There Can Be Composite Services When You Believe

Yu Lei

Abstract—Web services on the internet are distributed, heterogeneous, autonomous and dynamic, which leads to uncertainty of web services. These uncertain factors affect success rate of service composition. The web service composition problem should be considered as an uncertain planning problem. Therefore, we use an uncertainty planning method to deal with the uncertain planning problem for service composition. Without knowing complete information, our method uses an estimated value function to obtain a composite service. We give details and examples that how to compose services by our method. In addition, we shows relations among user requirements, service invocations and Quality of services. The experiments verify the validity of the algorithm.

Index Terms—Quality of Web service; incomplete information; partially observable markov decision process; temporal difference learning

I. INTRODUCTION

Who knows what composite services you can achieve, when you believe somehow you will. You will when you believe. They occasionally happen when you ask, and it is easy to give in to your fear, but when you are blinded by unstable QoS, cannot see your way safe through the workflow, POMDP is still a resilient voice, says mashup is very near.

Web services on the internet are distributed, heterogeneous, autonomous and dynamic, which leads to two kinds of uncertainty of web services. Uncertainty of web services refers to the uncertain results of invoking services and the uncertain QoS (Quality of Services) values, which are hard to precisely predict and affect the process of service composition.

For the first uncertainty, behaviors of web services or results of web services are the main reasons [1]. For the second uncertainty, values of QoS, such as response time and availability etc., are uncertain because other factors such as network delays constantly change QoS values. This phenomenon exists and is generally solved by providing an average value [2], but we suggest that using uncertain QoS values with probability distribution is more realistic.

One more significant and similar example is e-Marketplace, which is a dynamic supply chain for textile industry [24]. In addition, uncertain and dynamic environments are widespread, especially in wireless networks, and some researchers [19-21] already have studied on the service composition in their specific areas.

When taking into account the uncertain and dynamic nature of real-world cases, the following considerations have practical significance for reliable services composition: how to compose services based on implicit knowledge of business logics and how to model uncertain QoS values to ensure that composite services have higher QoS.

Because of the uncertainty of web services and QoS constraints, web service composition must take into account the results of services to dynamically adjust web service composition. Therefore, web service composition should not only be a composition plan under certain specific restrictions, but also be an optimal policy of web service composition in certain conditions. Focusing on uncertainty and QoS constraints of web services, this article views the problem of web service composition as an uncertainty planning by Partially Observable Markov Decision Processes (POMDP) [3].

Partially Observable Markov Decision Processes is a decision-making method solving a planning problem, which is an extension of Markov Decision Processes (MDP). POMDP assumes that the information of the system states is just partially knowable and cannot be observed directly, thus it models the system with incomplete information, and then makes decisions based on the current incomplete information. POMDP has a wide range of application areas [4]. Inspired by reinforcement learning [5], we use a learning method to compose web services in uncertain environments.

Contributions in this paper:

Integrating with reinforcement learning, we give details of how to use POMDP to model the problem of service composition in the uncertain environment.

After comparing three learning algorithms from both theoretical perspective and experimental perspective, and indicating sufficiency of these methods, we propose a method and verify the performance and adaption of this algorithm in the domain of service composition.

The rest of this paper is organized as follows. The second section introduces related research on the service composition based on MDP. The third section describes service composition based on POMDP. The fourth section proposes a learning algorithm to address the service composition problem (i.e. the POMDP problem). The fifth section shows extensive experiment results of the service composition. The final section summarizes the paper.

II. RELATED WORK

There are a variety of service composition methods based on the Artificial Intelligence planning, such as classic planning which is based on state-space [6], neoclassic planning which is based on propositions and constraints satisfaction, heuristics
and control policy planning which is based on heuristics function [7], logic planning which is based on model checking [8], graph planning which is based on Graphplan [9, 10], and uncertainty planning which is based on Markov Decision Processes.

Gao et al. [11] modeled service composition based on MDP, viewing a task collection as a state, and viewing success rate of invocations as transition probabilities. They compared and analyzed two algorithms that can solve MDP, which are the forward value iteration algorithm and the afterward value iteration algorithm, also they proved that the forward value iteration algorithm is more suitable for dynamic web services environments.

We are tracking the studies of Doshi. Doshi et al. [1] proposed a scene that a retailer requests products by querying product inventory from many providers, and then they used MDP to compose services that have the uncertain business logic, afterwards used a Bayesian learning method to learn the transition probability among services. Based on the previous literature, they [12] further proposed a composition method of hierarchical semi-Markov decision process, where they considered the service execution time and added the ability to decompose tasks. Recently they [13] proposed a method combined with risk preferences to adapt a composite service to different types of users.

Chen et al. [14] transferred the service composition problem into the artificial intelligence planning problem, and then combined MDP with Hierarchical Task Network (HTN) to solve the problem. The method first decomposed service requirements into sub-requirements by means of HTN, and then used MDP to solve the sub-requirements. The article indicated that the fewer are constraints of a service, the bigger is the probability that the service is selected. Wang et al. [15] proposed an adaptive approach for service composition, which mapped services to actions of MDP and constructed a map of service network. Each path generated by the method is a workflow path. Furthermore, to prove the adaptability of the composition method, the paper also conducted experiments and analysis on learning efficiency. Wang et al. [16] went a step further by combining the Semi-Markov Processes with the preference logic to solve the service composition problem. Semi-Markov Processes was used for modeling service execution time, and the preference logic was used for modeling the preference degree that different users prefer the different QoS criteria. Li et al. [17] extended a finite state machine model and introduced a probabilistic computation tree logic to denote MDP properties, such as the states accessibility and the probability of success for service composition, and then used a probabilistic model checker, PRISM, to analyze and verify reliability and cost of composed service. Fan et al. [18] proposed a new method for measuring QoS to adaptively update QoS values, and designed a reliable composition algorithm of web service based on MDP.

The current insufficiency of the service composition methods based on MDP is the assumption that QoS values can be accurately provided, but due to uncertain network environments and uncertain invocation results of Web services, this assumption is too optimistic. Therefore, this paper proposes a new service composition method based on learning algorithm to deal with this problem.

### III. Problem Formulation

In our problem of web service composition, the possible structure of the composite service is predefined, and multiple services with the same functionality have been grouped already. Our goal is to select a best business process and its services, and the principle of selection is based on rewards that will be described below.

#### A. Model parameters in POMDP

Web service composition can be modeled by POMDP as follows:

- **S**: State, a finite set of states of the world [15]. Suppose we have a service for an example. The state is whether or not the service is our needed service:
  - \( S = \{ \text{Yes, Not} \} \)
- **A**: Action, namely web service invocation. Results of web service invocations are uncertain.

  The actions can be to do nothing, invoke the service or query a service center about this service.

  - \( A = \{ \text{No-op, Invoke, Query} \} \)
- **T**: \( S \times A \times S \rightarrow [0,1] \). \( T \) is a state transition function, expressing the probability distribution that the current services transit to the next services after an invocation of the current services.

  The state transition functions:

<table>
<thead>
<tr>
<th>T( StartState, No-op, EndState )</th>
<th>EndState</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartState</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>0.98</td>
</tr>
<tr>
<td>Not</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T( StartState, Query, EndState )</th>
<th>EndState</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartState</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>0.98</td>
</tr>
<tr>
<td>Not</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T( StartState, Invoke, EndState )</th>
<th>EndState</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartState</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>1.00</td>
</tr>
<tr>
<td>Not</td>
<td>1.00</td>
</tr>
</tbody>
</table>

- **R**: \( S \times A \rightarrow \mathbb{R} \). \( R \) is a reward function, which means the reward that we take action \( A \) in the current service \( S \) and transit to the next service \( S' \).

  The reward function:
**Partially Observable Service Composition**

POSC (Partially Observable Service Composition) is a POMDP problem, or a belief-MDP problem. POSC = (B, A, Z, Γ, ρ). The service composition can be described by this 6-tuples, and a generated optimal policy is a composed service.

As we mentioned earlier, B is a belief state in the continuous space, meaning the belief degree that we select one service. The value of belief state is \( b(s) \in [0,1] \), where \( \sum b(s) = 1 \). All belief states form a geometric simplex with \( |S| - 1 \) dimensions. A and Z respectively are the finite sets of actions and observations, the same as those of MDP. \( \Gamma : \mathbb{B} \times \mathbb{A} \times \mathbb{Z} \rightarrow \mathbb{B} \) is a belief state transition function, meaning the probability of being the belief state \( B' \), when taking the action \( A \) in the current belief \( B \) and then obtaining the observation \( Z \). \( \rho : \mathbb{B} \times \mathbb{A} \rightarrow \mathbb{R} \) is a reward function in the belief space, namely \( \rho(b,a) = \sum s R(s,a) x(b,s) \).

Solving POMDP problems is to select an approximate optimal policy. Policy \( \pi \) is a function mapping belief state \( b \) (\( b \in \mathbb{B} \)) into action \( a \) (\( a \in \mathbb{A} \)):

\[
\pi(b) \rightarrow a
\]

(1)

For methods that not use belief states to infer the states, instead maintain a historical sequence, the policy also can be defined as:

\[
\pi(b_0, h_t) \rightarrow a
\]

(2)

\( h_t \) is the history at the moment \( t \), \( b_0 \) is the initial belief state. It's necessary to consider future \( h \) steps before making a decision. In addition, two problems need to be considered, that are how to decide a policy is the optimal policy, and how to compare two policies. Therefore, a criterion must be specified to decide what the optimal policy is. This paper uses the maximum-expected total-discount-benefits criterion to measure the policy, \( E[\sum_b R(s_t, a_t) \times \gamma^t] \).

Value function is a function mapping each action in a state into a reward so that we can see which action is better than others [23], thus it indicates how good one service invocation is. The optimal value function of POMDP reflects, from a long-term point of view, how many benefits a service contributes, as shown in formula 5. All of optimal value functions constitute an optimal policy.

\[
V^*_t(b_t) = \max_a [\rho(b_t, a) + \gamma \sum_{z \in Z} P(z|b_t, a) V^*_{t-1}(b')] \]

(3)
Optimal policy is a strategy that decides to take which action in each step (or state) [23], thus it indicates how good one composite service is. Its strategy is equal to a composite service. Its value reflects the overall rewards of this composite service, as shown below.

$$\pi^*_t(b) = \arg \max_a [\rho(b,a) + \gamma \sum_{z \in Z} P(z|b,a)V^*_t(b')]$$

The optimal policy may be:

<table>
<thead>
<tr>
<th>Pr( State=Yes )</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000 to 0.237</td>
<td>invoke</td>
</tr>
<tr>
<td>0.237 to 0.490</td>
<td>query</td>
</tr>
<tr>
<td>0.490 to 0.512</td>
<td>query</td>
</tr>
<tr>
<td>0.512 to 0.713</td>
<td>no-op</td>
</tr>
<tr>
<td>0.713 to 0.912</td>
<td>no-op</td>
</tr>
<tr>
<td>0.912 to 0.989</td>
<td>no-op</td>
</tr>
<tr>
<td>0.989 to 1.000</td>
<td>no-op</td>
</tr>
</tbody>
</table>

IV. GENERATE COMPOSITE SERVICES BASED ON LEARNING ALGORITHMS

A. Iterative formula

Belief state $B'$ at time $t$ can be denoted by belief state $B$ at time $t-1$ by means of recursive calculation. When we take action $A$ and then obtain observation $Z$, belief state $B$ will be updated to $B'$ based on Bayes rule, as follows:

$$b' = P(s'|b,a,z) = \frac{P(s',b,a,z)}{P(b,a,z)}$$
$$= \frac{P(z|s',b,a)p(s'|b,a)p(b,a)}{P(z|b,a)p(b,a)}$$
$$= \frac{P(z|s',a)p(s'|b,a)}{P(z|b,a)}$$
$$= \frac{P(z|s',a)}{P(z|b,a)}\sum_{s \in S}P(s'|s,a)\times b(s)$$

Where $P(z|b,a)$ can be obtained by Equ.6.

$$P(z|b,a) = \sum_{s' \in S} P(z|s',a) \times \sum_{s \in S} P(s'|s,a) \times b(s)$$

By iteratively updating belief state, web services can be composed according to the following circulate procedure: (1) when state is $s_{t-1}$ and belief state is $b_{t-1}$; (2) we select actions $a_{t-1}$; (3) obtain observations $z_{t-1}$ and rewards $r_{t-1}$; (4) move to a new service $s_t$ in accordance with state transition function $T(s,a,s')$, and then belief state is updated to $b_t$. New service $s_t$ is selected for the next service.

B. Using reinforcement learning

Reinforcement learning is a kind of Machine Learning algorithm in the Artificial Intelligence domain. It uses the observed reward to learn an optimal policy (or approximate optimal policy), in order to make the cumulative reward maximum.

When our method selects an action, it needs to reach a compromise between exploring (exploration) new strategies and taking advantage of (exploitation) the optimization strategies that are already retrieved, in order to avoid local optimal solutions.

We propose a Time-based Learning method (TL) that is suitable for service composition. The method decides how and when to explore and exploit. Its basic idea is to decline the number of exploration over time and to increase the number of exploitation over time.

$$TL(t) = \max \frac{1}{\ln t+1}\left[\frac{1}{\ln t+1}Q(b_{t+1},a_{t+1}) - Q(b_t,a_t)\right]$$

In the formula above, TL function will obtain the maximum value from the exploration (left part of max) and the exploitation (right part of max), i.e. the times of trying new services will decrease, and the times of using retrieved services will increase over time. Time-based learning method is shown below:

**Algorithm: Time-based learning method**

Inputs: the current belief $b_{t+1}$ and reward $r_{t+1}$

Outputs: an action $a_{t+1}$

Variables: $Q(b,a)$, which is a table storing belief $b$ and action $a$.

1. Steps:
2. Begin
3. If $b$ is not null
4. Then
5. \( TL(t) = \max_{a_{t+1}} \left\{ \frac{1}{\ln t} \ln Q(b_{t+1}, a_{t+1}) - Q(b_t, a_t) \right\} \)

6. \( Q(b, a) \leftarrow Q(b, a) + \alpha \left[ r_t + \gamma TL(t) - Q(b, a) \right] \)

7. Increase \( t \)
8. End If
9. If \( b_{t+1} \) is terminal
10. Then \( b_t, a_t, r_t \leftarrow \text{null} \)
11. Else
12. \( b_t \leftarrow b_{t+1} \)
13. \( a_t \leftarrow a_{t+1} \)
14. \( r_t \leftarrow r_{t+1} \)
15. End If
16. End Begin

In the 11th-14th line, we iteratively update the belief, action and reward. A strategy determines a composite service. We compose services by learning strategies.

V. SCENARIO

We give an example of service composition in the uncertain environment. Supposing there are two Web services, and each has one method, say method A and method B. One of the two methods can return data required by invokers. The invokers have following choices and corresponding rewards.

Table 1. Rewards, actions and states

<table>
<thead>
<tr>
<th>Rewards (R)</th>
<th>Method B returns the needed data (Nb)</th>
<th>Method A returns the needed data (Na)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke method A (Ia)</td>
<td>-100</td>
<td>80</td>
</tr>
<tr>
<td>Invoke method B ( Ib)</td>
<td>90</td>
<td>-100</td>
</tr>
<tr>
<td>Invoke nothing (In)</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

According to the above values, we draw two lines corresponding to \( Ia \) and \( Ib \) separately in the following figure. This figure shows that the expected rewards and optimal actions depend on the value of \( p \). If \( p = 0 \), then \( R(Ia) = 80 \) and \( R(Ib) = -100 \), thus we should invoke method A.

Supposing we have two ranks of QoS statistic values, say good QoS (\( Qg \)) and medium QoS (\( Qm \)). If the probability of good QoS of method B is 0.7, then the probability of medium QoS of method B is 0.3. Likewise, we give the following assumption:

\[
\begin{align*}
P(Qg \mid Nb) &= 0.7 \\
P(Qm \mid Nb) &= 0.3 \\
P(Qg \mid Na) &= 0.3 \\
P(Qm \mid Na) &= 0.7
\end{align*}
\]

If we obtain the value of \( Qg \), then we change our belief of invoking method B. This change can be described by:
\[ p' = P(Nb \mid Q_g) = \frac{P(Q_g \mid Nb)P(Nb)}{P(Q_g)} = \frac{0.7p}{0.7p + 0.3(1 - p)} \]

If we cannot sure which method is better, we can stay and obtain more accurate and latest values either from the Web service or a central server. In this case, the reward of staying is \(-1\) and we have new rewards of invoking method A and method B. The rewards of actions are given below.

\[
R(Ia \mid Q_g) = R(Ia, Nb)P(Nb \mid Q_g) + R(Ia, Na)P(Na \mid Q_g) = -100p + 80(1 - p')
\]
\[
R(Ib \mid Q_g) = R(Ib, Nb)P(Nb \mid Q_g) + R(Ib, Na)P(Na \mid Q_g) = 90p - 100(1 - p')
\]
\[
R(In \mid Q_g) = -1
\]

Likewise, if we obtain the value of \(Q_m\), then we change our belief and rewards. These changes can be described by:

\[
R(Ia \mid Q_m) = R(Ia, Na)P(Na \mid Q_m) + R(Ia, Nb)P(Nb \mid Q_m)
\]
\[
R(Ib \mid Q_m) = R(Ib, Na)P(Na \mid Q_m) + R(Ib, Nb)P(Nb \mid Q_m)
\]
\[
R(In \mid Q_m) = -1
\]

Our aim is to find the optimal action with maximum rewards in certain QoS. Through the following formula we can achieve our aim.

\[
\sum_j \left[ \max_i R(I_i \mid Q_j) \right] P(Q_j)
\]

We can deduce that the probability of \(Q_g\) is \(P(Q_g) = 0.7p + 0.3(1 - p)\). Therefore, we have new rewards after choosing stay as the following shows.

\[
\sum_j \left[ \max_i R(I_i \mid Q_j) \right] P(Q_j) = \max \left\{ R(Ia \mid Q_g) \right\}
\]

Consequently we have the expected rewards as the following figure shows. This figure means we have better choices than we have in Fig.1. After staying and obtaining more values, this figure shows the price of information. In this figure, we should choose one optimal action from two immediate actions and one discounted future action.

![Fig.2 Optimal actions](image)

VI. EXPERIMENTS

Our simulation computer has following configurations: 2.13GHZ Intel Core2 and 2GB RAM. In the simulation, we randomly generate transitions, rewards and observations. The related discount factor is 0.9, learning rate is 0.2, and \(\epsilon\) is 0.6.

A. Comparative methods

One traditional method and three learning-based methods are introduced. Three learning-based methods are compared with our algorithm, and we introduce their iterative formula respectively.

1) Dynamic programming

Dynamic programming solves our problem by breaking it down into a collection of states, and then solves each states just once and stores results. The following figure shows the search paths of dynamic programming.
Dynamic programming is not suit for our problem due to its time complexity is very high.

2) Monte Carlo method

Monte Carlo method repeatedly uses formula 8 (an estimated value function) to progressively approach the real value function.

\[ V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)] \]  

(8)

In the formula 8, \( \alpha \) is a learning rate, a predefined constant. \( R_t \) is a reward after time \( t \). The following picture shows the search paths of Monte Carlo.

3) Temporal Difference method

The simplest Temporal Difference is one-step forward method, namely TD(0). TD(0) simply modifies the values of adjacent states.

\[ V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \]  

(9)

4) Q-learning method

Q-learning estimates the value function based on actions, \( Q(s,a) \), to learn the optimal policy. The estimated value function of Q-learning is:

\[ Q(s,a) \leftarrow Q(s,a) + \alpha [r_t + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s,a)] \]  

(10)

Q-learning method needs to reach a compromise between exploration and exploitation in order to avoid local optimal solutions. The previous literatures use a \( \varepsilon \)-greedy policy to select the action \( (0<\varepsilon<1) \). This probability can ensure that each action has opportunities to be explored. Q-learning method greedily selects (with \( \varepsilon \) probability) the retrieved action that has the maximum \( Q(b,a) \), and explores (with \( 1-\varepsilon \) probability) new actions. It can guarantee the actions with maximum \( Q(b,a) \) have higher priority to be selected.

Q-learning accelerates the speed of convergence, meanwhile it ensures the effective learning. Q-learning is an advanced Temporal Difference method, and the difference between TD(0) and Q-learning is that Q-learning select the maximum actions as the following picture shows.
B. Simulation Results

Each service has four candidate services in the Table 2 and Table 4, which show that Q-learning and TD(0) are more efficient than Monte Carlo method. The reason is that TD(0) only modifies values of adjacent states. Monte Carlo method begins to iterate after a whole learning cycle, whereas TD(0) and Q-learning only need to search one step forward. Q-learning is slightly better than TD(0), because Q-learning uses a maximum action to be the current optimal action. TL is better than these methods. Because, with time passing, TL tries less new services and tries more existing optimization services to find the best composition.

Table 2 Comparison of learning speed w.r.t. services

<table>
<thead>
<tr>
<th>#Service</th>
<th>Monte Carlo (Episodes)</th>
<th>TD(0) (Episodes)</th>
<th>Q-learning (Episodes)</th>
<th>TL (Episodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>80</td>
<td>45</td>
<td>41</td>
<td>30</td>
</tr>
<tr>
<td>300</td>
<td>130</td>
<td>95</td>
<td>90</td>
<td>63</td>
</tr>
<tr>
<td>500</td>
<td>202</td>
<td>155</td>
<td>135</td>
<td>102</td>
</tr>
<tr>
<td>700</td>
<td>310</td>
<td>241</td>
<td>205</td>
<td>156</td>
</tr>
<tr>
<td>900</td>
<td>420</td>
<td>377</td>
<td>312</td>
<td>271</td>
</tr>
</tbody>
</table>

Table 3 Comparison of learning speed w.r.t. candidate services

<table>
<thead>
<tr>
<th>#Candidate Service</th>
<th>Monte Carlo (Episodes)</th>
<th>TD(0) (Episodes)</th>
<th>Q-learning (Episodes)</th>
<th>TL (Episodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>101</td>
<td>94</td>
<td>82</td>
<td>67</td>
</tr>
<tr>
<td>2000</td>
<td>203</td>
<td>180</td>
<td>150</td>
<td>126</td>
</tr>
<tr>
<td>3000</td>
<td>310</td>
<td>270</td>
<td>231</td>
<td>190</td>
</tr>
<tr>
<td>4000</td>
<td>440</td>
<td>382</td>
<td>350</td>
<td>262</td>
</tr>
<tr>
<td>5000</td>
<td>612</td>
<td>525</td>
<td>496</td>
<td>381</td>
</tr>
</tbody>
</table>

In summary, not only the experiments support our ideas, but also the theory shows advancement of Q-learning as the following figure shows.

Dynamic programming and exhaustive search are full backups, but they take too much time to obtain the optimal results. Whereas Temporal difference learning and Monte Carlo method use sample backups to reduce the complexity of problem (Google AlphaGo takes the same strategy). Temporal difference learning (Q-learning) just uses efficient shallow backups, meanwhile it finally keeps convergence.

The experiments show that more parameters make the model more realistic, TL method improves success rate of the service composition and reduces computing time. We randomly generated transition graphs, whose number of services is between 1000 and 4000, and number of candidate services is between 1000 and 10000.
The methods used in the paper [18] and in this paper are compared in two aspects: success rate of service composition and computation cost (computing time). Lit is the result of the paper [18], TL is the result of the method this paper used. Fig. 7 shows that our success rate of service composition in uncertain environments is higher than that obtained by [18].

Fig. 8 shows that, as the number of services increases, computing time of TL method is significantly shorter than that in the [18]. With the number of services dramatically increases, the computing time of the traditional methods dramatically increase.

Comparing our results with the results of literature [15], we observed that the cumulative reward is converged to the optimal policy very fast. From Table 4 and Table 5, we can see that our method is faster than that of literature [15].

### Table 4 Convergence time with respect to services

<table>
<thead>
<tr>
<th># Services</th>
<th>Lit (Episodes)</th>
<th>TL (Episodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>82</td>
<td>62</td>
</tr>
<tr>
<td>1500</td>
<td>103</td>
<td>86</td>
</tr>
<tr>
<td>2000</td>
<td>136</td>
<td>116</td>
</tr>
<tr>
<td>2500</td>
<td>155</td>
<td>125</td>
</tr>
<tr>
<td>3000</td>
<td>203</td>
<td>156</td>
</tr>
<tr>
<td>3500</td>
<td>258</td>
<td>188</td>
</tr>
<tr>
<td>4000</td>
<td>301</td>
<td>230</td>
</tr>
</tbody>
</table>
To solve the web service composition problem in uncertain environments, this paper proposes a time-based learning algorithm, which is a dynamic decision-making method. Our method uses an estimated value function to obtain a composite service. The proposed method is applied to the real scenario. We give details and examples that how to compose services by our method. In addition, we shows relations among user requirements, service invocations and QoS of services. Comparing to the methods based on MDP, the experiments show that our method achieves higher success rate of service composition and shorter computing time.

VII. CONCLUSION

ACKNOWLEDGMENT

This work was supported by Natural Science Foundation of Inner Mongolia Autonomous Region (2015BS0603), Scientific projects of higher school of Inner Mongolia (NJZY009), Programs of Higher-level talents of Inner Mongolia University (215005145143).

REFERENCES


