Continuous-time MAXQ Algorithm for Web Service Composition

Hao Tang
School of Electrical Engineering and Automation, Hefei University of Technology, Tunxi Road No.193, Hefei, Anhui 230009, P.R. China
Email: htang@hfut.edu.cn

Wenjing Liu, Wenjuan Cheng and Lei Zhou
School of Computer and Information, Hefei University of Technology, Tunxi Road No.193, Hefei, Anhui 230009, P.R. China
Email: {winnykris, zhouleizhl, wj_cheng}@163.com

***Abstract*—Web services composition present a technology to compose complex service applications from individual (atomic) services, that is, through web services composition, distributed applications and enterprise business processes can be integrated by individual service components developed independently. In this paper, we concentrate on the optimization problems of dynamic web service composition, and our goal is to find an optimal composite policy. Different from many traditional composite methods that do not scale to large continuous-time processes, we introduce a hierarchical reinforcement learning technique, i.e., a continuous-time unified MAXQ algorithm, to solve large-scale web service composition problems in the context of continuous-time semi-Markov decision process (SMDP) model under either average- or discounted-cost criteria. The proposed algorithm can avoid the "curse of modeling" and the "curse of dimensionality" existing in the optimization process. Finally, we use a travel reservation as an example to illustrate the high effectiveness of the proposed algorithm, and the simulation results show that, it has better optimization performance and faster learning speed than the flat Q-learning.**

***Index Terms*—web service composition, hierarchical reinforcement learning, semi-Markov decision process (SMDP), MAXQ**

I. INTRODUCTION

Web services are considered as self-contained, self-describing, and plat-form-independent applications that can be published (Web Services Description Language, WSDL), discovered (Universal Description Discovery and Integration, UDDI) and invoked (Simple Object Access Protocol, SOAP) over the internet [1, 2]. A single web service usually can not fulfill the requirements of a user, while web service compositions provide a way to combine a set of simple web services into more powerful services or new value-added services that can satisfy the user's needs. So, in modern high-tech world, web service composition has played more and more important role in many domains, typically in e-commerce and enterprise application integration, and has attracted many researchers' attention.

The composition process of web services has been divided into static and dynamic [3, 4]. Static composition is purely manual, whose process model should be built before run time, and each web service is executed one by one so as to achieve the desired requirement. It is a time-consuming task that requires a lot of effort. However, web service requests are always stochastic, and the environment in which web service composition operation is often dynamic. So, it is inappropriate to select and compose web services statically in such cases, and the dynamic composition process is more competitive. The later creates a process model and selects primitive (atomic) services automatically during the run time, so as to make the composite service adaptable in a high dynamic environment.

In this paper, we consider the dynamic web service composition and focus on how to compose services according to a user's request as well as some quality of service (QoS) properties, that is, given a set of possibly candidate web services and an optimization goal, how to achieve this goal by composing these web services into a web process (workflow) [2]. The QoS model for web service usually includes reliability, cost, response time, availability, and so on. So far, a number of methods have been proposed to achieve dynamic web service composition. For example, Gao et al. has discussed a composition method based on Markov decision process (MDP) [5], and introduced two value iteration algorithms for computing the optimal policy: one is backward recursive iteration and the other is forward iteration. But both algorithms are numerical and model-based, which suffers from the curse of modeling, and can not be used in a model-free case.

In [6], a new algorithm is applied for dynamic service composition based on reinforcement learning (RL) and logic of preference. However, in large-scale web service composition problems, current tabular reinforcement learning methods suffer from the "curse of

Partially supported by the Nature Science Foundation of Anhui Province (090412046), the Nature Science Foundation of Education Department of Anhui Province (KJ2008A058), and the Humanities and Social Sciences Project of Ministry of Education (09YJA630029)

doi:10.4304/jsw.7.5.943-950

dimensionality" [7], which is the exponential growth of computational and memory requirements with the number of system state variables. So, the method proposed in [6] may encounter low efficiency in the case of a large number of candidate services, although it can realize composition automatically in a model-free case. Typically, any element of a web process may be either a primitive web service or a composition of primitive web services, which itself is a web process. To that end, the web service composition problem as a task can be recursively decomposed into smaller and smaller subtask until reach the primitive tasks. In other words, realistic web processes may be nested as higher level web processes and lower level web processes, in which a higher level web process maybe composed from lower level web processes [8]. Thus, the large-scale web service composition problems can be solved by hierarchical methods [9–11]. One of the ways is to utilize hierarchical reinforcement learning.

Wang et al. has adapted MAXQ algorithm to dynamic service composition, which is based on discrete-time semi-Markov decision process (DT-SMDP) [11]. But in real-word applications, web service compositions are always related to run time. Specifically, the performance of a composition process is affected by the real time of network transmission and service processing, which is related to one of the QoS properties, i.e., the response time. Therefore, it is more practical to use continuous-time hierarchical reinforcement learning (HRL) algorithms to solve web service composition problems. In this article, we will propose a dynamic web service composition method based on MAXQ algorithm, in the context of continuous-time semi-Markov decision process (CT-SMDP). It is effective in dealing with the "curse of dimensionality" and the "curse of modeling" for practical large-scale service composition.

The rest of this paper is organized as follows: Section 2 presents the framework for dynamic web service composition. Section 3 describes the continuous-time MAXQ algorithm for web service composition. Section 4 demonstrates the experimental results of the approach introduced in section 3. And finally, the last section concludes this paper and discusses the possible future work.

II. Web Service Composition Model

The framework of the dynamic web service composition model is shown in Figure 1 [1].

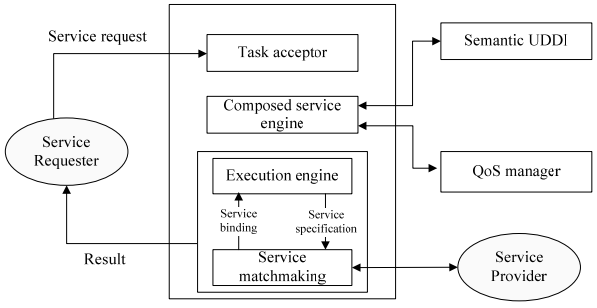

Figure 1 Web Service Composition Model.

The service composition system has two kinds of participants, service provider and service requester. The former proposes web services for use, and the latter consumes information or services offered by the former. The system also contains the following components: task acceptor, composed service engine, execution engine, service matchmaking, etc. Firstly, the service providers advertise their atomic services at a global market place. Once a service requester submits his requests and the task acceptor accepts the information, the composed service engine will try to solve the requirement by composing the atomic services advertised by the service providers. Then the execution engine will accept the corresponding flow chart, and send the service specification to the service matchmaking, which will find the most appropriate atomic web services and return the information to the execution engine. Finally, the execution engine invokes and executes each atomic service, and the result will be then sent back to the service requester.

The dynamic web service composition problem can be modeled as a SMDP [12–15], which is a more general model than MDP. When a composition process evolves to a task node, the composed service engine should decide to select a concrete web service, and the moment of making decision is called decision epoch. Here, we use t_n to denote the nth decision epoch with $t_0 = 0$. If there are l task nodes to be bound in a web service composition problem, the system state s is then defined as a conjunction of status of each task node, i.e., $s =< s_1,...,s_k,...,s_l >$, where s_k corresponds to the kth task node of this service composition. $s_k = 1$ represents that this node is active and has been bound to a concrete web service, while $s_k = 0$ means that this node is not active. Let Φ be the system state space, i.e., $s \in \Phi$. At time t_n and state s_n, the action a_n is selected from the set of all possible candidate web services $A(s_n)$, i.e., $a_n \in A(s_n)$, and write $A = \bigcup A(s_n)$. Then, a stationary policy π represents a mapping from states to actions, i.e., $\pi : \Phi \to A$. Here we use the reliability of a relative service as the transition probability $p(s_{n+1} \mid s_n, a_n)$ [9], that is, under a concrete action a_n, the system transits from current state s_n to next state s_{n+1} with probability $p(s_{n+1} \mid s_n, a_n)$ or still stays at state s_n with probability $1 - p(s_{n+1} \mid s_n, a_n)$. Let $\tau_{ss'}$ be the interval time between t_n and t_{n+1}, and it can also be called the sojourn-time of state s_n under action a_n, which follows a random distribution. Then, the web service composition problem can be modeled as a SMDP.

Suppose that the expected cost the system pays every unit time at state s_n under action a_n and before transiting to next state s_{n+1} is denoted by $f(s_n, a_n, s_{n+1})$. Then, we take the following infinite-horizon expected cost criteria [15].

$$\eta_{\alpha}^{\pi}(s)=E\left[\sum_{n=0}^{\infty}\int_{t_n}^{t_{n+1}}\alpha e^{-\alpha t}f(s_n,a_n,s_{n+1})dt\,|\,s_0=s\right],\ \forall s\in\Phi. \quad (1)$$

Here, α denotes a discount factor, and when $\alpha>0$, $\eta_{\alpha}^{\pi}(s)$ represents the long-run expected total discounted cost of states under policy π. As a special case, if $\alpha\to 0$, the limitation $\eta_{0}^{\pi}(s)$ represents the following infinite-horizon expected average cost

$$\eta^{\pi}=\lim_{N\to\infty}\frac{1}{t_N}E\left[\sum_{n=0}^{N-1}\int_{t_n}^{t_{n+1}}f(s_n,a_n,s_{n+1})dt\right]. \quad (2)$$

III. Continuous-time MAXQ Algorithm for Service Composition

The MAXQ algorithm is a well-known approach in hierarchical reinforcement learning, which provides a hierarchical decomposition of the given reinforcement learning problem into a set of sub-problems. Ghavamzadeh and Mahadevan has proposed a continuous-time MAXQ algorithm in the context of SMDP [17], which is an extension of the MAXQ method for discrete-time hierarchical reinforcement learning [18,19]. In that work, continuous-time discounted reward MAXQ algorithm and continuous-time average reward MAXQ algorithm were introduced respectively. Based on those work and by using the concept of performance potential, we will propose a unified optimization framework for both cost criteria in this section.

To construct MAXQ decomposition for the web service composition problem, we should first identify a set of individual subtasks that we believe important for solving the overall task. More formally, the MAXQ method decomposes the target task M into a set of subtasks $\{M_0,M_1,...,M_{m-1}\}$ and decomposes a hierarchical policy π into a set of policy $\{\pi_0,\pi_1,...,\pi_{m-1}\}$ with π_i corresponding to subtask M_i. Each subtask M_i is a triple $<T_i,A_i,R_i>$. The termination predicate partitions the state space Φ into a set of active states S_i and a set of terminal states T_i. The policy for subtask M_i can only be executed if the current state $s\in S_i$. A_i is a set of actions that can be performed to achieve subtask M_i, with elements being either primitive actions or other subtasks. R_i is the pseudo cost function, which specifies a pseudo-cost for each transition to a terminal state. In this case, let us define four tasks as follow:

- Root. This is the whole web service composition task.
- Input. In this subtask, the goal is to get the request information so as to invoke a concrete service.
- Output. In this subtask, the goal is to obtain the service output data.
- Comp. In this subtask, the goal is to select an appropriate composite model during the composition process. In other words, it is to move the service process from its current state to target state.

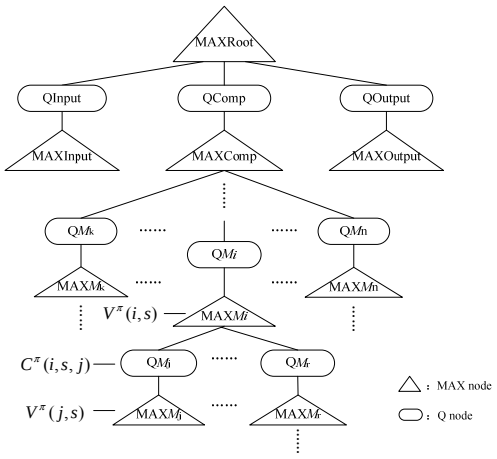

Figure 2 MAXQ graph for the web service composition problem.

By decomposing we can obtain a MAXQ graph for web service composition problem as shown in Figure 2 according to [19]. There are two kinds of nodes: MAX nodes (triangles) and Q nodes (rectangles). A MAX node with no children denotes a primitive action and the one with children represents a composite subtask. The immediate children of each MAX node are Q nodes corresponding to the actions that are available for each subtask. Specifically, each MAX node M_i can be viewed as computing the value of $V^{\pi}(i,s)$ for its subtask, which is the expected cumulative cost of executing π_i and the policies of all descendants of M_i starting at state s until M_i terminates. For a primitive MAX node, this information is stored in the node, while for a composite MAX node, this information is obtained by the Q node selected by π_i. Suppose that the policy π_i of M_i chooses subtask $\pi_i(s)=j$ at state s. Then, each Q node with parent task M_i, state s, and child task M_j can be viewed as computing the value of $Q^{\pi}(i,s,\pi_i(s))$ that is equal to the projected value function $V^{\pi}(j,s)$ of its child task M_j plus its completion function $C^{\pi}(i,s,j)$. Here, $C^{\pi}(i,s,j)$ is stored in the Q node, and represents the expected discounted cumulative cost of completing subtask M_i after invoking the subroutine for subtask M_j at state s.

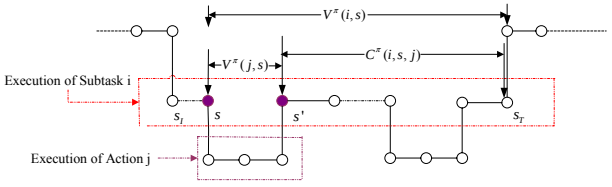

Figure 3 Note how the caption is centered in the column.

The decomposition for the $V^{\pi}(i,s)$ is also shown by Figure 3. Each circle is a state of the composition process, and suppose subtask M_i is initiated at state s_I and terminated at state s_T. If i is a primitive action, then $s=s_I$, $s'=s_T$. The interval time $\tau_{ss'}$ between state s and

its next state s' is the sojourn-time of state s, which is assumed to be exponential distribution during our experiments in the next section. If i is a composite action, $\tau_{ss'}$ is the cumulative time that can be separated into several interval times, and is no longer exponential distribution. Then, the value function $V^{\pi}(i,s)$ of state s for M_i in the MAXQ algorithm is broken into two parts: the value of the subtask M_j that is independent of the parent task M_i, and the value for completing M_i after executing subtask M_j that of course depends on the parent task M_i. So, we have

$$V^{\pi}(i,s)=\begin{cases} f'(s,i,s') & \text{if } i \text{ is primitive} \\ Q^{\pi}(i,s,\pi_i(s)) & \text{if } i \text{ is composite} \end{cases}. \quad (3)$$

where

$$f'(s,i,s')=k_1(s,i)+\int_0^{\tau_{ss'}} k_2(s,i,s')\times e^{-\alpha t}dt - T_{\alpha}(\tau_{ss'})\times\tilde{\eta}. \quad (4)$$

$$Q^{\pi}(i,s,j)=V^{\pi}(j,s)+C^{\pi}(i,s,j). \quad (5)$$

Here, $k_1(s,i)$ represents the immediate cost of executing primitive action i at state s, which represents the user's request in web service composition problem, like the price acceptable, the distance between the hotel and the airport, and so on; $k_2(s,i,s')$ represents the time-related cost that the system pays every unit time, such as response time. In addition, $\tilde{\eta}$ represents the estimate of average cost, satisfying

$$\tilde{\eta}:=\frac{S_f}{S_w}. \quad (6)$$

with s_f and s_w being learned, respectively, as follows

$$S_f := S_f+\beta_n(f'_{\alpha=0}(s,i,s')-S_f). \quad (7)$$

$$S_w := S_w+\beta_n(\tau_{ss'}-S_w). \quad (8)$$

Here, β_n is a stepsize, and $f'_{\alpha=0}(s,i,s')$ is calculated by (4) with $\alpha=0$, which denotes accumulated cost from s to s' under primitive action i.

The projected value function for the root is then decomposed into the ones for the individual subtasks and the individual completion functions recursively by equations (3) to (5). Furthermore, $V^{\pi}(i,s)$ and $C^{\pi}(i,s,j)$ are updated, respectively, as follows

$$V^{\pi}(i,s):=(1-\gamma)V^{\pi}(i,s)+\gamma\times f'(s,i,s'). \quad (9)$$

$$C^{\pi}(i,s,j):=(1-\gamma)C^{\pi}(i,s,j)+\gamma(e^{-dT}(C^{\pi}(i,s',j')+V^{\pi}(j',s'))-T_{\alpha}(T)\times\tilde{\eta}). \quad (10)$$

$$j'=\arg\min_{a'\in A_i(s')}(V^{\pi}(a',s')+C^{\pi}(i,s',a')). \quad (11)$$

Here, $T_{\alpha}(\tau)=\int_0^{\tau}e^{-\alpha t}dt=\frac{1-e^{-\alpha\tau}}{\alpha}$ for any discount factor $\alpha>0$ and time $\tau>0$, and let $T_0(\tau)=\lim_{\alpha\to 0}T_{\alpha}(\tau)=\tau$. In addition, γ is a stepsize, and T is the current total cumulative time for executing subtask M_i.

Due to the introduction of the term $T_{\alpha}(\bullet)\times\tilde{\eta}$ in equation (4) and (10) according to the idea of performance potential for SMDP [15, 16], the unified MAXQ algorithm can then be established for both discounted and average criteria, which is the difference between our algorithm and other MAXQ algorithms such as proposed in [14]. The continuous-time unified MAXQ algorithm for web service composition is depicted in Table I, and the flowchart of this algorithm is shown in Figure 4.

TABLE I.
MAXQ ALGORITHM FOR WEB SERVICE COMPOSITION

```
1.  function CTU_MAXQ(MaxNode i, State s)
2.  initialize Stack = {s, i, τ_ss', f', f'_α=0}
3.  if i is primitive MAXNode, then
4.    execute action i in state s, receive sojourn-time τ_ss', observe
      state s'; calculate the accumulated cost f'(s,i,s') and
      f'_α=0(s,i,s') according to (4)
5.    push the sample records into the top of the Stack{}; calculate
      η̃ according to (6)-(8) and compute the value of V^π(i,s)
      according to (3)-(5)
6.  else
7.    while s ∉ T_i do
8.      choose action j according to the current ε – greedy policy
        π_i
9.      let ChildStack = CTU_MAXQ(j, s) while executing action j
10.     observe result state s'
11.     let j' = arg min_{a'∈A_i(s')}(C^π(i,s',a') + V^π(a',s'))
12.     T=0
13.     for the element of the ChildStack from the beginning do
14.       T := T + τ_ss', calculate η̃ and compute C^π(i,s',j') via (10)
15.     end for
16.     append ChildStack onto the top of Stack
17.     let s = s'
18.   end while
19. end if
20. return Stack
21. end CTU_MAXQ

22. initialize V^π(i,s) and C^π(i,s,j) arbitrarily
23. CTU_MAXQ (0, s_0)
```

IV. EXPERIMENTS

In web service composition problems, a complete web service composition task is staring from service requester submitting requests to the system returning the results. After finishing a web service composition task, it will deal with another web service composition task immediately. In this section, we will demonstrate the high

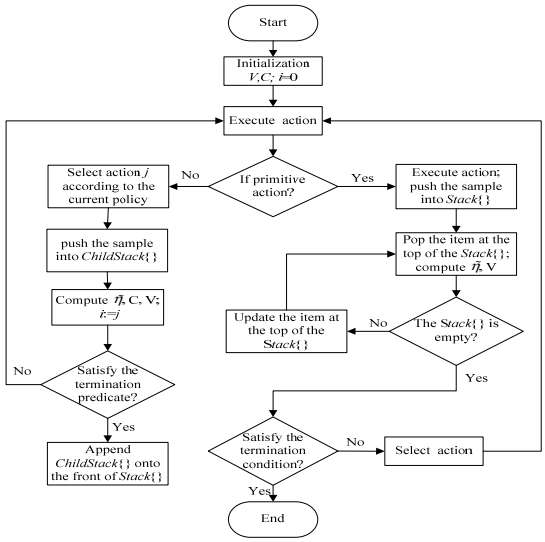

Figure 4 Flow chart for the continues-time MAXQ algorithm.

effectiveness of the continues-time unified MAXQ method by using a travel reservation as a simulation example.

A. Simulation model

Let us assume that a customer wants to travel to someplace. So he/she should first tell the travel agent who notes the customer's requests and generates a corresponding *trip request* document that may contain several needed *plane/train/bus tickets*, *hotels*, *car rental*, *excursions*, etc. The travel agent performs all bookings and when he is done, he puts the *trip request* either into the *canceled requests* or the *completed requests* data base. A completed document is sent to the customer as an answer to his request. If the booking fails, the customer is contacted again and the whole process re-iterates. Service providers (airlines, bus companies, hotel chains, etc) are providing web services for selection, and credit card companies are also providing services to guarantee payments made by consumers.

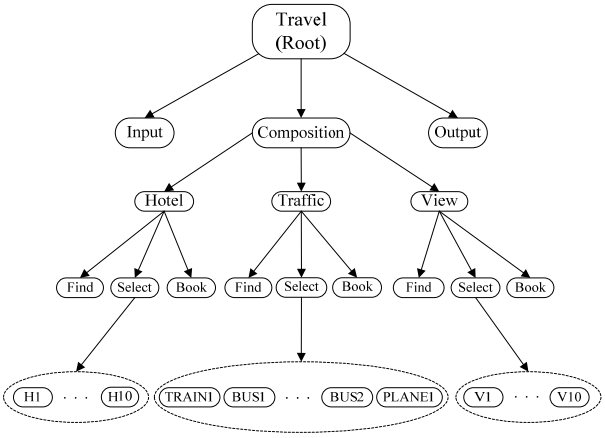

Figure 5 travel reservation model.

As Figure 5 shows, the travel reservation problem is decomposed into 4 levels. The highest level is the task of *input*, *composition* and *output*. Further more, the *composition* task is decomposed into *hotel*, *traffic* and *viewpoint*, which constitute the second level. Then they all decompose into three subtasks: *find*, *select*, *book* respectively. At last level, there are all primitive actions which are the candidate web services that can be bound to the corresponding parent task node. The goal is to find an optimal composite policy according to user's request.

B. Experimental Results

As a case study, we suppose there are three web service classes, such as hotel, traffic and viewpoint, and the number of candidate services for each web service class is 10, as shown in figure 5. For comparison, we firstly used flat Q-learning to simulate this problem. In Q-learning, $\varepsilon - greedy$ actions are necessary for exploration, especially at the beginning of the optimization. So we use $\varepsilon = 0.3$, learning step $\gamma = 1/(8 \times N(s,i)^{0.2})$. Here $N(s,i)$ is the number of state-action pairs (s,i) that has been visited. $k_1(s,i)$ is different for every state-action pairs (s,i) , and $k_2(s,i,s') = 0.8$.

First, we consider the average case, i.e., $\alpha = 0$. Two optimization plots are provided respectively in Figure 6, where each *y*-axis denotes the average cost of each algorithm in 1000 episodes. In this problem, a simulation episode is staring from any state to the termination state. We observed that, comparing with the Q-learning, the performance of the MAXQ method is more efficient. The reason is that the MAXQ method accelerates the learning speed and also improved the optimizing precision via

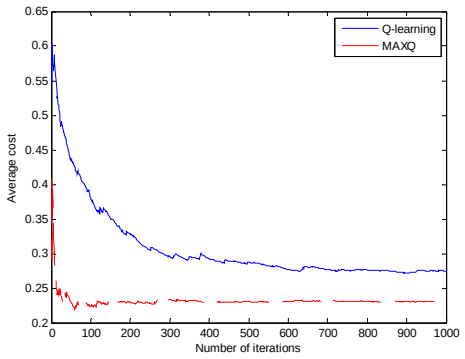

Figure 6 Real time Average cost of the Q-learning and MAXQ algorithm.

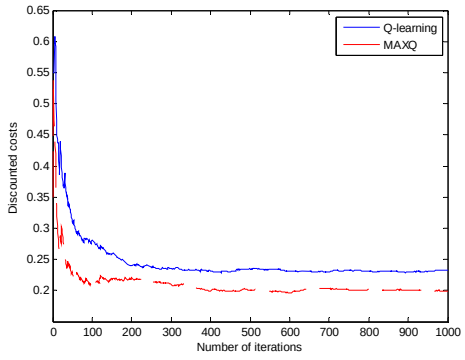

Figure 7 Real time Discounted costs of Q-learning and MAXQ algorithm ($\alpha = 0.01$).

hierarchy and action abstractions.

Figure 7 shows the results of Q-learning and MAXQ algorithm for discounted cases with the discount factor $\alpha = 0.01$. As we expected, the graph shows that the MAXQ algorithm can yield a good result faster than Q-learning. And due to the utilization of discount factor α, the curve for discounted case is flatter than it for average case. At the beginning, the curve of the MAXQ algorithm shows obvious fluctuation for either average case or discounted case. The reason is that each subtask is very different from each other, each of them has to be learned separately at first, then be reused (shared).

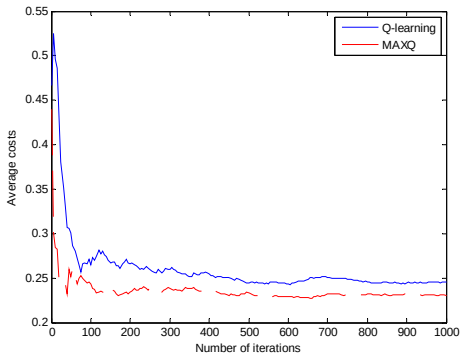

Figure 8 Policy evaluation of Q-learning and MAXQ algorithm for average case.

In addition, we also executed Monte-Carlo policy evaluation for each greedy policy yielded by these two algorithms at every 5 steps, respectively. Specifically, in order to evaluate the relevant performance values, we receive greedy policies at intervals of 5 steps, and then simulated the system by running 1000 steps according to each policy, respectively. Figure 8 and Figure 9 show the results which correspond to Figure 6 and Figure 7, respectively. From those two figures, we can see that the evaluated value of Q-learning method is higher than the MAXQ algorithm both in average and discounted case. From many independent and repeated runs, we conclude that such a unified MAXQ algorithm usually outperforms the flat Q-learning method.

On the other hand, we extended the number of tasks to test the other performance values, such as the success rate and the computation cost. The success execution rate $Succ(s_i)$ of web service s_i is a measure related to hardware and/or software configuration of web services and the network connections between the service requesters and providers. $Succ(s_i)$ is computed from data of past invocations using the follow expression

$$Succ(s_i) = N_c(s_i)/K. \tag{12}$$

where $N_c(s_i)$ is the number of times that service s_i has been successfully completed, and K is the total number of invocations.

So, we define the success rate $Succ(\pi)$ of a current policy π is calculated as follow

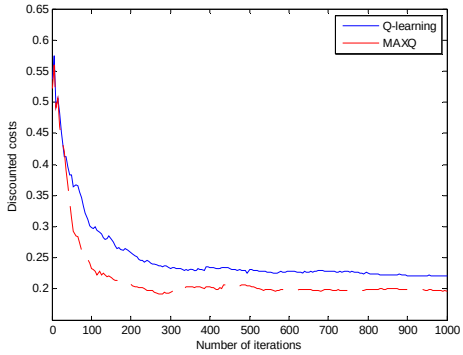

Figure 9 Policy evaluation of Q-learning and MAXQ algorithm for discounted cases ($\alpha = 0.01$).

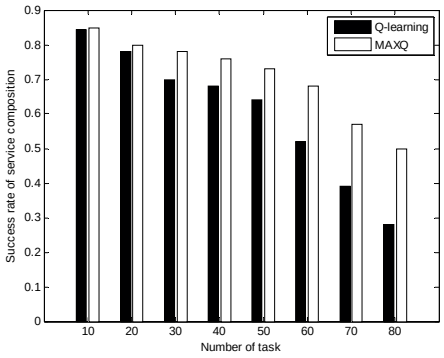

Figure 10 The web service composition success rates of both algorithms with different number of tasks.

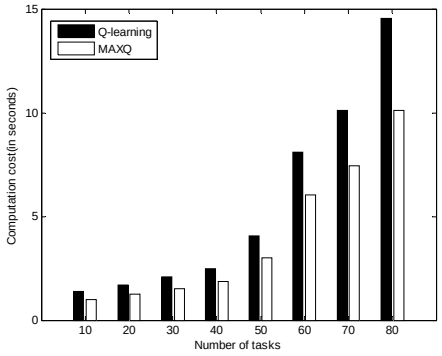

Figure 11 The computation cost of both algorithms.

$$Succ(\pi) = \prod_{i=1}^{l} Succ(s_i). \tag{13}$$

Figure 10 shows the success rates of obtained policy for Q-learning and MAXQ algorithm with different number of tasks, respectively. We can see that the success rates reduce as the number of tasks increased. And when the number of the tasks is 10, the success rates of these two algorithms are almost the same. But as the number of the tasks increased, the success rates of Q-learning reduce faster than that of MAXQ algorithm. In addition, the computation costs of both algorithms are shown by Figure 11. Obviously, when the number of tasks becomes large, the computation cost of Q-learning grows faster

than that of MAXQ algorithm. These two figures fully illustrate that the MAXQ algorithm is more effective in large-scale service composition problems.

The differences for the two algorithms are also listed in Table II. The average cost obtained by Q-learning is 0.2746, while the average cost obtained by MAXQ algorithm is 0.2311, which is almost 15.8% less. The reason is that policies learned in subproblems can be reused for multiple parent tasks, so the optimization performance has been improved. As we known, the MAXQ algorithm is more complex than Q-learning, so the time of completing a certain number of steps for MAXQ algorithm is more than for Q-learning. On the other hand, the value functions learned in subproblems can be shared, so when the subproblem is reused in a new task, learning of the overall value function for the new task is accelerated. When the curves tend to smooth, we consider that the algorithm get a good result. So we can see that the time spent for getting good results by MAXQ algorithm is about 26.4% less than that by Q-learning, as shown in Table II.

Furthermore, because of state abstraction, the overall value function can be represented compactly as the sum of separate terms that each depends on only a subset of the state variables. This more compact representation of the value function will require less data to learn, so as to improve the learning speed. For comparing with Q-learning, we define the reduction rate of memory units q for MAXQ algorithm as $q = ((C - B)/C) \times 100\%$. Here B is the number of memory units that MAXQ algorithm required, and C is the number of memory units that Q-learning required. In the model we proposed, the value of q is $(1-(8+8\times N+N\times W)/(3\times N\times W))\times 100\%$ with N being the number of tasks and W being the number of candidate services. According to this formula, we can obtain the limit of q is 66.67% when W tend to infinity and N is fixed, or the limit is $(1-(8+W)/(3\times W))\times 100\%$ when N tend to infinity and W is fixed. And the relation between q and N, W is shown in Table III.

From Table III, we can see that by applying state abstraction, the MAXQ required much less memory units than Q-learning. Obviously, q increases as the number of tasks and candidate services increased. It fully demonstrates that the MAXQ algorithm is more applicable to large-scale web services composition problems.

TABLE III.
COMPARISON OF DIFFERENT ALGORITHMS

Algorithms	**Q-learning**	**MAXQ**
Average cost	0.2746	0.2311
Time for 1000 steps	1.6688 s	1.7661 s
Time for getting good results	1.3678 s	1.0067 s

TABLE II.
REDUCTION RATES OF MEMORY UNITS FOR MAXQ ALGORITHM COMPARING WITH Q-LEARNING (%).

	N=10	N=20	N=30	N=40	N=50	N=60	N=70	N=80
W=10	37.33	38.67	39.11	39.33	39.47	39.56	39.61	39.67
W=20	52	52.67	52.89	53	53.07	53.11	53.14	53.16
W=30	56.89	57.33	57.48	57.56	57.60	57.63	57.65	57.67
W=40	59.3	59.67	59.78	59.83	59.87	59.89	59.90	59.91

V. CONCLUSIONS

The web service composition problems, under either average- or discounted-cost criteria, are solved effectively by using continues-time unified MAXQ algorithm. Compared with Q-learning, the proposed algorithm tends to be more suitable for solving the "curse of dimensionality" in large-scale web service composition problems. The simulation results also demonstrated that the MAXQ algorithm has the advantages of high effectiveness and high learning speed.

In our work, our optimization goal is concerned about the price requirements of user. But the algorithm we proposed in this paper can also apply to other elements of QoS issues like reliability. In addition, as part of our ongoing work, we will extend our algorithm to support multi-agent web service composition problems.

REFERENCES

[1] J. Rao, X. Su, "A Survey of Automated Web Service Composition Methods", *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*, 2004, pp. 43-54.

[2] H. Zhao, and P. Doshi, "Composing Nested Web Processes Using Hierarchical Semi-Markov Decision Process", *AAAI Workshop on AI-Driven Technologies for services-Oriented Computing*, pp. 75-84, 2006.

[3] F. H. Khan, S. Bashir, M. Y. Javed, A. Khan, M. S. H. Khiyal, "QoS Based Dynamic Web Services Composition & Execution", *Pages IEEE format, International Journal of Computer Science and Information Security, IJCSIS,* vol. 7 no. 2, pp. 147-152, USA, February 2010.

[4] F. Mustafa, T. L McCluskey., "Dynamic Web Service Composition", *International Conference on Computer Engineering and Technology*, 2009.

[5] A. Gao, D. Yang, S. Tang, and M. Zhang, "Web Service Composition Using Markov Decision Processes", *International Conference on Web-Age Information Management*, 2005, pp. 308-319.

[6] H. B. Wang, P. P. Tang, P. Hung, "RLPLA: A reinforcement learning Algorithm of Web service Composition with Preference Consideration", *IEEE Congress on Services Part II*, pp. 163-170, 2008.

[7] G. B. Andrew, S. Mahadevan, "Recent Advances in Hierarchical Reinforcement Learning", *Discrete Event Dynamic Systems*, vol.13 no.1-2, pp.41-77, 2003.

[8] H. Zhao, P. Doshi, "A hierarchical framework for composing nested web processes", *International Conference on Service Oriented Computing*, 2006.

[9] E. Sirin, B. Parsia, D. Wu, J. Hendler, D. Nau, "HTN planning for Web Service composition using SHOP2",

Web Semantics: Science, Services and Agents on the World Wide Web, pp. 377-396, 2004.
[10] K. Chen, J. Y. Xu, S. Reiff-Marganiec, "Markov-HTN Planning Approach to Enhance Flexibility of Automatic Web Service Composition", *IEEE International Conference on Web Services*, 2009, pp. 9-16.
[11] H. B. Wang, X. H. Guo, "Preference-Aware Web Service Composition Using Hierarchical Reinforcement Learning", *Web Intelligence/IAT Workshops*, pp. 315-318, 2009.
[12] S. J. Bradtke, and M. O. Duff, "Reinforcement learning methods for continuous-time Markov decision problems", *in Advances in Neural Information Processing Systems 7*, Cambridge, MA: MIT Press, pp. 393-400, 1995.
[13] S. Mahadevan, N. Marchalleck, T. Das, A. Gosavi, "Self-improving factory simulation using continuous-time average-reward reinforcement learning", *Proceedings of the 14th International Conference on Machine Learning*, 1997, pp. 202-210.
[14] R. Parr, "Hierarchical control and learning for Markov decision processes", PhD Thesis, University of California at Berkeley, 1998.
[15] X. R. Cao, "Semi-Markov decision problems and performance sensitivity analysis", *IEEE Transactions on Automatic Control*, 48(5): 758-769, 2003.
[16] H. Tang, T. Arai, "Look-ahead control of conveyor-serviced production station by using potential-based online policy iteration", *International Journal of Control*, 82(10): 1916-1928, 2009.
[17] M. Ghavamzadeh, S. Mahadevan, "Continuous-Time Hierarchical Reinforcement Learning", *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001, pp. 186-193.
[18] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition", *Journal of Artificial Intelligence Research*, 13:227-303, 2000.
[19] T. G. Dietterich, "The MAXQ Method for Hierarchical Reinforcement Learning", *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998, pp. 118-126.

Hao Tang was born in 1972. He received his B.E. degree from Anhui Institute of Technology, P.R. China, in 1995, M.E. degree from Institute of Plasma Physics, Chinese Academy of Sciences, in 1998, and Ph.D. degree from University of Science and Technology of China (USTC) in 2002. He has been a postdoctoral researcher at Advanced Robotics with Artificial Intelligence Lab in The University of Tokyo, Japan, from 2005 to 2007. He is currently a professor of Hefei University of Technology, P.R. China. His research interests include DEDSs, reinforcement learning, the methodology of neuro-dynamic programming, and intelligent optimization.

Wenjing Liu was born in 1985. She received her B.E. degree from Anhui Agricultural University, P.R. China, in 2007, M.E. degree from Hefei University of Technology, P.R. China, in 2011. Her research interests include DEDSs, reinforcement learning.

Lei Zhou was born in 1981. He is currently a lecturer and Ph.D. candidate of Hefei University of Technology, P.R. China. His research interests include DEDSs, reinforcement learning.