Received December 13, 2020, accepted January 21, 2021, date of publication January 27, 2021, date of current version February 5, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3054852

Automatic Hierarchical Reinforcement Learning for Reusing Service Process Fragments

RONG YANG[1], BING LI[2], (Member, IEEE), AND ZHENGLI LIU[2]
[1]College of Computer Science and Technology, Hubei University of Science and Technology, Xianning 437100, China
[2]School of Computer Science, Wuhan University, Wuhan 430072, China

Corresponding author: Rong Yang (yangrong@hbust.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1400602, in part by the Natural Science Foundation of Hubei Province under Grant 2019CFC888, and in part by the Research Development Fund of Hubei University of Science and Technology under Grant 2019-21GP06.

ABSTRACT Prevailing research trend is to use Web services for data publishing and sharing among organizations, but existing works often fall short of service reuse. Developing efficient solutions to achieve composite services has drawn significant attention in services computing. Services and service process fragments reuse is critical to improve the efficiency of software development and economize on human and material resources, meanwhile Reinforcement Learning (RL) is one commonly used approach in services computing. However, in service composition and service process fragments (SPFs) reusing scenarios, traditional RL methods cannot guarantee good efficiency for large-scale service processes construction problems. In this paper, we present a novel SPF reusing framework that combines automatic Hierarchical Reinforcement Learning (HRL) and extended Cocke-Kasami-Younger (CKY) algorithm. This framework has the ability to reuse any granularity of SPFs. We firstly get action models and trajectories by means of analysis on historical service process fragments. Furthermore, the ‘‘Causal Analysis’’ identifies the causal relationships among the actions in a trajectory, i.e. returning a causally annotated trajectory (CAT). Then, we utilize the SPF-Hierarchy algorithm to discover a coherent task hierarchy for each service process fragment. Finally, we map the hierarchy obtained from the previous stage to the HRL-CKY algorithm, which can fulfill the reuse and retrieval of any granularity of SPFs. The effectiveness and robustness of our approach are evaluated through a set of experiments.

INDEX TERMS Service reuse, service process fragment, hierarchical reinforcement learning.

I. INTRODUCTION

Service-oriented architectures (SOA) deal with the growing need for distributed applications, not only support the integration and collaboration among departments of the same organization but also enable industrial partnerships across distinct organizations. Since the capability provided by a single Web service is limited, Web services usually need to be composed as workflows (i.e., service processes) to achieve more complex tasks [1]–[3] or to mash up data from different data resources by using business process description languages.

How can we effectively structure a service process which meets user requirements? That is, for a user request, we do service composition by some means as soon as possible. There are a lot of themes about reliably constituting a composite service which have been researched and developed in the services computing field, such as Reinforcement Learning (RL) and service process fragments (SPFs) reuse.

Reinforcement Learning is an active area of machine learning approach, which is widely used in the fields of operations research, decision theory, and control engineering [4]. RL has been approved to be effective to construct composite service processes [5], [6]. Most RL algorithms take advantage of standard methods of stochastic dynamic programming (DP) so that they can solve problems with large state spaces. By focusing computational effort along behavioral trajectories and by utilizing function approximation methods for accumulating value function information, RL algorithms can do better than standard methods on those problems which have significant challenges [7]. However, traditional RL algorithms are not immune to the so called ‘‘curse of dimensionality’’. More specifically, the number of parameters to be

The associate editor coordinating the review of this manuscript and approving it for publication was Kaitai Liang.

learned grows exponentially with the increase of the size of states'.

To overcome this deficiency, researchers tried principled ways of exploiting temporal abstraction, where decisions are not required at each step, but rather invoke the execution of temporally-extended activities which have their own policies. All these work leads naturally to Hierarchical Reinforcement Learning (HRL), and HRL can solve the dimensional disaster problem of traditional RL [8]. Although there are a lot of successful studies about HRL, Option [9], HAM [10] (Hierarchy of Abstract Machine), and MAXQ [11] (MAXQ Value Function Decomposition) are three typical approaches.

Service process fragments reuse is also a well-explored subject in service composition. Reusing SPFs can not only decrease the composition time, but also improve the reliability of the composition process. Therefore, reusing SPFs can dramatically contribute to service composition. VGI [36] (Variable Granularity Index) can realize the unified index on both atomic and composite services and maximize the reuse of them. SCKY [13] (a service process fragment reusing method motivated by Cocke-Kasami-Younger algorithm) can reuse any granularity of service process fragment. However, the performance of VGI and SCKY both needs to be improved. Schumm *et al.* [47] proposed a method for Web service composition through shared process fragment libraries. A complete and integrated framework was proposed in [48], which can reuse and share service compositions over Web using stimulation workflows. The limitation of the work in [47] and [48] is that they paid no attention to how to search or locate the process fragment at all, but rather that they mainly focus on the complex control logic in the process. In addition, when the scale of service processes is very large, the efficiency is decreased significantly.

If we combine Hierarchical Reinforcement Learning with service process fragments reuse, what will the result be? Can we build composite services efficiently and reliably? To this end, in this work, we pay more attention on efficiently discovering and reusing useful service process fragments. Furthermore, we combine the idea of Cocke-Kasami-Younger (CKY, alternatively called CYK) [12] algorithm with hierarchical reinforcement learning, which is as a starting point for our research. The applicability of existing HRL methods requires a task graph, which can be generated by decomposing a service process plan into a task hierarchy. For each Web service of a service process fragment, we standardize its values of different QoS (Quality of Service) attributes and map them into the interval [0,1]. After that we can aggregate the various attributes into a single reward value. Then, for a user SPF query, our HRL method can receive a certain amount of reward by executing a specified workflow, which is equivalent to the cumulative reward of all the executed services. Finally, the service process fragment which has an optimal reward value is returned to the user.

A composite service typically runs in a dynamic environment, and consequently it is a central concern that a composition solution must be adaptable. Generally speaking, service composition should be able to adapt to those dynamic and uncertain factors to reach some degree of self-adaptivity. Service composition can be self-adaptable by means of traditional RL. However, when facing large-scale service composition problems, traditional RL methods cannot guarantee good efficiency. In this paper, we apply HRL to service process fragments reuse for doing service composition, where we can overcome performance deficiency. To the best of our knowledge, there has been no publication that combines Hierarchical Reinforcement Learning with service process fragments reuse.

A. MOTIVATION

For the sake of clarity, we first provide a one-day tour service process in Fig.1, in which there are 5 atomic services which can actually run. As shown in Fig. 1, S_1 is a weather service, and S_2 is an attraction searching service. S_3 and S_4 are bike renting and car renting service, respectively, but S_5 is a travel agency service. If the input conditions are met, each service produces corresponding outputs. For example, if date and city information are given to S_1, it can bring back exact weather conditions. Moreover, according to the input and output, the dependency relations among services can be easily constructed.

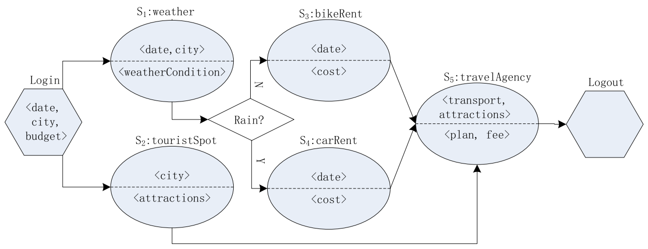


FIGURE 1. **A service process example.**

Generally speaking, if a user proposes a request with the preferences of date, city and budget, S_1 can do weather query by accepting two inputs: date and city. According to whether it is raining or not, S_3 or S_4 is chosen for the whole service process. Finally, S_5 takes attractions and transport as its input parameters, which come from S_2, S_3 or S_4 respectively, and then a detailed plan with fee will be returned to the user. S_4 is naturally chosen for the composite service process if it hits for user request and matches with the weather condition. However, if S_4 is out of order, the traditional method is to manually analyze and reconstruct the whole composite process despite some SPFs of Fig.1 can be reused by replacing S_4 with the other car renting service.

Actually, we can reuse SPF in any granularity [13]. As shown in Fig.1, we can reuse each atomic service (e.g. S_1, S_2), and the fragments (e.g. $S_1 \rightarrow S_3$, $S_1 \rightarrow S_4$, $S_1 \rightarrow S_3 \rightarrow S_5$). There are also some new challenges in the SPF reuse scenario:

- ♦ As a composite service process typically runs in a dynamic environment, how to improve the adaptability of the SPF reusing?

- If the service process repository is extremely large, how to accelerate learning and querying speed?

B. OUR APPROACH

Fig.2 indicates the high-level schema of our approach. After the system runs over a period of time, there may be a large number of service process fragments (SPFs) in the service process repository. Through the analysis on historical fragments, we can get action models and trajectories (action models and trajectories are detailed in Section 4). Then, the "Causal Analysis" module makes causal analysis to identify the causal relationships among the actions in a trajectory, i.e. returning a causally annotated trajectory (CAT). The CAT, along with the action models, is then provided to the SPF-Hierarchy algorithm, which discovers a coherent task hierarchy. It is worth noting that the task hierarchy can minimizes the number of intertask causal links, such that avoiding the "curse of dimensionality". Finally, through using the HRL-CKY algorithm, we can achieve a fast and accurate SPF query.

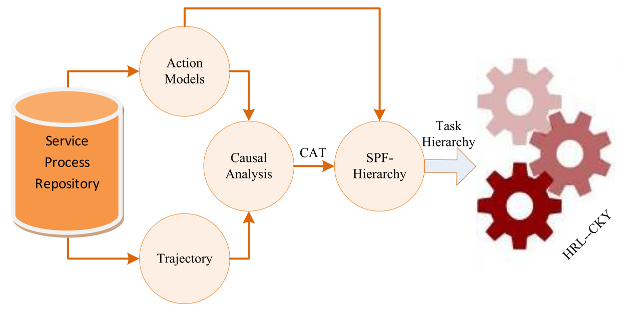


FIGURE 2. The architecture of our system.

We present an illustration for HRL-CKY in Fig.3. Given a target service process "$s_1s_2s_3s_4s_5$", which consists of five atomic services, by the idea of CKY, from bottom to up and left to right, we can map the above task hierarchy to the corresponding cells. That is to say, Cell[i, j] denotes a part of the task hierarchy, which is corresponding to the SPF that starts from s_{i+1}, ends with s_j. The leaf nodes of the task hierarchy (i.e. the five atomic services, s_1, s_2, s_3, s_4, and s_5) relate to Cell[0,1], Cell[1,2], Cell[2,3], Cell[3,4] and Cell[4,5], respectively. For those nodes of the task hierarchy, which are on the layer above the leaf nodes, they relate to Cell[0,2], Cell[1,3], Cell[2,4], and Cell[3,5], respectively. Let the root of a task hierarchy is the first layer. Then, in the same way, Cell[0,3], Cell[1,4] and Cell[2,5] relate to the nodes of the third layer. Cell[0,4] and Cell[1,5] represent the nodes of the second layer. At last, Cell[0,5] corresponds to the task hierarchy's root. As shown in Fig.3, if there is an element in Cell[1,4], which relates to one node in the task hierarchy. Using the depth or breadth-first traversal from this node, we gather all the leaf nodes during the traversal, and we can get "$s_2s_3s_4$". The above result works for all the cells in HRL-CKY.

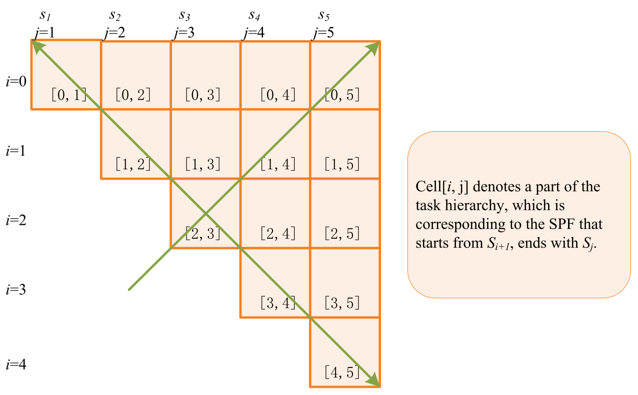


FIGURE 3. Illustration for HRL-CKY.

Generally speaking, if we intend to search the target service process "$s_1s_2s_3s_4s_5$", we firstly judge whether there is an element in Cell[0,5]. If Cell[0,5] has a value, we can reuse service process "$s_1s_2s_3s_4s_5$", and our query is successful. Similarly, if there are elements in Cell[0,2] and Cell[2,5], we can reuse "s_1s_2" and "$s_3s_4s_5$", and the query is still successful. This can be generalized to other cells. In a word, we can effectively discover and reuse any granularity of SPFs in terms of different requirements by means of HRL-CKY.

The remainder of this paper is organized as follows. In Section II, we discuss the related work relevant to our research. Then we formalize the used notions in Section III before we detail our approach in Section IV. We present the experiments in Section V and conclude this work in Section VI.

II. RELATED WORK

In this section, we discuss and analyze existing approaches for service composition, Reinforcement Learning (RL) issues in the area of services computing and service process fragments reuse.

A. SERVICE COMPOSITION

Some widely adopted methods for building composite services have been proposed from different perspectives. Tree-based method for service composition is one of them. For example, in [14], the exported behavior of a service was described in terms of a so-called execution tree, and thus the authors addressed the issue of automatic service composition. Petri Net is a tool that is often used in service composition. Through extending Coloured Petri Net (CPN) formalism, [15] incorporated transactional Web Services properties for service composition. Scalability is always a goal we should try our best to achieve for composing a series of atomic services. To achieve superior scalability and accuracy with respect to a large variety of composition scenarios, [16] designed a tool QSynth (QoS Synthesis) to use QoS objectives of service requests as the search directives. Similarly, for achieving scalability, [17] addressed a workflow orchestration to enable nested multilevel composition. QoS monitoring and forecasting are two important research topics in services computing [18], [19]. Service selection and classification are also widely addressed, which can make service composition more convenient [38], [39].

In this work, we exploit the method of service process fragments reuse for service composition, which can not only decrease the composition time, but also improve the reliability of the composition process.

B. SERVICE PROCESS FRAGMENTS REUSING

There are many studies about how to split and describe the process fragments. For example, [20] addressed the potential impact of using process fragment libraries in cross-enterprise collaboration and application integration. [21] proposed a method to decompose model into single-entry-single-exit (SESE) fragments. In [22], the researchers represented process fragments into different fragmentary knowledge in a formal way, which allows fragment models to be composed. Meanwhile, some novel methods have been proposed to contribute to the reuse of process fragment. [23] proposed a comprehensive framework for the dynamic, incremental, context-aware composition of process fragments into adaptable service-based applications. [24] used indexes to speed up query evaluation process by means of considering the semantic similarity between labels. In [25], to help process designers to adhere to compliance requirements relevant for their processes, an integrated approach was presented to compliance management. [26] addressed the issue, which can support users having different perspectives on process models and related data.

However, most of the above methods mainly focus on the complex control logic in the process. But when the scale of service processes is very large, the efficiency is most likely to decrease. In this work, we pay more attention to efficiently discovering and reusing any granularity of service process fragments. Furthermore, we combine the idea of Cocke-Kasami-Younger algorithm with hierarchical reinforcement learning, which is as a starting point for our work.

C. REINFORCEMENT LEARNING

Reinforcement Learning (RL) is one of the most active research areas in artificial intelligence, which is a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives when interacting with the environment [27]. To overcome the challenges of learning, planning, and representing knowledge at multiple levels of temporal abstraction, the authors [9] addressed those challenges within the mathematical framework of reinforcement learning and Markov decision processes (MDPs). RL is widely used in services computing. For example, [28] presented a multi-agent reinforcement learning model for Web service composition, which can address the scalability challenge, especially when the number of potential candidate services is large. [29] addressed a method CSSC-MDP, which modeled the constraint-satisfied service composition (CSSC) problem as a Markov decision process (MDP). With the development of GPS technology, a new Mobile Internet of Things (M-IoT) is emerging. [30] proposed a Mobile-IoT based multi-modal reinforcement learning service framework to deal with large scale and heterogeneous data. HRL is put forward to solve the curse of dimensionality problem of traditional RL and can achieve high efficiency [32]. In [31], the author built a relationship between HRL and decision making. [9] presented a method based on options, which makes decisions only when facing subgoal position and executes predetermined policy of option in other time. The HAM method can generate hierarchical, temporally abstract actions, where larger MDPs can be decomposed into smaller ones while maintaining a well-defined relationship between the smaller problem and the larger problem [10].

Though a lot of work in services computing, RL and HRL have not been applied to the research topic of service process fragments reuse. We have successfully attempted this in our work.

D. HIERARCHICAL CLASSIFICATION AND CLUSTERING

Besides Hierarchical Reinforcement Learning, Hierarchical Classification and Hierarchical Clustering are also typical research paradigms in machine learning related research areas. For example, [40] presented an adaptive resonance theory-supervised predictive mapping for hierarchical classification (ARTMAP-HC) network that allows incremental class learning for raw data regardless of normalization in advance, where each hierarchically stacked module incorporates two fuzzy ARTMAP networks. In [42], to address the memory problem and incorporating the knowledge of document structure, the authors proposed a hierarchical structured self-attention mechanism to create the sentence and document embeddings, and they defined the summarization task as a classification problem in which the model computes the respective probabilities of sentence-summary membership. [43] presented a deep hierarchical network (DHN) based on convolutional neural network (CNN) to address automatic modulation classification (AMC) problem. Similarly, Hierarchical Clustering has been extensively researched by the academe, and a large number of different methods have also been created. [41] proposed a novel approach k-Linkage, which calculates the distance by considering k observations from two clusters separately and can overcome the spurious clusters formation problem. By means of grouping a network into several clusters, a cluster-head being nominated for each cluster to make caching decision, [44] presented a two-layer hierarchical cluster-based caching solution to improve in-network caching efficiency, where the location and content popularity for caching both are considered. [45] addressed the Chameleon algorithm, whose selections are based on both interconnectivity and closeness, and it can yield accurate results for those highly variable clusters. A fuzzy semantic representation (FSR) method for rare words is presented in [46], which groups rare words together by means of a hierarchical clustering method and integrates it into the encoder-decoder framework.

III. PRELIMINARIES

In this section, we detail the notations used in our approach. For a clear understanding, we make use of the notations in Table 1.

TABLE 1. Notations.

Symbol	Description
S_i	i^{th} atomic service
ws	a Web service
$ws.I$ *and* $ws.O$	the sets of input and output of ws respectively
WS_i	i^{th} task
$US.I$	the set of user's input
$US.F$	the set of user's functional requirements
$US.Q$	user's QoS preferences
spf	a service process fragment
π	a policy
q_{ws}^{i}	the observed value of the i^{th} attribute of service ws
q_i^{max}	the maximum values of q_i for all services
q_i^{min}	the minimum values of q_i for all services
w_i	the weight of i^{th} QoS attribute
v_{ws}^{i}	the standardized value of ws's i^{th} QoS attribute
α	a time-varying learning-rate parameter
γ	a discount factor
$Q(\bullet, \bullet)$	Q-learning value
$C^{\pi}(\bullet, \bullet, \bullet)$	the expected discounted cumulative reward
$V^{\pi}(\bullet, \bullet)$	the Q-learning annotated value

A. SERVICE

A Web service ws can provide a reusable functionality that is specified in a service description document. Each atomic service owns input and output parameters, abbreviated as $ws.I$ and $ws.\ O$. Without the loss of generality, we assume a Web service ws can be formally described as follows:

Definition 1 (**Service**). A service ws is a 3-tuple $ws =< ws.I, ws.O, ws.Q>$, where I and O are the sets of input and output respectively. $I = \{I_1, I_2, \ldots, I_m\}$, and $O = \{O_1, O_2, \ldots, O_n\}$, m and n are the size of I and O, respectively. Q is the set of QoS(quality of service) attributes. $Q\,(ws)=\{q^1(ws), q^2(ws), \ldots, q^l(ws)\}$, where l is the dimension number and $q^i(ws)$ refers to the QoS value on i^{th} dimension of ws.

B. ABSTRACT SERVICE PROCESS FRAGMENT(ASPF)

An ASPF provides a control flow for two or more services, namely an ASPF is a workflow template, which contains service tasks instead of actual Web services. A task denotes an abstract functionality that can be performed by a concrete service.

Definition 2 (**ASPF**). ASPF=$\{WS_1, WS_2, \ldots, WS_n\}$, where $WS_i (i = 1, 2, \ldots, n)$ is a task, and $WS_i =< WS.\ I, WS.\ O, WS.Q >$.

C. SERVICE PROCESS FRAGMENT QUERY(SPF-QUERY)

If we only consider the QoS, input and output, a user may present a SPF query with the input and QoS constraints. Of course, if only the control structure sequence is considered, a SPF could be simplified as a directed graph, where each vertex and edge corresponds to Web service and dependence relationship between connected services, respectively. So, we can regard SPF-query as the traditional sub-graph matching problem. In this approach, the search restraint is just loose. A SPF query can be formally defined as follows.

Definition 3 (**SPF-query**). SPF-query=$\{US.I, US.F, US.Q\}$, where $US.I$ denotes the set of user's input, $US.F$ is the set of user's functional requirements, and $US.Q$ represents user's QoS preferences.

D. SEMI-MARKOV DECISION PROCESS (SMDP)

Semi-markov decision process (SMDP) is widely used in decision systems. During its generalization, the amount of time between one decision and the next is a random variable, either real- or integer-valued. In a discrete-time SMDP decisions can be made at integer multiples of an underlying time step [33]. In our SPF reuse scenario, a service process fragment may consist of several atomic services, where the services can take a variable amount of time steps to complete.

Definition 4 (**SMDP**). A SMDP is a five-tuple, i.e. SMDP=$<TS, SE, SA, P, R>$, where

– TS: Time steps.

– SE: The finite set of all possible states of environment.

– SA: The finite set of actions.

– P: The transition probability of the environment. Generally speaking, when an action is done, the environment may transit from its current state to a new state with time steps TS according to the probability distribution.

– R: when an action is done, the environment makes state changes, and the agent receives a real-valued reward.

E. SMDP FOR SERVICE PROCESS FRAGMENT(SPF-SMDP)

The build process of a SPF can be regarded as a semi-markov decision process, where each selected service may take a variable amount of time steps to complete. Moreover, sometimes for functional requirements, one atomic service may be called several times.

The SPF-SMDP model is defined as follows:

Definition 5 (**SPF-SMDP**). A Web Service Process Fragment (SPF) SMDP is a 6-tuple SPF-SMDP: SPF-SMDP=$< SE, se_0, SE_t, WS, P, R >$, where

– SE: The discrete set of environment states.

– se_0: The initial state. The SPF starts to execute from this state.

– SE_t: The set of terminal states. The SPF may end running with one of SE_t.

– WS: The set of Web services. Without loss of generality, a Web service can either be a composed Web service or an atomic Web service.

– P: The transition probability of the service system. When a Web service $ws \in WS$ is called, the service system makes a transition from its current state se to a resulting state se' with a probability $P(se', TS \mid se, ws)$.

– R: When a Web service of a SPF is performed, the service system environment makes state changes, and the service user receives an immediate reward r with expected value $R(se', TS \mid se, ws)$.

F. REWARD ASSESSMENT

In this approach, the evolved reinforcement learning applies to our service process fragment reuse scenario. First and foremost, we must properly model how to learn and update the reward, which relates to a SPF. A policy, π, is a mapping from states to service actions that tells what service action $ws = \pi(se)$ to perform when the service system is in state se. For simplicity, let all deterministic policies are proper– that is to say, all deterministic policies have a non-zero probability of reaching a terminal state when started in an arbitrary state. Given a user SPF query, each deterministic policy can uniquely return a service process fragment *spf*, and the user can receive a certain amount of reward by executing a specified SPF, which is equivalent to the cumulative reward of all the executed services.

There are two types of QoS attribute: the positive QoS attribute, i.e. a higher value implying a better performance or quality of a service; the negative QoS attribute, i.e. a higher value implying a worse quality [34]. For example, reliability, reputation degree and availability are all positive, however, service price and service time both are negative. The positive and the negative QoS attributes are normalized by formula 1 and formula 2, respectively.

$$v_{ws}^{i} = \begin{cases} \dfrac{q_i^{max} - q_{ws}^{i}}{q_i^{max} - q_i^{min}} & if q_i^{max} - q_i^{min} \neq 0 \\ 1 & if q_i^{max} - q_i^{min} = 0 \end{cases} \tag{1}$$

$$v_{ws}^{i} = \begin{cases} \dfrac{q_{ws}^{i} - q_i^{min}}{q_i^{max} - q_i^{min}} & if q_i^{max} - q_i^{min} \neq 0 \\ 1 & if q_i^{max} - q_i^{min} = 0 \end{cases} \tag{2}$$

where v_{ws}^{i} denotes the standardized value of service ws's i-th QoS attribute, q_{ws}^{i} represents the observed value of the i-th attribute of service ws, q_i^{max} and q_i^{min} represent the maximum and minimum values of q_i for all services.

Then, we can calculate the overall quality score for each service by Eq.3.

$$R(ws) = \sum_{i=1}^{k} wi \times v_{ws}^{i} wi \in [0, 1] \text{and} \sum_{i=1}^{k} wi = 1 \tag{3}$$

where k represents the number of QoS attributes, w_i represents the weight of each attribute according to the user preference.

Q-learning method is widely used in most RL algorithms, which is usually based on the DP (Dynamic Programming) backup but with the expected immediate reward and the expected maximum action-value of the successor state [8]. Q-learning algorithm updates an estimation of the value of performing action using the following equation:

$$Q(se, sa) \leftarrow (1-\alpha) * Q(se, sa) + \alpha * (r + \gamma * \max Q(se', sa')) \tag{4}$$

where α is a time-varying learning-rate parameter, and γ is a discount factor.

However, Eq.4 doesn't consider the different time steps that Web services take. In fact, our SPF reuse scenario utilizes SMDP and also considers the future benefits. So, it is a good strategy to take the time steps of each Web service into consideration. We update $Q(se, sa)$ value in our SPF-SMDP model by Eq.5, where TS is the time steps of Web service (or action) sa.

$$Q(se, sa) \leftarrow (1-\alpha) * Q(se, sa) + \alpha * (r + \gamma^{TS} * \max Q(se', sa')) \tag{5}$$

IV. TECHNICAL APPROACH

In this section, we address how to do SPF-query with automatic HRL and extended Cocke-Kasami-Younger (CKY) algorithm. Section 4.1 discusses action models of services. Task hierarchies and causal analysis are detailed in Section 4.3 and Section 4.2, respectively. At last, in Section 4.4, we present our algorithms.

A. ACTION MODELS OF SERVICE

In our SPF reuse approach, we integrate task hierarchies with extended CKY method. Since task hierarchies rely on compact and inspectable action models, we first detail how to construct an action model, which represents the effect that executing an action has on the state variables. For the sake of understanding, before diving into the details, we first illustrate the process using a simple example below.

As shown in Fig.1, assume that a tourist decides to take a one-day tour. He (or she) must arrange transportation according to the weather. If it rains, he has to rent a car. Otherwise, he prefers a bike. In order to learn about relevant scenic spots, he may search for nearby attractions, restaurants and hotels. The states and actions are described in Table 2. The service state is described through the following variables: $t.l$ represents the location of tourist t, $t.r$ indicates whether the tourist is empty-handed or having a resource (bike or car), $w.s$ indicates if it is raining, the binary $r.b$, $r.c$ variables indicate whether there is a bike or car for tourist's trip, and the binary

TABLE 2. State and action description.

Variable	Description
$t.l$	the location of tourist
$t.r$	tourist's resource
$r.b$	indicator for bike
$r.c$	indicator for car
$q.b$	quota indicator for bike
$q.c$	quota indicator for car
$w.s$	weather situation
Action	**Description**
GO	navigate to a location
RB	rent bike
RC	rent car
SA	search attractions

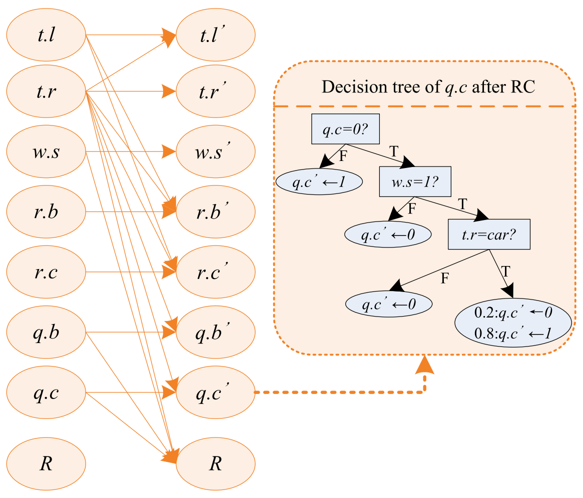


FIGURE 4. The dynamic bayesian network model for the RC action.

$q.b$, $q.c$ variables indicate whether the required quota of bike or car has been met.

The tourist has four actions: GO, causing his location to change; RB, causing him to ride a bike if it is sunny; RC, causing him to drive if it is raining; and SA denoting that the tourist searches attractions. If RB is executed when an empty tourist t is in the vicinity of a bike-hire company, then this will change $t.r$ to bike; otherwise, this action will have no effect. Similarly, a successful RC will change $t.r$ from empty to car.

As we can see from Fig.4, we employ dynamic Bayesian networks (DBNs) with context-specific independence to denote the action models. A DBN is a bipartite directed graph, where each node represents a state variable, and each edge denotes a direct causal dependence. The first stage (shown on the left in Fig.4) and the second stage (on the right) of the graph represent the state variables before and after the actions are executed, respectively. In the second stage, the node labeled with R represents the immediate reward received after the action is executed. On the far right of Fig.4 is a decision tree, it represents the conditional probability distributions of car quota ($q.c'$) after RC is executed. The tree structure represents the fact that it remains unchanged if the weather does not rain or the tourist does not have a resource of car. In a nutshell, the decision tree captures the fact that the probability distribution over $q.c$' depends on the context. Moreover, this is more compact than representing the probability distribution as a table, where we must enumerate all possible combinations of values of the parents.

B. CAUSAL ANALYSIS

As shown in Fig.2, we utilize SPF-Hierarchy to produce task hierarchies. However, a CAT (causal annotated trajectory) is the input of SPF-Hierarchy. So, in this section, we detail causal analysis as follows.

The input trajectory is a sequence of actions that achieves the overall goal in the source problem. For example, a trajectory in our SPF-reuse scenario is a sequence of Go, RB, RC, and SA actions that together achieves the requisite quota of bike or car, and attractions. Generally speaking, before a trajectory is fed to the SPF-Hierarchy algorithm, it should be annotated with causal information using the DBN models. Further speaking, the intent of this annotation is to identify how executing actions affects the state variables. It's worth noting that those state variables may be relevant to future actions and, ultimately, the goal of the overall task.

Fig.5 is a causally annotated trajectory example of the service process in Fig.1. As we can see from Fig.5, the annotation is based on the relevance of variables to actions, which comes from the DBNs. Specifically, in state se, given an action sa and a variable v, if the reward and transition dynamics for sa either check or change v, we affirm v is relevant to sa. In Fig.5, the variable $w.s$ is always relevant to RB and RC, because he arranges his transportation according to the weather.

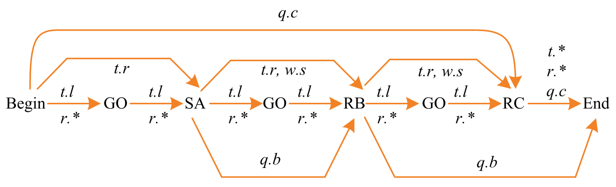


FIGURE 5. A causally annotated trajectory example.

C. TASK HIERARCHIES

Our HRL-CKY makes task hierarchies as inputs. So, for the SPF-reuse effect, it is really important that how we achieve succinct and efficient task hierarchies. We construct the automatic hierarchies based on HI-MAT [35] (Hierarchy Induction through Models and Trajectories). That is to say, we address the above issue by systematically integrating automatic task decomposition. Given the CAT (causal annotated trajectory), the DBN model, and the SMDP's goal as input, our algorithm SPF-Hierarchy can partition the CAT recursively to discover the hierarchical structure, and every partition corresponds to a candidate subtask (i.e. service process fragment).

With the aid of task hierarchies, we can arrange plan at multiple time scales. Concretely, plans at higher levels (larger temporal scales) can be refined into sub-plans at lower levels (finer temporal scales). For example, in Fig.1, a traveler plans to take a one-day tour. He may first do a weather query before choosing which day to go out, which in turn may be considered before choosing what kind of transportation to use.

In this work, our SPF-reuse approach is based on the MAXQ framework for representing task hierarchies [11], where each task has a goal or termination condition. The goal describes what the task is trying to achieve, but the termination condition indicates under which it can be invoked. Moreover, each task may have a set of sub-tasks that it can invoke recursively to achieve its goal and a set of relevant state variables, i.e. the state abstraction. The root of the task hierarchy corresponds to the over-all MDP, while the leaves correspond to primitive actions. Each MAXQ task hierarchy has a hierarchical policy, which is a collection of local policies, one for each task. During execution of a hierarchical policy, each sub-task follows its local policy until its goal or termination condition is true. When sub-task returns to the calling parent task, the reward obtained during its execution also is brought back to its parent.

The SPF-reuse domain is too complex for a nonhierarchical solution, because the more the candidate services, the larger the state space and the larger the number of candidate service process fragments available for a specific functional requirement. For example, in Fig.1, there may be a large number of car rental services, bike rental services, and weather services. So, for the one-day tour requirement, we can get many service processes. For each of those service processes, it consists of one weather service, one bike rental service or one car rental service. However, as we can see from Fig.6, the task hierarchy decomposes the overall problem into simpler sub-problems. The Root task tries to learn a policy for meeting the overall requirement in the original MDP by solving and combining the solutions of three subtasks, i.e. Get bike, Get weather, and Get car. Of course, we can only choose one between Get bike and Get car, which is on the basis of the value of Get weather. So, for indicating this dependency relationship, in Fig.6, we draw a directed edge with dotted line from Get weather to Get bike and Get car, respectively.

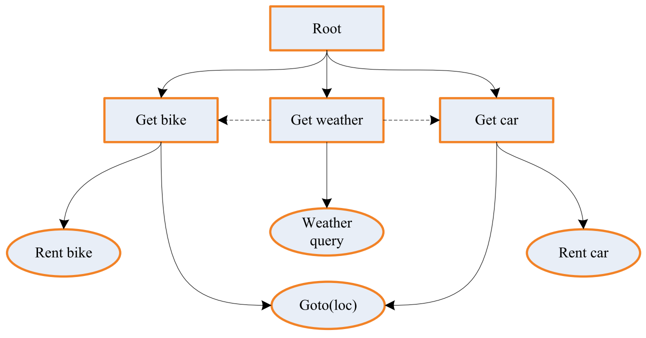


FIGURE 6. A task hierarchy for the one-day tour.

We perform the partitioning process based on the goals of the task. That process works backward from the goals of the task. By means of combining the goals and the action models, some appropriate preconditions can be discovered, and small CAT segments can be separated, which are responsible for achieving the goal. For each literal in the goal condition, we extract the corresponding segment of the trajectory through the process of finding the set of temporally contiguous actions in the CAT. CAT scanning is repeated until all literals are accounted for. During the partitioning process, if an extracted trajectory segment is equal to the entire CAT, we can conclude that the segment achieves only the literal emerging out of the ultimate action. As a result, we can split the trajectory into two new segments. One contains the ultimate action, but the other segment contains everything prior to the ultimate action. When extracting the segment, the same subtasks are merged, since they have same termination conditions and subtasks. As presented in [35], we also simplify the termination predicate. For example, there is a goal condition $q.b = 1 \wedge t.r = \text{empty} \wedge t.l = \text{home}$, if the first literal is true, we don't need to consider the second and third ones. Fig.7 is the hierarchy for the one-day tour.

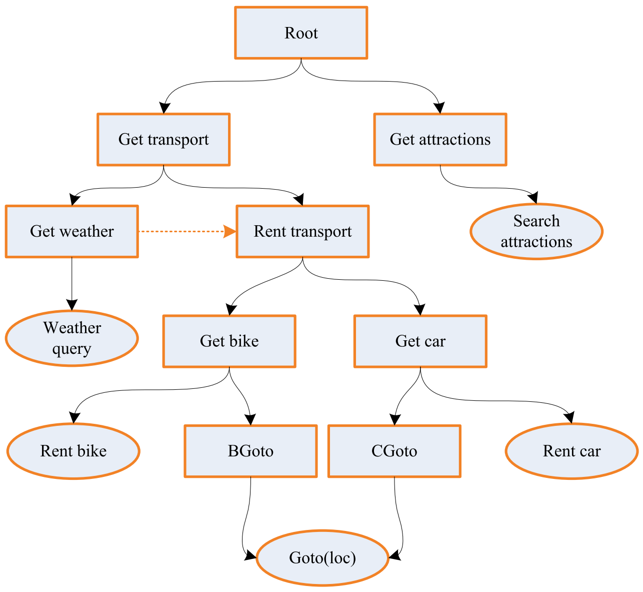


FIGURE 7. The hierarchy induced by SPF-hierarchy algorithm for the one-day tour.

In the graph of SPF-Hierarchy, for one node, let it corresponding to the subtask ST_{sa}, and its parent node is task ST_i, then we get the following equation for the value of ST_i:

$$C^{\pi}(i, se, sa) = \sum_{se', TS} P_i^{\pi}(se', TS|se, sa)\gamma^{TS} Q^{\pi}(i, se', \pi(se')) \quad (6)$$

which is the expected discounted cumulative reward of completing subtask ST_i after invoking the subroutine for subtask ST_{sa} in state se.

Then, for subtask ST_i, its Q-learning annotated value function in the form of a Bell equation is as follows:

$$V^{\pi}(i, se) = V^{\pi}(\pi i(se), se) + \sum_{se', TS} P_i^{\pi}(se', TS|se, \pi i(se))\gamma^{TS} V^{\pi}(i, se') \quad (7)$$

Based on Eq.6 and Eq.7, we can further get the Q value function for subtask ST_i as follows:

$$Q^{\pi}(i, se, sa) = V^{\pi}(sa, se) + \sum_{se', TS} P_i^{\pi}(se', TS|se, sa)\gamma^{TS} Q^{\pi}(i, se', \pi(se')) \quad (8)$$

where se' is the state after action sa is executed.

As stated before, the partitioning process works backward from the goals of the task. So, we can get the value function of root task (equal to the whole task) as the following process assumptions and real calculations. Assume that the agent chooses subtask ST_{sa1} according to the policy of subtask ST_0, then recursively the agent chooses ST_{sa2} according to the policy of ST_{sa1} and so on. As a result, Eq.9 indicates the computing process, where ws_m is a primitive subtask:

$$\begin{aligned} V^{\pi}(ws_m, se) &= \sum_{se'} P(se'|se, sa_m)R(se'|se, sa_m) \\ V^{\pi}(0, se) &= V^{\pi}(ws_m, se) + C^{\pi}(sa_m - 1, se, ws_m) \\ &\quad + \ldots + C^{\pi}(0, se, sa_1) \end{aligned} \tag{9}$$

D. ALGORITHMS

In this section, we detail two algorithms, i.e. SPF-Hierarchy and HRL-CKY. SPF-Hierarchy benefits the construction of task hierarchies, which is based on HI-MAT [35]. Moreover, its output is just the input of HRL-CKY, which is responsible for computing the value function to solve the SPF-reuse problem.

In Algorithm 1, the inputs are a causally annotated trajectory (CAT), DBN models and the goal predicate G. The algorithm discovers the hierarchical structure by recursively partitioning the CAT and each partition corresponds to a candidate subtask. As shown in Fig.7, it is the hierarchy induced by SPF-Hierarchy algorithm for the one-day tour example presented in Fig.1. The root task terminates when the requisite traffic tool ($q.b$=1 or $q.c$=1) and information for nearby attractions have been collected. And it can be separated into two subtasks, i.e. Get transport and Get attractions. In the same way, Get transport is decomposed into Get weather and Rent transport. Since a traveler must choose one means of transportation according to weather condition, there is a red directed edge from Get weather to Rent transport, which describes this dependency. The leaf node Goto(loc) is a primitive action that is not included to the task.

After executing Algorithm 1, we get MAXQ Hierarchies, which are imported into Algorithm 2. Specifically, MAXQ Hierarchies are the input of our Reinforcement Learning method. Through reward values calculation for all the service process fragments, our method outputs two 2-dimensional arrays, i.e. Link and Q-value. The former records the links among its constituent parts of subtasks, but the latter Q-value stores those values computed by value functions.

As shown in Fig.3, if there is an element (ST, Q) in Q-value [2], [3], it indicates that ST is a subtask and its value is Q. ST is mapped to Cell[2, 3], and if we do a depth traversal from the node of ST, by gathering all the leaves, we can get a service process fragment, which match services between s_i to s_{i+j-1}. To reproduce the alternative parts of an optimal derivation, we use Link to record the links among its constituent parts. In the hierarchy induced by Algorithm 1, for all the leaf nodes, their values are computed in lines 3-11.

From line 12 to line 26, the values for all the subtasks are computed.

E. COMPLEXITY ANALYSIS

Our method mainly consists of two steps, i.e. the automatic hierarchy construction (detailed in Algorithm 1) and the process of service process fragments reusing (seen in Algorithm 2). In the second step, we can effectively discover and reuse any granularity of SPFs in terms of different requirements by the hierarchy given by step1. Algorithm 1 has complexity O($|\S|$), where $|\S|$ denotes the number of subtasks of $\S$. In Algorithm 2, it firstly incurs complexity O(n) to compute values for all the leaf nodes (lines 3-11). Next, computing values for all the subtasks has complexity O($|\S|^2 \times n^3$) (lines 12-26). Thus, we obtain a total complexity of O($|\S|^2 \times n^3$). Actually, our method can effectively reduce the convergence time, which is shown in the experiments. This is because that our method is based on CATs, and we partition each CAT to smaller CAT segments and every segment corresponds to a deterministic subtask, the hierarchy generated by Algorithm 1 is unique for each unique CAT.

Algorithm 1 SPF-Hierarchy

```
Input: CAT $\S$, DBN models, Goal predicate $G$
Output: Task $T$
1.  if $\S$ contains a single action $sa$ then
2.  return task $T$ with termination $G$, state abstraction
    based on variables relevant to $sa$, and child $sa$
3.  else
4.      if actions in $\S$ have identical sets of relevance then
5.          return task $T$ with $G$, state abstraction based on
    $\S$'s relevance, actions in $\S$
6.  Extract trajectory segments from $\S$ for goal literals and
    any literals entering the segments
7.  if a segment $=\S$ then
8.      Create two segments: one with the ultimate action
    and the other with the rest and the ultimate action's
    precondition as its goal
9.  Merge all overlapping segments
10. for each $CAT$ segment do
11. Invoke SPF-Hierarchy recursively to discover the sub-
    task hierarchy and add it as a chid of $T$
12. Set termination for $T = G$
13. Set state abstraction for $T$ based on the relevant vari-
    ables from $\S$'s merged DBN
14. Add all primitive actions to $T$ that share DBN structure
    but are not already included
15. return task $T$
```

V. EXPERIMENTS AND RESULTS

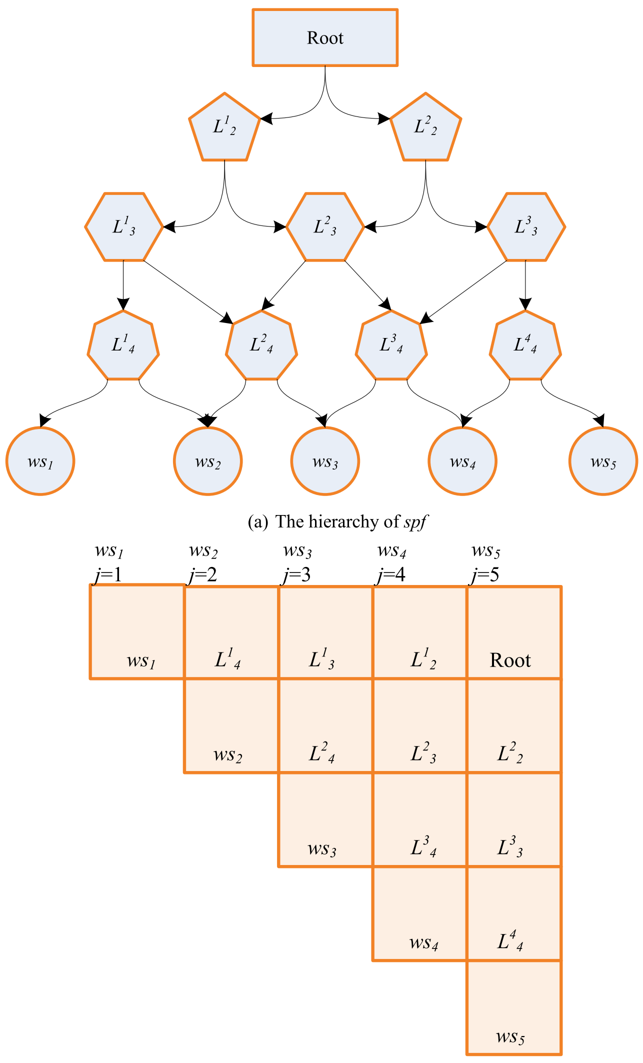
In this section, we will use some experiments to evaluate our approach. In service process fragment reusing research area, [36] presented VGI (Variable Granularity Index) method, which is based on SSM-Tree on service processes. VGI can realize the unified index on both atomic and composite services (i.e. service processes) and maximize reuse of them. However, it does not consider QoS attributes and it is slightly less efficient when it faces a large-scale dataset. SCKY [13] (a service process fragment reusing method motivated by Cocke-Kasami-Younger algorithm) can reuse any granularity of service process fragments, but the performance needs to be improved.

Algorithm 2 HRL-CKY

Input: MAXQ Hierarchy

Output: a 2-dimensional array Link, a 2-dimensional array Q-value

```
1.  Q-value[$n$, $n$]$\leftarrow \Phi$;
2.  Link[$n$, $n$]$\leftarrow \Phi$;
3.  **for** $i = 1$ **to** $n$
4.     **if** there is a primitive subtask $ST$ in Cell[$i$-1, $i$] **then**
5.        compute R_pattern($se$, $i$) and R_QoS($se$, $i$) ; //let $se$ is the current state
6.        r$\leftarrow$ R_Pattern($se$, $i$)+ R_QoS($se$, $i$);
7.        $V^{\pi}(si, se) \leftarrow (1-\alpha)V^{\pi}(si, se) + \alpha r$;
8.        Q-value[$i$, 1]$\leftarrow V^{\pi}(si, se)$;
9.        Link[$i$, 1] $\leftarrow$(‘$ST$’, 0, ‘$s_i$ ’);
10.    **End if**;
11. **End for;**
12.    **For** $j = 2$ **to** n
13.     **For** $i = 1$ **to** $n$-$j + 1$
14.      **For** $k = 1$ to $j$-1
15.       **For** each elem($ST_1$) $\in$ Q-value [$i$, $k$];
16.        **For** each elem($ST_2$) $\in$ Q-value [$i + k$, $j$-$k$];
17.          **if** $ST \rightarrow ST_1$ $ST_2$ **then** // $ST_1$ and $ST_2$ are $ST$’s children
            $Q1 \leftarrow (1-\alpha)C^{\pi}(i, se, ST_1) + \alpha\gamma^{j-i+1}V^{\pi}(ST_1, se)$
18.         $Q2 \leftarrow (1-\alpha)C^{\pi}(i, se, ST2) + \alpha\gamma^{j-i+1}V^{\pi}(ST2, se)$
            $Q \leftarrow Q1 + Q2$
19.          Q-value[$i$, j]$\leftarrow Q$;
20.          Link[$i$, j] $\leftarrow$($ST$, $k$, $ST_1$, $ST_2$);
21.         **End if**;
22.        **End for;**
23.       **End for;**
24.      **End for;**
25.    **End for;**
26. **End for;**
```



(a) The hierarchy of *spf*

(b) The mapping between the hierarchy and HRL-CKY for *spf*

FIGURE 8. A SPF reusing example.

Schumm *et al.* [47] presented a method for Web service composition through shared process fragment libraries. In [48], the authors proposed a complete and integrated framework to enable reuse and sharing of service compositions over Web using stimulation workflows. However, [47] and [48] do not address on how to search or locate the process fragment at all. Moreover, they mainly focus on the complex control logic in the process. Especially, when the scale of service processes is very large, they are likely to lose efficiency. In this work, we pay more attention to efficiently discover and reuse useful service process fragments. Furthermore, we combine the idea of Cocke-Kasami-Younger algorithm with hierarchical reinforcement learning, which is as a starting point for our research.

A. EXPERIMENT SETUP

Our dataset is constructed by the Web Service Challenge Testset Generator (CTG). We have invoked CTG many times since the number of generated processes based on CTG is limited each time. In this way, we get a large scale process repository which consists of about 400 thousand processes and each service contains 5 to10 input or output parameters. We extracted throughput and response time values from a file named Servicelevelagreements which is generated concurrently.

As presented in [13], several variable parameters are utilized: the amount of process (AP) in the process repository, the average number of abstract services (AS) and dependency relationships (AR) among them in each query process. Experimental parameters are set as follows: the discount factor γ of the Q-learner is 0.9, and the learning rate α is set to 0.2. All algorithms are implemented in Java. The hardware environment is a machine with the Intel(R) Core(TM) i5 CPU 760, 2.80 GHz, and 4 GB RAM running Windows 7 (64-bit).

Fig.8 is an example of SPF reusing induced by the above approach. We choose one service process fragment *spf* from our dataset, which consists of five atomic services (according to the input and output dependencies, they combine into a service process fragment), shorted as $ws_1, ws_2, \ldots, ws_5$. As we

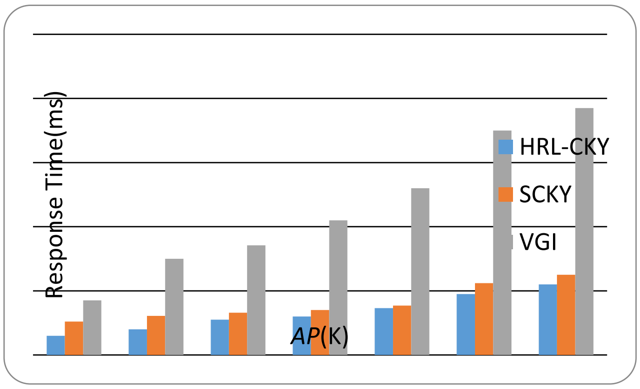


FIGURE 9. Efficiency evaluation with variable *AP* (*AS* = 5 and *AR* =4).

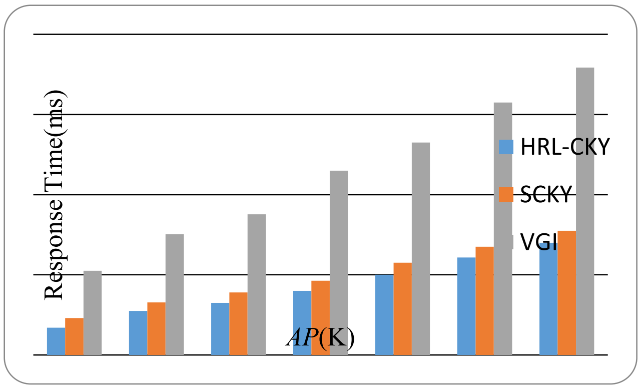


FIGURE 10. Efficiency evaluation with variable *AP* (*AS* = 12 and *AR* = 11).

can see, Fig.8a is the hierarchy induced by SPF-Hierarchy algorithm for *spf*, and Fig.8b is the mapping between the hierarchy and HRL-CKY for *spf*. L_i^j denotes the i^{th} layer of the hierarchy, and it indicates the service process fragment, which starts from ws_j. For example, L_2^1 indicates the service process fragment "$ws_1 ws_2 ws_3 ws_4$", but L_3^2 embodies "$ws_2 ws_3 ws_4$". Each leaf node of Fig.8a denotes an atomic service. Actually, if we do a breadth priority traversal from L_i^j, and combine all the atomic services, we can just get the corresponding service process fragment. As shown in Fig.8a, when it travels from L_3^3, it gets through "$L_4^3 L_4^4$", finally "$ws_3 ws_4 ws_5$". As a result, we can effectively discover and reuse any granularity of SPFs.

B. EXPERIMENTAL RESULT

In the first set of experiments, we evaluate and compare the efficiency of our approach with SCKY and VGI. In the original work of VGI, it did not consider QoS attributes. So, for the comparison, we added QoS attributes to VGI method. Given a query condition with functional requirements and QoS preferences, we find a SPF from the process repository. All experiments are conducted independently for 100 times and the average results are calculated. When *AP* varies from 100K to 400K, as shown in Fig.9 and Fig.10, the response time of VGI, SCKY and HRL-CKY grows. If the query condition becomes more complicated, i.e. from "*AS*=5 and *AR*=4" to "*AS*=12 and *AR*=11", three methods all need more query time. However, under the two experimental environments, HRL-CKY is faster than SCKY, especially better than VGI. This is because that HRL-CKY utilized Hierarchical Reinforcement Learning (HRL) technique, which can release the curse of dimensionality effectively.

In the second set of experiments, we focus on how the learning rate affects HRL-CKY's performance. We fix the amount of process (*AP*) to 400K, *AS*=8 and *AR*=7. We vary the learning rate from 0.1, 0.2, 0.7 to 0.8. As we can see from Fig.11, though higher learning rate can accelerate the learning process, it is easier to trap into local optimal and a smaller learning rate is helpful to avoid this problem although its convergence rate is a little slow. Through this set of experiments, we get that a higher learning rate can convergence

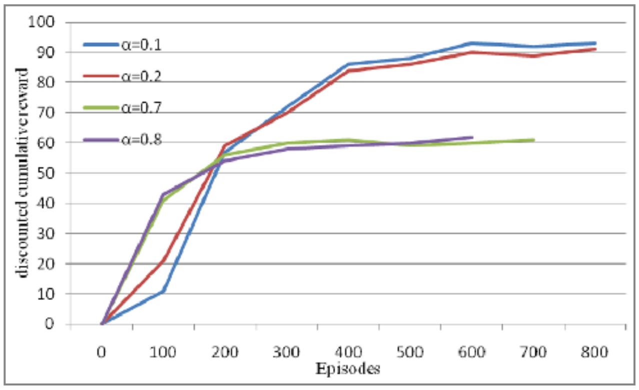


FIGURE 11. Performance on learning rate.

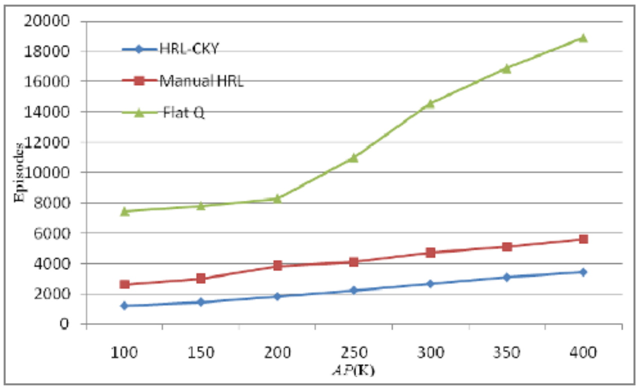


FIGURE 12. Learning speed with variable *AP*.

faster. But it is not that the faster the convergence, the better the performance is. Maybe a learning rate around 0.2 is a good choice.

In the third set of experiments, we try to address the influence of the number of *AP* on the learning speed. In Fig.12, we compare the efficiency of our approach with Manual HRL [11] and Flat Q [37]. We fix *AS*=8 and *AR*=7. Manual HRL is a hierarchical policy that is coded by hand. When *AP* varies from 100K to 400K, as shown in Fig.12, HRL-CKY converges fastest, and Manual HRL comes second. Actually,

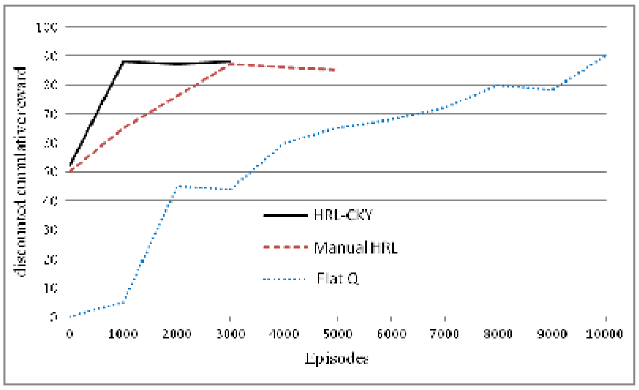


FIGURE 13. Three methods' adaptability.

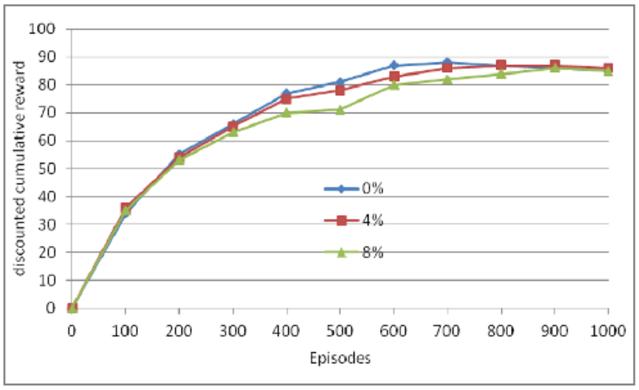


FIGURE 14. HRL-CKY's adaptability with different changing percents.

HRL-CKY converges roughly two times as fast than Manual HRL. In our hand-coded policy, knowledge of contextual information is used to choose operators. So, Manual HRL is surely better than Flat Q though inferior than HRL-CKY, and Flat Q's convergence time increases polynomially with the number of service processes variation. It is clear that the number of alternative service process fragments usually increase exponentially with the number of AP. However, with state abstraction, our HRL-CKY can release the curse of dimensionality effectively and the result looks flat. By means of state abstraction, subtasks can employ a lower-dimensional representation for their value functions, and this will speed learning. Furthermore, in our hierarchical policy, we merge the same subtasks whose relevances are identical. That is, they have same termination conditions and subtasks. As a result, this will further improve the convergence rate.

The purpose of the last set of experiments is to verify the adaptability of our method. We fix AP=250K, AS=8 and AR=7, but we change the QoS attributes of the services periodically to simulate the changes of the environment. In Fig.13, we vary 4% of the services' QoS attributes to compare the three methods' adaptability. As we can see from Fig.13, all the three approaches (HRL-CKY, Manual HRL and Flat Q) can still converge with delay, but the effectiveness is not same. It is clear that HRL-CKY performs more adaptively. In Fig.14, we vary 0%, 4% and 8% respectively to address the influence that different changing percents have on adaptability. From our experimental result, we can get that the more changes, the longer delay. But the increased delay is small compared with the whole convergence time. So our HRL-CKY gains decent adaptability.

C. DISCUSSION AND ANALYSIS

By these experiments conducted on a large scale dataset, we firstly find that HRL-CKY is much faster than VGI and SCKY. In HRL-CKY, we use link list to store dependency relationships of services. Moreover, through state abstractions, it can greatly reduce the search space. All these factors contribute to the good efficiency performance of HRL-CKY. In addition, the abstraction of any task in the task hierarchy is maximally compact, that is, it does not contain redundant state variables. For example, in Fig.8a, ws_2 relates to L_4^1 and L_4^2, but we only present ws_2 just once in the hierarchy. Compactness determines the speed of learning in the SPF reusing — the more compact the abstraction, the fewer the number of parameters to be learned, and hence the faster the learning. Fig.12 evaluated this conclusion. We will omit the detailed discussion due to the space limitation.

Although our method has many advantages mentioned above, it still has a lot to be improved upon. First, our automatic hierarchy is based on the HI-MAT [35], where an observed successful trajectory is necessary. Second, when transferring the hierarchy made by a source service process fragment (SPF) to a target service process fragment, there maybe limitations. Third, we mainly consider the variation of non-functional properties for the adaptability, however, the deterioration of candidate services may happen, or new functional services join in. In our future work, we will overcome these drawbacks.

VI. CONCLUSION AND FUTURE WORK

In this paper, we present a novel SPF (Service Process Fragment) reusing framework that combines automatic HRL and extended Cocke-Kasami-Younger (CKY) algorithm. This framework has the ability to reuse any granularity of SPFs through HRL-CKY. Through the analysis on historical fragments, action models and trajectories can be firstly returned. The "Causal Analysis" identifies the causal relationships between the actions in the trajectory, i.e. returning a causally annotated trajectory (CAT). Then, SPF-Hierarchy algorithm discovers a coherent task hierarchy. It is worth noting that the task hierarchy can minimize the number of intertask causal links, such that avoiding the "curse of dimensionality". Moreover, by means of state abstractions, our method can greatly reduce the search space. In addition, the abstraction of any task in the task hierarchy is maximally compact, that is, it does not contain redundant state variables. We conduct several groups of experiments to evaluate the efficiency and robustness of our approach, and experiments show that HRL-CKY performs well in service process fragments reuse.

Our future work is to investigate how to deal with service processes with more QoS attributes. And, we will focus on how to better generating a successful trajectory. We will try other new automatic HRL mechanisms to overcome those drawbacks as discussed in Section V.C.

ACKNOWLEDGMENT

The authors deepest gratitude goes to the anonymous reviewers for their careful work and thoughtful suggestions that will help us improve this paper substantially.

REFERENCES

[1] H. Ma, H. Zhu, K. Li, and W. Tang, "Collaborative optimization of service composition for data-intensive applications in a hybrid cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1022–1035, May 2019.

[2] Y. Zhang, F. Tao, Y. Liu, P. Zhang, Y. Cheng, and Y. Zuo, "Long/short-term utility aware optimal selection of manufacturing service composition toward industrial Internet platforms," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3712–3722, Jun. 2019.

[3] S. Chattopadhyay and A. Banerjee, "QoS-aware automatic Web service composition with multiple objectives," *ACM Trans. Web*, vol. 14, no. 3, pp. 1–38, Jul. 2020.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[5] A. Moustafa and M. Zhang, "Multi-objective service composition using reinforcement learning," in *Proc. Int. Conf. Service-Oriented Comput.*, 2013, pp. 298–312.

[6] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, and A. Bouguettaya, "Adaptive service composition based on reinforcement learning," in *Service-Oriented Computing*. San Francisco, CA, USA: Springer, 2010, pp. 92–107.

[7] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Mach. Learn.*, vol. 33, nos. 2–3, pp. 235–262, 1998.

[8] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, nos. 1–2, pp. 41–77, 2003.

[9] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, nos. 1–2, pp. 181–211, Aug. 1999.

[10] R. E. Parr, "Hierarchical control and learning for Markov decision processes," Ph.D. dissertation, Univ. California, Berkeley, Berkeley, CA, USA, 1998.

[11] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, Nov. 2000.

[12] M.-H. Zhang, "Quantitative structural information for inferring context free grammars with an extended Cocke–Younger–Kasami algorithm," *Pattern Recognit. Lett.*, vol. 32, no. 6, pp. 860–865, Apr. 2011.

[13] R. Yang, B. Li, J. Wang, L. He, and X. Cui, "SCKY: A method for reusing service process fragments," in *Proc. IEEE Int. Conf. Web Services*, Jul. 2014, pp. 209–216.

[14] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "Automatic service composition based on behavioral descriptions," *Int. J. Cooperat. Inf. Syst.*, vol. 14, no. 4, pp. 333–376, 2005.

[15] J. E. Haddad, M. Manouvrier, and M. Rukoz, "TQoS: Transactional and QoS-aware selection algorithm for automatic Web service composition," *IEEE Trans. Serv. Comput.*, vol. 3, no. 1, pp. 73–85, Jan./Mar. 2010.

[16] W. Jiang, C. Zhang, Z. Huang, M. Chen, S. Hu, and Z. Liu, "QSynth: A tool for QoS-aware automatic service composition," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Miami, FL, USA, Jul. 2010, pp. 42–49.

[17] I. Paik, W. Chen, and M. N. Huhns, "A scalable architecture for automatic service composition," *IEEE Trans. Serv. Comput.*, vol. 7, no. 1, pp. 82–95, Mar. 2014.

[18] P. Zhang, H. Jin, H. Dong, and W. Song, "M-BSRM: Multivariate BayeSian runtime QoS monitoring using point mutual information," *IEEE Trans. Serv. Comput.*, early access, Nov. 11, 2019, doi: 10.1109/TSC.2019.2952604.

[19] P. Zhang, H. Jin, H. Dong, W. Song, and L. Wang, "LA-LMRBF: Online and long-term Web service QoS forecasting," *IEEE Trans. Serv. Comput.*, early access, Feb. 26, 2019, doi: 10.1109/TSC.2019.2901848.

[20] D. Schumm, D. Karastoyanova, O. Kopp, F. Leymann, M. Sonntag, and S. Strauch, "Process fragment libraries for easier and faster development of process-based applications," *J. Syst. Integr.*, vol. 2, no. 1, pp. 39–55, 2011.

[21] J. Vanhatalo, H. Völzer, and F. Leymann, "Faster and more focused control-flow analysis for business process models through SESE decomposition," in *Proc. 5th Int. Conf. Service-Oriented Comput.*, 2007, pp. 43–55.

[22] H. Eberle, T. Unger, and F. Leymann, "Process fragments," in *Proc. 17th Int. Conf. Cooperat. Inf. Syst.*, 2009, pp. 398–405.

[23] A. Bucchiarone, A. Marconi, M. Pistore, and H. Raik, "Dynamic adaptation of fragment-based and context-aware business processes," in *Proc. ICWS*, Jun. 2012, pp. 33–41.

[24] T. Jin, J. Wang, M. L. Rosa, A. T. Hofstede, and L. Wen, "Efficient querying of large process model repositories," *Comput. Ind.*, vol. 64, no. 1, pp. 41–49, Jan. 2013.

[25] D. Schumm, O. Turetken, N. Kokash, A. Elgammal, F. Leymann, and W.-J. van den Heuvel, "Business process compliance through reusable units of compliant processes," in *Proc. 10th Int. Conf. Web Eng.*, 2010, pp. 325–337.

[26] J. Kolb, K. Kammerer, and M. Reichert, "Updatable process views for user-centered adaption of large process models," in *Proc. ICSOC*, 2012, pp. 484–498.

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning* (A Bradford Book), vol. 15, no. 7. Cambridge, MA, USA: MIT Press, 1998, pp. 665–685.

[28] H. Wang, X. Wang, X. Zhang, Q. Yu, and X. Hu, "Effective service composition using multi-agent reinforcement learning," *Knowl.-Based Syst.*, vol. 92, pp. 151–168, Jan. 2016.

[29] L. Ren, W. Wang, and H. Xu, "A reinforcement learning method for constraint-satisfied services composition," *IEEE Trans. Serv. Comput.*, vol. 13, no. 5, pp. 786–800, Oct. 2020.

[30] P. Wang, L. T. Yang, J. Li, X. Li, and X. Zhou, "MMDP: A mobile-IoT based multi-modal reinforcement learning service framework," *IEEE Trans. Serv. Comput.*, vol. 13, no. 4, pp. 675–684, Aug. 2020.

[31] M. M. Botvinick, "Hierarchical reinforcement learning and decision making," *Current Opinion Neurobiol.*, vol. 22, no. 6, pp. 956–962, 2012.

[32] T. Yu, Y. M. Wang, W. J. Ye, B. Zhou, and K. W. Chan, "Stochastic optimal generation command dispatch based on improved hierarchical reinforcement learning approach," *IET Gener., Transmiss. Distrib.*, vol. 5, no. 8, pp. 789–797, Aug. 2011.

[33] R. A. Howard, *Dynamic Probabilistic Systems: Semi-Markov and Decision Processes*. New York, NY, USA: Wiley, 1971.

[34] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for Web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.

[35] N. Mehta, S. Ray, P. Tadepalli, and T. Dietterich, "Automatic discovery and transfer of task hierarchies in reinforcement learning," *AI Mag.*, vol. 32, no. 1, pp. 35–50, 2011.

[36] C. Zeng, Z. Lu, J. Wang, P. C. K. Hung, and J. Tian, "Variable granularity index on massive service processes," in *Proc. ICWS*, Jun. 2013, pp. 18–25.

[37] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[38] W. Pan, B. Li, J. Liu, Y. Ma, and B. Hu, "Analyzing the structure of java software systems by weighted K-core decomposition," *Future Gener. Comput. Syst.*, vol. 83, pp. 431–444, Jun. 2018.

[39] W. Pan, H. Ming, C. Chang, Z. Yang, and D.-K. Kim, "ElementRank: Ranking java software classes and packages using a multilayer complex network-based approach," *IEEE Trans. Softw. Eng.*, early access, Oct. 8, 2019, doi: 10.1109/TSE.2019.2946357.

[40] J.-Y. Park and J.-H. Kim, "Incremental class learning for hierarchical classification," *IEEE Trans. Cybern.*, vol. 50, no. 1, pp. 178–189, Jan. 2020.

[41] P. Yildirim and D. Birant, "K-linkage: A new agglomerative approach for hierarchical clustering," *Adv. Electr. Comput. Eng.*, vol. 17, no. 4, pp. 77–88, 2017.

[42] K. Al-Sabahi, Z. Zuping, and M. Nadher, "A hierarchical structured self-attentive model for extractive document summarization (HSSAS)," *IEEE Access*, vol. 6, pp. 24205–24212, 2018.

[43] J. Nie, Y. Zhang, Z. He, S. Chen, S. Gong, and W. Zhang, "Deep hierarchical network for automatic modulation classification," *IEEE Access*, vol. 7, pp. 94604–94613, 2019.

[44] H. Yan, D. Gao, W. Su, C. H. Foh, H. Zhang, and A. V. Vasilakos, "Caching strategy based on hierarchical cluster for named data networking," *IEEE Access*, vol. 5, pp. 8433–8443, 2017.

[45] G. Karypis, E.-H. Han, and V. Kumar, "Chameleon: Hierarchical clustering using dynamic modeling," *Computer*, vol. 32, no. 8, pp. 68–75, 1999.

[46] M. Yang, S. Liu, K. Chen, H. Zhang, E. Zhao, and T. Zhao, "A hierarchical clustering approach to fuzzy semantic representation of rare words in neural machine translation," *IEEE Trans. Fuzzy Syst.*, vol. 28, no. 5, pp. 992–1002, May 2020.

[47] D. Schumm, D. Dentsas, M. Hahn, D. Karastoyanova, F. Leymann, and M. Sonntag, "Web service composition reuse through shared process fragment libraries," in *Proc. 12th Int. Conf. Web Eng.*, 2012, pp. 498–501.

[48] M. Sonntag and D. Karastoyanova, "Next generation interactive scientific experimenting based on the workflow technology," in *Proc. 21st IASTED Int. Conf.*, 2010, pp. 1–8.

RONG YANG received the Ph.D. degree from the School of Computer Science, Wuhan University, in 2015. He is currently an Associate Professor with the School of Computer Science and Technology, Hubei University of Science and Technology. His research interests include services computing and software engineering.

BING LI (Member, IEEE) received the B.E. degree from the Department of Computer Science, Huazhong University of Science and Technology (HUST), in 1990, the M.S. degree from HUST, in 1997, and the Ph.D. degree from the Computer Science School, HUST, in 2003. From 2003 to 2005, he has worked as a Postdoctoral Researcher with the State Key Laboratory of Software Engineering (SKLSE), School of Computer. He is currently a Professor with the Department of School of Computer, Wuhan University. He is a Senior Member of the China Computer Federation (CCF). He serves as a member with three CCF Technical Committee, such as open systems, software engineering, and service computing. In recognition of his contributions to scientific and technological progress, he received the Second Class Award of the Chinese National Scientific and Technological Progress Award in 2009, the First Class Award of the Hubei Province Scientific and Technological Progress Award in 2008 and 2011, and the First Class Award of the Natural Sciences Award of Chinese Ministry of Education in 2007.

ZHENGLI LIU is currently pursuing the Ph.D. degree with the School of Computer Science, Wuhan University, China. His current research interests include services computing and software engineering.