Dynamic Workflow Composition using Markov Decision Processes

Prashant Doshi
Dept. of Computer Science
Univ. of Illinois at Chicago
pdoshi@cs.uic.edu

Richard Goodwin and Rama Akkiraju
IBM T. J. Watson Research Center
Hawthorne, NY
{rgoodwin,akkiraju}@us.ibm.com

Kunal Verma
Dept. of Computer Science
Univ. of Georgia, Athens
verma@cs.uga.edu

Abstract

The advent of Web services has made automated workflow composition relevant to web based applications. One technique, that has received some attention, for automatically composing workflows is AI-based classical planning. However, classical planning suffers from the paradox of first assuming deterministic behavior of Web services, then requiring the additional overhead of execution monitoring to recover from unexpected behavior of services. To address these concerns, we propose using Markov decision processes (MDPs), to model workflow composition. Our method models both, the inherent stochastic nature of Web services, and the dynamic nature of the environment. The resulting workflows are robust to non-deterministic behaviors of Web services and adaptive to a changing environment. Using an example scenario, we demonstrate our method and provide empirical results in its support.

1. Introduction

Over the next decade several enterprises will subscribe to the distributed computing paradigm of Web services. These enterprises will provide a Web services based interface to their core business systems. The task of business process integration and management (BPIM) will then involve linking together both intra-enterprise and inter-enterprise services to achieve the desired business objectives. In Fig 1, we illustrate this observation.

Prevalent techniques for BPIM are time consuming and often involve the use of custom and proprietary technology for connecting partners. The advent of Web services has the potential to significantly reduce the cost by introducing standard protocols and helping to automate the process. Specifically, the ability to dynamically discover and bind to desired Web services using Web services discovery mechanisms facilitate automated workflow compositions. If we view workflow composition as a goal-oriented process, AI-inspired planning techniques appear suitable for the task. Preliminary efforts in this respect [12] utilize classical STRIPS-style planning algorithms that assume deterministic behavior of Web services. Additional execution monitoring overhead is required to recover from service failures.

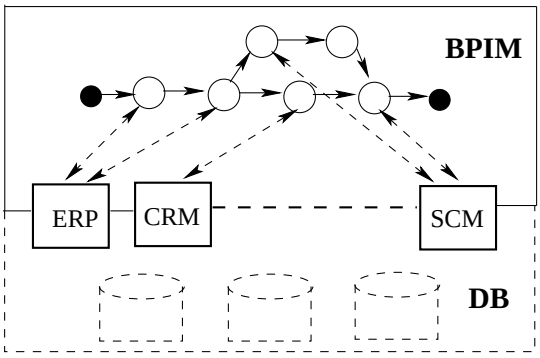

Figure 1. BPIM Layer in Enterprise IT Infrastructure

In this paper we present a novel policy-based method to dynamically choreograph Web services thereby generating sequential workflows. In particular, we focus on automatically establishing the workflow logic and avoid concentrating on the implementation details. Our method models both, the stochastic nature of services, and the dynamic nature of the environment producing workflows that are robust and adaptive. Specifically, we utilize Markov decision processes (MDPs) [9, 10] to model the problem of work-

flow composition. The solution of a MDP produces a policy that *optimally* guides a stateful workflow towards its goal. Additionally, the dynamics of the environment may render the previously computed policy sub-optimal. To account for the dynamic nature of the environment, we intersperse policy computation with model learning. This allows the workflow to gradually adapt itself to the changed environment. We empirically measure the learning convergence rate and report our findings.

The remainder of this paper is structured as follows. In Section 2, we briefly dwell on the workflow composition problem. We also touch upon some related work in this area. We introduce a motivating scenario in Section 3. This scenario is used as a running example for rest of the paper. In Section 4, we introduce our stochastic optimization framework and its application to workflow generation. Section 5 focuses on our model learning approach. Finally, we conclude this paper with a discussion in Section 6.

2. Dynamic Workflow Composition

Workflow composition in the context of Web services involves discovering and binding to Web services that collectively implement the required process functionality. Often, multiple candidate flows may exist that achieve the required process functionality. One criteria for selecting a workflow is to establish a model of optimality and select the workflow that is most optimal.

Definition 1 (Optimal Workflow) *Let $\mathcal{W}$ be the set of candidate workflows. Let C_i^w be the cost associated with each Web service invocation (WS_i) for some workflow,w. Let $\mathcal{N}$ be a fixed maximum number of invocations in each of the candidate workflows. The workflow, w^*, that results in the minimal expected cost is called an optimal workflow. Formally,*

$$w^* = \underset{w \in \mathcal{W}}{argmin}\ E(\sum_{i=1}^{\mathcal{N}} C_i^w) \tag{1}$$

The expectation operator, $E(\cdot)$, in the above equation is necessary due to the stochastic nature of the Web services.

Workflow composition is either static or dynamic, depending on the process environment. Highly dynamic environments require the workflow to adapt continuously. Specifically, the changing stochasticity of the Web services may cause the previously optimal workflow to become sub-optimal. Hence, a new workflow composition must be recomputed taking into account the changed process environment.

Limited prior work exists in the area of applying Web services towards workflow compositions. Laukkanen et. al. [7] outline four distinct steps for composing workflows. These include identifying the required functionality; semantic matching of Web services; creating or updating the workflow; and executing and monitoring the workflow. In this paper we give a method for creating, executing and adapting the workflow to dynamic environments. Our method deems monitoring of workflows for unexpected behavior, unnecessary. A separate paper [5] presents our semantic matchmaking framework for discovering Web services. Another related effort [12] investigates the application of classical STRIPS-style planning for choreographing Web services. However, classical planning assumes a static environment with deterministic Web services outcomes. Furthermore, classical planning exhibits a computational complexity of atleast NP-Complete, and in some cases even PSPACE-Complete [4]. In contrast, our method for generating workflows is P-Complete [8], and efficiently models the stochastic nature of Web services thereby producing robust workflows. The problem of composing adaptive workflows has also received some attention recently. One line of work [3] adopts a multiagent perspective to adaptive workflow composition and suggests the utilization of standard workflow languages for multiagent coordination. However, this work is introductory, and has not been described in sufficient detail to allow for implementation.

3. Motivating Scenario

In order to illustrate the composition of workflows using Web services we present an example motivating scenario. The scenario will serve as a running example for the rest of the paper. We define our example scenario below.

Example Scenario A manufacturer receives an order to deliver some merchandise to a retailer. The manufacturer may satisfy the order in one of several ways. He may satisfy the order from his own inventory if sufficient stock exists. The manufacturer may request parts from his preferred supplier in order to satisfy the order. The manufacturer may also search for a new supplier of parts, or look for parts in a Spot Market. A *costing analysis* reveals that the manufacturer will incur least cost if he is able to satisfy the order from his own inventory. The manufacturer will incur increasing costs as he tries to fulfil the order by procuring parts from his preferred supplier, a new supplier, and the Spot Market.

In Fig. 2, we illustrate our motivating scenario graphically. Clearly, the manufacturer may choose from several candidate workflows. For example, the manufacturer may initially attempt to satisfy the order from his inventory. If he is unable to do so, he may resort to ordering parts from his preferred supplier. Another potential workflow may involve bypassing the inventory check, since the manufacturer strongly believes that his inventory will not satisfy the order. He may then initiate a status check on his pre-

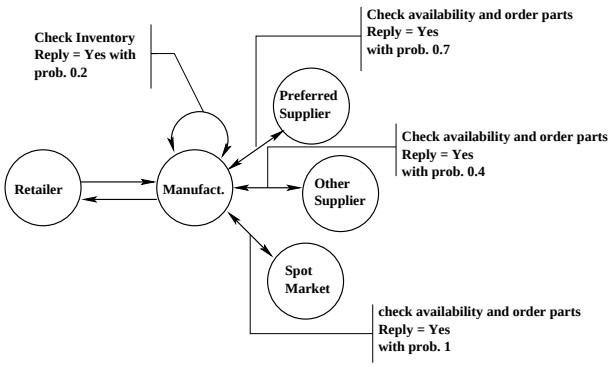

Figure 2. Motivating scenario with example probability values.

ferred supplier. These example workflows reveal two important factors for selecting the optimal one. First, the manufacturer must possess some estimate of certainty (belief) with which his order will be satisfied by his inventory, preferred supplier, supplier, and the Spot Market. One source of these estimates is historical data on past transactions maintained by the manufacturer. Suitable extrapolation techniques allow the estimates to be projected to the current time. In Section 5, we present an alternate approach where the manufacturer is initially ignorant about these estimates, and gradually learns them from his interactions. Pertaining to the second factor, at each stage, rather than greedily selecting an action with the least cost, the manufacturer must select the action which is expected to be optimal over the long term.

In the next section, we present a decision-theoretic framework whose solution is a *policy*. The policy prescribes the expected long-term optimal action to perform at each stage.

4. Markov Decision Processes

If we view workflow composition as a goal-driven problem, then classical planning algorithms appear suitable candidates for automatically composing workflows. However, the classical planning assumptions do not hold as well in this doamin as they might first appear to. It is not the case that web service invocations can be modeled as deterministic. Similarly, simply finding a plan that achieves a goal is insufficient. Workflow composition needs to take into account not only the cost and time needed to accomplish a task, but also the factors included in service level agreements. These aspects of the problem are better addressed by decision-theoretic planning techniques such as Markov decision processes (MDPs) [9, 10].

A MDP models the real-world as inherently stochastic, sequential and fully observable. Modeling the workflow composition problem as a MDP and solving it produces a policy or a "universal plan". A policy assigns to each state of the workflow, an action that is expected to be optimal over the period of consideration. Thus, if an agent executing the workflow has a policy, then no matter what the outcome of any action is, the agent will always know what to do next. We formally define an MDP below.

Definition 2 (Markov Decision Process (MDP)) *A Markov decision process is a sextuplet,* $\mathcal{M} = (S, A, T, C, H, s_0)$ *where* S *is the set of all possible states;* A *is the set of all possible actions;* T *is a transition function,* $T : S \times A \rightarrow \Delta(S)$*, which specifies the probability distribution over the next states given the current state and action;* C *is a cost function,* $C : S \times A \rightarrow \mathbb{R}$*, which specifies the cost of performing each action from each state;* H *is the period of consideration over which the plan must be optimal, also known as the horizon,* $0 < H \leq \infty$*; and* s_0 *is the starting state of the process.*

The transition function as defined in a MDP is Markovian, that is, the probability of reaching the next state depends only on the current state, and not on the history of earlier states. Inclusion of the transition function allows MDPs to model and reason with non-deterministic actions. Furthermore, the horizon may be either finite or infinite. If a MDP is solved over a finite horizon, then the resulting policy is *non-stationary*. It is a sequence of policies, indexed by time, that specifies the optimal web service to invoke in each state. The optimal web service to invoke can vary as the finite horizon is approached. If the horizon is infinite, then the resulting policy is *stationary* over time.

In order to gain insight into the functioning of MDPs, let us model the example scenario as a Markov decision problem.

Example 1 *The state of the workflow is captured by the random variables –* ***Inventory Availability****,* ***Preferred Supplier Availability****,* ***New Supplier Availability****,* ***Spot Market Availability****,* ***Order Assembled****, and* ***Order Shipped****. A state is then a conjunction of assignments of either* Yes*,* No*, or* Unknown *to each random variable. Actions are Web service invocations,* A={*Check Inventory Status, Check Preferred Supplier Status, Check New Supplier Status, Check Spot Market Status, Assemble Order, Ship Order*}*. The transition function,* T*, models the non-deterministic effect of each action on some random variable(s). For example, invoking the Web service Check Inventory Status will cause* ***Inventory Availability*** *to be assigned* Yes *with a probability of* T(***Inventory Availability***=Yes|*Check Inventory Status,* ***Inventory Availability***=Unknown)*, and assigned* No *or* Unknown *with a probability of (1-*T(***Inventory Availability***=Yes|*Check Inventory Sta-*

*tus,**Inventory Availability***=Unknown*)). The cost function, C, prescribes the cost of performing each action. We let $H \rightarrow \infty$ which implies that the* manufacturer *is concerned with getting the most optimal workflow possible. Since no information is available at the start state, all random variables will be assigned the value* Unknown.

Once our manufacturer has modeled his workflow composition problem as a MDP, he may apply standard MDP solution techniques to arrive at an optimal workflow. These solution techniques revolve around the use of stochastic dynamic programming [9] for calculation of the optimal policy. Bellman [2], via his Principle of Optimality, showed that the stochastic dynamic programming equation given below is guaranteed to find the optimal policy for the MDP.

$$V^n(s) = \begin{cases} \min\limits_{a \in A} \left\{ C(s,a) + \sum\limits_{s' \in S} T(s'|a,s) V^{n-1}(s) \right\} & \text{n>0} \\ 0 & \text{n=0} \end{cases} \quad (2)$$

where the function, $V^n : S \rightarrow \mathbb{R}$, quantifies the long-term negative value, or cost, of reaching each state with n actions remaining to be performed.

Once we know the cost associated with each state of the workflow, the optimal action for each state is the one which results in the minimum expected cost.

$$\pi^*(s) = \underset{a \in A}{argmin} \left\{ C(s,a) + \sum_{s' \in S} T(s'|a,s') V^{n-1}(s) \right\} \quad (3)$$

In Equation 3, π^* is the optimal policy which as we mentioned before, is simply a mapping from states to actions, $\pi^* : S \rightarrow A$. The reader at this point may wonder that how does an optimal policy such as π^* translate to an optimal workflow such as w^*. In Fig. 3, we give an algorithm that addresses this question. It takes the optimal policy, and the starting state of the workflow as input, and interleaves composition and execution of the workflow.

In addition to providing a guarantee of optimality, MDPs admit efficient solution techniques. In particular, the task of computing the policy of a MDP has been proved to be P-Complete [8]. Furthermore, we contend that employing a policy for workflow generation produces robust workflows. Since a policy is a mapping from each state to an action, no matter what the current state of the workflow is, the policy will always prescribe an optimal Web service to execute at that state. For example, if a Web service fails, then the state of the workflow remains unchanged. In such a situation, the policy may prescribe either the same action as before, or a completely different one, depending on which action is optimal then. Thus, our policy-based method for generating workflows is capable of optimally recovering from Web service failures. In contrast, execution of traditional classical plans needs to be monitored for unexpected interaction between the plan and the environment, and there is no fixed method for handling exceptions that may arise.

Algorithm for generating workflow
Input: π^*, s_0
$s \leftarrow s_0$
while goal state not reached
$a \leftarrow \pi^*(s)$
Execute Web service a
Get response of a and construct next state, s'
$s \leftarrow s'$
end while
end algorithm

Figure 3. Algorithm for translating a policy into a workflow

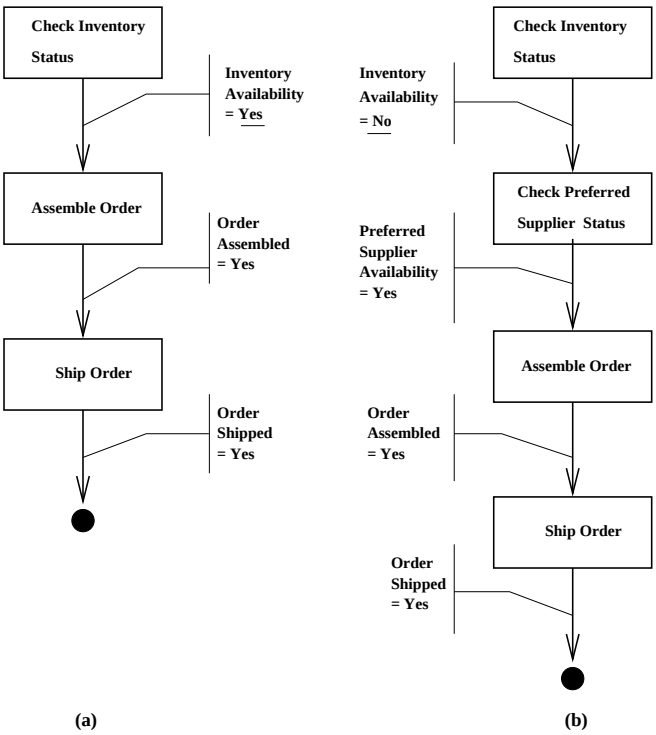

Figure 4. Variations in workflow execution as responses to Web service invocations change

In Fig. 4, we present two example workflow traces that arise when the manufacturer uses the algorithm in Fig. 3 to compose and execute his workflow. As expected, the workflow changes as responses to Web service invocations

change. Specifically, if the manufacturer's inventory is able to satisfy the order, then he will assemble the order and ship it to the retailer (Fig. 4.(a)). However, if the inventory is unable to satisfy the order (Fig. 4.(b)), the manufacturer checks his preferred supplier's status. On receiving a positive response, the manufacturer assembles the order using parts supplied by his preferred supplier, and ships the order to the retailer.

The current industry standard for representing workflows is BPEL4WS [1]. BPEL4WS which grew out of Microsoft's XLANG and IBM's WSFL specifications, is receiving widespread recognition by the industry. To integrate our method with industry standards, we have developed a tool for automatically converting a policy into a workflow in BPEL4WS specification using the BPWS4J API [6]. The resulting BPEL4WS flow can be executed in the BPEL engine that comes bundled with IBM's Websphere application suite.

5. Model Learning

In Section 3, we indicated that in order to select the optimal workflow from several candidate ones, the manufacturer must possess some estimate of certainty with which his requests will be satisfied. These estimates are a measure of the non-determinism of Web services and manifest themselves in the transition function, T, of the MDP. Clearly, the optimal policy, π^*, is dependent on these estimates, and the policy, and consequently the workflow changes as the probabilities vary.

Figure 5 shows how the manufacturer's optimal workflows depends on probability estimates. The optimal workflow changes from the one in Fig. 5.(a) to the one in Fig. 5.(b) as the manufacturer's estimate of the probability that inventory can satisfy his request drops. If this estimate is sufficiently low, then at the start state, the expected value of checking the inventory (computed using the expression in parenthesis in Equation 3) is exceeded by the expected value of checking the preferred supplier's availability. In such an event, the optimal workflow changes as shown in the figure.

Though it is possible to derive reasonably accurate estimates of the true probabilities from historical records of transactions, in this part of our work, we take the view that the manufacturer is initially clueless about these probabilities. Hence, in true Bayesian manner, the manufacturer assigns equal probabilities to each response of a Web service invocation. An initial workflow is generated and executed using the algorithm in Fig 3. Using a Bayesian learning algorithm, Web service invocation responses obtained during workflow execution are used to update the manufacturer's estimates of "ground truth".[1] In this manner, we

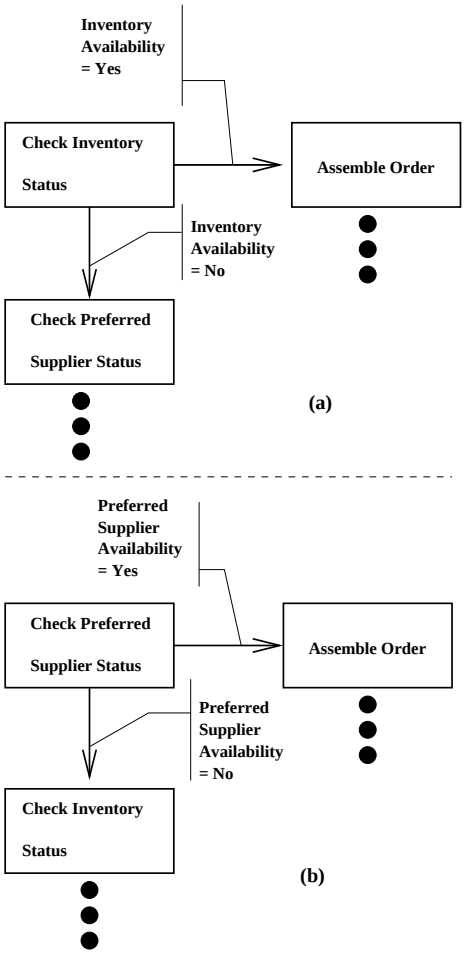

Figure 5. When T(Inv Avail = *Yes*|Chk Inv Stat, Inv Avail = *Unknown*) drops below some threshold, the optimal workflow changes from (a) to (b)

interleave workflow generation with model learning, and repeat this procedure until the sequence of workflows converges. Convergence arises out of the proposition that updating the probabilities using a Bayesian learning algorithm leads almost surely to the true probability. Thus, the manufacturer's workflows slowly adapt themselves to the ground truth through repeated interactions with the environment. In the unusual case of a dynamic environment (the true probabilities are varying), model learning allows the workflow to "keep up" with the changing environment.

Our Bayesian learning algorithm is rather simple, and has its roots in [11]. The algorithm maintains an experience counter, *exper*, initialized to 1, for each value of a random variable. During workflow execution, when a Web ser-

1 This form of learning is frequently called parameter learning in Bayesian statistics

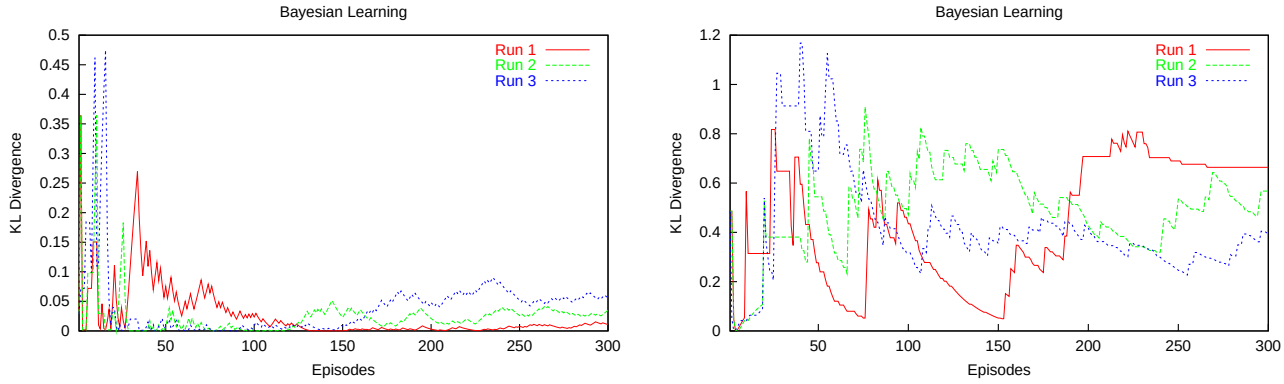

Figure 6. Plots displaying the performance of Bayesian learning of distributions over (a) Inventory Availability (b) Preferred Supplier Availability

vice invocation, a, causes some random variable(s), $\mathbf{X}$, to change its value from x to x', the experience associated with the new value is incremented by 1. The updated probability, $T'(\mathbf{X} = x'|a, \mathbf{X} = x)$, for that value is calculated from the prior probability, $T(\mathbf{X} = x'|a, \mathbf{X} = x)$, as

$$T'(\mathbf{X} = x'|a, \mathbf{X} = x) := \frac{T(\mathbf{X} = x'|a, \mathbf{X} = x) \times \textit{exper} + 1}{\textit{exper'}}$$

where *exper'* is the incremented counter. In order to make the probability distribution over $\mathbf{X}$ sum to 1, probabilities of rest of the values of $\mathbf{X}$ are updated in the following manner:

$$T'(\mathbf{X} = y|a, \mathbf{X} = x) := \frac{T(\mathbf{X} = y|a, \mathbf{X} = x) \times \textit{exper}}{\textit{exper'}}$$

The example given below, illustrates our learning process.

Example 2 *Let us apply the Bayesian learning algorithm outlined above to update the probability distribution over the random variable,* ***Inventory Availability (X)****. The* manufacturer *initially assigns a uniform probability distribution to the random variable. We assume that on invoking the Web service,* Check Inventory Status (a)*, a positive response is obtained, and* ***Inventory Availability*** *is assigned* Yes*. The updated probability distribution,* $T'(\cdot|a, \boldsymbol{X} = \text{Unknown})$*, is:*

$$T'(\boldsymbol{X} = \text{Yes}|a, \boldsymbol{X} = \text{Unknown}) := \frac{0.33 \times 1 + 1}{2} = 0.67$$
$$T'(\boldsymbol{X} = \text{No}|a, \boldsymbol{X} = \text{Unknown}) := \frac{0.33 \times 1}{2} = 0.165$$
$$T'(\boldsymbol{X} = \text{Unknown}|a, \boldsymbol{X} = \text{Unknown}) := \frac{0.33 \times 1}{2} = 0.165$$

T' becomes the new prior for the next Bayesian learning phase.

We empirically measured the performance of our Bayesian learning algorithm. In particular, we were interested in knowing the number of *episodes* that must elapse before the estimated probabilities converge to the true probability. An episode consists of a single pass through the algorithm given in Fig 3, combined with Bayesian updating of the probability estimates. At the end of each episode, we measured the distance between the updated estimates and the true probability distributions. In order to measure the distance between probability distributions, we require a metric. We used *Kullbach Leibler Divergence* (KLD), also known as relative entropy, as the metric for measuring the distance between two probability distributions.

Definition 3 (Kullbach-Leibler Divergence (KLD)) *The KLD between two probability distributions, p and q, is defined as:*

$$D(p||q) := \sum_x p(x) log_2 \frac{p(x)}{q(x)} \tag{4}$$

Note that KLD is not a true metric. Specifically, it is asymmetric and does not obey the triangle inequality, though it is non-negative. Furthermore, $D(p||q) = 0$ if $p = q$.

The performance plots are displayed in Fig. 6. Our methodology for generating these plots involved running an episode of workflow generation using current probability estimates interleaved with Bayesian learning; measuring the KLD between the estimated probability distribution and the true distribution, for each random variable; and plotting the KLD value. Fig. 6.(a) shows the plot of three independent runs of learning the probability distribution over the random variable **Inventory Availability**, measured over 300 episodes. Clearly, after initial fluctuations, the estimated probabilities have almost converged to the true ones. Notice that convergence has almost been attained by the 100^{th} episode. In Fig. 6.(b), we show the plot of three independent runs for the probability distribution over the random variable **Preferred Supplier Availability**, over 300 episodes.

In this case, even after 300 episodes, the distribution is yet to converge, exhibiting a large KLD variance in all the 3 runs. We observed that in several workflows, the Web service Check Preferred Supplier Status was not invoked, since the order was satisfied by the inventory itself. Subsequently, few episodes effected a Bayesian update of the distribution over **Preferred Supplier Availability** thereby slowing down its convergence. When the same experiment was carried out over 500 episodes, KLD almost converges to zero (as expected).

Our experiments provide two important conclusions. First, the experiments validate our hypothesis that the Bayesian learning approach is effective for model learning. Second, the results reveal an important observation that during learning, models of events less likely to be observed take more runs to converge. This is a natual result of our use of maximum entropy to initialize the model and the structure of the Markov model. It has the beneficial effect that it tends to focus information gathering on the most relevant states and provides a mechanism for gradually reducing exploration as probabilities converge.

6. Discussion

In this paper, we have presented a novel policy-based approach for dynamically choreographing Web services resulting in workflows. Our primary focus has been on assembling the workflow at an abstract level, ignoring the implementation-level details. We believe that our work is novel in two respects: (1) Instead of ignoring the non-determinism inherent in real-world Web services, we have utilized a stochastic optimization framework, namely Markov decision processes, that permit us to model this uncertainty and reason with it. Furthermore, MDPs allow us to associate a measure of quality with each workflow, thereby facilitating selection of the optimal one. By efficiently generating policies, they produce workflows that are tolerant of service failures and uncertainties. (2) We have interleaved workflow generation and execution with model learning, thereby acknowledging that the true stochastic information (”ground truth”) may not be accurately known *a’priori*. Through empirical experiments, we have demonstrated the effectiveness of our Bayesian learning algorithm for learning the true probability models. The net result is that our method generates robust and adaptive workflows.

We believe that our method will scale well to composing complex workflows. In order to validate this belief, we are currently testing our approach with several large real-world workflows. The current industry standard for representing workflows is BPEL4WS. We feel that this standard is not expressive enough for representing intricate workflows. Hence, another focus of our future work will be to improve the richness of BPEL4WS thereby facilitating its widespread adoption.

References

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, and D. Roller. Business process execution language for web services. version 1.1. Technical report, IBM, May 2003.

[2] R. Bellman. *Dynamic Programming*. Dover Publications, 1957.

[3] P. A. Buhler and J. M. Vidal. Adaptive workfbw enactment = web services + agents. In *International Conference on Web Services*, 2003.

[4] T. Bylander. Complexity results for planning. In *Proceedings IJCAI 12*, pages 274–279. IJCAII, 1991.

[5] P. Doshi, R. Goodwin, R. Akkiraju, and S. Roeder. Parameterized semantic matchmaking for workfbw composition. Technical Report RC23133(W0403-026), IBM Research, 2004.

[6] IBM. Business process execution language for web services java runtime. http://www.alphaworks.ibm.com/tech/bpws4j, 2002.

[7] M. Laukkanen and H. Helin. Composing workfbws of semantic web services. In *Workshop on Web Services and Agent-based Engineering, AAMAS*, Melbourne, Australia, 2003.

[8] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[9] M. L. Puterman. *Markov Decision Processes:Discrete Stochastic Dynamic Programming*. Wiley series in probability and mathematical statistics. Wiley-Interscience, 1994.

[10] S. Russell and P. Norvig. *Artifi cial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[11] D. Spiegelhalter, A. Dawid, S. Lauritzen, and R. Cowell. Bayesian analysis in expert systems, 1993.

[12] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. Automatic web services composition using shop2. In *Workshop on Planning for Web Services, ICAPS*, June 2003.