decision maker's possible strategies by a set CP of conditional plans cp which correspond to decision trees. Fig. 1(a) shows the decision tree of a k -step conditional plan for the Tiger POMDP with discrete observations. Nodes are labeled by actions and edges are labeled by observations. The execution of a conditional plan starts at the root, performing the actions of the nodes traversed and following the edges labeled with the observations received from the environment. For example, the conditional plan in Fig. 1(a), will lead to opening a door if two successive observations confirm the same tiger location. We can define recursively a k -step conditional plan $cp_k = \langle a, os_{k-1} \rangle$ as a tuple consisting of an action a with an observation strategy $os_{k-1} : \mathcal{Z} \rightarrow \mathcal{CP}_{k-1}$ that maps observations to $(k-1)$ -step conditional plans. For POMDPs with continuous observations, conditional plans cp are decision trees with infinitely many branches and observation strategies os are continuous functions. The value function $\alpha_{cp}(b)$ of a conditional plan cp is the expected sum of discounted rewards that will be earned when starting in belief state b . This value function is often called an α -vector since it is linear with respect to the belief space and therefore parameterized by a vector $\alpha_{cp}(s)$ that has one component per state (i.e., $\alpha_{cp}(b) = \sum_s b(s)\alpha_{cp}(s)$). Fig. 1(b) shows the 5 α -vectors corresponding to the conditional plans shown in Fig. 1(a). The α -vectors corresponding to the conditional plans starting with an open door action (α_4 and α_5) have high value at one extreme of the belief space (when the certainty about the tiger location is high), but very low value at the other extreme.

A collection of conditional plans forms a policy π . The value function V^π of a policy π is the best value achieved by any of its conditional plans (i.e., $V^\pi(b) = \max_{cp \in \pi} \alpha_{cp}(b)$).

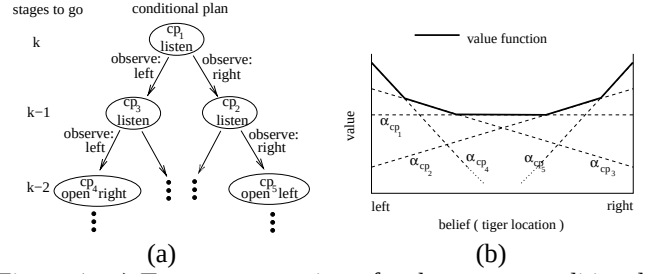


Figure 1: a) Tree representation of a three-step conditional plan for the simple tiger problem, starting with a uniform belief about the tiger location. b) Corresponding value function of composed of 5 α -vectors

Fig. 1(b) shows the value function of a policy for the Tiger problem, composed of 5 conditional plans, each of which is highest for some part of the belief space. The goal is to find an optimal policy π^* with the highest value (i.e., $V^{\pi^*}(b) \geq V^\pi(b) \forall \pi$). The optimal value function $V^*(b)$ for an infinite planning horizon can be computed by *value iteration* which computes successive approximations $V^k(b)$ by *dynamic programming*

$$V^{k+1}(b) = \max_a R^a(b) + \gamma \sum_z \Pr(z|b, a) V^k(b_z^a)$$

where $R^a(b) = \sum_s b(s)R(s, a)$, $\Pr(z|b, a) = \sum_{s, s'} b(s) \Pr(s'|s, a) \Pr(z|a, s')$ and b_z^a is the revised belief state after executing a and observing z . When V^k is formed by the set of α_{cp} , a new set of $\alpha_{cp'}$ for V^{k+1} can be constructed by *point-based dynamic programming backups*. A point-based backup computes the best conditional plan $cp'^* = \langle a^*, os^* \rangle$ (and corresponding α -vector) from a set of conditional plans cp for a given belief state b :

$$\alpha_{\langle a^*, os^* \rangle}(b) = R^{a^*}(b) + \gamma \sum_z \Pr(z|b, a^*) \alpha_{os^*(z)}(b_z^{a^*}) \quad (1)$$

$$\begin{aligned} \text{s.t.} \quad & os^*(z) = \operatorname{argmax}_{cp} \alpha_{cp}(b_z^{a^*}) \quad (2) \\ & a^* = \operatorname{argmax}_a R^a(b) + \gamma \sum_z \Pr(z|b, a) \alpha_{os^*(z)}(b_z^a) \quad (3) \end{aligned}$$

In practice, we cannot perform such point-based backups for every belief state since the belief space is continuous. However, as noted by Smallwood and Sondik [22], finite-horizon optimal value functions are composed by a finite number of α -vectors, which means that we only need to compute a finite number of point-based backups, one per α -vector to be constructed. Exact algorithms such as Linear Support [6] and Witness [12] perform this finite set of point-based backups at carefully chosen belief points. Alternatively, approximate algorithms such as Point-Based Value Iteration [27; 17; 24] heuristically sample a set of belief points at which they perform point-based backups.

3 Policy-directed Observation Aggregation

In a decision process, observations provide information to the decision maker for deciding the future course of action. When the observation space is rich (i.e., continuous observations or many discrete observations), the decision maker can devise

rich policies with a different course of action for each possible observation. However, for many POMDPs, there often exists good policies that are quite simple. These policies tend to select the same course of action for many different observations that share similar features. For POMDPs with continuous observations, this will allow us to implicitly discretize the observation space without introducing any error. From the point of view of the application domain, this also gives us insights regarding the relevant features of the observation space. In this section we discuss how simple policies allow us to aggregate many observations, effectively reducing the complexity of the observation space.

Recall that a conditional plan $cp = \langle a, os \rangle$ is a tuple consisting of an action a and an observation strategy os . The observation strategy $os(z) = cp'$ indicates for each observation z the conditional plan cp' encoding the future course of action. Intuitively, all the observations that select the same conditional plan are indistinguishable and can be aggregated. We can therefore view observation strategies as partitioning the observation space into regions mapped to the same conditional plan.¹ In the continuous observation tiger problem, for example, as long as a sound is heard coming “from the left”, the best choice of action may be to open the right door. Although the precise location of the sound here is not important, the decision boundary is (e.g. how far right can the sound be heard before the decision maker would listen again).

In each point-based backup, we compute an observation strategy os which partitions the observation space into regions Z_{cp} that select the same conditional plan cp . Let's examine how these regions arise. Recall from Eq. 2 that for each observation z , the conditional plan selected is the one that maximizes $\alpha_{cp}(b_z^a)$. Hence, we define $Z_{cp}^* = \{z | cp^* = \operatorname{argmax}_{cp} \alpha_{cp}(b_z^a)\}$ to be the set of observations for which cp^* is the best conditional plan to execute in b_z^a .

For each region Z_{cp} , we can compute the aggregate probability $\Pr(Z_{cp}|a, s')$ that any observation $z \in Z_{cp}$ will be made if action a is taken and state s' is reached by integrating $pdf(z|a, s')$ over region Z_{cp} (i.e., $\Pr(Z_{cp}|a, s') = \int_{z \in Z_{cp}} pdf(z|a, s') dz$).² The aggregate probabilities can be used to perform point-based dynamic programming backups

$$\alpha_{\langle a^*, os^* \rangle}(b) = R^{a^*}(b) + \gamma \sum_{cp} \Pr(Z_{cp}|b, a^*) \alpha_{cp}(b_{Z_{cp}}^{a^*}) \quad (4)$$

with $\Pr(Z_{cp}|b, a^*) = \sum_{s, s'} b(s) \Pr(s'|s, a^*) \Pr(Z_{cp}|a^*, s')$ and $b_{Z_{cp}}^{a^*}(s') \propto \sum_s b(s) \Pr(s'|s, a^*) \Pr(Z_{cp}|a^*, s')$. This is equivalent to Eq. 1, except that we have replaced the sum over observations z with a sum over regions Z_{cp} , in each of which a particular conditional plan is dominant.

4 One-Dimensional Observation Space

We now discuss how to find the implicit discretization of the observation space induced by a set of conditional plans (or α -vectors) when the observation space is one-dimensional continuous. For this special case, the regions, Z_{cp} , over which

¹Note that the observations that are mapped to the same conditional plan may not form a contiguous region though.

²If Z_{cp} is not a contiguous region, then several integrals must be computed, one for each contiguous sub-region.

observations can be aggregated are segments of a line corresponding to a range of observations for which the same conditional plan is optimal. Segment boundaries are observations for which there are two (or more) conditional plans yielding the highest $\alpha_{cp}(b_z^a)$. To make clear that z is the only variable here, define $\beta_{cp}^{b,a}(z)$ to be a function in z that corresponds to $\alpha_{cp}(b_z^a)$ where b and a are fixed (i.e., $\beta_{cp}^{b,a}(z) \equiv \alpha_{cp}(b_z^a)$). We can find these boundaries by solving $\beta_{cp_i}^{b,a}(z) - \beta_{cp_j}^{b,a}(z) = 0$ for every pair cp_i, cp_j of conditional plans.³ Analytically solving this equation will be difficult in general, and is not possible for observation functions in the exponential family (e.g. Gaussians). However, efficient numerical solutions can be used for many well-behaved functions. We used the Mathematica function `IntervalRoots`, that finds all the roots of an arbitrary one-dimensional function in a given interval by interval bisection combined with gradient-based methods. Once all potential regions are identified, the aggregate probabilities $P(Z_{cp}|b, a)$ can be computed by exact integration (or Monte Carlo approximation) for each conditional plan cp .

Consider again the continuous Tiger problem introduced in Sect. 2. We now illustrate how to find the observation regions induced by a set of conditional plans and how to use them in a point-based backup. Suppose that the doors are located at $z = 1$ and $z = -1$, and the decision maker is at $z = 0$. Due to a lack of accuracy, the binary microphone array reports the tiger's location corrupted by some zero-mean, normally distributed noise of standard deviation $\sigma = 0.965$. Listening costs 1, while meeting the tiger costs 100, but opening the correct door is rewarded with 10. The discount is $\gamma = 0.75$.

Suppose we have 3 conditional plans with corresponding α -vectors shown in Fig. 2(b) and we would like to perform a point-based backup. When considering belief state $b(\text{tiger_location} = \text{left}) = 0.85$ and action $a = \text{listen}$, Fig. 2(a) shows the β functions of each conditional plan. Since the observation function is Gaussian, the β -function is a linear combination of Gaussian distributions. By finding the roots of $\beta_{cp_1}^{b,a}(z) - \beta_{cp_2}^{b,a}(z)$ and $\beta_{cp_1}^{b,a}(z) - \beta_{cp_3}^{b,a}(z)$, we obtain the boundaries $z = 0.28$ and $z = 1.33$ delimiting the observation regions Z_{cp_1} , Z_{cp_2} and Z_{cp_3} .

We can thus form a discrete observation function $\Pr(Z_{cp}|s, a)$ by integrating the original Gaussian observation distributions over each region. We analytically integrate each Gaussian over each region using the complementary error function erfc .⁴ Fig. 2(c) shows the two Gaussian observation distributions, and the aggregate observation probabilities for each region. Using Eq. 4, we can then compute the value of the conditional plan $cp' = \langle a, \sigma \rangle$ where $a = \text{listen}$ and $\sigma(z) = cp_i$ if $z \in Z_{cp_i}$.

In contrast, the original discrete version of the tiger problem partitions the observation space in two halves at $z = 0$, resulting in a discrete, binary observation function with $P(\text{observe} = \text{right} | \text{tiger_location} = \text{right}) = \frac{1}{2} \operatorname{erfc}(\frac{-1}{\sqrt{2}\sigma})$. A dynamic partition induced by the current set of conditional

³This method may find more boundaries than are necessary, for a third β_{cp_k} may have higher value than both β_{cp_i} and β_{cp_j} at their intersection points.

⁴ $\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$.

plans has the advantage that no information is lost. Fig. 2(d) compares the value of the policies obtained with our dynamic discretization scheme and the naive binary discretization as we vary the variance σ^2 . The dynamic discretization outperforms the fixed binary discretization for all variances. The solutions are the same for almost perfectly observable cases ($\sigma \leq 0.1$) and approach one another for almost unobservable cases ($\sigma \rightarrow \infty$).

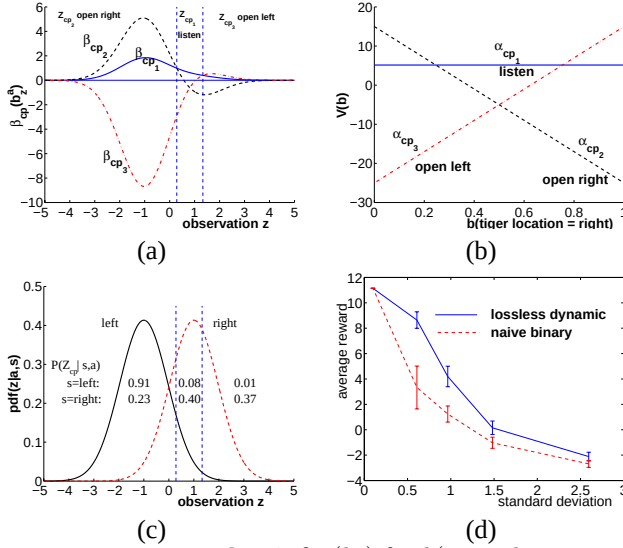


Figure 2: Tiger example. a) $\beta_{cp}(b_z^a)$ for $b(\text{tiger location} = \text{left}) = 0.85$ and $a = \text{listen}$, showing regions over which each conditional plan will be selected. b) α -vectors of 3 conditional plans. c) Observation function and aggregate probabilities of each observation region for each state. d) Average discounted reward achieved over 10 trials of 100 simulated runs of 50 steps each, for different observation variances in the continuous 1D tiger problem.

5 Multi-dimensional observation space

In many applications, observations are composed of several sensor readings. For example, a mobile robot may have a number of sonar, laser or tactile sensors, which together give rise to multi-dimensional observations. When the observation space is multi-dimensional, analytically finding the regions and computing their aggregate probability will be difficult for many observation functions.

We examine two approaches. The first, discussed in Sect. 5.1, considers the special case of POMDPs with observations composed of conditionally independent variables. For this special case, it is possible to define an equivalent POMDP where the observation variables are processed sequentially in isolation, essentially reducing the observation space to one dimension. For arbitrary multi-dimensional observations, we present a general sampling-based technique in Sect. 5.2.

5.1 Conditionally Independent Observations

In some applications, the random variables corresponding to the individual sensor measurements may be conditionally independent given the last action executed and the state of

the world. In this case, it is possible to factor the observation function into a product of small observation functions, one for each random variable (i.e., $\Pr(z_1, z_2|a, s') = \Pr(z_1|a, s') \Pr(z_2|a, s')$). For example, consider a mobile robot with sonars pointing forwards and to the side. The readings from each sonar are conditionally independent given the location of the robot and a map of the robot's environment.

This factorization can be exploited to process the observations sequentially. For n conditionally independent observation variables, we divide each time step into n sub-steps such that only one observation variable is observed per sub-step. This can be easily accomplished by constructing a POMDP with an additional state variable, *sub_step*, that keeps track of the current sub-step. The observation function encodes the probabilities of a single, but different, observation variable at each sub-step. The transition function is the same as in the original POMDP when *sub_step*=1, but becomes the identity function otherwise. The rewards are gathered and the discount factor applied only when *sub_step*=1.

When all observation variables are conditionally independent, this effectively reduces the dimensionality of continuous observations to one, allowing the approach of Sect. 4 to be used. For discrete observation variables, an exponential reduction is also achieved since the domain of a single variable is exponentially smaller than the cross-domain of several variables. Note however that the complexity of the equivalent POMDP remains the same since the reduction is achieved by multiplying the horizon and the number of states by n .

5.2 Sampling

For arbitrary multi-dimensional observations, an effective approximation technique for computing the aggregate probabilities consists of sampling. Recall from Sect. 3 that for each conditional plan cp , we can aggregate in one region Z_{cp} all the observations z for which cp yields the highest value (i.e., $\beta_{cp}^{b,a}(z) \geq \beta_{cp'}^{b,a}(z) \forall cp' \in \mathcal{CP}$). Hence, for each $\langle a, s' \rangle$ -pair, we sample k observations from $\text{pdf}(z|a, s')$ and set $\Pr(Z_{cp}|a, s')$ to the fraction of observations for which $\beta_{cp}^{b,a}(z)$ is maximal, breaking ties by favoring the conditional plan with the lowest index.

This sampling technique allows us to build an approximate discrete observation function $\Pr(Z_{cp}|a, s')$ which can be used to perform a point-based backup for a set $\mathcal{CP}$ of conditional plans and a belief state b . The number of observations sampled is $k|\mathcal{S}||\mathcal{A}|$ for each point-based backup. The quality of the approximation improves with k . In particular, using Hoeffding's bound [9], we can guarantee that $\Pr(Z_{cp}|a, s')$ has an error of at most ϵ with probability $1 - \delta$ when $k = \ln(2|\mathcal{CP}|/\delta)/(\epsilon^2)$. Interestingly, k doesn't depend on the dimensionality (nor any other complexity measure) of the observation space. It depends only on the number of regions, which is at most the number of conditional plans. While this is true in a single DP backup, the number of conditional plans may increase exponentially (in the worst case) with the number of observations at each DP backup [12]. On the other hand, several algorithms [18; 8; 17; 24] can mitigate policy complexity by searching for good yet small policies represented by a bounded number of condi-

tional plans or α -vectors. Perseus [24], a randomized point-based value iteration algorithm, is such an algorithm since the number of α -vectors is bounded by the number of belief points initially sampled. Hence k depends on policy complexity, which generally depends on observation complexity, but can be made independent by restricting policies to have a bounded representation.

This sampling technique can also be used for POMDPs with many discrete observations. In particular, when the observations are factored into several random variables, the number of observations is exponential in the number of variables, but as long as the number of conditional plans remains small, the number of samples will also be relatively small.

Note also that we can often weaken the dependency between the number of samples and the size of the action and state spaces. Instead of sampling k observations from each of the $|\mathcal{A}||\mathcal{S}|$ densities $pdf(z|a, s')$, we can sample j observations from one proposal distribution $p(z)$. This sample of j observations can be used to approximate each $\Pr(\mathcal{Z}_{cp}|a, s')$ as follows. First, we assign a weight $pdf(z|a, s')/p(z)$ to each sampled observation z to obtain an unbiased sample of $pdf(z|a, s')$. Then, for each conditional plan cp , we find the subset of sampled observations z for which $\beta_{cp}^{b,a}(z)$ is maximal, and set $\Pr(\mathcal{Z}_{cp}|a, s')$ to the (normalized) sum of the weights of the observations in that subset. The number of sampled observations j necessary to guarantee an error of at most ϵ with probability $1 - \delta$ depends on how similar the proposal distribution $p(z)$ is with each density $pdf(z|a, s')$. When the proposal is relatively similar to each of the densities then j tends to be close to k , significantly reducing the dependencies on $|\mathcal{A}|$ and $|\mathcal{S}|$. However, as the differences between the proposal and each of the densities increase, j also increases and may become arbitrarily large. In Sect. 6, we use $pdf(z|b, a)$ as a proposal distribution.

6 Experiments

This section presents experiments with a POMDP that assists people with cognitive difficulties to complete activities of daily living. Focusing on the task of handwashing, we present a simplified POMDP for guiding patients with verbal prompts as they wash their hands. The goal of such a system is to minimize the human caregiver burden, and is part of an ongoing research initiative applying intelligent reasoning to assistive living [4]. In this paper, we present results from simulations of our methods on a simplified POMDP model for the handwashing task. Fig. 3(a) shows the graphical model of the handwashing POMDP. The POMDP's actions are the verbal prompts (corresponding to the canonical steps of handwashing: *turn on water*, *wet hands*, *use soap*, *dry hands* and *turn off water*) and a *null* action where the system waits. The states are defined by the variables *hands_state*, which can be {*dirty*, *soapy*, *clean*}, *hand_location*, which can be {*away*, *tap*, *water*, *soap*, *towel*}, *hands_wet*, which can be {*wet*, *dry*}, and *water*, which can be {*on*, *off*}. We assume the hands start *dirty* and *dry*, and the goal is to get them *clean* and *dry*, which can only happen if they become *soapy* and *wet* at some intermediate time. The water starts *off* and must be *off* for task completion. The cost of a prompt is 0.2, and a large reward

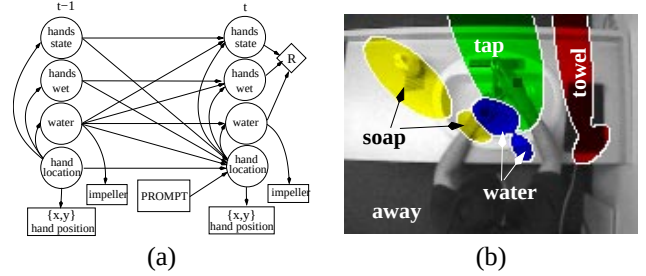


Figure 3: a) POMDP model of handwashing. b) Example image showing the regions induced by the observation function alone, $\max_i P(x, y|hand_location = i)$

(20) is given when the hands are dry and clean and the water is off. Smaller rewards are given for progressions through the task: if the hands are clean (1), soapy (0.5) and wet (0.2), but other variables are not as in the final goal state. The discount was $\gamma = 0.85$.

We model the system as being equipped with an impeller in the water pipe, which returns 0 when there is no water flowing and 1 when the water is on full. The sensor's noise is zero-mean Gaussian with standard deviation $\sigma = 0.4825$, which gives it an 85% accuracy (when the most likely state of *water* is considered). The position of the hands in the horizontal plane is measured using a camera mounted in the ceiling above the sink, connected to a computer vision system that returns an estimate of the $\{x, y\}$ position of the patient's dominant hand in the image using skin color [14]. Fig. 3(b) shows an example image from the camera. The observation function that relates these measured hand positions to the actual *hand_location* was learned from a set of data taken from the computer vision system. An actor simulated the repeated handwashing trials for about 10 minutes. The vision system tracked and reported the $\{x, y\}$ position of the right (dominant) hand, while a researcher annotated the data with the actual *hand_location*.⁵ The functions $P(x, y|hand_location = i)$ were then learned by fitting a mixture of Gaussians to the data annotated with the value *hand_location* = i .⁶ The mixture models were fit using a K-means initialization followed by the expectation-maximization algorithm. Figure 3(b) shows the most likely *hand_locations* for each $\{x, y\}$ position induced by the learned mixtures of Gaussians. The water flow observation function was not learned from data.

The water flow and hand position readings yield a 3D observation space. Although water flow and hand positions are conditionally independent, the $\{x, y\}$ coordinates of the measured hand positions are dependent. Hence, we cannot process the observations sequentially as suggested in Sect. 5.1 and must resort to the sampling technique of Sect. 5.2. We extended Perseus [24] with the sampling technique described

⁵The vision system only reports the hand position every 2 seconds, or when the hand location changes and is located consistently for 5 frames (1 second).

⁶The model orders were selected by minimizing the minimum description length (MDL): $-\log(P(z|hand_location)) + \frac{1}{2}M\log N$, where M is the number of parameters and N is the number of data points. This yielded between 2 mixture components for each state. The *away* state was fixed to have 1 mixture component.

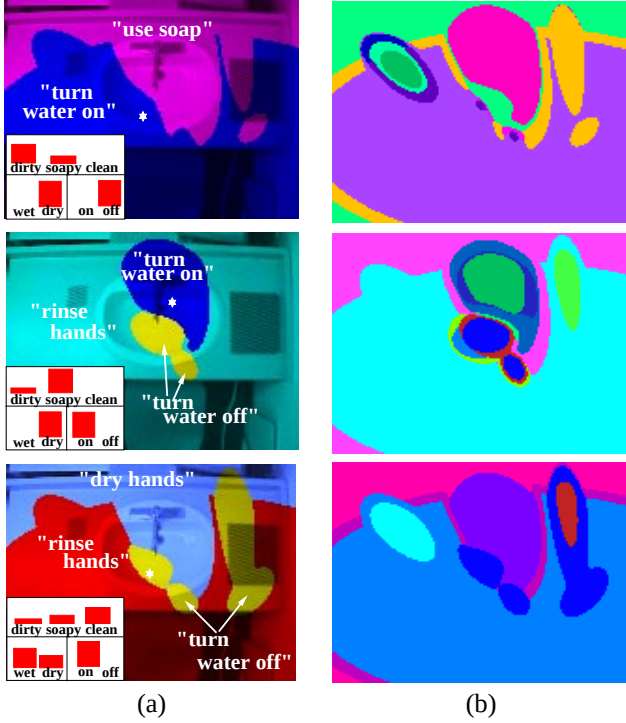


Figure 4: Example regions during a simulated trial at three stages (from top to bottom, after 1, 3 and 4 actions): (a) shows the best actions to take, the current belief state, and the current location of the hand (star); (b) shows the regions Z_{cp} , each different shade corresponding to a different conditional plan that will be followed if an observation is made in the region.

in Sect. 5.2 to compute an approximate discrete observation function at each point-based backup. More precisely, 100 observations were sampled before each point-based backup to approximate the aggregate probabilities of each observation region induced by some conditional plan. We ran the algorithm for 50 iterations with 227 sampled belief states. For comparison purposes, we also ran the original Perseus algorithm with a fixed discrete observation function obtained by partitioning the plane of hand positions according to the regions shown in Fig. 3(b) and by partitioning water flow readings in two regions at 0.5. This is a natural discretization that arises from the observation function by considering the regions where each $pdf(x, y | hand_location)$ and $pdf(impeller | water_flow)$ are highest. The computed policies were then simulated, and actions were selected by monitoring the belief state and computing the best conditional plan $\arg \max_{cp} \sum_s b(s) \alpha_{cp^*}(s)$. The discounted rewards (averaged over 10 trials of 100 simulated runs of 50 steps each) of the policies obtained by dynamic partitioning are 13.7 ± 1.4 , while those for fixed partitioning are 4.7 ± 0.3 , showing that our sample-based dynamic partitioning technique outperforms the fixed discretization technique. The final conditional plans were represented with 64 α vectors for our dynamic algorithm, and 77 α vectors for the fixed discretization.

Figure 4 shows examples of the dynamic partitions found by our technique at three stages during a simulated trial. Since we cannot show 3D partitions, we show the 2D partitions of the $\{x, y\}$ plane (ignoring the water flow sensor). At

the beginning of the trial (top row, stage 1), the system will either prompt to use the soap or turn the water on, based on where it sees the hands. At this stage, the regions Z_{cp} mainly distinguish the areas surrounding the soap and the taps, since these are the usual first steps in handwashing. Once the hands are believed to be soapy and the water on (stage 3, middle row), the system will prompt to rinse the hands, unless the patient has rinsed their hands or has turned the water off, in which case the prompt will be to turn the water off or on, respectively. We see in Figure 4(b) (middle row) that there are many regions now in the areas near the tap or under the water. This is the most uncertain area for this system (see Figure 3(b)), calling for many different conditional plans. Finally, at stage 4 (bottom row), the system believes the hands are clean, and will prompt the user to dry their hands or turn the water off. In this case, fewer possibilities remain, and so there are fewer regions in Figure 4(b) (bottom row).

7 Conclusion

Exploiting the fact that observations are useful only to the extent where they lead to different courses of actions, the paper describes how to dynamically partition observation spaces without any loss of information based on the current policy. For policies with a small number of conditional plans, observations can be aggregated in a small number of regions corresponding to the relevant observation features of the application domain. The region-based observation function can generally be constructed by numerical root-finding and integration algorithms for uni-dimensional observations or multi-dimensional observations composed of conditionally independent variables. For general multi-dimensional observations a general sampling technique was also described and demonstrated on a realistic assisted living task.

Note that the dynamic partitioning technique proposed in this paper is tightly integrated with point-based backups. More precisely, a lossless dynamic partition of the observation space can be computed only with respect to a given belief state and a set of α -vectors. As a result, our technique cannot be integrated with algorithms that do not use point-based backups (e.g., Incremental Pruning [26], Bounded Policy Iteration [18]). Furthermore, it cannot be integrated with the linear programs that find belief points prior to point-based backups in the Witness algorithm [12]. At the moment, full integration is only possible with Linear Support [6], PBVI [17], and Perseus [24] since these algorithms make use of the observation function only in point-based backups. Dynamic lossless observation partitioning for a broader range of algorithms is a possible direction for future research.

This paper tackles POMDPs with continuous observations, but discrete states. As mentioned earlier, such POMDPs are common in user modeling, event recognition and spoken-dialog systems, since the observations correspond to continuous sensor readings and the states are abstract discrete features. Note also that our dynamic partitioning technique doesn't require the state space to be discrete. In fact, Porta et. al [11] recently proposed an extension to Perseus that can handle continuous state spaces. Point-based backups are performed in a similar fashion, but given the continuous nature

of the state space, α -functions are generated instead of α -vectors. Integrating our dynamic partitioning technique with such continuous point-based backups should be possible and is subject to future research.

Our current work in using POMDPs for assistive living tasks involves learning model structure and user behaviors from sequence data, rather than imposing our own structure on tasks. The POMDP models we learn have observation functions which are themselves dynamic Bayesian networks (DBNs) with video frame observations at each time step, leading to a hierarchical model. Preliminary work along these lines is reported in [10]. We wish to use the techniques we have described in this paper both to solve these POMDPs and to learn models of human behaviors from video sequence data. Another potential research direction includes the exploration of automated feature detection in application domains such as image processing and speech recognition by policy-directed observation aggregation.

Acknowledgements

The authors wish to thank Alex Mihailidis for help with the handwashing data, Darius Braziumas for pointing out some inconsistencies in some early experiments, Nikos Vlassis for helpful comments and Jason Williams for early discussions. The first author gratefully acknowledges the support of Intel Corporation and the American Alzheimer Association.

References

- [1] K. J. Åström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.
- [2] D. Aberdeen and J. Baxter. Scaling internal-state policy-gradient methods for POMDPs. In *ICML*, pages 3–10, Sydney, Australia, 2002.
- [3] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, Belmont, MA, 1996.
- [4] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and Alex Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proc. IJCAI*, Edinburgh, 2005.
- [5] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *AAAI*, Seattle, WA, 1994.
- [6] H.-T. Cheng. *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, Vancouver, 1988.
- [7] T. Darrell and A. P. Pentland. Active gesture recognition using partially observable Markov decision processes. In *IEEE Intl. Conf. on Pattern Recognition*, Vienna, Austria, 1996.
- [8] Z. Feng and E. Hansen. Approximate planning for factored POMDPs. In *Proc. ECP*, Toledo, Spain, 2001.
- [9] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
- [10] Jesse Hoey, Pascal Poupart, Craig Boutilier, and Alex Mihailidis. Semi-supervised learning of patient-caregiver interactions using partially observable Markov decision processes. Working paper, 2005.
- [11] M. Spaan J. Porta and N. Vlassis. Value iteration for continuous-state POMDPs. Technical Report IAS-UVA-04-04, Informatics Institute, University of Amsterdam, 2004.
- [12] Leslie Pack Kaelbling, Michael Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.
- [13] N. Meuleau, L. Peshkin, K.-E. Kim, and L. P. Kaelbling. Learning finite-state controllers for partially observable environments. In *UAI*, pages 427–436, Stockholm, 1999.
- [14] A. Mihailidis, B. Carmichael, and J. Boger. The use of computer vision in an intelligent environment to support aging-in-place, safety, and independence in the home. *IEEE Trans. on Information Technology in Biomedicine (Spec. Issue on Pervasive Healthcare)*, 8(3):1–11, 2004.
- [15] M. Montemerlo, J. Pineau, N. Roy, S. Thrun, and V. Verma. Experiences with a mobile robotic guide for the elderly. In *AAAI*, pages 587–592, Edmonton, AB, 2002.
- [16] A. Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *UAI*, pages 406–415, Stanford, CA, 2000.
- [17] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: an anytime algorithm for POMDPs. In *IJCAI*, Acapulco, Mexico, 2003.
- [18] P. Poupart and C. Boutilier. Bounded finite state controllers. In *NIPS*, Vancouver, BC, 2003.
- [19] P. Poupart and C. Boutilier. VDCBPI: an approximate scalable algorithm for large POMDPs. In *NIPS*, Vancouver, BC, 2004.
- [20] N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In *NIPS*, pages 1635–1642, Vancouver, BC, 2002.
- [21] N. Roy, J. Pineau, and S. Thrun. Spoken dialog management using probabilistic reasoning. In *ACL*, Hong Kong, 2000.
- [22] R. Smallwood and E. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.
- [23] S. Thrun. Monte Carlo POMDPs. In *NIPS*, pages 1064–1070, Denver, 1999.
- [24] N. Vlassis and M. T. J. Spaan. A fast point-based algorithm for POMDPs. In *Proc. Belgian-Dutch Conference on Machine Learning*, Brussels, Belgium, 2004.
- [25] J. Williams, P. Poupart, and S. Young. Using factored Markov decision processes with continuous observations for dialogue management. Technical Report CUED/F-INFEG/TR.520, Cambridge University, Engineering Department, 2005.
- [26] N. Zhang and W. Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, Hong Kong University of Science and Technology, 1996.
- [27] N. Zhang and W. Zhang. Speeding up the convergence of value-iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 14:29–51, 2001.

Optimal Nonmyopic Value of Information in Graphical Models – Efficient Algorithms and Theoretical Limits

Andreas Krause
Carnegie Mellon University

Carlos Guestrin
Carnegie Mellon University

Abstract

Many real-world decision making tasks require us to choose among several expensive observations. In a sensor network, for example, it is important to select the subset of sensors that is expected to provide the strongest reduction in uncertainty. It has been general practice to use heuristic-guided procedures for selecting observations. In this paper, we present the first efficient optimal algorithms for selecting observations for a class of graphical models containing Hidden Markov Models (HMMs). We provide results for both selecting the optimal subset of observations, and for obtaining an optimal conditional observation plan. For both problems, we present algorithms for the filtering case, where only observations made in the past are taken into account, and the smoothing case, where all observations are utilized. Furthermore we prove a surprising result: In most graphical models tasks, if one designs an efficient algorithm for chain graphs, such as HMMs, this procedure can be generalized to polytrees. We prove that the value of information problem is NP^{PP} -hard even for discrete polytrees. It also follows from our results that even computing conditional entropies, which are widely used to measure value of information, is a $\#\text{P}$ -complete problem on polytrees. Finally, we demonstrate the effectiveness of our approach on several real-world datasets.

1 Introduction

In probabilistic reasoning, where one can choose among several possible but expensive observations, it is often a central issue to decide which variables to observe in order to most effectively increase the expected utility. In a medical expert system [14], for example, multiple tests are available, and each test has a different cost. In such systems, it is thus important to decide which tests to perform in order to become most certain about the patient's condition, at a minimum cost. Occasionally, the cost of testing can even exceed the value of information for any possible outcome.

The following running example motivates our research and is empirically evaluated in Section 7. Consider a temperature monitoring task, where wireless temperature sensors are distributed across a building. The task is to become most certain about the temperature distribution, whilst minimizing energy expenditure, a critically constrained resource [4].

Many researchers have suggested the use of myopic (greedy) approaches to select observations [13; 15; 5; 1]. Unfortunately, this heuristic does not provide any performance

guarantees. In this paper, we present efficient algorithms, which guarantee optimal nonmyopic value of information in chain graphical models such as Hidden Markov Models (HMMs). We address two settings: *subset selection*, where the optimal subset of observations is obtained in an open-loop fashion, and *conditional plans*, a closed-loop plan where the observation strategy depends on the actual value of the observed variables (c.f. Fig. 1(a)). To our knowledge, these are the first optimal and efficient algorithms for these tasks for this class of graphical models. For both settings, we address the *filtering* and the *smoothing* versions: Filtering is important in online decision making, where our decisions can only utilize observations made in the past. Smoothing arises for example in structured classification tasks, where there is no temporal dimension in the data, and hence all observations can be taken into account. We evaluate our algorithms empirically on three real-world datasets, and also show that they are well-suited for interactive classification of sequential data.

Most problems in graphical models, such as probabilistic inference and the most probable explanation, that can be solved efficiently for chain-structured graphs, can also be solved efficiently for polytrees. We prove that the problem of maximizing value of information is NP^{PP} -hard even for discrete polytree graphical models, giving a complexity theoretic classification of a core artificial intelligence problem. NP^{PP} -hard problems are believed to be significantly harder than NP -complete or even $\#\text{P}$ -complete problems commonly arising in the context of graphical models. As a special case, we also prove that computing conditional entropies is $\#\text{P}$ -complete even in the case of discrete polytrees. This is a surprising result about a measure of uncertainty that is frequently used in practice.

2 Optimization criteria

In order to maximize value of information, our objective functions should depend on probability distributions over variables. Let $S = \{X_1, \dots, X_n\}$ be a set of discrete random variables. We consider a class of *local reward* functions R , which are defined on the marginal probability distributions of the variables. This class has the computational advantage that local rewards can be evaluated using probabilistic inference techniques. The total reward will then be the sum of all local rewards.

Let O be a subset of S . Then $P(X_j \mid O = o)$ denotes the marginal distribution of variable X_j conditioned on observations o . For classification purposes, it can be more appropriate to consider the max-marginals $P^{\text{max}}(X_j = x_j \mid O = o) = \max_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}, X_j = x_j \mid O = o)$, that is, for X_j set to value

x_j , the probability of the most probable assignment to all other random variables conditioned on the observations o . The *local reward* R_j is a functional on the probability distribution P or P^{max} over X_j . We write

$$R_j(X_j | O) \triangleq \sum_o P(O = o) R_j(P(X_j | O = o))$$

as an abbreviation to indicate *expected local rewards*, where the expectation is taken over all assignments o to the observations O . Important measures for value of information include:

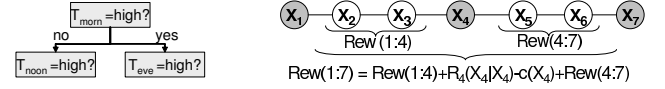
- **Entropy.** If we set $R_j(P(X_j | O)) = -H(X_j | O) = \sum_{x_j, o} P(x_j, o) \log P(x_j | o)$, the objective in the optimization problem becomes to minimize the sum of residual entropies. We choose this reward function in our running example to measure the uncertainty about the temperature distribution.
- **Maximum expected utility.** The concept of local reward functions also includes the concept of utility nodes in influence diagrams. If $U_j : A_j \times \text{dom } X_j \rightarrow \mathbb{R}$ is a utility function mapping an action $a \in A_j$ and an outcome $x \in \text{dom } X_j$ to a reward, then the *maximum expected utility principle* states that actions should be selected as to maximize $EU_j(a | o) = \sum_x P(x | o) U_j(a, x)$. The more certain we are about X_j , the more economically we can choose our action. Hence we can define our local reward function $R(P(X_j | O)) = \sum_o P(o) \max_a EU_j(a | o)$.
- **Margin.** We can also consider the margin of confidence: $R_j(P^{max}(X_j | O)) = \sum_o P(o) [P^{max}(x_j^* | o) - P^{max}(\bar{x}_j | o)]$, where $x^* = \text{argmax}_{x_j} P^{max}(x_j | o)$ and $\bar{x} = \text{argmax}_{x_j \neq x^*} P^{max}(x_j | o)$, which describes the margin between the most likely outcome and the closest runner up. This reward function is very useful for structured classification purposes, as shown in Sec. 7.

These examples demonstrate the generality of our notion of local reward. One can generalize the algorithms even more, e.g., to measure the total entropy or the margin between the most probable explanation and its runner up. Details are omitted here due to space limitations.

We also want to capture the constraint that observations are expensive. This can mean that each observation X_j has an associated positive *penalty* C_j that effectively decreases the reward. In our example, we might be interested in trading off accuracy with sensing energy expenditure. Alternatively, it is also possible to define a *budget* B for selecting observations, where each one is associated with an integer *cost* β_j . Here, we want to select observations whose sum cost is within the budget, but these costs do not decrease the reward. In our running example, the sensors could be powered by solar power, and regain a certain amount of energy per day, which allows a certain amount of sensing. Our formulation of the optimization problems allows both for penalties and budgets. To simplify notation we also write $C(O) = \sum_{X_j \in O} C_j$ and $\beta(O) = \sum_{X_j \in O} \beta_j$ to extend C and β to sets.

3 Decomposing Rewards

In the following Sections 4 and 5, we present efficient algorithms for two problems of optimizing value of information in the class of chain graphical models.



(a) Example conditional plan. (b) Decomposition of the reward.

Figure 1: Example, and decomposing reward idea

The set of random variables $S = \{X_1, \dots, X_n\}$ forms a chain graphical model (a chain), if X_i is conditionally independent of X_k given X_j whenever $i < j < k$. We can assume that the joint distribution is specified by the prior $P(X_1)$ and conditional probability distributions $P(X_{i+1} | X_i)$. The time series model for the temperature measured by one sensor in our example can be formulated as a chain graphical model.

Chain graphical models originating from time series have additional, specific properties: In a system for online decision making, only observations from the past and present time steps can be taken into account, not observations which will be made in the future. This is in general referred to as the *filtering* problem. In this setting, the notation $P(X_i | O)$ will refer to the distribution of X_i conditional on observations in O prior to and including time i . For structured classification problems as discussed in Section 7, in general observations made anywhere in the chain must be taken into account. This situation is usually referred to as the *smoothing* problem. We will provide algorithms both for filtering and smoothing.

We will now describe the key insight, which allows for efficient optimization in chains. Consider a set of observations $O \subseteq S$. If the j variable is observed, i.e., $X_j \in O$, then the local reward is simply $R(X_j | O) = R(X_j | X_j)$. Now consider $X_j \notin O$, and let O_j be the subset of O containing the closest ancestor (and for the smoothing problem also the closest descendant) of X_j in O . The conditional independence property of the graphical model implies that, given O_j , X_j is independent of the rest of the observed variables, i.e., $P(X_j | O) = P(X_j | O_j)$. Thus, it follows that $R(X_j | O) = R(X_j | O_j)$.

These observations imply that the expected reward of some set of observations decomposes along the chain. For simplicity of notation, we add two independent dummy variables X_0 and X_{n+1} , where $R_0 = C_0 = \beta_0 = R_{n+1} = C_{n+1} = \beta_{n+1} = 0$. Let $O = \{X_{i_0}, \dots, X_{i_{m+1}}\}$ where $i_l < i_{l+1}$, $i_0 = 0$ and $i_{m+1} = n + 1$. Using this notation, the total reward $R(O) = \sum_j R_j(X_j | O)$ for the smoothing case is given by:

$$\sum_{v=0}^m \left(R_{i_v}(X_{i_v} | X_{i_v}) - C_{i_v} + \sum_{j=i_v+1}^{i_{v+1}-1} R_j(X_j | X_{i_v}, X_{i_{v+1}}) \right).$$

In filtering settings, we simply replace $R_j(X_j | X_{i_v}, X_{i_{v+1}})$ by $R_j(X_j | X_{i_v})$. Figure 1(b) illustrates this decomposition.

Consider now a Hidden Markov Model unrolled for n time steps, i.e., S can be partitioned into the hidden variables $\{X_1, \dots, X_n\}$ and the emission variables $\{Y_1, \dots, Y_n\}$. In HMMs, the Y_i are observed and the variables X_i form a chain. In many applications, some of which are discussed in Section 7, we can observe some of the hidden variables, e.g., by asking an expert, in addition to observing the emission variables. In these cases, the problem of selecting expert labels also belongs to the class of chain graphical models addressed by this paper.

4 Subset Selection

In the *subset selection* problem, we want to find a most informative subset of the variables to observe *in advance*, i.e., before any observations are made. In our running example, we would, before deploying the sensors, identify k time points that are expected to provide the most informative sensor readings according to our model.

First, define the objective function L on subsets of S by

$$L(O) = \sum_{j=1}^n R_j(X_j | O) - C(O). \quad (4.1)$$

The *subset selection* problem is to find the optimal subset

$$O^* = \operatorname{argmax}_{O \subseteq S, \beta(O) \leq B} L(O)$$

maximizing the sum of expected local rewards minus the penalties, subject to the constraint that the total cost must not exceed the budget B .

We solve this optimization problem using a dynamic programming algorithm, where the chain is broken into sub-chains using the insight from Sec. 3. Consider a sub-chain from variable X_a to X_b . We define $L_{a:b}(k)$ to represent the expected total reward for the sub-chain $X_a, \dots, X_b$, where X_a (and X_b in the smoothing case) is observed, and with a budget level of k . More formally:

$$L_{a:b}(k) = \max_{\substack{O \subseteq \{X_{a+1} \dots X_{b-1}\} \\ \beta(O) \leq k}} \sum_{j=a+1}^{b-1} R_j(X_j | O \cup \{X_a\}) - C(O),$$

for the filtering version, and

$$L_{a:b}(k) = \max_{\substack{O \subseteq \{X_{a+1} \dots X_{b-1}\} \\ \beta(O) \leq k}} \sum_{j=a+1}^{b-1} R_j(X_j | O \cup \{X_a, X_b\}) - C(O),$$

for the smoothing version. Note that $L_{0:n+1}(B) = \max_{O: \beta(O) \leq B} L(O)$, as in Eq. (4.1), i.e., by computing the values for $L_{a:b}(k)$, we compute the maximum expected total reward for the entire chain.

We can compute $L_{a:b}(k)$ using dynamic programming. The base case is simply:

$$L_{a:b}(0) = \sum_{j=a+1}^{b-1} R_j(X_j | X_a),$$

for filtering, and

$$L_{a:b}(0) = \sum_{j=a+1}^{b-1} R_j(X_j | X_a, X_b),$$

for smoothing. The recursion for $L_{a:b}(k)$ has two cases: we can choose not to spend any more of the budget, reaching the base case, or we can break the chain into two sub-chains, selecting the optimal observation X_j , where $a < j < b$:

$$L_{a:b}(k) = \max\{L_{a:b}(0), \max_{j: a < j < b, \beta_j \leq k} \{-C_j + R_j(X_j | X_j) + L_{a:j}(0) + L_{j:b}(k - \beta_j)\}\}.$$

At first, it may seem that this recursion should consider the optimal split of the budget between the two sub-chains. However,

Input: Budget B , rewards R_j , costs β_j and penalties C_j
Output: Optimal selection O of observation times
begin

```

for  $0 \leq a < b \leq n + 1$  do compute  $L_{a:b}(0)$ ;
for  $k = 1$  to  $B$  do
  for  $0 \leq a < b \leq n + 1$  do
     $sel(-1) := L_{a:b}(0)$ ;
    for  $j = a + 1$  to  $b - 1$  do  $sel(j) :=$ 
       $-C_j + R_j(X_j | X_j) + L_{a:j}(0) + L_{j:b}(k - \beta_j)$ ;
     $L_{a:b}(k) = \max_{a < j < b} sel(j)$ ;
     $\Lambda_{a:b}(k) = \operatorname{argmax}_{a < j < b} sel(j)$ ;
  end
end
 $a := 0; b := n + 1; k := B; O := \emptyset$ ;
repeat
   $j := \Lambda_{a:b}(k)$ ;
  if  $j \geq 0$  then  $O := O \cup \{X_j\}; k := k - \beta_j$ ;
until  $j = -1$ ;
end
```

Algorithm 1: Optimal subset selection.

since the subset problem is open-loop and the order of the observations is irrelevant, we only need to consider split points where the first sub-chain receives zero budget.

A pseudo code implementation is given in Alg. 1. If we do not consider different costs β , we would simply choose $\beta_j = 1$ for all variables and compute $L_{a:b}(N)$. Alg. 1 uses the quantities $\Lambda_{a:b}$ to recover the optimal subset by tracing the maximal values occurring in the dynamic programming equations. Using an induction proof, we obtain:

Theorem 1. *The dynamic programming algorithm described above computes the optimal subset with budget B in $(\frac{1}{6}n^3 + \mathcal{O}(n^2))B$ evaluations of expected local rewards.* $\square$

If the variables X_i are continuous, our algorithm is still applicable when the integrations and inferences necessary for computing the expected rewards can be performed efficiently.

5 Conditional Plan

In the *conditional plan* problem, we want to compute an optimal querying policy: We sequentially observe a variable, pay the penalty, and depending on the observed values, select the next query as long as our budget suffices. The objective is to find the plan with the highest expected reward, where, for each possible sequence of observations, the budget B is not exceeded. For filtering, we can only select observations in the future, whereas in the smoothing case, the next observation can be anywhere in the chain. In our running example, the filtering algorithm would be most appropriate: The sensors would sequentially follow the conditional plan, deciding on the most informative times to sense based on the previous observations. Fig. 1(a) shows an example of such a conditional plan.

The formal definition of the objective function J is given recursively. The base case considers the exhausted budget:

$$J(O = o; 0) = \sum_{X_j \in S} R_j(X_j | O = o) - C(O).$$

The recursion, $J(O = o; k)$, represents the maximum expected reward of the conditional plan for the chain where $O = o$ has

been observed and the budget is limited to k :

$$J(O = o; k) = \max\{J(O = o; 0), \max_{X_j \notin O} \{ \sum_y P(X_j = y | O = o) \cdot J(O = o, X_j = y; k - \beta_j) \} \}.$$

The optimal plan has reward $J(\emptyset; B)$.

We propose a dynamic programming algorithm for obtaining the optimal conditional plan that is similar to the subset algorithm presented in Sec. 4. Again, we utilize the decomposition of rewards described in Section 3. The difference here is that the observation selection and budget allocation now depend on the actual values of the observations.

We again consider sub-chains $X_a, \dots, X_b$. The base case deals with the zero budget setting:

$$J_{a:b}(x_a; 0) = \sum_{j=a+1}^{b-1} R_j(X_j | X_a = x_a),$$

for filtering, and

$$J_{a:b}(x_a, x_b; 0) = \sum_{j=a+1}^{b-1} R_j(X_j | X_a = x_a, X_b = x_b),$$

for smoothing. The recursion defines $J_{a:b}(x_a; k)$ ($J_{a:b}(x_a, x_b; k)$ for smoothing), the expected reward for the problem restricted to the sub-chain $X_a, \dots, X_b$ conditioned on the values of X_a (and X_b for smoothing), and with budget limited by k . To compute this quantity, we again iterate through possible split points j , such that $a < j < b$. Here we observe a notable difference between the filtering and the smoothing case. For smoothing, we now must consider all possible splits of the budget between the two resulting sub-chains, since an observation at time j might require us to make an additional, earlier observation:

$$J_{a:b}(x_a, x_b; k) = \max\{J_{a:b}(x_a, x_b; 0), \max_{a < j < b} \{-C_j + \sum_{x_j} P(X_j = x_j | X_a = x_a, X_b = x_b) \{R_j(X_j | X_j) + \max_{0 \leq l \leq k - \beta_j} [J_{a:j}(x_a, x_j; l) + J_{j:b}(x_j, x_b; k - l - \beta_j)]\}\}\}.$$

Looking back in time is not possible in the filtering case, hence the recursion simplifies to

$$J_{a:b}(x_a; k) = \max\{J_{a:b}(x_a; 0), \max_{a < j < b: \beta_j \leq k} \{-C_j + \sum_{x_j} P(X_j = x_j | X_a = x_a) \{R_j(X_j | X_j) + J_{a:j}(x_a; 0) + J_{j:b}(x_j; k - \beta_j)\}\}\}.$$

The optimal reward is obtained by $J_{0:n+1}(\emptyset; B) = J(\emptyset; B)$. Alg. 5 presents a pseudo code implementation for the smoothing version – the filtering case is a straight-forward modification. The plan itself is compactly encoded in the quantities $\pi_{a:b}$ which determines the next variable to query and $\sigma_{a:b}$, which determines the allocation of the budget. Considering the exponential number of possible sequences of observations, it is remarkable that the optimal plan can even be represented using only polynomial space. Alg. 5 indicates how the computed plan can be executed. The procedure is recursive, requiring the parameters $a := 0, x_a := 1, b := n + 1, x_b := 1$ and $k := B$ for the initial call. Again, by induction, we obtain:

Theorem 2. *The algorithm for smoothing presented above computes an optimal conditional plan in $d^3 \cdot B^2 \cdot (\frac{1}{6}n^3 + \mathcal{O}(n^2))$ evaluations of local rewards, where d is the maximum domain size of the random variables $X_1, \dots, X_n$. In the filtering case, or if no budget is used, the optimal plan can be computed using $d^3 \cdot B \cdot (\frac{1}{6}n^3 + \mathcal{O}(n^2))$ or $d^3 \cdot (\frac{1}{6}n^4 + \mathcal{O}(n^3))$ evaluations respectively. $\square$*

The faster computation for the no budget case is obtained by observing that we do not require the third maximum computation, which distributes the budget into the sub-chains.

Input: Budget B , rewards R_j , costs β_j and penalties C_j
Output: Optimal conditional plan $(\pi_{a:b}, \sigma_{a:b})$
begin
 for $0 \leq a < b \leq n + 1, x_a \in \text{dom } X_a, x_b \in \text{dom } X_b$ **do**
 compute $J_{a:b}(x_a, x_b; 0)$;
 for $k = 1$ **to** B **do**
 for $0 \leq a < b \leq n + 1, x_a \in \text{dom } X_a, x_b \in \text{dom } X_b$ **do**
 $sel(-1) := J_{a:b}(0)$;
 for $a < j < b$ **do**
 $sel(j) := -C_j + R_j(X_j | X_j)$;
 for $a < j < b, x_j \in \text{dom } X_j$ **do**
 for $0 \leq l \leq k - \beta_j$ **do** ;
 $bd(l) :=$
 $J_{a:j}(x_a, x_j; l) + J_{j:b}(x_j, x_b; k - l - \beta_j)$;
 $sel(j) := sel(j) + P(x_j | x_a, x_b) \cdot \max_l bd(j)$;
 $\sigma(j, x_j) = \text{argmax}_l bd(j)$;
 end
 $J_{a:b}(k) = \max_{a < j < b} sel(j)$;
 $\pi_{a:b}(x_a, x_b; k) = \text{argmax}_{a < j < b} sel(j)$;
 for $x_j \in \text{dom } X_{\pi_{a:b}(k)}$ **do**
 $\sigma_{a:b}(x_a, x_b, x_j; k) = \sigma(\pi_{a:b}(k), x_j)$;
 end
 end
 end
 end

Algorithm 2: Computation of optimal conditional plan.

Input: Budget k , observations $X_a = x_a, X_b = x_b, \sigma, \pi$
begin
 $j := \pi_{a:b}(x_a, x_b; k)$;
 if $j \geq 0$ **then**
 Observe $X_j = x_j$;
 $l := \sigma_{a:b}(x_a, x_b, x_j; k)$;
 Recurse with $k := l, a := a, b := j$;
 Recurse with $k := k - l - \beta_j, a := j, b := b$;
 end
end

Algorithm 3: Observation selection using conditional plan.

6 Theoretical Limits

Many problems that can be solved efficiently for discrete chain graphical models can also be efficiently solved for discrete polytrees. Examples include probabilistic inference and the most probable explanation (MPE). Surprisingly, we prove that for the optimization problems discussed in this paper, this generalization is not possible, unless $\mathbf{P} = \mathbf{NP}$. All proofs in this section are stated in the Appendix.

In order to solve the optimization problems, we will most likely have to evaluate the objective function, i.e., the expected local rewards. Our first result states that this problem is intractable even for discrete polytrees.

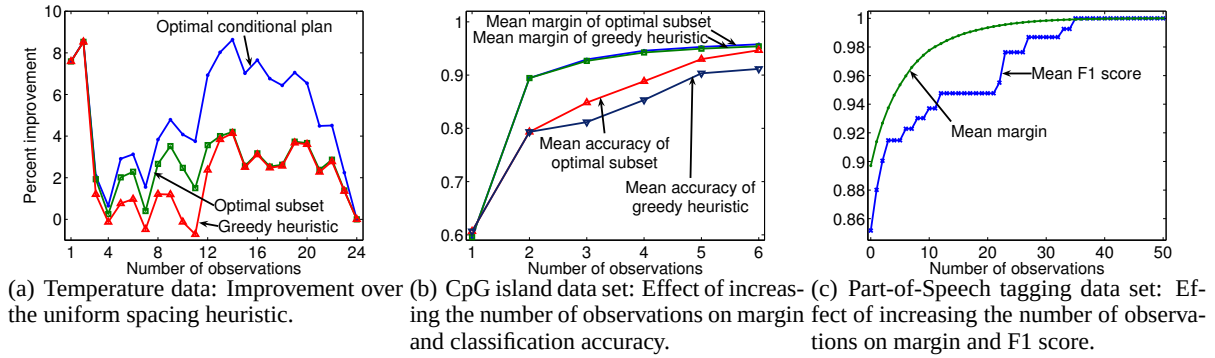


Figure 2: Experimental results.

Theorem 3. *The computation of expected local rewards for discrete polytrees is $\#P$ -complete.*¹ $\square$

This negative result can be specialized to the conditional entropy, one of the most frequently used reward function to characterize the residual uncertainty in value of information problems.

Corollary 4. *The computation of conditional entropy for discrete polytrees is $\#P$ -complete.* $\square$

Since evaluating local rewards is $\#P$ -complete, it can be suspected that the subset selection problem is at least $\#P$ -hard. We show that it is even NP^{PP} -complete², a complexity class containing problems that are believed to be significantly harder than NP or $\#P$ complete problems. This result provides a complexity theoretic classification of value of information, a core AI problem.

Theorem 5. *Subset Selection is NP^{PP} -complete even for discrete polytrees.* $\square$

For our running example, this implies that the generalized problem of optimally selecting k sensors from a network of correlated sensors is most likely computationally intractable without resorting to heuristics. A corollary extends the hardness of subset selection to the hardness of conditional plans.

Corollary 6. *Computing conditional plans is NP^{PP} -hard even for discrete polytrees.* $\square$

7 Experiments

In this section, we evaluate the proposed methods for several real world data sets. A special focus is on the comparison of the optimal methods with the greedy heuristic and other heuristic methods for selecting observations, and on how the algorithms can be used for interactive structured classification.

7.1 Temperature time series

The first data set consists of temperature time series collected from a sensor network deployed at Intel Research Berkeley [4] as described in our running example. Data was continuously collected for 19 days, linear interpolation was used in case of missing samples. The temperature was measured once every

¹ $\#P$ contains problems such as counting the number of satisfying assignments to a Boolean formula.

² NP^{PP} is natural for AI planning problems [9]. A complete problem is *EMAJSAT*, where one has to find an assignment to the first k variables of a 3CNF formula, such that the formula is satisfied under the majority of assignments to the remaining variables.

60 minutes, and it was discretized into 10 bins of 2 degrees Kelvin. To avoid overfitting, we used pseudo counts $\alpha = 0.5$ when learning the model. Using parameter sharing, we learned four sets of transition probabilities: from 12 am - 7am, 7 am - 12 pm, 12 pm - 7 pm and 7 pm - 12 am. Combining the data from three adjacent sensors, we got 53 sample time series.

The goal of this task was to select k out of 24 time points during the day, during which sensor readings are most informative. The experiment was designed to compare the performance of the optimal algorithms, the greedy heuristic, and a uniform spacing heuristic, which distributed the k observations uniformly over the day. Fig. 2(a) shows the relative improvement of the optimal algorithms and the greedy heuristic over the uniform spacing heuristic. The performance is measured in decrease of expected entropy, with zero observations as the baseline. It can be seen that if k is less than about the half of all possible observations, the optimal algorithms decreased the expected uncertainty by several percent over both heuristics. The improvement gained by the optimal plan over the subset selection algorithms appears to become more drastic if a large number of observations (over half of all possible observations) is allowed. Furthermore, for a large number of observations, the optimal subset and the subset selected by the greedy heuristic were almost identical.

7.2 CpG-Island detection

We then studied the bioinformatics problem of finding CpG islands in DNA sequences. CpG islands are regions in the genome with a high concentration of the cytosine-guanine sequence. These areas are believed to be mainly located around the promoters of genes, which are frequently expressed in the cell. In our experiment, we considered the gene loci HS381K22, AF047825 and AL133174, for which the GenBank annotation listed three, two and one CpG islands each. We ran our algorithm on a 50 base window at the beginning and end of each island, using the transition and emission probabilities from [6] for our Hidden Markov Model, and we used the sum of margins as reward function.

The goal of this experiment was to locate the beginning and ending of the CpG islands more precisely by asking experts, whether or not certain bases belong to the CpG region or not. Fig. 2(b) shows the mean classification accuracy and mean margin scores for an increasing number of observations. The results indicate that, although the expected margin scores are similar for the optimal algorithm and the greedy heuristic, the mean classification performance of the optimal algorithm was still better than the performance of the greedy heuristic.

7.3 Part-of-Speech Tagging

In our third experiment, we investigated the structured classification task of part-of-speech (POS) tagging [3]. Problem instances are sequences of words (sentences), where each word is part of an entity (e.g., “United States of America”), and each entity belongs to one of five categories: Location, Miscellaneous, Organization, Person or Other. Imagine an application, where automatic information extraction is guided by an expert: Our algorithms compute an optimal conditional plan for asking the expert, trying to optimize classification performance while requiring as little expert interaction as possible.

We used a conditional random field for the structured classification task, where each node corresponds to a word, and the joint distribution is described by node potentials and edge potentials. The sum of margins was used as reward function. Measure of classification performance was the F1 score, the geometric mean of precision and recall. The goal of this experiment was to analyze how the addition of expert labels increases the classification performance, and how the indirect, decomposing reward function used in our algorithms corresponds to real world classification performance.

Figure 2(c) shows the increase of the mean expected margin and F1 score for an increasing number of observations, summarized over ten 50 word sequences. It can be seen that the classification performance can be effectively enhanced by optimally incorporating expert labels. Requesting only three out of 50 labels increased the mean F1 score from by more than five percent. The following example illustrates this effect: In one scenario both words of an entity, the sportsman ‘P. Simmons’, were classified incorrectly – ‘P.’ as *Other* and ‘Simmons’ as *Miscellaneous*. The first request of the optimal conditional plan was to label ‘Simmons’. Upon labeling this word correctly, the word ‘P.’ was automatically labeled correctly also, resulting in an F1 score of 100 percent.

8 Related Work

Decision Trees [12] popularized the value of information as a criterion for creating conditional plans. Unfortunately, there are no guarantees on the performance of this greedy method. Bayer-Zubek [1] proposed a heuristic method based on the Markov Decision Process framework. Several researchers [15; 5] suggested myopic, i.e., greedy approaches for selectively gathering evidence in graphical models. Heckerman *et al.* [7] propose a method to compute the maximum expected utility for specific sets of observations. While their work considers more general graphical models than this paper, they provide only large sample guarantees for the evaluation of a given sequence of observations, and use a heuristic without guarantees to select such sequences. The subset selection problem as an instance of feature selection is a central issue in machine learning, with a vast amount of literature (see [10] for a survey). The problem of choosing observations also has a strong connection to the field of active learning [2] in which the learning system designs experiments based on its observations.

9 Conclusions

We have described novel efficient algorithms for optimal subset selection and conditional plan computation in chain graphical models, including HMMs. Empirical evaluation indicates that these algorithms can improve upon commonly used

heuristics for decreasing expected uncertainty. Our algorithms can also effectively enhance performance in interactive structured classification tasks.

Unfortunately, the optimization problems become intractable for even a slight generalization of chains. We presented surprising theoretical limits, which indicate that commonly used local reward functions, such as conditional entropies, cannot be efficiently computed even in discrete polytree graphical models. We also identified optimization of value of information as a new class of problems that are intractable (NP^{PP} -complete) for polytrees.

Our hardness results, along with other recent results for polytree graphical models, the NP -completeness of maximum a posteriori assignment [11] and NP -hardness of inference in conditional linear Gaussian models [8], suggest the possibility of developing a generalized complexity characterization of problems that are hard in polytree graphical models.

In light of these theoretical limits for computing optimal solutions, it is a natural question to ask whether approximation algorithms with non-trivial performance guarantees can be found. We are currently focusing our research in this direction.

Acknowledgements. We would like to thank Ben Taskar for providing the part-of-speech tagging model, and Reuters for making their news archive available. We would also like to thank Brigham Anderson and Andrew Moore for helpful comments and discussions.

Appendix

Proof of Theorem 3. Membership in $\#P$ is straightforward. To show hardness, we use a construction similar to the one presented in [11] for the maximum a posteriori problem. Let ϕ be an instance of $\#3SAT$, where we have to count the number of assignments to $X_1, \dots, X_n$ satisfying ϕ . Let $C = \{C_1, \dots, C_m\}$ be the set of clauses. Now create a Bayesian network with nodes U_i for each X_i , each with uniform Bernoulli prior. Add variables Y_0 , which uniformly varies over $\{1, \dots, m\}$ and $Y_1, \dots, Y_n$ with CPTs defined the following way:

$$Y_i \mid [Y_{i-1} = j, U_i = u_i] \sim \begin{cases} 0, & \text{if } j = 0, \text{ or } U_i = u_i \\ & \text{satisfies clause } C_j; \\ j, & \text{otherwise.} \end{cases}$$

In this model, $Y_n = 0$ iff $U_1, \dots, U_n$ encode a satisfying assignment of ϕ . Let all nodes have zero reward, except for Y_n , which is assigned the following reward:

$$R(Y_n \mid O = o) = \begin{cases} 2^n, & \text{if } P(Y_n = 0 \mid O = o) = 1; \\ 0, & \text{otherwise.} \end{cases}$$

Since the prior probability of any assignment is 2^{-n} , the expected reward $R(Y_n \mid U_1, \dots, U_n)$ is exactly the number of satisfying assignments to ϕ . $\square$

Proof of Corollary 4. We start from the same construction as in the proof of Theorem 3, and add an additional random variable Z after Y_n on the chain. $Z \mid Y_n$ is 0 if $Y_n = 0$, and takes uniformly random values in $\{0, 1\}$ if $Y_n \neq 0$. Then $H(Z \mid U = u)$ is 0 if u is a satisfying assignment, and 1 otherwise. Hence $H(Z \mid O) = 1 - K2^{-n}$, where K is the number of satisfying assignments to ϕ . $\square$

Proof of Theorem 5. Membership follows from Theorem 3. Let ϕ be an instance of $EMAJSAT$, where we have to find an

instantiation of $X_1, \dots, X_n$ such that $\phi(X_1, \dots, X_{2n})$ is true for the majority of assignments to $X_{n+1}, \dots, X_{2n}$. Let $C = \{C_1, \dots, C_m\}$ be the set of 3CNF clauses. Create the Bayesian network shown in Fig. 3, with nodes U_i , each having a uniform Bernoulli prior. Add bivariate variables $Y_i = (sel_i, par_i)$, $0 \leq i \leq 2n$, where sel_i takes values in $\{0, \dots, m\}$ and par_i is a parity bit. The CPTs for Y_i are defined as: sel_0 uniformly varies over $\{1, \dots, m\}$, $par_0 = 0$, and for $Y_1, \dots, Y_{2n}$:

$$sel_i \mid [sel_{i-1} = j, U_i = u_i] \sim \begin{cases} 0, & \text{if } j = 0, \text{ or } u_i \text{ satisfies } C_j; \\ j, & \text{otherwise;} \end{cases}$$

$$par_i \mid [par_{i-1} = b_{i-1}, U_i] \sim b_{i-1} \oplus U_i,$$

where $\oplus$ denotes the parity (XOR) operator.

We now add variables Z_i^T and Z_i^F for $1 \leq i \leq n$ and let

$$Z_i^T \mid [U_i = u_i] \sim \begin{cases} \mathcal{I}(\{0, 1\}), & \text{if } u_i = 1; \\ 0, & \text{otherwise;} \end{cases}$$

where $\mathcal{I}$ denotes the uniform distribution. Similarly, let

$$Z_i^F \mid [U_i = u_i] \sim \begin{cases} \mathcal{I}(\{0, 1\}), & \text{if } u_i = 0; \\ 0, & \text{otherwise.} \end{cases}$$

Intuitively, $Z_i^T = 1$ guarantees us that $U_i = 1$, whereas $Z_i^T = 0$ leaves us uncertain about U_i . The case of Z_i^F is symmetric.

We use the subset selection algorithm to choose the Z_i s that encode the solution to *EMAJSAT*. If Z_i^T is chosen, it will indicate that X_i should set to true, similarly Z_i^F indicates a false assignment to X_i . The parity function is going to be used to ensure that exactly one of $\{Z_i^T, Z_i^F\}$ is observed for each i .

We first assign penalties ∞ to all nodes except Z_i^T, Z_i^F for $1 \leq i \leq n$, and U_j for $n+1 \leq j \leq 2n$, which are assigned zero penalty. Let all nodes have zero reward, except for Y_{2n} , which is assigned the following reward:

$$R(Y_{2n} \mid O = o) = \begin{cases} 4^n, & \text{if } P(sel_{2n} = 0 \mid O = o) = 1 \text{ and} \\ & [P(par_{2n} = 1 \mid O = o) = 1 \text{ or} \\ & P(par_{2n} = 0 \mid O = o) = 1]; \\ 0, & \text{otherwise.} \end{cases}$$

Note that $sel_{2n} = 0$ with probability 1 iff $U_1, \dots, U_{2n}$ encode a satisfying assignment of ϕ , as in the proof of Theorem 3. Furthermore, we get positive reward only if we are both certain that $sel_{2n} = 0$, i.e., the chosen observation set must contain a proof that ϕ is satisfied, and we are certain about par_{2n} . The parity certainty will only occur if we are certain about the assignment $U_1, \dots, U_{2n}$. It is only possible to infer the value of each U_i with certainty by observing one of U_i, Z_i^T or Z_i^F . Since, for $i = 1, \dots, n$, the cost of observing U_i is ∞ , to receive any reward we must observe at least one of Z_i^T or Z_i^F . Assume that we compute the optimal subset $\hat{O}$ for budget $2n$, then we can only receive positive reward by observing exactly one of Z_i^T or Z_i^F .

We interpret the selection of Z_i^T and Z_i^F as an assignment to the first n variables of *EMAJSAT*. Let $\hat{R} = R(Y_{2n} \mid \hat{O})$. We claim that $\phi \in \text{EMAJSAT}$ if and only if $\hat{R} > 0.5$. First let $\phi \in \text{EMAJSAT}$, with assignment $x_1, \dots, x_n$ to the first n variables. Now add $U_{n+1}, \dots, U_{2n}$ to O and add Z_i^T to O iff $x_i = 1$ and Z_i^F to O iff $x_i = 0$. This selection guarantees $\hat{R} > 0.5$.

Now assume $\hat{R} > 0.5$. We call an assignment to $U_1, \dots, U_{2n}$ *consistent* if for any $1 \leq i \leq n$, if $Z_i^T \in \hat{O}$,

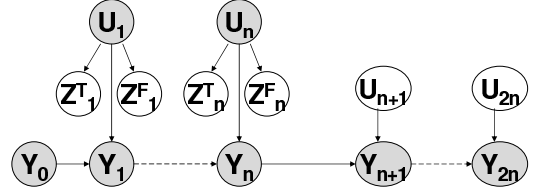


Figure 3: Graphical model used in proof of Theorem 6.

then $U_i = 1$ and if $Z_i^F \in \hat{O}$ then $U_i = 0$. For any consistent assignment, the chance that the observations Z_i prove the consistency is 2^{-n} . Hence $\hat{R} > 0.5$ implies that the majority of all provably consistent assignments satisfy ϕ and hence $\phi \in \text{EMAJSAT}$. This proves that subset selection is NP^{PP} complete. $\square$

Proof of Corollary 6. The construction in the proof of Theorem 5 also proves that computing conditional plans is NP^{PP} -hard, since, in this instance, any plan with positive reward must observe all $U_{n+1}, \dots, U_{2n}$ and one each of the $Z_1, \dots, Z_n$, to satisfy the parity condition. In this case, the order of selection is irrelevant, and, hence, the conditional plan effectively performs subset selection. $\square$

References

- [1] V. Bayer-Zubek. Learning diagnostic policies from examples by systematic search. In *UAI*, 2004.
- [2] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *J AI Research*, 4:129–145, 1996.
- [3] CoNLL. Conference on computational natural language learning shared task. <http://cnts.uia.ac.be/conll2003/ner/>, 2003.
- [4] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.
- [5] S. Dittmer and F. Jensen. Myopic value of information in influence diagrams. In *UAI*, pages 142–149, San Francisco, 1997.
- [6] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1999.
- [7] D. Heckerman, E. Horvitz, and B. Middleton. An approximate nonmyopic computation for value of information. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15:292–298, 1993.
- [8] U. Lerner and R. Parr. Inference in hybrid networks: Theoretical limits and practical algorithms. In *UAI*, 2001.
- [9] M. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.
- [10] L. Molina, L. Belanche, and A. Nebot. Feature selection algorithms: A survey and experimental evaluation. In *ICDM*, 2002.
- [11] J. D. Park and A. Darwiche. Complexity results and approximation strategies for map explanations. *Journal of Artificial Intelligence Research*, 21:101–133, February 2004.
- [12] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [13] T. Scheffer, C. Decomain, and S. Wrobel. Active learning of partially hidden markov models for information extraction. In *ECML/PKDD Workshop on Instance Selection*, 2001.
- [14] P. D. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2:369–409, 1995.
- [15] L. van der Gaag and M. Wessels. Selective evidence gathering for diagnostic belief networks. *AISB Quart.*, 86:23–34, 1993.

An MCMC Approach to Solving Hybrid Factored MDPs

Branislav Kveton

Intelligent Systems Program
University of Pittsburgh
bkveton@cs.pitt.edu

Milos Hauskrecht

Department of Computer Science
University of Pittsburgh
milos@cs.pitt.edu

Abstract

Hybrid approximate linear programming (HALP) has recently emerged as a promising framework for solving large factored Markov decision processes (MDPs) with discrete and continuous state and action variables. Our work addresses its major computational bottleneck – constraint satisfaction in large structured domains of discrete and continuous variables. We analyze this problem and propose a novel Markov chain Monte Carlo (MCMC) method for finding the most violated constraint of a relaxed HALP. This method does not require the discretization of continuous variables, searches the space of constraints intelligently based on the structure of factored MDPs, and its space complexity is linear in the number of variables. We test the method on a set of large control problems and demonstrate improvements over alternative approaches.

1 Introduction

Markov decision processes (MDPs) [Bellman, 1957; Puterman, 1994] offer an elegant mathematical framework for solving sequential decision problems in the presence of uncertainty. However, traditional techniques for solving MDPs are computationally infeasible in real-world domains, which are factored and contain both discrete and continuous state and action variables. Recently, approximate linear programming (ALP) [Schweitzer and Seidmann, 1985] has emerged as a promising approach to solving large factored MDPs [de Farias and Roy, 2003; Guestrin *et al.*, 2003]. This work focuses on hybrid ALP (HALP) [Guestrin *et al.*, 2004] and addresses its major computational bottleneck – constraint satisfaction in the domains of discrete and continuous variables.

If the state and action variables are discrete, HALP involves an exponential number of constraints, and if any of the variables are continuous, the number of constraints is infinite. To approximate the constraint space in HALP, two techniques have been proposed: ε -HALP [Guestrin *et al.*, 2004] and Monte Carlo constraint sampling [de Farias and Roy, 2004; Hauskrecht and Kveton, 2004]. The ε -HALP formulation relaxes the continuous portion of the constraint space to an ε -grid, which can be compactly satisfied by the methods for discrete-state ALP [Guestrin *et al.*, 2001; Schuurmans and

Patrascu, 2002]. However, these methods are exponential in the treewidth of the discretized constraint space, which limits their application to real-world problems. In addition, the ε -grid discretization is done blindly and impacts the quality of the approximation. Monte Carlo methods [de Farias and Roy, 2004; Hauskrecht and Kveton, 2004] offer an alternative to the ε -grid discretization and approximate the constraint space in HALP by its finite sample. Unfortunately, the efficiency of Monte Carlo methods is heavily dependent on an appropriate choice of sampling distributions. The ones that yield good approximations and polynomial sample size bounds are closely related to the optimal solutions and rarely known a priori [de Farias and Roy, 2004].

To overcome the limitations of the discussed constraint satisfaction techniques, we propose a novel Markov chain Monte Carlo (MCMC) method for finding the most violated constraint of a relaxed HALP. The method directly operates in the domains of continuous variables, takes into account the structure of factored MDPs, and its space complexity is proportional to the number of variables. Such a separation oracle can be easily embedded into the ellipsoid or cutting plane method for solving linear programs, and therefore constitutes a key step towards solving HALP efficiently.

The paper is structured as follows. First, we introduce hybrid MDPs and HALP [Guestrin *et al.*, 2004], which are our frameworks for modeling and solving large-scale stochastic decision problems. Second, we review existing approaches to solving HALP and discuss their limitations. Third, we compactly represent the constraint space in HALP and formulate an optimization problem for finding the most violated constraint of a relaxed HALP. Fourth, we design a Markov chain to solve this optimization problem and embed it into the cutting plane method. Finally, we test our HALP solver on a set of large control problems and compare its performance to alternative approaches.

2 Hybrid factored MDPs

Factored MDPs [Boutilier *et al.*, 1995] allow a compact representation of large stochastic planning problems by exploiting their structure. In this section, we review hybrid factored MDPs [Guestrin *et al.*, 2004], which extend this formalism to the domains of discrete and continuous variables.

A *hybrid factored MDP with distributed actions (HMDP)* [Guestrin *et al.*, 2004] is a 4-tuple $\mathcal{M} = (\mathbf{X}, \mathbf{A}, P, R)$, where

$\mathbf{X} = \{X_1, \dots, X_n\}$ is a state space represented by a set of state variables, $\mathbf{A} = \{A_1, \dots, A_m\}$ is an action space represented by a set of action variables, $P(\mathbf{X}' | \mathbf{X}, \mathbf{A})$ is a stochastic transition model of state dynamics conditioned on the preceding state and action choice, and R is a reward model assigning immediate payoffs to state-action configurations¹.

State variables: State variables are either discrete or continuous. Every discrete variable X_i takes on values from a finite domain $\text{Dom}(X_i)$. Following Hauskrecht and Kveton 2004, we assume that every continuous variable is bounded to the $[0, 1]$ subspace. The state is represented by a vector of value assignments $\mathbf{x} = (\mathbf{x}_D, \mathbf{x}_C)$ which partitions along its discrete and continuous components $\mathbf{x}_D$ and $\mathbf{x}_C$.

Action variables: The action space is distributed and represented by action variables $\mathbf{A}$. The composite action is defined by a vector of individual action choices $\mathbf{a} = (\mathbf{a}_D, \mathbf{a}_C)$ which partitions along its discrete and continuous components $\mathbf{a}_D$ and $\mathbf{a}_C$.

Transition model: The transition model is given by the conditional probability distribution $P(\mathbf{X}' | \mathbf{X}, \mathbf{A})$, where $\mathbf{X}$ and $\mathbf{X}'$ denote the state variables at two successive time steps. We assume that the model factors along $\mathbf{X}'$ as $P(\mathbf{X}' | \mathbf{X}, \mathbf{A}) = \prod_{i=1}^n P(X'_i | \text{Par}(X'_i))$ and can be compactly represented by a *dynamic Bayesian network (DBN)* [Dean and Kanazawa, 1989]. Usually, the parent set $\text{Par}(X'_i) \subseteq \mathbf{X} \cup \mathbf{A}$ is a small subset of state and action variables which allows for a local parameterization of the model.

Parameterization of transition model: One-step dynamics of every state variable is described by its conditional probability distribution $P(X'_i | \text{Par}(X'_i))$. If X'_i is a continuous variable, its transition function is represented by a mixture of beta distributions [Hauskrecht and Kveton, 2004]:

$$P(x | \text{Par}(X'_i)) = \sum_j \pi_{ij} \frac{\Gamma(\alpha_j + \beta_j)}{\Gamma(\alpha_j)\Gamma(\beta_j)} x^{\alpha_j-1} (1-x)^{\beta_j-1}, \quad (1)$$

where π_{ij} is the weight assigned to the j -th component of the mixture, and $\alpha_j = \phi_{ij}^\alpha(\text{Par}(X'_i))$ and $\beta_j = \phi_{ij}^\beta(\text{Par}(X'_i))$ are arbitrary positive functions of the parent set. The mixture of beta distributions provides a very general class of transition functions and yet allows closed-form solutions to the integrals in HALP. If X'_i is a discrete variable, its transition model is parameterized by $|\text{Dom}(X'_i)|$ nonnegative discriminant functions $\theta_j = \phi_{ij}^\theta(\text{Par}(X'_i))$ [Guestrin *et al.*, 2004]:

$$P(j | \text{Par}(X'_i)) = \frac{\theta_j}{\sum_{j=1}^{|\text{Dom}(X'_i)|} \theta_j}. \quad (2)$$

Reward model: The reward function is an additive function $R(\mathbf{x}, \mathbf{a}) = \sum_j R_j(\mathbf{x}_j, \mathbf{a}_j)$ of local reward functions defined on the subsets of state and action variables $\mathbf{X}_j$ and $\mathbf{A}_j$.

Optimal value function and policy: The quality of a policy is measured by the *infinite horizon discounted reward* $E[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma \in [0, 1)$ is a *discount factor* and r_t is the reward obtained at the time step t . This optimality criterion guarantees that there always exists an *optimal policy* π^*

¹General state and action space MDP is an alternative name for a hybrid MDP. The term *hybrid* does not refer to the dynamics of the model, which is discrete-time.

which is stationary and deterministic [Puterman, 1994]. The policy is greedy with respect to the *optimal value function* V^* , which is a fixed point of the Bellman equation [Bellman, 1957; Bertsekas and Tsitsiklis, 1996]:

$$V^*(\mathbf{x}) = \sup_{\mathbf{a}} \left[R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'_D} \int_{\mathbf{x}'_C} P(\mathbf{x}' | \mathbf{x}, \mathbf{a}) V^*(\mathbf{x}') d\mathbf{x}'_C \right]. \quad (3)$$

3 Hybrid ALP

Value iteration, policy iteration, and linear programming are the most fundamental dynamic programming (DP) methods for solving MDPs [Puterman, 1994; Bertsekas and Tsitsiklis, 1996]. However, their computational complexity grows exponentially in the number of used variables, which makes them unsuitable for factored MDPs. Moreover, they are built on the assumption of finite support for the optimal value function and policy, which may not exist if continuous variables are present. Recently, Feng *et al.* 2004 showed how to solve general state-space MDPs by performing DP backups of piecewise constant and piecewise linear value functions. This approximate method has a lower scale-up potential than HALP, but does not require the design of basis functions.

Linear value function: Value function approximation is a standard approach to solving large factored MDPs. Due to its favorable computational properties, *linear value function approximation* [Bellman *et al.*, 1963; Roy, 1998]:

$$V^w(\mathbf{x}) = \sum_i w_i f_i(\mathbf{x})$$

has become extremely popular in recent research [Guestrin *et al.*, 2001; Schuurmans and Patrascu, 2002; de Farias and Roy, 2003; Hauskrecht and Kveton, 2004; Guestrin *et al.*, 2004]. This approximation restricts the form of the value function V^w to the linear combination of $|\mathbf{w}|$ basis functions $f_i(\mathbf{x})$, where $\mathbf{w}$ is a vector of tunable weights. Every basis function can be defined over the complete state space $\mathbf{X}$, but often is restricted to a subset of state variables $\mathbf{X}_i$.

3.1 HALP formulation

Various methods for fitting of the linear value function approximation have been proposed and analyzed [Bertsekas and Tsitsiklis, 1996]. We adopt a variation on approximate linear programming (ALP) [Schweitzer and Seidmann, 1985], *hybrid ALP (HALP)* [Guestrin *et al.*, 2004], which extends this framework to the domains of discrete and continuous variables. The HALP formulation is given by:

$$\begin{aligned} & \text{minimize}_{\mathbf{w}} \sum_i w_i \alpha_i \\ & \text{subject to: } \sum_i w_i F_i(\mathbf{x}, \mathbf{a}) - R(\mathbf{x}, \mathbf{a}) \geq 0 \quad \forall \mathbf{x}, \mathbf{a}; \end{aligned}$$

where α_i denotes *basis function relevance weight*:

$$\alpha_i = \sum_{\mathbf{x}_D} \int_{\mathbf{x}_C} \psi(\mathbf{x}) f_i(\mathbf{x}) d\mathbf{x}_C, \quad (4)$$

$\psi(\mathbf{x})$ is a *state relevance density function* weighting the quality of the approximation, and $F_i(\mathbf{x}, \mathbf{a}) = f_i(\mathbf{x}) - \gamma g_i(\mathbf{x}, \mathbf{a})$

is the difference between the basis function $f_i(\mathbf{x})$ and its discounted *backprojection*:

$$g_i(\mathbf{x}, \mathbf{a}) = \sum_{\mathbf{x}'_D} \int_{\mathbf{x}'_C} P(\mathbf{x}' | \mathbf{x}, \mathbf{a}) f_i(\mathbf{x}') d\mathbf{x}'_C. \quad (5)$$

We say that the HALP is *relaxed* if only a subset of the constraints is satisfied.

The HALP formulation reduces to the discrete-state ALP [Schweitzer and Seidmann, 1985; Schuurmans and Patrascu, 2002; de Farias and Roy, 2003; Guestrin *et al.*, 2003] if the state and action variables are discrete, and to the continuous-state ALP [Hauskrecht and Kveton, 2004] if the state variables are continuous. The quality of this approximation was studied by Guestrin *et al.* 2004 and bounded with respect to $\min_{\mathbf{w}} \|V^* - V^{\mathbf{w}}\|_{\infty, 1/L}$, where $\|\cdot\|_{\infty, 1/L}$ is a max-norm weighted by the reciprocal of the Lyapunov function $L(\mathbf{x}) = \sum_i w_i^L f_i(\mathbf{x})$. The integrals in the objective function (Equation 4) and constraints (Equation 5) have closed-form solutions if the basis functions and state relevance densities are chosen appropriately [Hauskrecht and Kveton, 2004]. For example, the mixture of beta transition model (Equation 1) yields a closed-form solution to Equation 5 if the basis function $f_i(\mathbf{x}')$ is a polynomial. Finally, we need an efficient constraint satisfaction procedure to solve HALP.

3.2 Constraint satisfaction in HALP

If the state and action variables are discrete, HALP involves an exponential number of constraints, and if any of the variables are continuous, the number of constraints is infinite. To approximate such a constraint space, two techniques have been proposed recently: ε -HALP [Guestrin *et al.*, 2004] and Monte Carlo constraint sampling [de Farias and Roy, 2004; Hauskrecht and Kveton, 2004].

ε -HALP: The ε -HALP formulation relaxes the continuous portion of the constraint space to an ε -grid by the discretization of continuous variables $\mathbf{X}_C$ and $\mathbf{A}_C$. The new constraint space spans discrete variables only and can be compactly satisfied by the methods for discrete-state ALP [Guestrin *et al.*, 2001; Schuurmans and Patrascu, 2002]. For example, Schuurmans and Patrascu 2002 search for the most violated constraint with respect to the solution $\mathbf{w}^{(t)}$ of a relaxed ALP:

$$\arg \min_{\mathbf{x}, \mathbf{a}} \left[\sum_i w_i^{(t)} [f_i(\mathbf{x}) - \gamma g_i(\mathbf{x}, \mathbf{a})] - R(\mathbf{x}, \mathbf{a}) \right] \quad (6)$$

and add it to the linear program. If no violated constraint is found, $\mathbf{w}^{(t)}$ is an optimal solution to the ALP.

The space complexity of both constraint satisfaction methods [Guestrin *et al.*, 2001; Schuurmans and Patrascu, 2002] is exponential in the treewidth of the constraint space. This is a serious limitation because the cardinality of discretized variables grows with the resolution of the ε -grid. Roughly, if the discretized variables are replaced by binary, the treewidth increases by a multiplicative factor of $\log_2(1/\varepsilon + 1)$, where $(1/\varepsilon + 1)$ is the number of discretization points in a single dimension. Therefore, even problems with a relatively small treewidth are intractable for small values of ε . In addition, the ε -grid discretization is done blindly and impacts the quality of the approximation.

Monte Carlo constraint sampling: Monte Carlo methods approximate the constraint space in HALP by its finite sample. De Farias and Van Roy 2004 analyzed constraint sampling for discrete-state ALP and bounded the sample size for achieving good approximations by a polynomial in the number of basis functions and state variables. Hauskrecht and Kveton 2004 applied random constraint sampling to solve continuous-state factored MDPs and later refined their sampler by heuristics [Kveton and Hauskrecht, 2004].

Monte Carlo constraint sampling is easily applied in continuous domains and can be implemented in a space proportional to the number of variables. However, proposing an efficient sampling procedure that guarantees a polynomial bound on the sample size is as hard as knowing the optimal policy itself [de Farias and Roy, 2004]. To lower a high number of sampled constraints in a relaxed HALP, Monte Carlo samplers can be embedded into the cutting plane method. A technique similar to Kveton and Hauskrecht 2004 yields significant speedup with no drop in the quality of the approximation.

4 MCMC constraint sampling

To address the deficiencies of the discussed constraint satisfaction techniques, we propose a novel Markov chain Monte Carlo (MCMC) method for finding the most violated constraint of a relaxed HALP. Before we proceed, we compactly represent the constraint space in HALP and formulate an optimization problem for finding the most violated constraint in this representation.

4.1 Compact representation of constraints

Guestrin *et al.* 2001 and Schuurmans and Patrascu 2002 showed that the compact representation of constraints is essential in solving ALP efficiently. Following their ideas, we define *violation magnitude* $\tau^{\mathbf{w}}(\mathbf{x}, \mathbf{a})$:

$$\tau^{\mathbf{w}}(\mathbf{x}, \mathbf{a}) = - \sum_i w_i [f_i(\mathbf{x}) - \gamma g_i(\mathbf{x}, \mathbf{a})] + R(\mathbf{x}, \mathbf{a}) \quad (7)$$

to be the amount by which the solution $\mathbf{w}$ violates the constraints of a relaxed HALP. We represent $\tau^{\mathbf{w}}(\mathbf{x}, \mathbf{a})$ compactly by an influence diagram (ID), where $\mathbf{X}$ and $\mathbf{A}$ are decision nodes and $\mathbf{X}'$ are random variables. The ID representation is built on the transition model $P(\mathbf{X}' | \mathbf{X}, \mathbf{A})$, which is already factored and contains dependencies among the variables $\mathbf{X}$, $\mathbf{X}'$, and $\mathbf{A}$. We extend the diagram by three types of reward nodes, one for each term in Equation 7: $R_j = R_j(\mathbf{x}_j, \mathbf{a}_j)$ for every local reward function, $H_i = -w_i f_i(\mathbf{x})$ for every basis function, and $G_i = \gamma w_i f_i(\mathbf{x}')$ for every backprojection. The construction is completed by adding arcs that represent the dependencies of the reward nodes on the variables. Finally, we verify that $\tau^{\mathbf{w}}(\mathbf{x}, \mathbf{a}) = E_{P(\mathbf{x}'|\mathbf{x}, \mathbf{a})} [\sum_i (H_i + G_i) + \sum_j R_j]$. Therefore, the decision that maximizes the expected utility in the ID corresponds to the most violated constraint.

We conclude that any algorithm for solving IDs can be used to find the most violated constraint. Moreover, special properties of the ID representation allow its further simplification. If the basis functions are chosen conjugate to the transition model (Section 3.1), we obtain a closed-form solution to $E_{P(\mathbf{x}'|\mathbf{x}, \mathbf{a})} [G_i]$ (Equation 5), and thus the random variables

$\mathbf{X}'$ can be marginalized out of the diagram. This new representation contains no random variables and is known as a *cost network* [Guestrin *et al.*, 2001].

4.2 Separation oracle

To find the most violated constraint in the cost network, we use the Metropolis-Hastings (MH) algorithm [Metropolis *et al.*, 1953; Hastings, 1970] and construct a Markov chain whose invariant distribution converges to the vicinity of $\arg \max_{\mathbf{z}} \tau^{\mathbf{w}}(\mathbf{z})$, where $\mathbf{z} = (\mathbf{x}, \mathbf{a})$ and $\mathbf{Z} = \mathbf{X} \cup \mathbf{A}$ is a joint set of state and action variables. The Metropolis-Hastings algorithm accepts the transition from a state $\mathbf{z}$ to a proposed state $\mathbf{z}^*$ with the *acceptance probability* $A(\mathbf{z}, \mathbf{z}^*) = \min \left\{ 1, \frac{p(\mathbf{z}^*)q(\mathbf{z}|\mathbf{z}^*)}{p(\mathbf{z})q(\mathbf{z}^*|\mathbf{z})} \right\}$, where $q(\mathbf{z}^* | \mathbf{z})$ is a *proposal distribution* and $p(\mathbf{z})$ is a *target density*. Under mild restrictions on $p(\mathbf{z})$ and $q(\mathbf{z}^* | \mathbf{z})$, the chain always converges to the target density $p(\mathbf{z})$ [Andrieu *et al.*, 2003]. In the rest of this section, we discuss the choice of $p(\mathbf{z})$ and $q(\mathbf{z}^* | \mathbf{z})$ to solve our optimization problem.

Target density: The violation magnitude $\tau^{\mathbf{w}}(\mathbf{z})$ is turned into a density by the transformation $p(\mathbf{z}) = \exp[\tau^{\mathbf{w}}(\mathbf{z})]$. Due to its monotonic character, $p(\mathbf{z})$ retains the same set of global maxima as $\tau^{\mathbf{w}}(\mathbf{z})$, and thus the search for $\arg \max_{\mathbf{z}} \tau^{\mathbf{w}}(\mathbf{z})$ can be performed on $p(\mathbf{z})$. To prove that $p(\mathbf{z})$ is a density, we show that it has a normalizing constant $\sum_{\mathbf{z}_D} \int_{\mathbf{z}_C} p(\mathbf{z}) d\mathbf{z}_C$, where $\mathbf{z}_D$ and $\mathbf{z}_C$ are the discrete and continuous components of the vector $\mathbf{z} = (\mathbf{z}_D, \mathbf{z}_C)$. As the integrand $\mathbf{z}_C$ is restricted to the finite space $[0, 1]^{|\mathbf{z}_C|}$, the integral is proper as long as $p(\mathbf{z})$ is bounded, and therefore it is Riemann integrable and finite. To prove that $p(\mathbf{z})$ is bounded, we bound $\tau^{\mathbf{w}}(\mathbf{z})$. Let $R_{\max}$ denote the maximum one-step reward in the HMDP. If the basis functions are of unit magnitude, w_i can be typically bounded by $|w_i| \leq \gamma(1 - \gamma)^{-1} R_{\max}$, and consequently $|\tau^{\mathbf{w}}(\mathbf{z})| \leq (|\mathbf{w}| \gamma(1 - \gamma)^{-1} + 1) R_{\max}$. Therefore, $p(\mathbf{z})$ is bounded and can be treated as a density function.

To find the mode of $p(\mathbf{z})$, we adopt the simulating annealing approach [Kirkpatrick *et al.*, 1983] and simulate a non-homogeneous Markov chain whose invariant distribution equals to $p^{1/T_t}(\mathbf{z})$, where T_t is a decreasing cooling schedule with $\lim_{t \rightarrow \infty} T_t = 0$. Under weak regularity assumptions on $p(\mathbf{z})$, $p^\infty(\mathbf{z})$ is a probability density that concentrates on the set of global maxima of $p(\mathbf{z})$ [Andrieu *et al.*, 2003]. If the cooling schedule decreases such that $T_t \geq c / \log_2(t + 2)$, where c is a problem-specific constant independent of t , the chain converges to the vicinity of $\arg \max_{\mathbf{z}} \tau^{\mathbf{w}}(\mathbf{z})$ with the probability converging to 1 [Geman and Geman, 1984]. However, this schedule can be too slow in practice, especially for a high initial temperature c . Following the suggestion of Geman and Geman 1984, we overcome this limitation by selecting a smaller value of c than is required by the convergence criterion. As a result, convergence to the global optimum $\arg \max_{\mathbf{z}} \tau^{\mathbf{w}}(\mathbf{z})$ is no longer guaranteed.

Proposal distribution: We take advantage of the factored character of $\mathbf{Z}$ and adopt the following proposal distribution [Geman and Geman, 1984]:

$$q(\mathbf{z}^* | \mathbf{z}) = \begin{cases} p(z_i^* | \mathbf{z}_{-i}) & \text{if } \mathbf{z}_{-i}^* = \mathbf{z}_{-i} \\ 0 & \text{otherwise} \end{cases},$$

where $\mathbf{z}_{-i}$ and $\mathbf{z}_{-i}^*$ are the assignments to all variables but Z_i in the original and proposed states. If Z_i is a discrete variable, its conditional $p(z_i^* | \mathbf{z}_{-i}) = \frac{p(z_1, \dots, z_{i-1}, z_i^*, z_{i+1}, \dots, z_{n+m})}{\sum_{z_i} p(z_1, \dots, z_{i-1}, z_i, z_{i+1}, \dots, z_{n+m})}$ can be derived in a closed form. If Z_i is a continuous variable, a closed form of its cumulative density function is not likely to exist. To allow sampling from its conditional, we embed another MH step within the original chain. In the experimental section, we use the Metropolis algorithm with the acceptance probability $A(z_i, z_i^*) = \min \left\{ 1, \frac{p(z_i^* | \mathbf{z}_{-i})}{p(z_i | \mathbf{z}_{-i})} \right\}$, where z_i and z_i^* correspond to the original and proposed values of Z_i . Note that sampling from both conditionals can be performed in the space of $\tau^{\mathbf{w}}(\mathbf{z})$ and locally.

Finally, we get a non-homogenous Markov chain with the acceptance probability $A(\mathbf{z}, \mathbf{z}^*) = \min \left\{ 1, \frac{p^{1/T_t-1}(z_i^* | \mathbf{z}_{-i})}{p^{1/T_t-1}(z_i | \mathbf{z}_{-i})} \right\}$ that converges to the vicinity of the most violated constraint. A similar chain was derived by Yuan *et al.* 2004 and applied to find the maximum a posteriori (MAP) configuration of random variables in Bayesian networks.

4.3 Constraint satisfaction

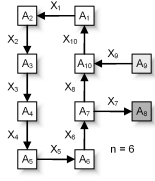
If the MCMC oracle converges to a violated constraint (not necessarily the most violated) in a polynomial time, it guarantees that the ellipsoid method can solve HALP in a polynomial time [Bertsimas and Tsitsiklis, 1997]. However, convergence of our chain within an arbitrary precision requires an exponential number of steps [Geman and Geman, 1984]. Even if this bound is too weak to be of practical interest, it suggests that the time complexity of finding a violated constraint dominates the time complexity of solving HALP. Therefore, the search for violated constraints should be performed efficiently. Convergence speedups that directly apply to our work include hybrid Monte Carlo (HMC) [Duane *et al.*, 1987], slice sampling [Higdon, 1998], and Rao-Blackwellization [Casella and Robert, 1996].

To evaluate the MCMC oracle, we embed it into the cutting plane method for solving linear programs, which results in a novel approach to solving HALP. As the oracle is not guaranteed to converge to the most violated constraint, we run the cutting plane method for a fixed number of iterations rather than having the same stopping criterion as Schuurmans and Patrascu 2002 (Section 3.2). Our MCMC solver differs from the Monte Carlo solver in that it samples constraints based on their potential to improve the existing solution, which substitutes for an unknown problem-specific sampling distribution. Comparing to the ε -HALP method, the MCMC oracle directly operates in the domains of continuous variables and its space complexity is linear in the number of variables.

5 Experiments

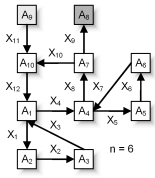
The performance of the MCMC solver is evaluated against two alternative constraint satisfaction techniques: ε -HALP and uniform Monte Carlo sampling. Due to space limitations, our comparison focuses on two irrigation-network problems [Guestrin *et al.*, 2004], but our conclusions are likely to generalize across a variety of control optimization tasks. The irrigation-network problems are challenging for state-of-the-art MDP solvers due to the factored state and action spaces

Ring topology



		$n = 6$			$n = 12$			$n = 18$		
		OV	Reward	Time	OV	Reward	Time	OV	Reward	Time
ε -HALP	$\varepsilon = 1/4$	24.3	35.1 ± 2.3	11	36.2	54.2 ± 3.0	43	48.0	74.5 ± 3.4	85
	$\varepsilon = 1/8$	55.4	40.1 ± 2.4	46	88.1	62.2 ± 3.4	118	118.8	84.9 ± 3.8	193
	$\varepsilon = 1/16$	59.1	40.4 ± 2.6	331	93.2	63.7 ± 2.8	709	126.1	86.8 ± 3.8	1 285
MCMC	$N = 10$	63.7	29.1 ± 2.9	43	88.0	51.8 ± 3.6	71	113.9	57.3 ± 4.6	101
	$N = 50$	69.7	41.1 ± 2.6	221	111.0	63.3 ± 3.4	419	149.0	84.8 ± 4.2	682
	$N = 250$	70.6	40.4 ± 2.6	1 043	112.1	63.0 ± 3.1	1 864	151.8	86.0 ± 3.9	2 954
MC		51.2	39.2 ± 2.8	1 651	66.6	60.0 ± 3.1	3 715	81.7	83.8 ± 4.3	5 178

Ring-of-rings topology



		$n = 6$			$n = 12$			$n = 18$		
		OV	Reward	Time	OV	Reward	Time	OV	Reward	Time
ε -HALP	$\varepsilon = 1/4$	28.4	40.1 ± 2.7	82	44.1	66.7 ± 2.7	345	59.8	93.1 ± 3.8	861
	$\varepsilon = 1/8$	65.4	48.0 ± 2.7	581	107.9	76.1 ± 3.8	2 367	148.8	104.5 ± 3.5	6 377
	$\varepsilon = 1/16$	68.9	47.1 ± 2.8	4 736	113.1	77.6 ± 3.7	22 699	156.9	107.8 ± 3.9	53 600
MCMC	$N = 10$	68.5	45.0 ± 2.7	69	99.9	67.4 ± 3.8	121	109.1	39.4 ± 4.1	173
	$N = 50$	81.1	47.4 ± 2.9	411	131.5	76.2 ± 3.7	780	182.7	104.3 ± 4.1	1 209
	$N = 250$	81.9	47.1 ± 2.5	1 732	134.0	78.2 ± 3.6	3 434	185.8	106.7 ± 4.1	5 708
MC		55.6	43.6 ± 2.9	2 100	73.7	74.8 ± 3.9	5 048	92.1	102.0 ± 4.2	6 897

Figure 1: Comparison of HALP solvers on two irrigation-network topologies of varying sizes (n). The solvers are compared by the objective value of a relaxed HALP (OV), the expected discounted reward of a corresponding policy, and computation time (in seconds). The expected discounted reward is estimated by the Monte Carlo simulation of 100 trajectories. The ε -HALP and MCMC solvers are parameterized by the resolution of ε -grid (ε) and the number of iterations (N).

(Figure 1). The goal of an irrigation network operator is to select discrete water-routing actions $\mathbf{A}_D$ to optimize continuous water levels $\mathbf{X}_C$ in multiple interconnected irrigation channels. The transition model is parameterized by beta distributions and represents water flows conditioned on the operation modes of regulation devices. The reward function is additive and given by a mixture of two normal distributions for each channel. The optimal value function is approximated by a linear combination of four univariate piecewise linear basis functions for each channel [Guestrin *et al.*, 2004]. The state relevance density function $\psi(\mathbf{x})$ is uniform. A comprehensive description of the irrigation-network problems can be found in Guestrin *et al.* 2004.

The ε -HALP solver is implemented with the method of Schuurmans and Patrascu 2002 as described in Section 3.2. The Monte Carlo solver uniformly generates one million constraints and establishes a baseline for the comparison to an uninformatively behaving sampling method. The chain of the MCMC oracle is simulated for 500 steps from the initial temperature $c = 0.2$, which yields a decreasing cooling schedule from $T_0 = 0.2$ to $T_{500} \approx 0.02$. These parameters were chosen empirically to demonstrate the characteristics of our approach rather than to maximize the performance of the MCMC oracle. All experiments were performed on a Dell Precision 340 workstation with 2GHz Pentium 4 CPU and 1GB RAM. All linear programs were solved by the simplex method in the LP-SOLVE package. The results of the experiments are reported in Figure 1.

Based on our results, we draw the following conclusions. First, the MCMC solver ($N = 250$) achieves the highest objective values on all problems. Higher objective values can be interpreted as closer approximations to the constraint space in HALP since the solvers operate on relaxed versions of HALP. Second, the quality of the MCMC policies ($N = 250$) surpasses the Monte Carlo ones while both solvers consume approximately the same computation time. This result is due to

the informative search for violated constraints in the MCMC solver. Third, the quality of the MCMC policies ($N = 250$) is close to the ε -HALP ones ($\varepsilon = 1/16$), but does not surpass them significantly. In the irrigation-network problems, the ε -HALP policies ($\varepsilon = 1/16$) are already close to optimal, and therefore hard to improve. Even if the MCMC solver reaches higher objective values than the ε -HALP solver, the policies may not improve due to the suggestive relationship between our true objective $\min_{\mathbf{w}} \|V^* - V^{\mathbf{w}}\|_{\infty, 1/L}$ and the objective of HALP $\min_{\mathbf{w}} \|V^* - V^{\mathbf{w}}\|_{1, \psi}$ [Guestrin *et al.*, 2004].

Finally, the computation time of the ε -HALP solver is seriously affected by the topologies of tested networks, which can be explained as follows. For a small ε and large n , the time complexity of formulating cost networks grows approximately by the rates of $(1/\varepsilon + 1)^2$ and $(1/\varepsilon + 1)^3$ for the ring and ring-of-rings topologies, respectively. The ε -HALP solver spends a significant amount of time by formulating cost networks, which makes its decent time complexity on the ring topology (quadratic in $1/\varepsilon + 1$) deteriorate on the ring-of-rings topology (cubic in $1/\varepsilon + 1$). A similar cross-topology comparison of the MCMC solver shows that its computation times differ only by a multiplicative factor of 2. This difference is due to the increased complexity of sampling $p(z_i^* | \mathbf{z}_{-i})$, which is caused by more complex local dependencies, and not the treewidth of the ring-of-rings topology.

6 Conclusions

Development of scalable algorithms for solving large factored MDPs is a challenging task. The MCMC approach presented in this paper is a small but important step in this direction. In particular, our method overcomes the limitations of existing approaches to solving HALP and works directly with continuous variables, generates constraints based on their potential to improve the existing solution, and its space complexity is linear in the number of variables. Moreover, the MCMC

solver seems to be less affected by larger treewidth than the ε -HALP method while delivering substantially better results than uniform Monte Carlo sampling. Empirical results on two large control problems confirm the expected benefits of the approach and its potential to tackle complex real-world optimization problems. The objective of our future research is to eliminate the assumptions placed on the transition model and basis functions, which would make the framework applicable to a broader class of problems.

Acknowledgment

This work was supported in part by National Science Foundation grants CMS-0416754 and ANI-0325353. The first author was supported by an Andrew Mellon Predoctoral Fellowship for the academic year 2004-05. We thank anonymous reviewers for providing insightful comments that led to the improvement of the paper.

References

- [Andrieu *et al.*, 2003] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50:5–43, 2003.
- [Bellman *et al.*, 1963] Richard Bellman, Robert Kalaba, and Bella Kotkin. Polynomial approximation - a new computational technique in dynamic programming. *Mathematics of Computation*, 17(8):155–161, 1963.
- [Bellman, 1957] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.
- [Bertsekas and Tsitsiklis, 1996] Dimitri Bertsekas and John Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.
- [Bertsimas and Tsitsiklis, 1997] Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Belmont, MA, 1997.
- [Boutilier *et al.*, 1995] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1104–1111, 1995.
- [Casella and Robert, 1996] George Casella and Christian Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1): 81–94, 1996.
- [de Farias and Roy, 2003] Daniela Pucci de Farias and Benjamin Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–856, 2003.
- [de Farias and Roy, 2004] Daniela Pucci de Farias and Benjamin Van Roy. On constraint sampling for the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, 29(3):462–478, 2004.
- [Dean and Kanazawa, 1989] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.
- [Duane *et al.*, 1987] Simon Duane, A. D. Kennedy, Brian Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222, 1987.
- [Feng *et al.*, 2004] Zhengzhu Feng, Richard Dearden, Nicolas Meuleau, and Richard Washington. Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 154–161, 2004.
- [Geman and Geman, 1984] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.
- [Guestrin *et al.*, 2001] Carlos Guestrin, Daphne Koller, and Ronald Parr. Max-norm projections for factored MDPs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 673–682, 2001.
- [Guestrin *et al.*, 2003] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.
- [Guestrin *et al.*, 2004] Carlos Guestrin, Milos Hauskrecht, and Branislav Kveton. Solving factored MDPs with continuous and discrete variables. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 235–242, 2004.
- [Hastings, 1970] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their application. *Biometrika*, 57:97–109, 1970.
- [Hauskrecht and Kveton, 2004] Milos Hauskrecht and Branislav Kveton. Linear program approximations for factored continuous-state Markov decision processes. In *Advances in Neural Information Processing Systems 16*, pages 895–902, 2004.
- [Higdon, 1998] David Higdon. Auxiliary variable methods for Markov chain Monte Carlo with applications. *Journal of the American Statistical Association*, 93(442):585–595, 1998.
- [Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220 (4598):671–680, 1983.
- [Kveton and Hauskrecht, 2004] Branislav Kveton and Milos Hauskrecht. Heuristic refinements of approximate linear programming for factored continuous-state Markov decision processes. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, pages 306–314, 2004.
- [Metropolis *et al.*, 1953] Nicholas Metropolis, Arianna Rosenbluth, Marshall Rosenbluth, Augusta Teller, and Edward Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.
- [Puterman, 1994] Martin Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York, NY, 1994.
- [Roy, 1998] Benjamin Van Roy. *Planning Under Uncertainty in Complex Structured Environments*. PhD thesis, Massachusetts Institute of Technology, 1998.
- [Schuurmans and Patrascu, 2002] Dale Schuurmans and Relu Patrascu. Direct value-approximation for factored MDPs. In *Advances in Neural Information Processing Systems 14*, 2002.
- [Schweitzer and Seidmann, 1985] Paul Schweitzer and Abraham Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110:568–582, 1985.
- [Yuan *et al.*, 2004] Changhe Yuan, Tsai-Ching Lu, and Marek Druzdzel. Annealed MAP. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 628–635, 2004.

BLOG: Probabilistic Models with Unknown Objects*

Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong and Andrey Kolobov

Computer Science Division

University of California

Berkeley, CA 94720-1776

{milch, bhaskara, russell, dsontag, dlong, karaya1}@cs.berkeley.edu

Abstract

This paper introduces and illustrates BLOG, a formal language for defining probability models over worlds with unknown objects and identity uncertainty. BLOG unifies and extends several existing approaches. Subject to certain acyclicity constraints, every BLOG model specifies a unique probability distribution over first-order model structures that can contain varying and unbounded numbers of objects. Furthermore, complete inference algorithms exist for a large fragment of the language. We also introduce a probabilistic form of Skolemization for handling evidence.

1 Introduction

Human beings and AI systems must convert sensory input into some understanding of what's out there and what's going on in the world. That is, they must make inferences about the objects and events that underlie their observations. No pre-specified list of objects is given; the agent must infer the existence of objects that were not known initially to exist.

In many AI systems, this problem of unknown objects is engineered away or resolved in a preprocessing step. However, there are important applications where the problem is unavoidable. *Population estimation*, for example, involves counting a population by sampling from it randomly and measuring how often the same object is resampled; this would be pointless if the set of objects were known in advance. *Record linkage*, a task undertaken by an industry of more than 300 companies, involves matching entries across multiple databases. These companies exist because of uncertainty about the mapping from observations to underlying objects. Finally, *multi-target tracking* systems perform *data association*, connecting, say, radar blips to hypothesized aircraft.

Probability models for such tasks are not new: Bayesian models for data association have been used since the 1960s [Sittler, 1964]. The models are written in English and mathematical notation and converted by hand into special-purpose code. In recent years, *formal representation languages* such as graphical models [Pearl, 1988] have led to general inference algorithms, more sophisticated models, and automated model selection (structure learning). In Sec. 7, we review several *first-order probabilistic languages* (FOPLs)

that explicitly represent objects and the relations between them. However, most FOPLs only deal with fixed sets of objects, or deal with unknown objects in limited and *ad hoc* ways. This paper introduces BLOG (Bayesian LOGic), a compact and intuitive language for defining probability distributions over outcomes with varying sets of objects.

We begin in Sec. 2 with three example problems, each of which involves possible worlds with varying object sets and identity uncertainty. We describe generative processes that produce such worlds, and give the corresponding BLOG models. Sec. 3 observes that these possible worlds are naturally viewed as model structures of *first-order logic*. It then defines precisely the set of possible worlds corresponding to a BLOG model. The key idea is a generative process that constructs a world by adding objects whose existence and properties depend on those of objects already created. In such a process, the existence of objects may be governed by many random variables, not just a single population size variable. Sec. 4 discusses how a BLOG model specifies a probability distribution over possible worlds.

Sec. 5 solves a previously unnoticed “probabilistic Skolemization” problem: how to specify evidence about objects—such as radar blips—that one didn’t know existed. Finally, Sec. 6 briefly discusses inference in unbounded outcome spaces, stating a sampling algorithm and a completeness theorem for a large class of BLOG models, and giving experimental results on one particular model.

2 Examples

In this section we examine three typical scenarios with unknown objects—simplified versions of the population estimation, record linkage, and multitarget tracking problems mentioned above. In each case, we provide a short BLOG model that, when combined with a suitable inference engine, constitutes a working solution for the problem in question.

Example 1. *An urn contains an unknown number of balls—say, a number chosen from a Poisson distribution. Balls are equally likely to be blue or green. We draw some balls from the urn, observing the color of each and replacing it. We cannot tell two identically colored balls apart; furthermore, observed colors are wrong with probability 0.2. How many balls are in the urn? Was the same ball drawn twice?*

The BLOG model for this problem, shown in Fig. 1, describes a stochastic process for generating worlds. The first 4

*This work was supported by DARPA under award 03-000219, and by an NSF Graduate Research Fellowship to B. Milch.

```

1 type Color; type Ball; type Draw;
2 random Color TrueColor(Ball);
3 random Ball BallDrawn(Draw);
4 random Color ObsColor(Draw);

5 guaranteed Color Blue, Green;
6 guaranteed Draw Draw1, Draw2, Draw3, Draw4;

7 #Ball ~ Poisson[6]();
8 TrueColor(b) ~ TabularCPD[[0.5, 0.5]]();
9 BallDrawn(d) ~ Uniform({Ball b});
10 ObsColor(d)
11   if (BallDrawn(d) != null) then
12     ~ TabularCPD[[0.8, 0.2], [0.2, 0.8]]
13     (TrueColor(BallDrawn(d)));

```

Figure 1: BLOG model for the urn-and-balls scenario of Ex. 1 with four draws.

lines introduce the types of objects in these worlds—colors, balls, and draws—and the functions that can be applied to these objects. Lines 5–7 specify what objects may exist in each world. In every world, the colors are blue and green and there are four draws; these are the *guaranteed* objects. On the other hand, different worlds have different numbers of balls, so the number of balls that exist is chosen from a prior—a Poisson with mean 6. Each ball is then given a color, as specified on line 8. Properties of the four draws are filled in by choosing a ball (line 9) and an observed color for that ball (lines 10–13). The probability of the generated world is the product of the probabilities of all the choices made.

Example 2. *We have a collection of citations that refer to publications in a certain field. What publications and researchers exist, with what titles and names? Who wrote which publication, and which publication does each citation refer to? For simplicity, we just consider the title and author-name strings in these citations, which are subject to errors of various kinds, and we assume only single-author publications.*

Fig. 2 shows a BLOG model for this example, based on the model in [Pasula *et al.*, 2003]. The BLOG model defines the following generative process. First, sample the total number of researchers from some distribution; then, for each researcher r , sample the number of publications by that researcher. Sample the publications’ titles and researchers’ names from appropriate prior distributions. Then, for each citation, sample the publication cited by choosing uniformly at random from the set of publications. Sample the title and author strings used in each citation from string corruption models conditioned on the true attributes of the cited publication; finally, generate the full citation string according to a formatting model.

Example 3. *An unknown number of aircraft exist in some volume of airspace. An aircraft’s state (position and velocity) at each time step depends on its state at the previous time step. We observe the area with radar: aircraft may appear as identical blips on a radar screen. Each blip gives the approximate position of the aircraft that generated it. However, some blips may be false detections, and some aircraft may not be detected. What aircraft exist, and what are their trajectories? Are there any aircraft that are not observed?*

The BLOG model for this scenario (Fig. 3) describes the following process: first, sample the number of aircraft in the

```

1 guaranteed Citation Cite1, Cite2, Cite3, Cite4;
2 #Researcher ~ NumResearchersDistrib();
3 #Publication: (Author) -> (r) ~ NumPubsDistrib();

4 Title(p) ~ TitlePrior();
5 Name(r) ~ NamePrior();

6 PubCited(c) ~ Uniform(Publication p);

7 TitleString(c) ~ TitleObs(Title(PubCited(c)));
8 AuthorString(c) ~ AuthorObs(Name(Author(PubCited(c))));
9 CitString(c) ~ CitDistrib(TitleString(c), AuthorString(c));

```

Figure 2: BLOG model for Ex. 2 with four observed citations (type and function declarations are omitted).

area. Then, for each time step t (starting at $t = 0$), choose the state (position and velocity) of each aircraft given its state at time $t - 1$. Also, for each aircraft a and time step t , possibly generate a radar blip r with $\text{Source}(r) = a$ and $\text{Time}(r) = t$. Whether a blip is generated or not depends on the state of the aircraft—thus the number of objects in the world depends on certain objects’ attributes. Also, at each step, generate some false alarm blips r' with $\text{Source}(r') = \text{null}$. Finally, sample the position for each blip given the true state of its source aircraft (or using a default distribution for a false-alarm blip).

3 Syntax and Semantics: Possible Worlds

3.1 Outcomes as first-order model structures

The possible outcomes for Ex. 1 through Ex. 3 are structures containing many related objects, which we will treat formally as *model structures* of *first-order logic*. A model structure provides interpretations for the symbols of a first-order language. (Usually, first-order languages are described as having predicate, function, and constant symbols. For conciseness, we view all symbols as function symbols; predicates are just functions that return a Boolean value, and constants are just zero-ary functions.)

For Ex. 1, the language has function symbols such as $\text{TrueColor}(b)$ for the true color of ball b ; $\text{BallDrawn}(d)$ for the ball drawn on draw d ; and Draw1 for the first draw. (Statisticians might use indexed families of random variables such as $\{\text{TrueColor}_b\}$, but this is mainly a matter of taste.)

```

1 type Aircraft; type Blip;
2 random R6Vector State(Aircraft, NaturalNum);
3 random R3Vector ApparentPos(Blip);

4 nonrandom NaturalNum Pred(NaturalNum) = Predecessor;

5 generating Aircraft Source(Blip);
6 generating NaturalNum Time(Blip);

7 #Aircraft ~ NumAircraftDistrib();

8 State(a, t)
9   if t = 0 then ~ InitState()
10  else ~ StateTransition(State(a, Pred(t)));

11 #Blip: (Source, Time) -> (a, t)
12   ~ DetectionDistrib(State(a, t));

13 #Blip: (BlipTime) -> (t)
14   ~ NumFalseAlarmsDistrib();

15 ApparentPos(r)
16   if (BlipSource(r) = null) then ~ FalseAlarmDistrib()
17   else ~ ObsDistrib(State(Source(r), Time(r)));

```

Figure 3: BLOG model for Ex. 3.

To eliminate meaningless random variables, we use a *typed, free* logical language, in which every object has a type and in which functions may be partial. Each function symbol f has a *type signature* $(\tau_0, \dots, \tau_k)$, where τ_0 is the return type of f and $\tau_1, \dots, \tau_k$ are the argument types. A partial function applied to arguments outside its domain returns the special value null, which is not of any type.

The truth of any first-order sentence is determined by a *model structure* for the corresponding language. A model structure specifies the *extension* of each type and the *interpretation* for each function symbol:

Definition 1. A model structure ω of a typed, free, first-order language consists of an extension $[\tau]^\omega$ for each type τ , which may be an arbitrary set, and an interpretation $[f]^\omega$ for each function symbol f . If f has return type τ_0 and argument types $\tau_1, \dots, \tau_k$, then $[f]^\omega$ is a function from $[\tau_1]^\omega \times \dots \times [\tau_k]^\omega$ to $[\tau_0]^\omega \cup \{\text{null}\}$.

A BLOG model defines a probability distribution over a particular set of model structures. In Ex. 1, identity uncertainty arises because $[\text{BallDrawn}]^\omega$ (Draw1) might be equal to $[\text{BallDrawn}]^\omega$ (Draw2) in one structure but not another. The set of balls, $[\text{Ball}]^\omega$, can also vary between structures.

3.2 Outcomes with fixed object sets

BLOG models for fixed object sets have five kinds of statements. A *type declaration*, such as the two statements on line 1 of Fig. 3, introduces a type. Certain types, namely Boolean, NaturalNum, Integer, String, Real, and RkVector (for each $k \geq 2$) are already provided. A *random function declaration*, such as line 2 of Fig. 3, specifies the type signature for a function symbol whose values will be chosen randomly in the generative process. A *nonrandom function definition*, such as the one on line 4 of Fig. 3, introduces a function whose interpretation is fixed in all possible worlds. In our implementation, the interpretation is given by a Java class (Predecessor in this example). A *guaranteed object statement*, such as line 5 in Fig. 1, introduces and names some distinct objects that exist in all possible worlds. For the built-in types, the obvious sets of guaranteed objects and constant symbols are predefined. The set of guaranteed objects of type τ in BLOG model M is denoted $G_M(\tau)$. Finally, for each random function symbol, a BLOG model includes a *dependency statement* specifying how values are chosen for that function. We postpone further discussion of dependency statements to Sec. 4.

The first four kinds of statements listed above define a particular typed first-order language $\mathcal{L}_M$ for a model M . The set of *possible worlds* of M , denoted Ω_M , consists of those model structures of $\mathcal{L}_M$ where the extension of each type τ is $G_M(\tau)$, and all nonrandom function symbols (including guaranteed constants) have their given interpretations.

For each random function f and tuple of appropriately typed guaranteed objects $o_1, \dots, o_k$, we can define a random variable (RV) $V_f[o_1, \dots, o_k](\omega) \triangleq [f]^\omega(o_1, \dots, o_k)$. The possible worlds are in one-to-one correspondence with full instantiations of these basic RVs. Thus, a joint distribution for the basic RVs defines a distribution over possible worlds.

3.3 Unknown objects

In general, a BLOG model defines a generative process in which objects are added iteratively to a world. To describe

such processes, we first introduce *generating function declarations*, such as lines 5–6 of Fig. 3. Unlike other functions, generating functions such as Source or Time have their values set when an object is added. A generating function must take a single argument of some type τ (namely Blip in the example); it is then called a τ -generating function.

Generative steps that add objects to the world are described by *number statements*. For instance, line 11 of Fig. 3 says that for each aircraft a and time step t , the process adds some number of blips r such that $\text{Source}(r) = a$ and $\text{Time}(r) = t$. In general, the beginning of a number statement has the form: $\# \tau: (g_1, \dots, g_k) \rightarrow (x_1, \dots, x_k)$ where τ is a type, $g_1, \dots, g_k$ are τ -generating functions, and $x_1, \dots, x_k$ are logical variables. (For types that are generated *ab initio* with no generating functions, the $() \rightarrow ()$ is omitted, as in Fig. 1.) The inclusion of a number statement means that for each appropriately typed tuple of objects $o_1, \dots, o_k$ (the appropriate types are the return types of $g_1, \dots, g_k$), the generative process adds some number (possibly zero) of objects q of type τ such that $[g_i]^\omega(q) = o_i$ for $i = 1, \dots, k$.

Object generation can even be recursive: for instance, in a model for phylogenetic trees, we could have a generating function Parent that takes a species and returns a species; then we could model speciation events with a number statement that begins: $\# \text{Species}: (\text{Parent}) \rightarrow (p)$.

We can also view number statements more declaratively:

Definition 2. Let ω be a model structure of $\mathcal{L}_M$, and consider a number statement for type τ with generating functions $g_1, \dots, g_k$. An object $q \in [\tau]^\omega$ satisfies this number statement applied to $o_1, \dots, o_k$ in ω if $[g_i]^\omega(q) = o_i$ for $i = 1, \dots, k$, and $[g]^\omega(q) = \text{null}$ for all other τ -generating functions g .

Note that if a number statement for type τ omits one of the τ -generating functions, then this function takes on the value null for all objects satisfying that number statement. For instance, Source is null for objects satisfying the false-detection number statement on line 13 of Fig. 3. Also, a BLOG model cannot contain two number statements with the same set of generating functions. This ensures that each object o has exactly one generation history, which can be found by tracing back the generating functions on o .

The set of possible worlds Ω_M is the set of model structures that can be constructed by M 's generative process. To complete the picture, we must explain not only *how many* objects are added on each step, but also *what* these objects are. It turns out to be convenient to define the generated objects as follows: when a number statement with type τ and generating functions $g_1, \dots, g_k$ is applied to generating objects $o_1, \dots, o_k$, the generated objects are tuples $\{(\tau, (g_1, o_1), \dots, (g_k, o_k), n) : n = 1, \dots, N\}$, where N is the number of objects generated. Thus in Ex. 3, the aircraft are pairs (Aircraft, 1), (Aircraft, 2), etc., and the blips generated by aircraft are nested tuples such as (Blip, (Source, (Aircraft, 2)), (Time, 8), 1). The tuple encodes the object's generation history; of course, it is purely internal to the semantics and remains invisible to the user.

The set of all objects (nested tuples) of type τ that can be generated from the guaranteed objects via finitely many recursive applications of number statements is called the *universe* of τ , and denoted $U_M(\tau)$. In each possible world, the extension of τ is some subset of $U_M(\tau)$.

Definition 3. For a BLOG model M , the set of possible worlds Ω_M is the set of model structures ω of $\mathcal{L}_M$ such that:

1. for each type τ , $G_M(\tau) \subseteq [\tau]^\omega \subseteq U_M(\tau)$;
2. nonrandom functions have the specified interpretations;
3. for each number statement in M with type τ and generating functions $g_1, \dots, g_k$, and each appropriately typed tuple of generating objects $(o_1, \dots, o_k)$ in ω , the set of objects in $[\tau]^\omega$ that satisfy this number statement applied to these generating objects is $\{(\tau, (g_1, o_1), \dots, (g_k, o_k), n) : n = 1, \dots, N\}$ for some natural number N ;
4. for every type τ , each element of $[\tau]^\omega$ satisfies some number statement applied to some objects in ω .

Note that by part 3 of this definition, the number of objects generated by any given application of a number statement in world ω is a finite number N . However, a world can still contain infinitely many non-guaranteed objects if some number statements are applied recursively, or are triggered for every natural number (like the ones generating radar blips in Ex. 3).

With a fixed set of objects, it was easy to define a set of basic RVs such that a full instantiation of the basic RVs uniquely identified a possible world. To achieve the same effect with unknown objects, we need two kinds of basic RVs:

Definition 4. For a BLOG model M , the set $\mathbf{V}_M$ of basic random variables consists of:

- for each random function f with type signature $(\tau_0, \dots, \tau_k)$ and each tuple of objects $(o_1, \dots, o_k) \in U_M(\tau_1) \times \dots \times U_M(\tau_k)$, a function application RV $V_f[o_1, \dots, o_k](\omega)$ that is equal to $[f]^\omega(o_1, \dots, o_k)$ if $o_1, \dots, o_k$ all exist in ω , and null (or false for Boolean RVs) otherwise;
- for each number statement with type τ and generating functions $g_1, \dots, g_k$ that have return types $\tau_1, \dots, \tau_k$, and each tuple of objects $(o_1, \dots, o_k) \in U_M(\tau_1) \times \dots \times U_M(\tau_k)$, a number RV $N_{(\tau, g_1, \dots, g_k)}[o_1, \dots, o_k](\omega)$ equal to the number of objects that satisfy this number statement applied to $o_1, \dots, o_k$ in ω .

Intuitively, each step in the generative world-construction process determines the value of a basic variable. The crucial result about basic RVs is the following:

Proposition 1. For any BLOG model M and any complete instantiation of $\mathbf{V}_M$, there is at most one model structure in Ω_M consistent with this instantiation.

Equating objects with tuples might seem unnecessarily complicated, but it becomes very helpful when we define a Bayes net over the basic RVs (which we do in Sec. 4.2). For instance, the sole parent of $V_{\text{ApparentPos}}[(\text{Blip}, (\text{Source}, (\text{Aircraft}, 2)), (\text{Time}, 8), 1)]$ is $V_{\text{State}}[(\text{Aircraft}, 2), 8]$. It might seem more elegant to assign numbers to objects as they are generated, so that the extension of each type in each possible world would be simply a prefix of the natural numbers. Specifically, we could number the aircraft arbitrarily, and then number the radar blips lexicographically by aircraft and time step. Then we would have basic RVs such as $V_{\text{ApparentPos}}[23]$, representing the apparent aircraft position for blip 23. But blip 23 could be generated by any aircraft at any time step. In fact, the parents of $V_{\text{ApparentPos}}[23]$ would have to include all the $\# \text{Blip}$ and State variables in the model. So, defining objects as tuples yields a much simpler Bayes net.

4 Syntax and Semantics: Probabilities

4.1 Dependency statements

Dependency and number statements specify exactly how the steps are carried out in our generative process. Consider the dependency statement for $\text{State}(a, t)$ beginning on line 8 of Fig. 3. This statement is applied for every basic RV of the form $V_{\text{State}}[a, t]$ where $a \in U_M(\text{Aircraft})$ and $t \in \mathbb{N}$. If $t = 0$, the conditional distribution for $\text{State}(a, t)$ is given by the elementary CPD InitState ; otherwise it is given by the elementary CPD StateTransition , which takes $\text{State}(a, \text{Pred}(t))$ as an argument. These elementary CPDs define distributions over objects of type R6Vector (the return type of State). In our implementation, elementary CPDs are Java classes with a method `getProb` that returns the probability of a particular value given a list of CPD arguments.

A dependency statement begins with a function symbol f and a tuple of logical variables $x_1, \dots, x_k$ representing the arguments to this function. In a number statement, the variables $x_1, \dots, x_k$ represent the generating objects. In either case, the rest of the statement consists of a sequence of *clauses*. When the statement is not abbreviated, the syntax for the first clause is `if cond then ~ elem-cpd(arg1, ..., argN)`. The *cond* portion is a formula of $\mathcal{L}_M$ (in which only $x_1, \dots, x_k$ can occur as free variables) specifying the condition under which this clause should be used to sample a value for a basic RV. More precisely, if the possible world constructed so far is ω , then the applicable clause is the *first* one whose condition is satisfied in ω (assuming for the moment that ω is complete enough to determine the truth values of the conditions). If no clause's condition is satisfied, or if the basic RV refers to objects that do not exist in ω , then the value is set by default to false for Boolean functions, null for other functions, and zero for number variables. If the condition in a clause is just “true”, then the whole string “if true then” may be omitted.

In the applicable clause, each CPD argument is evaluated in ω . The resulting values are then passed to the elementary CPD. In the simplest case, the arguments are terms or formulas of $\mathcal{L}_M$, such as $\text{State}(a, \text{Pred}(t))$. An argument can also be a *set expression* of the form $\{\tau \ y : \varphi\}$, where τ is a type, y is a logical variable, and φ is a formula. The value of such an expression is the set of objects $o \in [\tau]^\omega$ such that ω satisfies φ with y bound to o . If the formula φ is just true it can be omitted: this is the case on line 9 of Fig. 1.

We require that the elementary CPDs obey two rules related to non-guaranteed objects. First, a CPD should never assign positive probability to objects that do not exist in the partially completed world ω . Thus, we allow an elementary CPD to assign positive probability to a non-guaranteed object only if the object was passed in as a CPD argument. Second, an elementary CPD cannot “peek” at the tuple representations of objects that are passed in: it must be invariant to permutations of the non-guaranteed objects.

4.2 Declarative semantics

So far we have explained BLOG semantics procedurally, in terms of a generative process. To facilitate both knowledge engineering and the development of learning algorithms, we would like to have declarative semantics. The standard approach — which is used in most existing FOPLs — is to say

that a BLOG model defines a certain Bayesian network (BN) over the basic RVs. In this section we discuss how that approach needs to be modified for BLOG.

We will write σ to denote an instantiation of a set of RVs $\text{vars}(\sigma)$, and σ_X to denote the value that σ assigns to X . If a BN is finite, then the probability it assigns to each complete instantiation σ is $P(\sigma) = \prod_{X \in \text{vars}(\sigma)} p_X(\sigma_X | \sigma_{\text{Pa}(X)})$, where p_X is the CPD for X and $\sigma_{\text{Pa}(X)}$ is σ restricted to the parents of X . In an infinite BN, we can write a similar expression for each *finite* instantiation σ that is closed under the parent relation (that is, $X \in \text{vars}(\sigma)$ implies $\text{Pa}(X) \subseteq \text{vars}(\sigma)$). If the BN is acyclic and each variable has finitely many ancestors, then these probability assignments define a unique distribution [Kersting and De Raedt, 2001].

The difficulty is that in the BN corresponding to a BLOG model, variables often have infinite parent sets. For instance, the BN for Ex. 1 has an infinite number of basic RVs of the form $V_{\text{TrueColor}}[b]$: if it had only a finite number N of these RVs, it could not represent outcomes with more than N balls. Furthermore, each of these $V_{\text{TrueColor}}[b]$ RVs is a parent of each $V_{\text{ObsColor}}[d]$ RV, since if $\text{BallDrawn}(d)$ happens to be b , then the observed color on draw d depends directly on the color of ball b . So the $V_{\text{ObsColor}}[d]$ nodes have infinitely many parents. In such a model, assigning probabilities to finite instantiations that are closed under the parent relation does not define a unique distribution: in particular, it tells us nothing about the $V_{\text{ObsColor}}[d]$ variables.

We required instantiations to be closed under the parent relation so that the factors $p_X(\sigma_X | \sigma_{\text{Pa}(X)})$ would be well-defined. But we may not need the values of *all* of X 's parents in order to determine the conditional distribution for X . For instance, knowing $V_{\text{BallDrawn}}[d] = (\text{Ball}, 13)$ and $V_{\text{TrueColor}}[(\text{Ball}, 13)] = \text{Blue}$ is sufficient to determine the distribution for $V_{\text{ObsColor}}[d]$: the colors of all the other balls are irrelevant in this context. We can read off this context-specific independence from the dependency statement for ObsColor in Fig. 1 by noting that the instantiation ($V_{\text{BallDrawn}}[d] = (\text{Ball}, 13)$, $V_{\text{TrueColor}}[(\text{Ball}, 13)] = \text{Blue}$) determines the value of the sole CPD argument $\text{TrueColor}(\text{BallDrawn}(d))$. We say this instantiation *supports* the variable $V_{\text{ObsColor}}[d]$ (see [Milch et al., 2005]).

Definition 5. An instantiation σ supports a basic RV V of the form $V_f[o_1, \dots, o_k]$ or $N_p[o_1, \dots, o_k]$ if all possible worlds consistent with σ agree on (1) whether all the objects $o_1, \dots, o_k$ exist, and, if so, on (2) the applicable clause in the dependency or number statement for V and the values for the CPD arguments in that clause.

Note that some RVs, such as $N_{\text{Ball}}[]$ in Ex. 1, are supported by the empty instantiation. We can now generalize the notion of being closed under the parent relation.

Definition 6. A finite instantiation σ is self-supporting if its instantiated variables can be numbered $X_1, \dots, X_N$ such that for each $n \leq N$, the restriction of σ to $\{X_1, \dots, X_{n-1}\}$ supports X_n .

This definition lets us give semantics to BLOG models in a way that is meaningful even when the corresponding BNs contain infinite parent sets. We will write $p_V(v | \sigma)$ for the probability that V 's dependency or number statement assigns to the value v , given an instantiation σ that supports V .

Definition 7. A distribution P over Ω_M satisfies a BLOG model M if for every finite, self-supporting instantiation σ with $\text{vars}(\sigma) \subseteq \mathbf{V}_M$:

$$P(\Omega_\sigma) = \prod_{n=1}^N p_{X_n}(\sigma_{X_n} | \sigma_{\{X_1, \dots, X_{n-1}\}})$$

where Ω_σ is the set of possible worlds consistent with σ and $X_1, \dots, X_N$ is a numbering of σ as in Def. 6.

A BLOG model is *well-defined* if there is exactly one probability distribution that satisfies it. Recall that a BN is well-defined if it is acyclic and each variable has a finite set of ancestors. Another way of saying this is that each variable can be “reached” by enumerating its ancestors in a finite, topologically ordered list. The well-definedness criterion for BLOG is similar, but deals with finite, self-supporting instantiations rather than finite, topologically ordered lists of variables.

Theorem 1. Let M be a BLOG model. Suppose that $\mathbf{V}_M$ is at most countably infinite,¹ and for each $V \in \mathbf{V}_M$ and $\omega \in \Omega_M$, there is a self-supporting instantiation that agrees with ω and includes V . Then M is well-defined.

The theorem follows from a result in [Milch et al., 2005] that deals with distributions over full instantiations of a set of RVs. Prop. 1 makes the connection between full instantiations of $\mathbf{V}_M$ and possible worlds.

To check that the criterion of Thm. 1 holds for a particular example, we need to consider each basic RV. In Ex. 1, the number RV for balls is supported by the empty instantiation, so in every world it is part of a self-supporting instantiation of size one. Each $\text{TrueColor}(b)$ RV depends only on whether its argument exists, so these variables participate in self-supporting instantiations of size two. Similarly, each BallDrawn variable depends only on what balls exist. To sample an $\text{ObsColor}(d)$ variable, we need to know $\text{BallDrawn}(d)$ and $\text{TrueColor}(\text{BallDrawn}(d))$, so these variables are in self-supporting instantiations of size four. Similar arguments can be made for Examples 2 and 3. Of course, we would like to have an algorithm for checking whether a BLOG model is well-defined; the criteria given in Thm. 2 in Sec. 6.2 are a first step in this direction.

5 Evidence and Queries

Because a well-defined BLOG model M defines a distribution over model structures, we can use arbitrary sentences of $\mathcal{L}_M$ as evidence and queries. But sometimes such sentences are not enough. In Ex. 3, the user observes radar blips, which are not referred to by any terms in the language. The user could assert evidence about the blips using existential quantifiers, but then how could he make a query of the form, “Did *this* blip come from the same aircraft as *that* blip?”

A natural solution is to allow the user to extend the language when evidence arrives, adding constant symbols to refer to observed objects. In many cases, the user observes some new objects, introduces some new symbols, and assigns the symbols to the objects in an uninformative order. To handle such cases, BLOG includes a special macro. For instance, given 4 radar blips at time 8, one can assert:

$$\{\text{Blip } r: \text{Time}(r) = 8\} = \{\text{Blip1}, \text{Blip2}, \text{Blip3}, \text{Blip4}\};$$

¹This is satisfied if the *Real* and *RkVector* types are not arguments to random functions or return types of generating functions.

This introduces the new constants $\text{Blip1}, \dots, \text{Blip4}$ and asserts that there are exactly 4 radar blips at time 8.

Formally, the macro augments the model with dependency statements for the new symbols. The statements implement sampling without replacement; for our example, we have

```
Blip1 ~ Uniform({Blip r : {Time(r) = 8}});
Blip2 ~ Uniform({Blip r : {Time(r) = 8} & (Blip1 != r)});
```

and so on. Once the model has been extended this way, the user can make assertions about the apparent positions of Blip1 , Blip2 , etc., and then use these symbols in queries.

These new constants resemble Skolem constants, but conditioning on assertions about the new constants is *not* the same as conditioning on an existential sentence. For example, suppose you go into a new wine shop, pick up a bottle at random, and observe that it costs \$40. This scenario is correctly modeled by introducing a new constant Bottle1 with a Uniform CPD. Then observing that Bottle1 costs over \$40 suggests that this is a fancy wine shop. On the other hand, the mere *existence* of a \$40+ bottle does not suggest this, because almost every shop has *some* bottle at over \$40.

6 Inference

Because the set of basic RVs of a BLOG model can be infinite, it is not obvious that inference for well-defined BLOG models is even decidable. However, the generative process intuition suggests a rejection sampling algorithm. We present this algorithm not because it is particularly efficient, but because it demonstrates the decidability of inference for a large class of BLOG models (see Thm. 2 below) and illustrates several issues that any BLOG inference algorithm must deal with. At the end of this section, we present experimental results from a somewhat more efficient likelihood weighting algorithm.

6.1 Rejection sampling

Suppose we are given a partial instantiation e as evidence, and a query variable Q . Our rejection sampling algorithm starts by imposing an arbitrary numbering on the basic RVs. To generate each sample, it starts with an empty instantiation σ . Then it repeats the following process: scan the basic RVs in the imposed order until we reach the first RV V that is supported by σ but not already instantiated in σ ; sample a value v for V according to V 's dependency statement; and augment σ with the assignment $V = v$. It continues until all the query and evidence variables have been sampled. If the sample is consistent with the evidence e , then it increments a counter N_q , where q is the sampled value of Q . Otherwise, it rejects this sample. After N accepted samples, the estimate of $P(Q = q \mid e)$ is N_q/N .

This algorithm requires a subroutine that determines whether a partial instantiation σ supports a basic RV V , and if so, returns a sample from V 's conditional distribution. For a basic RV V of the form $V_f[o_1, \dots, o_k]$ or $N_p[o_1, \dots, o_k]$, the subroutine begins by checking the values of the relevant number variables in σ to determine whether all of $o_1, \dots, o_k$ exist. If some of these number variables are not instantiated, then σ does not support V . If some of $o_1, \dots, o_k$ do not exist, the subroutine returns the default value for V . If they do all exist, the subroutine follows the semantics for dependency statements discussed in Sec. 4.1. First, it iterates over the clauses in the dependency (or number) statement until it

reaches a clause whose condition is either undetermined or determined to be true given σ (if all the conditions are determined to be false, then it returns the default value for V). If the condition is undetermined, then σ does not support V . If it is determined to be true, then the subroutine evaluates each of the CPD arguments in this clause. If σ determines the values of all the arguments, then the subroutine samples a value for V by passing those values to the `sample` method of this clause's elementary CPD. Otherwise, σ does not support V .

To evaluate terms and quantifier-free formulas, we use a straightforward recursive algorithm. The base case looks up the value of a particular function application RV in σ and returns "undetermined" if it is not instantiated. A formula may be determined even if some of its subformulas are not determined: for example, $\alpha \wedge \beta$ is determined to be false if α is false. It is more complicated to evaluate set expressions such as $\{\text{Blip } r : \text{Time}(r) = 8\}$, which can be used as CPD arguments. A naive algorithm for evaluating this expression would first enumerate all the objects of type Blip (which requires certain number variables to be instantiated), then select the blips r that satisfy $\text{Time}(r) = 8$. But Fig. 3 specifies that there may exist some blips for each aircraft a and each natural number t : since there are infinitely many natural numbers, some worlds contain infinitely many blips. Fortunately, the number of blips r with $\text{Time}(r) = 8$ is necessarily finite: in every world there are a finite number of aircraft, and each one generates a finite number of blips at time 8. We have an algorithm that scans the formula within a set expression for *generating function restrictions* such as $\text{Time}(r) = 8$, and uses them to avoid enumerating infinite sets when possible. A similar method is used for evaluating quantified formulas.

6.2 Termination criteria

In order to generate each sample, the algorithm above repeatedly instantiates the first variable that is supported but not yet instantiated, until it instantiates all the query and evidence variables. When can we be sure that this will take a finite amount of time? The first way this process could fail to terminate is if it goes into an infinite loop while checking whether a particular variable is supported. This happens if the program ends up enumerating an infinite set while evaluating a set expression or quantified formula. We can avoid this by ensuring that all such expressions in the BLOG model are finite once generating function restrictions are taken into account.

The sample generator also fails to terminate if it never constructs an instantiation that supports a particular query or evidence variable. To see how this can happen, consider calling the subroutine described above to sample a variable V . If V is not supported, the subroutine will realize this when it encounters a variable U that is relevant but not instantiated. Now consider a graph over basic variables where we draw an edge from U to V when the evaluation process for V hits U in this way. If a variable is never supported, then it must be part of a cycle in this graph, or part of a receding chain of variables $V_1 \leftarrow V_2 \leftarrow \dots$ that is extended infinitely.

The graph constructed in this way varies from sample to sample: for instance, sometimes the evaluation process for $V_{\text{ObsColor}}[d]$ will hit $V_{\text{TrueColor}}[(\text{Ball}, 7)]$, and sometimes it will hit $V_{\text{TrueColor}}[(\text{Ball}, 13)]$. However, we can rule out cycles and infinite receding chains in all these graphs by considering a more abstract graph over function symbols and types.

Definition 8. The symbol graph for a BLOG model M is a directed graph whose nodes are the types and random function symbols of M , where the parents of a type τ or function symbol f are:

- the random function symbols that occur on the right hand side of the dependency statement for f or some number statement for τ ;
- the types of variables that are quantified over in formulas or set expressions on the right hand side of such a statement;
- the types of the arguments for f or the return types of generating functions for τ .

If the sampling subroutine for a basic RV V hits a basic RV U , then there must be an edge from U 's function symbol (or type, if U is a number RV) to V 's function symbol (or type) in the symbol graph. This property, along with ideas from [Milch *et al.*, 2005], allows us to prove the following:

Theorem 2. Suppose M is a BLOG model where:

1. uncountable built-in types do not serve as function arguments or as the return types of generating functions;
2. each quantified formula and set expression ranges over a finite set once generating function restrictions are taken into account;
3. the symbol graph is acyclic.

Then M is well-defined. Also, for any evidence instantiation e and query variable Q , the rejection sampling algorithm described in Sec. 6.1 converges to the posterior $P(Q|e)$ defined by the model, taking finite time per sampling step.

The criteria in Thm. 2 are very conservative: in particular, when we construct the symbol graph, we ignore all structure in the dependency statements and just check for the occurrence of function and type symbols. These criteria are satisfied by the models in Figures 1 and 2. However, the model in Fig. 3 does not satisfy the criteria because there is a self-loop from State to State. The criteria do not exploit the fact that $\text{State}(a, t)$ depends only on $\text{State}(a, \text{Pred}(t))$, and the non-random function Pred is acyclic. Friedman *et al.* [1999] have already dealt with this issue in the context of probabilistic relational models; their algorithm can be adapted to obtain a stronger version of Thm. 2 that covers Fig. 3.

6.3 Experimental results

Milch *et al.* [2005] describe a guided likelihood weighting algorithm that uses backward chaining from the query and evidence nodes to avoid sampling irrelevant variables. This algorithm can also be adapted to BLOG models. We applied this algorithm for Ex. 1, asserting that 10 balls were drawn and all appeared blue, and querying the number of balls in the urn. The top graph of Fig. 4 shows that when the prior for the number of balls is uniform over $\{1, \dots, 8\}$, the posterior puts more weight on small numbers of balls; this makes sense because the more balls there are in the urn, the less likely it is that they are all blue. The bottom graph, using a Poisson(6) prior, shows a similar but less pronounced effect. Note that the posterior probabilities computed by the likelihood weighting algorithm are very close to the exact values (computed by exhaustive enumeration). These results could not be obtained using any algorithm that constructed a single fixed BN, since the number of potentially relevant $V_{\text{TrueColor}}[b]$ variables is infinite in the Poisson case.

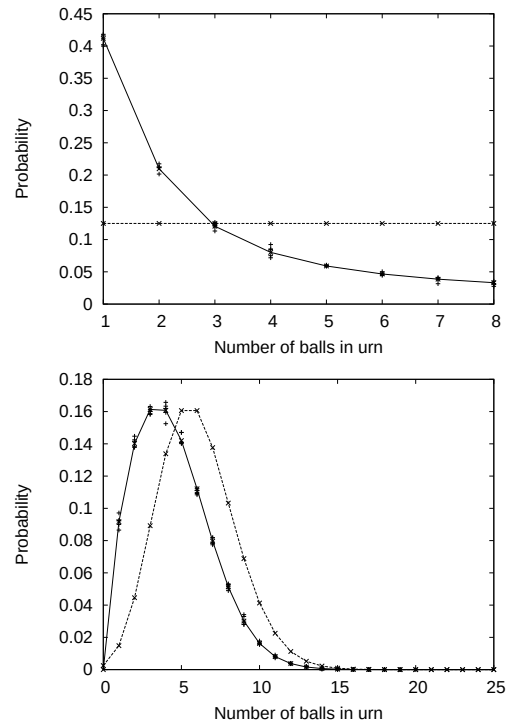


Figure 4: Distribution for the number of balls in the urn (Ex. 1). Dashed lines are the uniform prior (top) or Poisson prior (bottom); solid lines are the exact posterior given that 10 balls were drawn and all appeared blue; and plus signs are posterior probabilities computed by 5 independent runs of 20,000 samples (top) or 100,000 samples (bottom).

7 Related Work

Gaifman [1964] was the first to suggest defining a probability distribution over first-order model structures. Halpern [1990] defines a language in which one can make statements about such distributions: for instance, that the probability of the set of worlds that satisfy $\text{Flies}(\text{Tweety})$ is 0.8. *Probabilistic logic programming* [Ng and Subrahmanian, 1992] can be seen as an application of this approach to Horn-clause knowledge bases. Such an approach only defines *constraints* on distributions, rather than defining a unique distribution.

Most first-order probabilistic languages (FOPLs) that define unique distributions fix the set of objects and the interpretations of (non-Boolean) function symbols. Examples include relational Bayesian networks [Jaeger, 2001] and Markov logic models [Domingos and Richardson, 2004]. Prolog-based languages such as probabilistic Horn abduction [Poole, 1993], PRISM [Sato and Kameya, 2001], and Bayesian logic programs [Kersting and De Raedt, 2001] work with *Herbrand models*, where the objects are in one-to-one correspondence with the ground terms of the language.

There are a few FOPLs that allow explicit *reference uncertainty*, i.e., uncertainty about the interpretations of function symbols. Among these are two languages that use indexed RVs rather than logical notation: BUGS [Gilks *et al.*, 1994] and indexed probability diagrams (IPDs) [Mjolsness, 2004]. Reference uncertainty can also be represented in probabilistic relational models (PRMs) [Koller and Pfeffer, 1998], where

a “single-valued complex slot” corresponds to an uncertain unary function. PRMs are unfortunately restricted to unary functions (attributes) and binary predicates (relations). Probabilistic entity-relationship models [Heckerman *et al.*, 2004] lift this restriction, but represent reference uncertainty using relations (such as $\text{Drawn}(d, b)$) and special mutual exclusivity constraints, rather than with functions such as $\text{BallDrawn}(d)$. Multi-entity Bayesian network logic (MEBN) [Laskey, 2004] is similar to BLOG in allowing uncertainty about the values of functions with any number of arguments.

The need to handle unknown objects has been appreciated since the early days of FOPL research: Charniak and Goldman’s plan recognition networks (PRNs) [1993] can contain unbounded numbers of objects representing hypothesized plans. However, external rules are used to decide what objects and variables to include in a PRN. While each possible PRN defines a distribution on its own, Charniak and Goldman do not claim that the various PRNs are all approximations to some single distribution over outcomes.

Some more recent FOPLs do define a single distribution over outcomes with varying objects. IPDs allow uncertainty over the index range for an indexed family of RVs. PRMs and their extensions allow a variety of forms of uncertainty about the number (or existence) of objects satisfying certain relational constraints [Koller and Pfeffer, 1998; Getoor *et al.*, 2001] or belonging to each type [Pasula *et al.*, 2003]. However, there is no unified syntax or semantics for dealing with unknown objects in PRMs. MEBNs take yet another approach: an MEBN model includes a set of unique identifiers, for each of which there is an “identity” RV indicating whether the named object exists.

Our approach to unknown objects in BLOG can be seen as unifying the PRM and MEBN approaches. Number statements neatly generalize the various ways of handling unknown objects in PRMs: number uncertainty [Koller and Pfeffer, 1998] corresponds to a number statement with a single generating function; existence uncertainty [Getoor *et al.*, 2001] can be modeled with two or more generating functions (and a CPD whose support is $\{0, 1\}$); and domain uncertainty [Pasula *et al.*, 2003] corresponds to a number statement with no generating functions. There is also a correspondence between BLOG and MEBN logic: the tuple representations in a BLOG model can be thought of as unique identifiers in an MEBN model. The difference is that BLOG determines which objects actually exist in a world using number variables rather than individual existence variables.

Finally, it is informative to compare BLOG with the IBAL language [Pfeffer, 2001], in which a program defines a distribution over outputs that can be arbitrary nested data structures. An IBAL program could implement a BLOG-like generative process with the outputs viewed as logical model structures. But the declarative semantics of such a program would be less clear than the corresponding BLOG model.

8 Conclusion

BLOG is a representation language for probabilistic models with unknown objects. It contributes to the solution of a very general problem in AI: intelligent systems must represent and reason about objects, but those objects may not be known *a priori* and may not be directly and uniquely identified by

perceptual processes. Our approach defines generative models that create first-order model structures by adding objects and setting function values; everything else follows naturally from this design decision. Much remains to be done, especially on inference: we expect to employ MCMC with user-defined and possibly adaptive proposal distributions, and to develop algorithms that work directly with objects rather than at the lower level of basic RVs.

References

- [Charniak and Goldman, 1993] E. Charniak and R. P. Goldman. A Bayesian model of plan recognition. *AIJ*, 64(1):53–79, 1993.
- [Domingos and Richardson, 2004] P. Domingos and M. Richardson. Markov logic: A unifying framework for statistical relational learning. In *Proc. ICML Wksp on Statistical Relational Learning and Its Connections to Other Fields*, 2004.
- [Friedman *et al.*, 1999] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proc. 16th IJCAI*, pages 1300–1307, 1999.
- [Gaifman, 1964] H. Gaifman. Concerning measures in first order calculi. *Israel J. Math.*, 2:1–18, 1964.
- [Getoor *et al.*, 2001] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of relational structure. In *Proc. 18th ICML*, pages 170–177, 2001.
- [Gilks *et al.*, 1994] W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43(1):169–177, 1994.
- [Halpern, 1990] J. Y. Halpern. An analysis of first-order logics of probability. *AIJ*, 46:311–350, 1990.
- [Heckerman *et al.*, 2004] D. Heckerman, C. Meek, and D. Koller. Probabilistic models for relational data. Technical Report MSR-TR-2004-30, Microsoft Research, 2004.
- [Jaeger, 2001] M. Jaeger. Complex probabilistic modeling with recursive relational Bayesian networks. *Annals of Math and AI*, 32:179–220, 2001.
- [Kersting and De Raedt, 2001] K. Kersting and L. De Raedt. Adaptive Bayesian logic programs. In *11th Int. Conf. on ILP*, 2001.
- [Koller and Pfeffer, 1998] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proc. 15th AAAI*, pages 580–587, 1998.
- [Laskey, 2004] K. B. Laskey. MEBN: A logic for open-world probabilistic reasoning. Technical report, George Mason Univ., 2004.
- [Milch *et al.*, 2005] B. Milch, B. Marthi, D. Sontag, S. Russell, D. L. Ong, and A. Kolobov. Approximate inference for infinite contingent Bayesian networks. In *10th AISTATS Wksp*, 2005.
- [Mjolsness, 2004] E. Mjolsness. Labeled graph notations for graphical models. Technical Report 04-03, School of Information and Computer Science, UC Irvine, 2004.
- [Ng and Subrahmanian, 1992] R. T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.
- [Pasula *et al.*, 2003] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In *NIPS 15*. MIT Press, Cambridge, MA, 2003.
- [Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Francisco, revised edition, 1988.
- [Pfeffer, 2001] A. Pfeffer. IBAL: A probabilistic rational programming language. In *Proc. 17th IJCAI*, pages 733–740, 2001.
- [Poole, 1993] D. Poole. Probabilistic Horn abduction and Bayesian networks. *AIJ*, 64(1):81–129, 1993.
- [Sato and Kameya, 2001] T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *JAIR*, 15:391–454, 2001.
- [Sittler, 1964] R. W. Sittler. An optimal data association problem in surveillance theory. *IEEE Trans. Military Electronics*, MIL-8:125–139, 1964.

Continuous Time Particle Filtering

Brenda Ng

Harvard University
Cambridge, MA 02138
bmng@eecs.harvard.edu

Avi Pfeffer

Harvard University
Cambridge, MA 02138
avi@eecs.harvard.edu

Richard Dearden

University of Birmingham
Birmingham B15 2TT, UK
R.W.Dearden@cs.bham.ac.uk

Abstract

We present the continuous-time particle filter (CTPF) – an extension of the discrete-time particle filter for monitoring continuous-time dynamic systems. Our methods apply to hybrid systems containing both discrete and continuous variables. The dynamics of the discrete state system are governed by a Markov jump process. Observations of the discrete process are intermittent and irregular. Whenever the discrete process is observed, CTPF samples a trajectory of the underlying Markov jump process. This trajectory is then used to estimate the continuous variables using the system dynamics determined by the discrete state in the trajectory. We use the unscented Kalman-Bucy filter to handle nonlinearities and continuous time. We present results showing that CTPF is more stable in its performance than discrete-time particle filtering, even when the discrete-time algorithm is allowed to update many more times than CTPF. We also present a method for online learning of the Markov jump process model that governs the discrete states.

1 Introduction

The work described here is directly motivated by problems encountered in creating a state estimation system for the K-9 experimental Mars rover at NASA Ames Research Center [Willeke and Dearden, 2004]. As with many robotic systems, K-9 contains numerous sensors that report on its performance. These sensors produce telemetry at different rates, and frequently at rates that vary over time. As a result, using a fixed time-step state estimator, for example a Kalman filter, is extremely difficult. In this paper we describe a continuous-time state estimation algorithm based on particle filtering that can handle telemetry arriving at different rates and changes in system behavior between observations. The system can also update its model of system behavior to improve its performance and to detect gradual degradation of the system.

State estimation is typically done either using discrete diagnosis algorithms such as Livingstone [Williams and Nayak, 1996], or more recently, using hybrid system representations. The discrete approach is well suited for domains where there are occasional transitions between relatively stable states, and where sensor readings tend to be very reliable. Unfortunately,

this is not the case for K-9 and discrete approaches have largely proven unhelpful [Washington *et al.*, 2000]. For this reason, we use a hybrid system model for state representation and particle filtering for state estimation.

Probabilistic hybrid automata (PHAs) [Hofbaur and Williams, 2002] are commonly used for representing hybrid systems in diagnosis and state estimation (other representations have similar expressive power). In the PHA model, a system consists of a set of discrete *modes* and a set of continuous variables. For each mode, a set of differential equations describes the continuous behavior in that mode. Transitions between modes are stochastic; a transition matrix represents the probability of being in state s' at time $t + 1$ given that the system is in state s at time t . Transition probabilities may depend on the values of the continuous variables.

The problem with applying the PHA model (and other similar representations) to systems such as K-9 is the assumption that a transition matrix applied at fixed time intervals is sufficient to describe the evolution of the system. This implies that only one discrete transition can occur per interval, and also that it doesn't matter when in the interval the transition occurs. These assumptions lead to two problems. First, they make the system model unwieldy as they force a single transition matrix to represent the effects of multiple discrete transitions. Second, they reduce efficiency by forcing model steps to be small enough for the assumptions to be reasonable. The continuous time model avoids these assumptions, allowing state updates to be performed only when a new observation arrives or when a significant event occurs.

The approach we take is to build a continuous-time model of the system (see Section 2). As before, there is a differential equation model governing the continuous system behavior for every discrete system mode, but now the discrete state transitions are represented as a continuous-time Markov jump process. That is, we keep a set of transition probabilities for each discrete mode, as well as the distribution over *how long* the system will remain in the mode before transitioning out.

As we said above, we use particle filtering to track the system state (see Section 4). For efficiency, and to make it possible to track large systems, we use Rao-Blackwellized particle filters [Doucet *et al.*, 2000a], in which the discrete mode is sampled, just as in a standard particle filter, but the continuous state is tracked using an unscented Kalman-Bucy filter.

This work addresses the key challenge that many mode changes on the rover, as with most systems, are not observable directly. Commanded changes such as moving from

stopped to driving are observable, but the occurrence of faults, and other mode transitions triggered by the environment or the continuous state, do not produce telemetry directly, and can only be inferred from their effects on the continuous system behavior. The algorithm we have developed to address these difficulties, the *continuous-time particle filter*, is described in Section 5.

One major advantage of the continuous-time approach is that it allows resource-bounded computation. Although the cost of a single update may be higher than that of a discrete update (because of the Kalman-Bucy filter equations), computation need only be performed when new observations arrive. Indeed, if computation is limited, observations can even be ignored until sufficient computation is available to update the estimate. For discrete approaches, an update must be performed for each time step, even if no telemetry has arrived.

In Section 6, we discuss how the parameters of the continuous-time Markov jump process can be learned by observing the system. Section 7 applies the approach to the simulated rover model presented in Section 3.

2 Continuous-time hybrid-state processes

A continuous-time hybrid-state process consists of interacting discrete-state and continuous-state random variables.

The discrete-state variables $\mathbf{Z}_t$ evolve according to dynamics described by continuous-time Markov jump processes. A Markov jump process is a random variable Z_t that is parametrized by time $t \in [0, \infty)$. Z_t starts in an initial state z and remains in this state for a random amount of time before it makes a transition to a different state. The time that Z_t stays in a particular state is exponentially distributed, due to the Markovian property of the process.

Mathematically, a Markov jump process Z_t is characterized by its *intensity matrix*

$$Q = \begin{bmatrix} -q_1 & q_{12} & \dots \\ q_{12} & -q_2 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

where

$$q_{ij} = \lim_{\Delta t \rightarrow 0} \frac{P\{Z_{t+\Delta t} = j | Z_t = i\}}{\Delta t}, \quad i \neq j \quad \text{and} \quad q_i = \sum_{j \neq i} q_{ij}$$

in which q_{ij} is the transition rate that defines the probability per time unit that the system makes a transition from state i to state j and q_i is the total transition rate out of state i . The amount of time that the process stays in state i is distributed according to the exponential distribution $f_i(t) = q_i \exp(-q_i t)$. When the process leaves state i , it enters the next state j with probability $P_{ij} = \frac{q_{ij}}{\sum_j q_{ij}}$ if $i \neq j$ and 0 otherwise. In this paper, we assume that the process is stationary, i.e. the intensity matrix is the same at all times.

In recent work, [Nodelman *et al.*, 2002] introduced continuous-time Bayesian networks (CTBNs) as a way to factorize Markov jump processes over discrete-valued variables. Our work extends the inference to hybrid-state processes.

The instantiation of $\mathbf{Z}_t$ corresponds to a unique set of equations that govern the dynamics of the continuous variables $\mathbf{X}_t$. Let $\tau_1, \tau_2, \dots$ denote the times at which $\mathbf{Z}_t$ changes

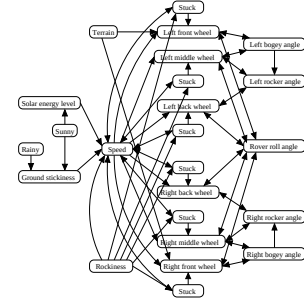


Figure 1: Rover CTBN. Note that a CTBN may contain cycles.

value. Let $\mathbf{z}_i$ denote the value of $\mathbf{Z}_t$ in the interval $[\tau_i, \tau_{i+1}]$. During the time span of $[\tau_i, \tau_{i+1}]$ the variables $\mathbf{X}_t$ follow nonlinear dynamics given by

$$\frac{d}{dt} \mathbf{X}_t = F_{\mathbf{z}_i}(\mathbf{X}_t, \mathbf{U}_t) + W_t \quad (1)$$

and the observations $\mathbf{Y}_t$ are given by

$$\mathbf{Y}_t = H_{\mathbf{z}_i}(\mathbf{X}_t, \mathbf{U}_t) + V_t \quad (2)$$

where $W_t \sim \mathcal{N}(0, \Phi_t)$ and $V_t \sim \mathcal{N}(0, \Psi_t)$ are the respective process and measurement noise, and $\mathbf{U}_t$ is the control input.

In this work, we assume that some variables in $\mathbf{Z}_t$ may be observed, but only at random discrete points in time. On the other hand, $\mathbf{Y}_t$ are observed more frequently, but as noisy measurements of the true continuous state $\mathbf{X}_t$.

3 A continuous-time rover model

As a testbed, we use a continuous-time rover model. The rover employs a rocker-bogey suspension system for its six wheels. On each side, the front two wheels are attached to the bogey, which pivots around its center. The center of the bogey is attached to the rocker, which is attached to the main rover chassis and the rear wheels. Motion from the bogey can cause the rocker to pivot around the differential axle, a mechanism that centers the rover chassis between the left and right sets of wheels. From the rocker angles and the bogey angles, we can infer the rover orientation and thus the wheel heights relative to the rover chassis.

We are interested in reasoning about the rover wheel dynamics under the effects of weather, terrain, ground conditions and faulty wheel behavior. The model consists of 20 state variables and 4 transient variables. There are two groups of state variables: 5 that model background conditions and 15 that directly model the rover. The background variables model environmental factors such as weather, terrain, ground rockiness and ground stickiness, while the rover variables model the solar energy level available to the rover, the rover speed, the wheel heights, the wheel stuck conditions, and the rover roll angle. The CTBN structure is shown in Figure 1.

The model comprises 11 binary and ternary discrete variables and 9 continuous variables. The only observable discrete variables are *Sunny*, *Rainy*, *Terrain* and *Rockiness*. The observation of these variables at sparse time points corresponds to situations when the rover takes measurements of its surroundings, to plan navigation or to manage power consumption. The wheel stuck states are never observed and are

inferred from observations of the continuous wheel variables. The wheel stuck condition is more frequently induced when the rover travels at high speed and when the ground is rocky.

The rover dynamics are adapted from the kinematic analysis in [Hacot, 1998]. The front wheels receive an input proportional to some height perturbation induced by the terrain. The change in the middle wheel height is proportional to the difference between the front and middle wheel height, taken relative to the rover body which experiences roll (torsional rotation) on uneven terrain. To illustrate the system's nonlinearity, the equation for the middle wheel height is:

$$\frac{d}{dt} z_t^M = \kappa_t \cos(\Theta_t) (l_1 \sin(\theta_t^B) - l_2 \sin(\theta_t^B + \beta)) \quad (3)$$

where Θ_t is the roll angle, θ_t^B is the bogey angle, and l_1, l_2 and β are constant parameters of the physical rover model. The change in the back wheel is defined in a similar manner. For each wheel, the proportionality constant κ_t is dependent on the speed and whether the wheel is stuck. The speed is affected by the availability of solar energy and by ground surface characteristics such as stickiness and rockiness.

Noisy measurements of the continuous variables that model the bogey and rocker angles are observed more frequently than the discrete observations. The bogey angle is determined from the front and middle wheel heights. The rocker angle depends on the middle and back wheel heights as well as the bogey angle.

4 Preliminaries

Let the hybrid system state be represented as $\mathbf{S}_t = \{\mathbf{Z}_t, \mathbf{X}_t\}$. We denote $y_{1:t}$ as the history of observations from time 1 to time t . The aim is to track the probability $p(\mathbf{S}_t|y_{1:t})$ that any given state may be the true system state, conditioned upon the sequence of past observations $y_{1:t}$. This probability distribution, also referred to as the *belief state*, can be recursively estimated given the previous belief state $p(\mathbf{S}_{t-1}|y_{1:t-1})$ and the current observation y_t :

$$p(\mathbf{S}_t|y_{1:t}) = p(y_t|\mathbf{S}_t) \frac{p(\mathbf{S}_t|y_{1:t-1})}{p(y_t|y_{1:t-1})} \quad (4)$$

where $p(\mathbf{S}_t|y_{1:t-1}) = \int p(\mathbf{S}_t|\mathbf{S}_{t-1})p(\mathbf{S}_{t-1}|y_{1:t-1})d\mathbf{S}_{t-1}$. However, the computation of this integral leads to intractability in all but the smallest conditional linear Gaussian models. As a result, one must resort to approximate inference methods to compute the updated belief state.

4.1 Particle filtering

Particle filtering (PF) [Doucet *et al.*, 2000b] approximates (4) by a discrete sum of *particles* or samples of possible states drawn from that distribution:

$$p(\mathbf{S}_t|y_{1:t}) \approx \sum_{i=1}^N \frac{1}{N} \delta(s_t^{(i)}) \quad (5)$$

where $\delta(\cdot)$ denotes the Dirac delta function. Since it is difficult to sample from $p(\mathbf{S}_t|y_{1:t})$ directly, importance sampling is used, in which particles are drawn from a more tractable *proposal* distribution, then each particle is weighted to account for this bias. Since variance increases as the state space

increases, the number of particles required to achieve decent accuracy increases as well, thus making PF an expensive solution for tracking complex, high-dimensional systems.

Rao-Blackwellization [Doucet *et al.*, 2000a] is a technique that improves PF by analytically marginalizing out some of the variables. This method is especially applicable to our domain since the structure of the rover model can be factorized:

$$p(\mathbf{S}_t|y_{1:t-1}) = \int p(\mathbf{X}_t|\mathbf{Z}_t, \mathbf{X}_{t-1})p(\mathbf{Z}_t|\mathbf{S}_{t-1})p(\mathbf{S}_{t-1}|y_{1:t-1})d\mathbf{S}_{t-1} \quad (6)$$

Hence, *Rao-Blackwellized Particle Filtering* (RBPf) can be used to sample only from the (lower-dimensional) discrete distribution and the continuous distribution can subsequently be computed using the Kalman filter.

4.2 The unscented Kalman-Bucy filter

The Kalman filter [Grewal and Andrews, 2001] is an efficient, recursive method that finds the least-square estimate of the state of a discrete-time, linear stochastic process with Gaussian noise. The Kalman-Bucy filter is the continuous-time counterpart to the discrete-time Kalman filter. It uses a differential equation to generate an estimate $\hat{\mathbf{x}}_t$ of the continuous state, and generates from that an estimate $\hat{\mathbf{y}}_t$ of the observation. It also generates P_t , the covariance of the state estimate error $\mathbf{x}_t - \hat{\mathbf{x}}_t$, and R_t , the covariance of $\mathbf{y}_t - \hat{\mathbf{y}}_t$. The update of P_t also uses a differential equation. Given these estimates, the probability of the observation $\mathbf{y}_t$ is given by a normal distribution with mean $\hat{\mathbf{y}}_t$ and covariance R_t .

The Kalman-Bucy filter assumes a linear transition model and observation model. To handle nonlinear models, we adopt an approach similar to the discrete-time unscented Kalman filter (UKF) [Wan and van der Merwe, 2000] and extend the Kalman-Bucy filter by applying the unscented transformation [Julier and Uhlman, 2002] to the state estimation. Thus, the *unscented Kalman-Bucy filter* (UKB) is a continuous-time filter that allows for nonlinear models. UKB is used in RBPf as an approximation to the Kalman-Bucy filter. This approximation introduces bias and does not provide the theoretical guarantee that variance is reduced in RBPf.

5 The continuous-time particle filter

In the continuous-time particle filter (CTPF), we perform an update each time a discrete observation is received. If no discrete-state process is observed, updates can be triggered by an asynchronous request for update. Technically, this is treated the same way as observing the discrete-state process with a vacuous observation. If the new observation is received at time t_n and the previous observation was at time t_{n-1} , an update means generating particles of the state at t_n , using the particles generated at t_{n-1} . We assume that observations of the discrete-state processes are sparse and intermittent. We make no assumption that observations come at regular intervals or that the observations coincide with any discrete transitions. Let ζ_n be the observation at time t_n . We may observe ζ_n as the entirety of $\mathbf{Z}_{t_n}$, or ζ_n may be an assignment of values to a subset of the variables in $\mathbf{Z}_{t_n}$, or ζ_n may be vacuous, assigning values to none of the variables. In any case, we will

say that a value $\mathbf{z}$ agrees with the observation ζ_n if it assigns the same value to all variables assigned by ζ_n .

At a high level, the CTPF works as follows. When updating from t_{n-1} to t_n , we sample a trajectory from the Markov jump process for the time interval $[t_{n-1}, t_n]$. This results in a set of time points $\tau_0 = t_{n-1}, \tau_1, \dots, \tau_k = t_n$ at which $\mathbf{Z}$ changes value. Let $\mathbf{z}_i$ denote the value of $\mathbf{Z}$ in interval $[\tau_i, \tau_{i+1}]$. We then perform Rao-Blackwellization to obtain updates for the continuous variables. We go through the sequence of intervals, starting at $[\tau_0, \tau_1]$ and ending at $[\tau_{k-1}, \tau_k]$, propagating our estimate of the mean and variance of the continuous variables $\mathbf{X}$ using UKB. In the time period $[\tau_i, \tau_{i+1}]$, the continuous dynamics are governed by the value $\mathbf{z}_i$. As we propagate the continuous variables, we also compute the probability of the trajectory, which is used as the particle weight. We repeat this process to obtain a set of weighted particles at time t_n , and resample from these to get a new set of unweighted particles.

The behavior of the Markov jump process governing $\mathbf{Z}$ is characterized by the intensity matrix $Q_{\mathbf{Z}}$. The process of generating a trajectory of the Markov jump process from time t_{n-1} to t_n is as follows. For each particle at time t_{n-1} , let $\mathbf{z}_0$ be the value of $\mathbf{Z}$ in that particle. Set $\tau_0 = t_{n-1}$ and $i = 0$. We use the intensity matrix to sample a time from the exponential distribution $f_{\mathbf{z}_i}(t) = q_{\mathbf{z}_i} \exp(-q_{\mathbf{z}_i} t)$ to obtain the length of time γ until $\mathbf{Z}$ next changes value. If $\tau_i + \gamma < t_n$, we sample $\mathbf{z}_{i+1}$ from the intensity matrix, and continue to loop, setting $\tau_{i+1} = \tau_i + \gamma$. Otherwise, we know that $\mathbf{Z} = \mathbf{z}_i$ at t_n . If this value agrees with the evidence ζ_n , the particle is kept, otherwise it is rejected.

Once we have the jump process, we proceed to propagate the continuous variables $\mathbf{X}$ through the time points $\tau_0, \dots, \tau_k$. Since there may be many observations of the continuous variables in the interval $[\tau_i, \tau_{i+1}]$, when we propagate them from τ_i to τ_{i+1} , we need to consider all the observations between those times. Let δ_0 be τ_i , δ_ℓ be τ_{i+1} and $\delta_1, \dots, \delta_{\ell-1}$ be the time points at which continuous observations were received. We loop through the intervals $[\delta_j, \delta_{j+1}]$ in turn. For each interval, we propagate the mean and variance of the continuous variables forward using UKB. We also obtain the probability of the observation at time δ_{j+1} , given by the predictive density $p(\mathbf{y}_{\delta_{j+1}} | \mathbf{y}_{\delta_1:\delta_j}) = \mathcal{N}(\mathbf{y}_{\delta_{j+1}}; \hat{\mathbf{y}}_{\delta_{j+1}}, R_{\delta_{j+1}})$ where $\hat{\mathbf{y}}_{\delta_{j+1}}$ and $R_{\delta_{j+1}}$ are the estimate and covariance of $\mathbf{y}_{\delta_{j+1}}$ under UKB. We then multiply the probability of all the observations together over all time points δ_j in $[\tau_i, \tau_{i+1}]$ and over all τ_i to obtain the probability of the sequence of observations. This becomes the weight of the particle.

We repeat this procedure for each of the M particles from time t_{n-1} . Each particle consists of a candidate Markov jump process for each discrete variable in $\mathbf{Z}_t$, and the mean and covariance for the continuous variables in $\mathbf{X}_t$. The Markov jump process candidates that are far from the ground truth will lead to poor tracking of the continuous variables due to incorrect parametrization of the continuous-state dynamics. Particles associated with the ill-fit Markov jump processes will have low weights and will be resampled out.

The pseudocode for CTPF is shown in Figure 2.

```

1: let  $M$  be the number of particles
2: let  $N$  be the number of discrete observations
3: for  $n = 1$  to  $N$  do
4:   Dynamics propagation
5:   for  $m = 1$  to  $M$  do
6:     let  $\zeta_n$  be the  $n^{\text{th}}$  observation
7:     let  $t_n$  be the observed time of  $\zeta_n$ 
8:     Simulate a Markov jump process
9:     let  $i = 0, \tau_0 = t_{n-1}$ 
10:    let  $\mathbf{z}_0$  be the value of  $\mathbf{Z}$  in the  $m^{\text{th}}$  particle at time  $t_{n-1}$ 
11:    while  $\tau_i < t_n$  do
12:      Sample  $\gamma$  from the PDF  $f(t) = q_{\mathbf{z}_i} \exp(-q_{\mathbf{z}_i} t)$ 
13:      if  $\tau_i + \gamma < t_n$  then
14:        let  $\tau_{i+1} = \tau_i + \gamma$ 
15:        Sample  $\mathbf{z}_{i+1}$  using the intensity matrix  $Q_{\mathbf{Z}}$ 
16:      else
17:        let  $\tau_{i+1} = t_n; \text{let } \mathbf{z}^{(m)} = \mathbf{z}_i$ 
18:         $i \leftarrow i + 1$ 
19:      if  $\mathbf{z}^{(m)}$  disagrees with  $\zeta_n$ , reject the particle
20:      let  $k = i$ 
21:      Propagate continuous variables
22:      let  $\hat{\mathbf{x}}_{t_{n-1}}$  and  $P_{t_{n-1}}$  be the mean and covariance of
        the continuous variables in the  $m^{\text{th}}$  particle at time  $t_{n-1}$ 
23:      let  $w^{(m)} = 1$ 
24:      for  $i = 0$  to  $k - 1$  do
25:        let  $\delta_1, \dots, \delta_{\ell-1}$  be the times of
          continuous observations in  $[\tau_i, \tau_{i+1}]$ 
26:        let  $\delta_0 = \tau_i, \delta_\ell = \tau_{i+1}$ 
27:        for  $j = 0$  to  $\ell - 1$  do
28:          Compute  $\hat{\mathbf{x}}_{\delta_{j+1}}, P_{\delta_{j+1}}, \hat{\mathbf{y}}_{\delta_{j+1}}$  and  $R_{\delta_{j+1}}$ 
            using UKB beginning with  $\hat{\mathbf{x}}_{\delta_j}$  and  $P_{\delta_j}$ ,
            under the dynamics determined by  $\mathbf{z}_i$ 
29:           $w^{(m)} \leftarrow w^{(m)} \times \mathcal{N}(\mathbf{y}_{\delta_{j+1}}; \hat{\mathbf{y}}_{\delta_{j+1}}, R_{\delta_{j+1}})$ 
30:        let  $\hat{\mathbf{x}}^{(m)}$  and  $P^{(m)}$  be  $\hat{\mathbf{x}}_{t_n}$  and  $P_{t_n}$ 
31:      Resampling
32:      Normalize  $w^{(m)} \leftarrow \frac{w^{(m)}}{\sum w^{(m)}}$ 
33:      Select  $M$  new particles where the probability of
         $(\mathbf{z}^{(m)}, \hat{\mathbf{x}}^{(m)}, P^{(m)})$  is  $w^{(m)}$ 

```

Figure 2: Continuous-time particle filter

6 Learning

In real-world applications, we may not know the exact parametrization of $\mathbf{Z}_t$. Instead, we may have a prior distribution over the intensity matrices $Q_{\mathbf{Z}}$ and need to fine tune our estimate of $Q_{\mathbf{Z}}$ as observations about $\mathbf{Z}_t$ are made.

In this case, we model $Q_{\mathbf{Z}}$ as a random variable and we attribute a probability distribution $\phi(Q_{\mathbf{Z}})$ over the space of admissible intensity matrices, $\mathcal{Q}_{\mathbf{Z}}$. Since the observations are exponentially distributed, we model $\phi(Q_{\mathbf{Z}})$ as a Gamma distribution since Gamma and exponential are conjugates.

Assume that the elements of $Q_{\mathbf{Z}}$ are distributed according to $q_{ij} \sim \Gamma\left(\frac{1}{\beta_i}, \alpha_{ij}\right)$. The prior distribution $Q_{\mathbf{Z}}$ is given by:

$$\phi(Q_{\mathbf{Z}}) \propto \prod_{i=1}^m \prod_{j \neq i} q_{ij}^{\alpha_{ij}-1} e^{-q_{ij} \beta_i} \quad (7)$$

Upon the arrival of a discrete-state observation ζ_n , we simulate a Markov jump process J over the time interval $[t_{n-1}, t_n]$. From J , we can compute the likelihood of Q_Z :

$$L(Q_Z) = \prod_{i=1}^m \prod_{j \neq i} q_{ij}^{N_{ij}} e^{-q_{ij} R_i} \quad (8)$$

where N_{ij} is the number of transitions from state i to j in the process J and R_i is the time spent in state i during process J . From this, we update the Q_Z -distribution

$$\psi(Q_Z) = L(Q)\phi(Q) = \prod_{i=1}^m \prod_{j \neq i} q_{ij}^{[N_{ij} + \alpha_{ij}] - 1} e^{-q_{ij} [R_i + \beta_i]} \quad (9)$$

Upon the arrival of the next discrete-state observation, we sample $q_{ij} \sim \Gamma\left(\frac{1}{R_i + \beta_i}, N_{ij} + \alpha_{ij}\right)$ to get an instantiation for Q_Z that reflects the updated distribution.

The learning procedure is integrated into algorithm 2 as follows: we sample $Q_Z \sim \phi(Q_Z)$ right before line (8) and perform the Q_Z -distribution update, as prescribed in Equations (7)-(9), after line (20) of the pseudocode.

This procedure may seem like magic. After all, we are using statistics generated from simulating from Q_Z itself to obtain an updated estimate of Q_Z . The reason it works is that not all Q_Z s from the previous set of particles have the same chance of being accepted into the next set of particles. First of all, if the trajectory disagrees with the discrete observation ζ_n , it will be rejected. Second, the trajectories will be weighted by the probability of the continuous observations, and these weighted particles will be resampled. A trajectory, and thus a new intensity matrix, that agrees well with the observations, is more likely to be kept. In this way, the set of intensity matrices that are kept will be influenced by the observations, both discrete and continuous.

Other learning approaches have been explored in [Nodelman *et al.*, 2003] for parameter learning of CTBNS, and in [Bladt and Sørensen, 2003] for statistical inference of discretely observed Markov jump processes. However, these approaches are offline learning approaches where the sequence of discrete-time observations are assumed known a priori to the learning process. Our approach integrates online learning into the particle filtering framework, where we also take advantage of the continuous observations in eliminating poor candidates for Q_Z . This advantage allows for relatively quick convergence to the neighborhood of the true Q_Z value.

7 Experimental results

We evaluate the performance of the CTPF algorithm on a simulated small model and the rover model. The small model consists of one discrete ternary variable, two continuous state variables, and two continuous observation variables. Depending on the state of the discrete-valued process, the continuous behavior can be either linear, polynomial or sinusoidal. The low dimensionality of the small model makes it feasible to compare against the discrete-time particle filter and to evaluate the performance of the learning procedure.

Figure 3 shows an experiment comparing CTPF to the discrete-time particle filter (DTPF). The results shown are averaged over 50 runs. The first graph shows the performance of CTPF with 10 particles and 55 updates, comparing the estimate of one of the continuous state variables produced by CTPF with the actual value of that variable. While CTPF updates the discrete variables at infrequent intervals, DTPF performs an update at equally-spaced intervals fixed by a preset time granularity. Both algorithms perform continuous inference, via UKB or UKF, at many intermediate time points, corresponding to when the continuous variables are observed.

The plotted points are the estimates at these intermediate points. From the graph, we see that CTPF is consistently able to track the variable through the different mode changes. The second graph (DTPF1) shows DTPF's performance using the same number of particles and the same number of updates as CTPF. We see that DTPF1 has worse tracking throughout, and does particularly poorly from time 75 to time 100. CTPF outperforms DTPF because it is able to perform updates precisely when they are needed.

DTPF is considerably faster than CTPF because it does not require solving differential equations. This was exacerbated by a slow Matlab implementation of the Kalman-Bucy filter, compared to a well-optimized implementation of the Kalman filter. We expect this difference to lessen with a more efficient implementation of CTPF. Nonetheless, we ran experiments that allowed DTPF the same running time as CTPF.

We first allowed DTPF to use more particles than CTPF, while maintaining the same number of updates. The results of DTPF, with 58 particles and 55 updates, are shown as DTPF2, in the third graph of Figure 3. When the number of updates is not sufficient for DTPF to track the variables, increasing the number of particles does not particularly help. We then allowed DTPF to update more frequently than CTPF, but use the same number of particles. The results, with 10 particles and 584 updates, are shown in the fourth graph and are labeled DTPF3. The DTPF3 results show a significant improvement in the performance of DTPF, although DTPF still does not perform as well as CTPF when the continuous variables are oscillating rapidly. Furthermore, we found the variance of the estimates in DTPF3 to be higher than that produced by CTPF.

Figure 4 shows the results of learning the Q matrix on the small model. The results were quite promising. The first graph compares the estimation error achieved by CTPF with learning the Q matrix and CTPF with using the true Q matrix. Both sets of CTPF experiments used 10 particles. We see that the error of CTPF with learning comes close to that of CTPF without learning after 1000 time steps. The second graph shows the distance of the learned Q matrix from the true Q matrix over time, computed as the sum of the element-wise differences between the two matrices. The graph shows the learned Q matrix converging towards the true Q matrix.

Lastly, we tested the CTPF algorithm on our main application, the continuous-time rover model. Despite the nonlinearity of the model, CTPF is able to track the wheel heights relatively well with 10 particles. The results of the CTPF inference, averaged over 20 runs, are presented in Figure 5 (one graph per rover wheel). The tracking error is mainly due to incorrect estimation of the wheel stuck variable. Whenever

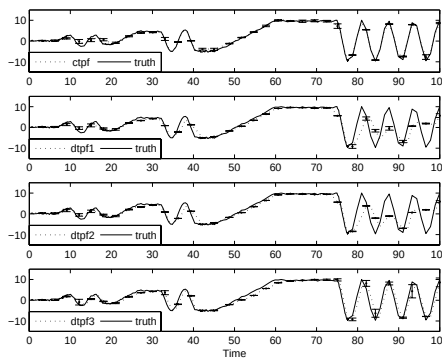


Figure 3: Estimation results

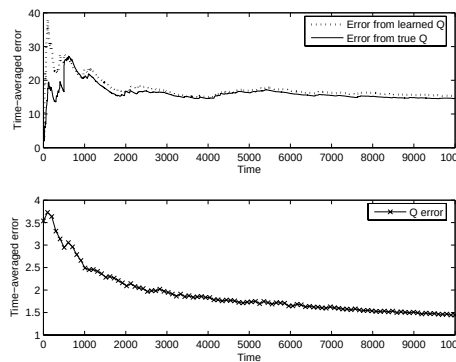


Figure 4: Learning results

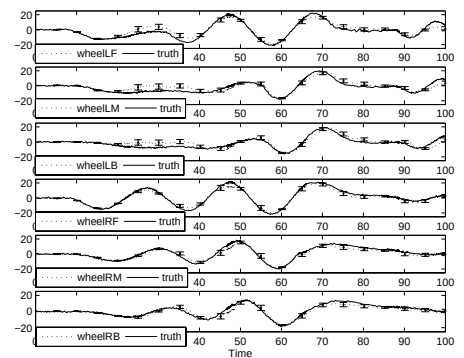


Figure 5: CTPF results on the rover

the CTPF estimate drifts from the ground truth, it is able to quickly recover from the error and continue to closely track the wheel heights. The empirical results confirm CTPF's applicability for monitoring complex continuous-time systems.

8 Conclusion and future work

Since most real-world systems evolve in continuous time, CTPF is a natural way to address the time granularity problem that is often associated with systems that have components that evolve at different rates. By performing state estimation in continuous time, CTPF is also better suited for resource-bounded computation since inference is no longer performed at every fixed time step (as in the discrete-time particle filter). Our results show that CTPF can effectively track the state of complex systems such as the rover. Moreover, when comparison is possible with discrete-time approaches, CTPF tracks the state more accurately. We believe that the approach is an important step in making recent advances in particle filtering applicable to a much larger set of continuous-time problems.

An important aspect in hybrid system modelling is the issue of *autonomous transitions*, in which the continuous state triggers a discrete transition. We model this in our approach by making the intensity matrices dependent on the continuous variables. This leads to difficulties in that while the system is in a state, the distribution over the time the system remains there may vary. At present we approximate this by only computing the intensity matrix at updates or when the discrete state changes. This is consistent with our aim of resource-bounded computation. We plan to investigate other alternatives such as resampling from an updated intensity matrix when the current distribution deviates drastically from the assumed distribution used for sampling.

Other areas of future work include applying this approach with factored state spaces [Ng *et al.*, 2002] to reduce the complexity of the Kalman-Bucy updates, and implementing this approach onboard an actual robot.

References

[Bladt and Sørensen, 2003] M. Bladt and M. Sørensen. Statistical inference for discretely observed markov processes. Technical report, University of Copenhagen, 2003.

- [Doucet *et al.*, 2000a] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proc. of the 16th UAI*, 2000.
- [Doucet *et al.*, 2000b] A. Doucet, S. Godsill, and C. Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 2000.
- [Grewal and Andrews, 2001] M. S. Grewal and A. P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB*. Wiley-Interscience, 2001.
- [Hacot, 1998] H. Hacot. Analysis and traction control of a rocker-bogie planetary rover. Master's thesis, MIT, 1998.
- [Hofbaur and Williams, 2002] M. W. Hofbaur and B. C. Williams. Hybrid diagnosis with unknown behaviour modes. In *Proc. of the 13th Intl. Workshop on Principles of Diagnosis*, 2002.
- [Julier and Uhlman, 2002] S. Julier and J. Uhlman. The scaled unscented transformation. In *Proc. of the IEEE American Control Conference*, 2002.
- [Ng *et al.*, 2002] B. Ng, L. Peshkin, and A. Pfeffer. Factored particles for scalable monitoring. In *Proc. of the 18th UAI*, 2002.
- [Nodelman *et al.*, 2002] U. Nodelman, C. R. Shelton, and D. Koller. Continuous time bayesian networks. In *Proc. of the 18th UAI*, 2002.
- [Nodelman *et al.*, 2003] U. Nodelman, C. R. Shelton, and D. Koller. Learning continuous time Bayesian networks. In *Proc. of the 19th UAI*, 2003.
- [Wan and van der Merwe, 2000] E. Wan and R. van der Merwe. The unscented kalman filter for nonlinear estimation. In *Proc. of IEEE Symposium*, 2000.
- [Washington *et al.*, 2000] R. Washington, K. Golden, and J. L. Bresina. Plan execution, monitoring, and adaptation for planetary rovers. *Electron. Trans. Artif. Intell.*, 2000.
- [Willeke and Dearden, 2004] T. Willeke and R. Dearden. Building hybrid rover models: Lessons learned. In *Proc. of the 15th Intl. Workshop on Principles of Diagnosis*, 2004.
- [Williams and Nayak, 1996] B. C. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. In *Proc. of the 13th AAAI and the 8th IAAI*, 1996.

Accurate and Low-cost Location Estimation Using Kernels

Jeffery Junfeng Pan^a James T. Kwok^a Qiang Yang^a Yiqiang Chen^b

^a Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong

^b Institute of Computer Technology, Chinese Academy of Sciences, China

{panjf,jamesk,qyang}@cs.ust.hk, yqchen@ict.ac.cn

Abstract

We present a novel method for indoor-location estimation using a vector-space model based on signals received from a wireless client. Our aim is to obtain an accurate mapping between the signal space and the physical space without incurring too much human calibration effort. This problem has traditionally been tackled through probabilistic models trained on manually labeled data, which are expensive to obtain. In this paper, we present a novel approach to building a mapping between the signal-vector space and the physical location space using kernel canonical correlation analysis (KCCA). Its training requires much less human labor. Moreover, unlike traditional location-estimation systems that treat grid points as independent and discrete target classes during training, we use the physical location as a continuous feedback to build a similarity mapping using KCCA. We test our algorithm in a 802.11 wireless LAN environment, and demonstrate the advantage of our method in both accuracy and its ability to utilize a much smaller set of labeled training data than previous methods.

1 Introduction

Location estimation is a major area of pervasive computing applications that range from context-dependent content delivery to people monitoring. However, accurate location information may not be always available. For example, many buildings cannot be reached by location-determination devices such as GPS. In such environments, a commonly-used technique is to leverage the signal strength values of a wireless network such as the IEEE 802.11b wireless LAN network (WLAN). Many systems utilize the signal strength values received from the access points to infer the location of mobile devices, based on deterministic or probabilistic techniques [Ladd *et al.*, 2002; Gentile and Berndt, 2004; Youssef *et al.*, 2003; Bahl and Padmanabhan, 2000; Ni *et al.*, 2003]. Similarly, in a sensor network, the location information must also be inferred from signals received from various deployed sensors. A major issue in these applications is how to build an accurate mapping between the signal space and the physical space while incurring minimal human effort.

In general, location-estimation systems using (RF)-based signal-strength values function in two phases: an *offline training phase* and an *online, localization phase*. In the offline phase, a *probabilistic model* is trained by considering the signal strength values received from the access points at selected

locations in the area of interest. These values comprise the training data gathered from a physical region, which are used to calibrate a probabilistic location-estimation system. In the online localization phase, the real-time signal strength samples received from the access points are used to estimate the current location based on the learned model.

In the training phase, the learned model is essentially a mapping function $f(\mathcal{S}, \mathcal{P})$ between the signal space $\mathcal{S}$ and the physical location space $\mathcal{P}$. Deterministic techniques build such a mapping function by assuming a signal propagation model based on properties of electro-magnetic fields. These signal-propagation models often cannot accurately describe a realistic environment due to uncertainty in the devices and the obstacles in the signal paths. Probabilistic techniques are often more accurate by directly handling the uncertainty. These techniques construct the function $f(\cdot, \cdot)$ by maximizing the likelihood of the observation given the probabilistic distribution of the training data used for calibration.

However, the probabilistic methods suffer from several major shortcomings. First, these techniques require a huge amount of labeled data for the trained model to be accurate. In reality, the data collection is a time-consuming process. In our experience, for example, in our earlier test it took many hours to collect and label the signal-strength data in a limited indoor environment. Second, the mapping between the signal and physical spaces are very difficult to construct due to nonlinearity. Nonlinearity exists when similar locations have very different signal signatures. Therefore, a direct mapping between the two spaces may not always work well, even when much data are collected. Third, many probabilistic techniques adopt a classification-based approach, by treating the locations as discrete classes to be classified. However, this approach ignores the important information inherent in the physical proximity of the locations. Furthermore, previous methods that apply regression based techniques treat the physical dimensions (x, y) separately, they cannot fully exploit the geometric information imparted by considering x and y together.

In this paper, we present a similarity-based approach to building a mapping function. Our main intuition is to perform a kernel-based transformation of the signal and physical spaces to capture the nonlinear relationship between the signals and locations. Furthermore, we perform a kernel canonical correlation analysis (KCCA) in the two spaces for feature extraction. This allows the pairwise similarity of samples that are measured in both spaces to be maximally correlated. We apply a Gaussian kernel to adapt the noisy characteristic of signal strength and use a Matérn kernel to sense the change

in physical locations.

A major advantage of our proposed technique is that we can obtain higher accuracy while reducing the training cost by requiring only a fraction of the labeled samples as compared to previous methods. We demonstrate this result in a series of tests on WLAN location estimation using the data collected in a realistic environment.

2 Related Work

2.1 Location Estimation

Location-estimation systems are an integral component for probabilistic plan-recognition systems in AI [Liao *et al.*, 2004; Bui *et al.*, 2002]. In general, they operate in two phases: an *offline training phase* and an *online localization phase*. During the offline phase, a *radio map* is built which tabulates the signal strength received from the access points at selected locations in the area of interest. During the localization phase, the signal strength samples received from the access points are used to search the radio map to estimate a user's location. In building the radio maps, two techniques have been attempted: deterministic and probabilistic.

Deterministic techniques rely on radio propagation models which take into account such environmental factors as path-loss functions [Roos *et al.*, 2002a; Kaemarungsi and Krishnamurthy, 2004]. When this is possible to do where uncertainty is relatively low, signal distribution models can be derived with a relatively small number of samples using knowledge on propagation channels [Hashemi, 1993]. This also enables the nearest neighbor or triangulation techniques to be used [Bahl and Padmanabhan, 2000; Smailagic and Kogan, 2002; Bhasker *et al.*, 2004]. For example, the RADAR system by Microsoft Research [Bahl and Padmanabhan, 2000] uses nearest neighbor heuristics and triangulation methods to infer a user's location. Each signal strength sample is compared against the radio map and the coordinates of the best matches are averaged to give location estimation. The accuracy of RADAR is about three meters with fifty percent probability. In [Bhasker *et al.*, 2004], an online procedure based on feedback from users was employed to correct the location estimation of the system. A similar technique for sensor networks has been applied; for example, the LANDMARC system [Ni *et al.*, 2003] utilizes the concept of reference tags to alleviate the effects caused by the fluctuation in RF signal strength. The method first computes the distance between the signal-strength vectors received from the tracking tags and those from different reference-tags' respectively. It then uses k nearest reference tags' coordinates to calculate the approximate coordinate of the tracking tag.

Probabilistic techniques confront the uncertainty in an indoor wireless environment, by constructing the signal-strength distributions over different locations in the radio map and use probabilistic inference methods for localization [Roos *et al.*, 2002b; Youssef *et al.*, 2003]. [Youssef *et al.*, 2003] used a joint clustering technique to group locations together in order to reduce the computational cost of the system. The method first determines a most likely cluster within which to search for the most probable location, then applies

a maximum likelihood estimation (MLE) method to infer the most probable location within the cluster.

More recently, intense research efforts have been spent to seek additional knowledge in order to boost the accuracy of the location determination systems. For example, to handle the sequential characteristic of user traces, [Ladd *et al.*, 2002] suggest a sensor fusion model and shows a strong correlation between consecutive samples. The robotics-based location sensing system in [Ladd *et al.*, 2002] applies Bayesian inference to compute the conditional probabilities over locations based on received signal-strength samples from various access points. Then a postprocessing step, which utilizes the spatial constraints of a user's movement trajectories, refines the location estimation and rejects the estimates that show significant changes in the location space. Depending on whether the postprocessing step is used or not, the accuracy of this method is 83% or 77% within 1.5 meters. [Yin *et al.*, 2005; Ni *et al.*, 2003] study the dynamic features of signal strength and build a regression model to adapt the change of radio map with reference points. A radio map is constructed using data mining methods in an effort to reduce the samples collected for calibration [Chai and Yang, 2005].

In contrast to the above works, our approach builds a similarity based mapping function by making full use of the continuous location information in kernel-based transformation. In the next section, we first review kernels and kernel canonical correlation analysis.

2.2 (Kernel) Canonical Correlation Analysis

Canonical Correlation Analysis

Given two sets of variables, canonical correlation analysis (CCA) [Hotelling, 1936] attempts to find a basis for each set such that the correlation between the projections of the variables onto these basis vectors are mutually maximized. Mathematically, given n instances $S = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$ of the pair $(\mathbf{x}, \mathbf{y})$, CCA finds directions (*canonical vectors*) $\mathbf{w}_x$ and $\mathbf{w}_y$ so that the transformed variables (*canonical variates*) $(\langle \mathbf{w}_x, \mathbf{x}_1 \rangle, \langle \mathbf{w}_x, \mathbf{x}_2 \rangle, \dots, \langle \mathbf{w}_x, \mathbf{x}_n \rangle)$ and $(\langle \mathbf{w}_y, \mathbf{y}_1 \rangle, \langle \mathbf{w}_y, \mathbf{y}_2 \rangle, \dots, \langle \mathbf{w}_y, \mathbf{y}_n \rangle)$, are maximally correlated. Now, define the total covariance matrix by

$$\mathbf{C} = \hat{\mathbb{E}} \left[\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}^T \right] = \begin{bmatrix} \mathbf{C}_{xx} & \mathbf{C}_{xy} \\ \mathbf{C}_{yx} & \mathbf{C}_{yy} \end{bmatrix},$$

where $\hat{\mathbb{E}}[\cdot]$ is the empirical expectation operator, $\mathbf{C}_{xx}, \mathbf{C}_{yy}$ are the within-sets covariance matrices and $\mathbf{C}_{xy} = \mathbf{C}_{yx}'$ is the between-sets covariance matrix. It can be shown that $\mathbf{w}_x$ can be obtained by solving the generalized eigenproblem $\mathbf{C}_{xy} \mathbf{C}_{yy}^{-1} \mathbf{C}_{yx} \mathbf{w}_x = \lambda^2 \mathbf{C}_{xx} \mathbf{w}_x$, and then subsequently obtain $\mathbf{w}_y$ as $\mathbf{w}_y = \frac{1}{\lambda} \mathbf{C}_{yy}^{-1} \mathbf{C}_{yx} \mathbf{w}_x$ [Hardoon *et al.*, 2004].

Kernel Canonical Correlation Analysis

A major limitation of CCA is that it can only exploit linear relationships between $\mathbf{x}$ and $\mathbf{y}$. As is now well-known, the use of kernels offers an efficient, nonlinear extension of CCA. Kernel canonical correlation analysis (KCCA) [Hardoon *et al.*, 2004] implicitly maps $\mathbf{x}$ and $\mathbf{y}$ to $\phi_x(\mathbf{x})$ and $\phi_y(\mathbf{y})$, and then performs traditional CCA in the two high-dimensional

feature spaces. To control the flexibility of the projections, the norms of the associated weight vectors are penalized.

Using the dual representations for the projection directions $\mathbf{w}_{\phi_x(\mathbf{x})}$ and $\mathbf{w}_{\phi_y(\mathbf{y})}$:

$$\mathbf{w}_{\phi_x(\mathbf{x})} = \mathbf{S}'_{\phi_x(\mathbf{x})} \boldsymbol{\alpha}, \quad \text{and} \quad \mathbf{w}_{\phi_y(\mathbf{y})} = \mathbf{S}'_{\phi_y(\mathbf{y})} \boldsymbol{\beta}, \quad (1)$$

where $\mathbf{S}_{\phi_x(\mathbf{x})} = [\phi_x(\mathbf{x}_1), \dots, \phi_x(\mathbf{x}_n)]'$, $\mathbf{S}_{\phi_y(\mathbf{y})} = [\phi_y(\mathbf{y}_1), \dots, \phi_y(\mathbf{y}_n)]'$, and $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{R}^n$. Denote the corresponding kernel functions by $k_x(\cdot, \cdot)$ and $k_y(\cdot, \cdot)$, and the kernel matrices (defined on all n instances) by $\mathbf{K}_x$ and $\mathbf{K}_y$. It can be shown that $\boldsymbol{\alpha}$ can be obtained by solving the generalized eigenproblem

$$(\mathbf{K}_x + \kappa \mathbf{I})^{-1} \mathbf{K}_y (\mathbf{K}_y + \kappa \mathbf{I})^{-1} \mathbf{K}_x \boldsymbol{\alpha} = \lambda^2 \boldsymbol{\alpha}, \quad (2)$$

where $\mathbf{I}$ is the identity matrix and κ is a user-defined regularization parameter. Subsequently, $\boldsymbol{\beta}$ can be obtained as

$$\boldsymbol{\beta} = \frac{1}{\lambda} (\mathbf{K}_y + \kappa \mathbf{I})^{-1} \mathbf{K}_x \boldsymbol{\alpha}. \quad (3)$$

Given any $\tilde{\mathbf{x}}$, its projection on $\mathbf{w}_{\phi_x(\mathbf{x})}$ is given by

$$P_{\mathbf{x}}(\tilde{\mathbf{x}}) = \phi_x(\tilde{\mathbf{x}})' \mathbf{w}_{\phi_x(\mathbf{x})} = \mathbf{k}'_{\tilde{\mathbf{x}}} \boldsymbol{\alpha}, \quad (4)$$

where $\mathbf{k}_{\tilde{\mathbf{x}}} = [k_x(\tilde{\mathbf{x}}, \mathbf{x}_1), k_x(\tilde{\mathbf{x}}, \mathbf{x}_2), \dots, k_x(\tilde{\mathbf{x}}, \mathbf{x}_n)]'$. Similarly, the projection of any $\tilde{\mathbf{y}}$ onto $\mathbf{w}_{\phi_y(\mathbf{y})}$ is $P_{\mathbf{y}}(\tilde{\mathbf{y}}) = \mathbf{k}'_{\tilde{\mathbf{y}}} \boldsymbol{\beta}$, where $\mathbf{k}_{\tilde{\mathbf{y}}} = [k_y(\tilde{\mathbf{y}}, \mathbf{y}_1), k_y(\tilde{\mathbf{y}}, \mathbf{y}_2), \dots, k_y(\tilde{\mathbf{y}}, \mathbf{y}_n)]'$.

The generalized eigenproblem in (2) can be solved by using the (complete) Cholesky decomposition. However, the kernel matrices $\mathbf{K}_x$ and $\mathbf{K}_y$ are of size n , and so obtaining the Cholesky decomposition can become computational expensive for large training sets. In that case, the incomplete Cholesky decomposition or the partial Gram-Schmidt orthogonalization can be used instead [Hardoon *et al.*, 2004].

3 Location Estimation using KCCA

3.1 Motivation

Consider the two-dimensional location estimation problem¹. Its goal is to obtain a mapping between the space of signal strengths obtained at p access points $\mathcal{S} = \{\mathbf{s} \equiv [s_1, s_2, \dots, s_p]' \in \mathbb{R}^p\}$ and the physical space $\mathcal{P} = \{\boldsymbol{\ell} \equiv [x, y]' \in \mathbb{R}^2\}$. Traditional approaches typically learn the two mappings, $\mathcal{S} \rightarrow \{x\}$ and $\mathcal{S} \rightarrow \{y\}$, independently [Brunato and Battiti, 2004]. In this paper, we consider x and y together and emphasize the *correlation* between signal and physical spaces, observing that the pairwise similarity in the signal space should match the pairwise similarity in the physical space. For example, in Figure 1, signal S_A should be more similar to S_B than S_C , since A is closer to B in the physical space. Consequently, we consider both x and y together and use KCCA to learn the mapping between the two spaces.

3.2 The LE-KCCA Algorithm

Offline Training Phase

In the training phase, the following steps are taken:

¹Extension to the three-dimensional (or even higher-dimensional) case is straight-forward

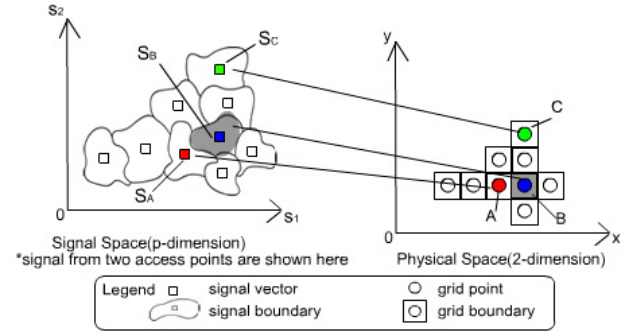


Figure 1: Correlation between the signal and physical spaces.

1. Signal strengths are collected at various grid locations.
2. KCCA, with appropriate choices for the two kernels (Section 3.3), is used to learn the relationship between the signal and physical spaces. In particular, λ_i 's and $\boldsymbol{\alpha}_i$'s are obtained from the generalized eigenproblem in (2), and the corresponding $\boldsymbol{\beta}_i$'s from (3).
3. For each training pair $(\mathbf{s}_i, \boldsymbol{\ell}_i)$, its projections

$$P(\mathbf{s}_i) = [P_1(\mathbf{s}_i), P_2(\mathbf{s}_i), \dots, P_T(\mathbf{s}_i)]' \quad (5)$$

on the T canonical vectors are obtained from (4).

Online Localization Phase

In the localization phase, the location of a new signal strength vector $\tilde{\mathbf{s}}$ is estimated as follows:

1. Use (4) to project $\tilde{\mathbf{s}}$ onto the canonical vectors and obtain $P(\tilde{\mathbf{s}}) = [P_1(\tilde{\mathbf{s}}), P_2(\tilde{\mathbf{s}}), \dots, P_T(\tilde{\mathbf{s}})]'$.
2. Among the projections (5) from the training samples, find the M neighbors closest to $P(\tilde{\mathbf{s}})$. In this paper, the weighted Euclidean distance $d_i = \sum_{j=1}^T \lambda_j (P_j(\tilde{\mathbf{s}}) - P_j(\mathbf{s}_i))^2$ is employed.
3. Interpolate these neighbors' physical locations to predict the physical location of $\tilde{\mathbf{s}}$. In this paper, we simply output the median (or mean for continuous location estimation) of the (x, y) coordinates of these M neighbors.

3.3 Choice of Kernels

Signal-strength values are often noisy. The empirical distribution (*radio map*) is difficult to obtain, especially when the training samples are scarce. As a first approximation, the Gaussian distribution has been used in characterizing the non-linearity of the signal strengths [Roos *et al.*, 2002b]. Hence, in this paper, we also use the Gaussian kernel $G(\mathbf{x}_1, \mathbf{x}_2)$:

$$\exp(-w_G^2 \|\mathbf{x}_1 - \mathbf{x}_2\|^2) \quad (6)$$

for the $\mathcal{S}$ space. Here, w_G is a user-defined parameter that reflects the smoothness of the radio map.

On the other hand, measurements of the physical locations are relatively clean. As in [Schwaighofer *et al.*, 2004], we adopt the Matérn kernel $M(\mathbf{x}_1, \mathbf{x}_2)$ (Figure 2):

$$\frac{2(\sqrt{\nu} w \|\mathbf{x}_1 - \mathbf{x}_2\|)^\nu}{\Gamma(\nu)} K_\nu(2\sqrt{\nu} w_M \|\mathbf{x}_1 - \mathbf{x}_2\|), \quad (7)$$

where ν is a smoothness parameter, $\Gamma(\nu)$ is the Gamma function and $K_\nu(r)$ is the modified Bessel function of the second

kind. It can be shown that when $\nu \rightarrow \infty$, the Matérn kernel degenerates to the Gaussian kernel, and to the exponential kernel $\exp(-\sqrt{2}\nu w\|\mathbf{x}_1 - \mathbf{x}_2\|)$ when $\nu = 0.5$. Note that both the Gaussian and Matérn kernels are isotropic and so are invariant to the location of origin and to arbitrary rotation.

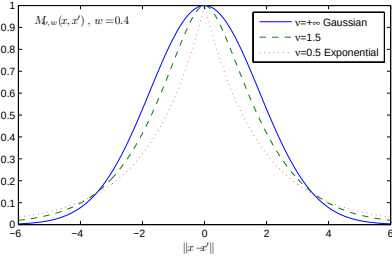


Figure 2: The Matérn kernel.

4 Experiments

In this Section, experiments are performed to measure the accuracy of LE-KCCA, and also test whether this is useful in reducing calibration effort. For comparison, we also run: (1) support vector machine (SVM) [Brunato and Battiti, 2004]; (2) maximum likelihood estimation (MLE) [Youssef *et al.*, 2003]; and (3) RADAR [Bahl and Padmanabhan, 2000].

4.1 Setup

Experiments are performed based on the office layout in the Department of Computer Science, The Hong Kong University of Science and Technology (Figure 3). There are four hallways with a total of 99 grids, where each grid measures $1.5\text{m} \times 1.5\text{m}$. There are only three access points in Figure 3, though a total of eight access points (including some from other floors) are detected. Each sample is thus an 8-dimensional signal strength vector, with the measurements averaged in one second. 100 such samples are collected at the center of each grid, with a total of 9,900 samples obtained. We randomly use 65% of the 9,900 samples for training and the rest for testing. To reduce statistical variability, results here are based on averages over 10 repetitions.

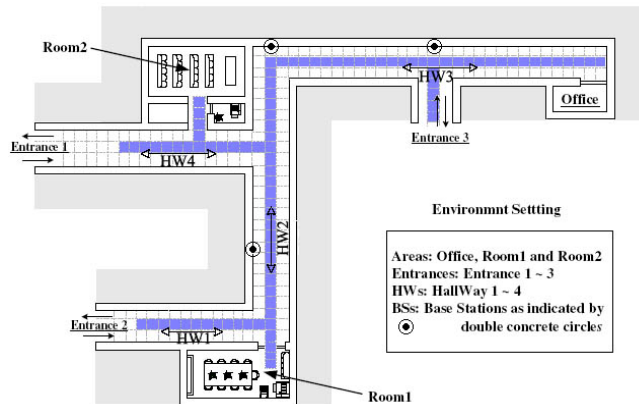


Figure 3: Office layout used in the experiments.

We set w_G in (6) to 0.1, w_M in (7) to 0.05, and M (the number of neighbors used in the localization phase) to 17. Preliminary studies show that the accuracy only changes

slightly (within 1%) when M varies in the range 7 to 31. Moreover, ν in (7) is set to 0.5, and thus the Matérn kernel degenerates to the exponential kernel.

4.2 Location Estimation Accuracy

Figure 4 plots the average testing accuracies (and the corresponding standard deviations) at different acceptable error distances. At an error distance of 3.0m, the accuracy of LE-KCCA is 91.6% while those of SVM, MLE and RADAR are 87.8%, 86.1% and 78.8% respectively. Thus, by utilizing the pairwise distance similarities in physical locations, LE-KCCA obtains better performance than the other methods.

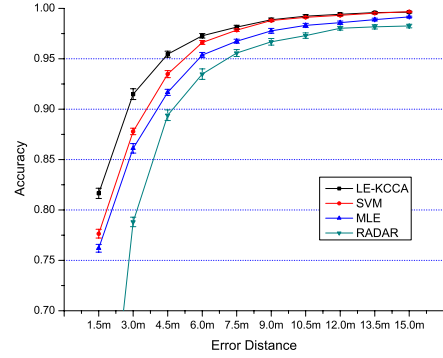


Figure 4: Testing accuracies at different error distances.

4.3 Reduction in Calibration Effort

In the first experiment, we use all the 99 grid locations but only a random subset of the signal samples available at each location for training. Figure 5 shows the testing accuracies at error distances of 1.5m or 3.0m. As can be seen, by using only 10-15 random training samples at each location, LE-KCCA can already outperform the other methods that use a full set.

In the second experiment, we select a subset of the available grid locations, and then use all the training signal samples at those selected locations for training. Evaluation is performed on both the unseen signal samples at the selected grid locations and also at the unseen locations. Again, Figure 6 shows that LE-KCCA yields better performance. Thus, LE-KCCA, by maximizing the correlation between signal and physical spaces, can reduce the calibration effort dramatically. With the physical location similarity as feedback, samples from the same locations move closer together, while those from different locations are pushed away under the feature-space mapping built by LE-KCCA.

We believe that LE-KCCA performs well. To validate this claim, we conduct another three experiments for comparison. The first one treats all the grid locations separately, which can be achieved by using a large value for w_M in the Matérn kernel (in this experiment, we set $w_M = 50$). This is denoted as Isolated-KCCA in the sequel. Another extreme case is to modify the LE-KCCA algorithm so that the signal space is correlated with x and y separately (denoted 1D-KCCA). A third experiment is to modify both Gaussian and Matérn kernels to linear ones (Linear CCA). Experiments are performed in the same setting as for Figure 5. As can be seen in Figure 7, all three variants lead to degraded performance as expected.

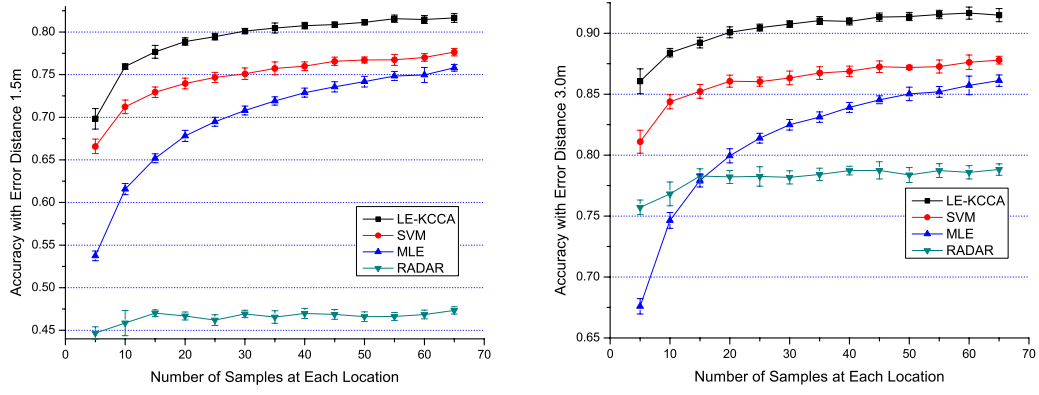


Figure 5: Effect on varying the number of training samples at each grid location.

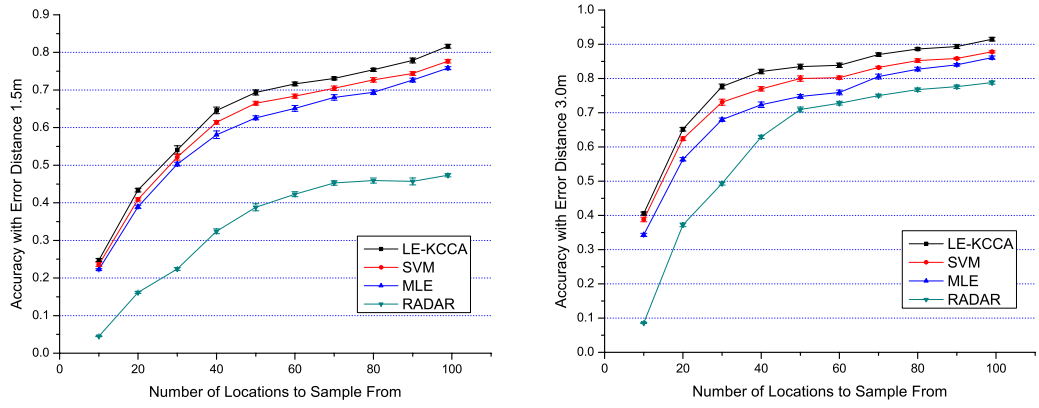


Figure 6: Effect on varying the number of grid locations in training.

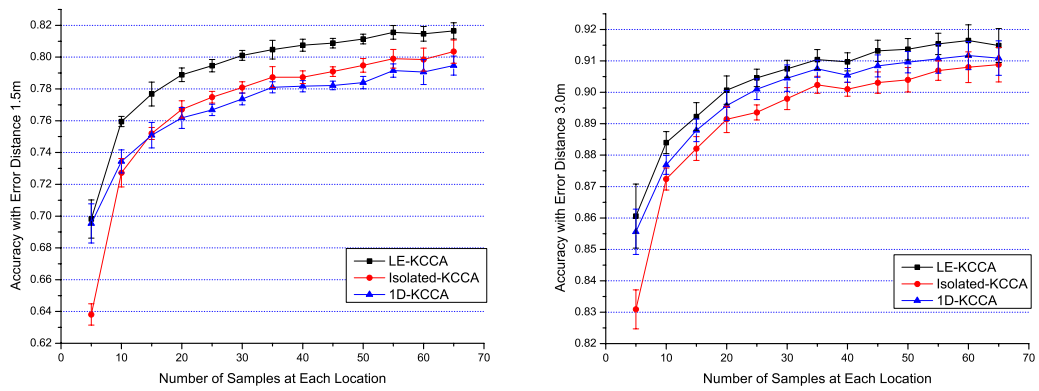


Figure 7: The effect when the information in both physical dimensions (x and y) are not used together. Isolated-KCCA considers each grid location as independent, while 1D-KCCA considers x and y as independent. Linear CCA has even lower accuracy and the curve is not shown in the figure.