Reinforcement Learning for Interactive QoS-aware Services Composition

Pegah ALIZADEH[1], Aomar OSMANI[2], Mohamed Essaid KHANOUCHE[3,4], Abdelghani CHIBANI[3], Yacine AMIRAT[3]
[1] Léonard de Vinci Pôle Universitaire, Research Center, 92 916 Paris La Défense, France
[2] Laboratoire LIPN-UMR CNRS 7030. PRES Sorbonne Paris-cité, FRANCE
[3] LISSI Laboratory, UPEC University, Vitry-Sur-Seine, 94400, France
[4] Medical Computing Laboratory, Faculty of Exact Sciences, University of Bejaia, 06000, Algeria

***Abstract*—An important and challenging research problem in Web of Things (WoT) is how to select an appropriate composition of concrete services in a dynamic and unpredictable environment. The main goal of this work is to select from all possible compositions the optimal one without knowing *a priori* the users' Quality of Service (QoS) preferences. From a theoretical point of view, we give bounds on the problem search space. As the QoS user's preferences are unknown, we propose a Vector valued Markov Decision Process (VMDP) approach for finding the optimal QoS-aware services composition. The algorithm alternatively solves MDP with dynamic programming and learns the preferences via direct queries to the user. An important feature of the proposed algorithm is that it is able to get the optimal composition and at the same time limiting the number of interactions with the user. Experiments on a real-world large size data-set with more than 3500 web services show that our algorithm finds the optimal composite services with around 50 interactions with the user.**

***Index Terms*—Web of things, Services Composition, Reinforcement Learning, Quality of Services.**

I. Introduction

The Web of Things (WoT) opens the way for the development of new intelligent applications that can be used to provide innovative and valuable services in several domains, such as smart cities, smart homes, ambient assisted living and connected cars [11], [35]. One of the WoT challenges is to deal with services composition to ensure suitable services flexibility and customization. This is crucial to meet the evolving needs of users, which can be expressed through several parameters of quality of service (such as response time, throughput, availability, price, popularity, etc.).

Following the web service paradigm [4] services composition problems are specified as a workflow involving abstract and concrete services. A concrete service refers to an executable service, where an abstract service describes the abstract functionality of the concrete service. In environments such as the WoT, concrete services are web interfaces to objects functionalities, which are exposed by using functional and non-functional meta-data. At a given run time and for a given user, each abstract service can be achieved using several concrete services with the same functionality but possibly with different non-functional parameters, such as the Quality of Service (QoS). A composite service (workflow) specifies *what are the concrete services that should be selected for each abstract service and what is the best services permutation to satisfy the user's requirements in terms of the QoS*. An important and challenging research problem is therefore how to select an appropriate composition of concrete services in a dynamic and unpredictable environment [32].

Composing services to achieve complex workflows that match with users' requirements can be formulated as a mathematical composition, which is always associative and commutative [15]. To limit the combinatorial explosion of the composition, specific domain constraints must be taken into account: same time execution for several concrete services is allowed, abstract services are used to instantiate unlimited number of concrete services and each concrete service appears at most once in the composition. Therefore, the services composition problem can be viewed as the selection of concrete services offering the overall best QoS.

While state-of-the-art approaches find the services composition workflow by knowing the users' priorities in advance [20], [26], [36], these approaches are limited and static, therefore they can not handle the following cases: a) the presence of the user uncertainties before the service executions, b) unpredictable services manners such as service failures. We propose a *Reinforcement Learning* (RL) based approach to dynamically obtain the optimal service composition, according to the user's expectations on the quality of services. Markov Decision Processes (MDP) with unknown rewards enable us to tackle this challenge by optimizing rewards according to the answered queries for any user [1], [2]. Based on our approach and according to the given theoretical results, our algorithm minimizes the number of queries needed to select the best service composition. The proposed algorithm is validated with a large number of experiments on the most representative real-world data-set for our problem [50]. The experiments show that the proposed approach is able to provide the best composite service integrating dynamically user QoS preferences during the algorithm computation process. We also show some theoretical lower and upper bounds on the size of the search space of the composition problem.

The main contributions of this paper are the following:

- We propose the first approach in the literature able to provide a web services composition that explicitly takes into account the users' expectations and preferences on the QoS.
- Within the given framework, we show a theoretical lower bound for the number of possible services compositions by respecting the mentioned constraints.
- Our method is able to select an optimal services composition among all the possible permutations without knowing

a priori the users' preferences.
- We validate the algorithm on a real world data-set of web services.

The paper is organized as follows: the next section summarizes the related work on QoS-aware services composition. Section IV provides the problem formulation, together with the theoretical results on the search space. Section V formalizes the services composition problem as an MDP. Section VI details the proposed interactive Reinforcement Learning algorithm to deal with the services composition problem and section VII summarizes the experimental results. Finally, Section VIII gives the conclusion and some perspectives.

II. Motivating scenario

One the scenarios studied during the European project called web of objects (WoO) (https://itea3.org/project/web-of-objects.html), which motivated the present study, is the ad-hoc instantiation of applications in smart spaces populated by thousands see millions of devices. These applications are defined as compositions of abstract services, which will be turned into compositions of concrete services that better match the functional and nonfunctional requirements expressed by the designers and the operators of the applications. In a smart space, such as smart building or a smart city, a concrete service corresponds to a sensing or actuation functionality that can be supplied by a device. In such environments, an abstract service can be associated with a myriad of concrete services, which are exposed in the web by using different XML or JSON schemas and can invoked by the orchestrators by using different web services protocols such as HTTP, SOAP, XMPP, etc. In addition, a set of concrete services exposed by using the same schema and invoked by using the same protocol may be supplied in different quality levels that depend on the agreement between the services owners and the application operator or the availability of computing and communication resources in the run-time environment. Therefore, considering the heterogeneity and quality dynamics of such environments, humans cannot afford the manual composition of the concrete and instantiate the targeted application in an acceptable time. To argue, let us consider the following scenario. A smart city with a web of objects infrastructure that is designed to quickly instantiate applications for the needs of the monitoring of the security of people and facilities. This infrastructure may be composed of hundred may be thousands of connected cameras and may be millions of sensors measuring different parameters such as air quality, lighting, temperature, humidity, etc. Let's consider a typical scenario of an application that must be instantiated to follow the movements of crowds, to detect abnormal movements from the cameras. In addition, this application must on the other hand measure and verify from the other sensors if the air quality is within the standards and the air quality is not reaching toxic levels in the areas where crowds are moving. Every information or event that is handled in this scenario is defined as an abstract service. The application is composed by associating the abstract services with the concrete services exposed by the camera and sensors devices present in the same place. Given that the number of services matching the abstract service and the expected quality in the same location is may be huge, the composition must take into account only the concrete services offering the best overall quality to ensure that crowd tracking is as efficient as possible and reliable in the sense that there will be no abnormal event will be ignored due to concrete service quality downgrade.

III. Related works

Usually, acquiring user preferences is not easy in practice, this is due to the fact that the users are not always certain about their preferences until the service execution. Moreover, collecting user preferences can be complex in terms of time and memory [33]. For these reasons, most of the services composition studies in the literature are based on the assumption of knowing the user preferences on the QoS attributes in advance.

For what concerns services composition problems where the user preferences are known in advance, several works are proposed in the literature [6], [14], [24], [36]. Such works can be classified accordingly to the techniques used: graph search [12], [31], Pareto optimality [44], [46], constraints decomposition [3], meta-heuristic population [7], [17], [39], [42], [49], planning [8], [51], integer linear programming [47], [48], recommender system [9], [22], [25] and machine learning [13], [26], [29], [30], [40]. In the following, we will describe with more details some of the mentioned papers.

A services selection algorithm combining qualitative and quantitative QoS attributes is proposed in [39]. The qualitative attributes considered in this approach are provider, location and platform, whereas the quantitative attributes are response time, throughput, reliability and availability. The services selection problem is solved using a global optimization algorithm and a genetic algorithm.

RL techniques have been often used to deal with dynamic and uncertain environments in services composition problems. In [30], [40], the services composition problem in a dynamic environment is modeled as a Markov Decision Process and solved using a Q-learning method. In these approaches, the QoS attribute values are extracted through executing the services, while the optimal workflow of elementary service invocation actions is updated according to the change occurred in the environment. In [40], the authors use a normalized reward function combining the QoS attributes and the known user preferences on the attributes, while the approached proposed in [30] aims at maximizing the expected cumulative rewards which is on behalf of the satisfaction degree of the user's QoS constraints. In [26], two multi-objective MDP methods are proposed to handle services composition in uncertain and dynamic environments. The first approach addresses the single policy multi-objective composition scenarios, while the second approach addresses the multiple-policies composition scenarios. In the first approach, the user's preferences are given *a priori*, while in the second one the set of all suitable services compositions are represented as a convex hull of extreme services. Recently, several works have been proposed to use RL methods on large-scale services composition. For instance, [37] uses hierarchical Reinforcement Learning, [21], [38] utilize recurrent neural networks and deep RL approaches and [45] proposes an actor-critic method as a deep RL approach.

Many services composition approaches are based on machine learning technique different from RL. For instance [18], [23] propose services selection approaches in the context of QoS-aware services composition. In [23], the services selection with global QoS constraints is first formulated as a set-based optimization problem, while the k-Means clustering method is then used to find the composite service by maximizing the QoS value and satisfying the global QoS constraints. On the other hand, [18] presents an algorithm which first uses the k-means clustering to obtain clusters of candidates services, then, for each cluster it obtains the composite service in terms of QoS using a heuristic method.

To the best of our knowledge, very few approaches have been proposed for solving the services composition problem in interaction with the users dynamically and without knowing the user's preferences *a priori*. The closest existing approaches are [9], [10], [20], [26], [36]. In [9], the authors propose to extract the user preferences on the QoS attributes in advance, with a Location-aware Web service Recommendation (LoRec) system. This system first collects users' QoS observations related to the past usage experience of different Web services. The users are then clustered, according to their locations and their observed QoS, in order to propose personalized service recommendations to each user. In [20], the services composition problem in highly-dynamic environments is modeled as an uncertainly planning problem using a Partially Observable Markov Decision Process. A time-based RL approach is then proposed to find the composite service satisfying the user's requirements. The proposed approach does not require knowledge of complete information about services. It uses historical information to estimate the success probability of a services composition using results. Finally, [36] suggests a user-centric service composition to identify the subset of correlated services that best match the users' requirements from a designed ontology.

[10], [26] introduce a method for finding the list of all non-dominated workflows, regardless of the user preferences on the QoS attributes. Our approach goes beyond the identification of the non-dominated workflows: it proposes an approach for identifying one optimal workflow (among the set of non-dominated workflows) for each system user.

IV. Problem formulation

For the sake of simplicity, the problem of identifying a services composition able to satisfy the end-to-end user requirements can be defined as follows. Given a set of n abstract services $S = \{S_1, \ldots, S_n\}$ where each abstract service S_i can be implemented by n_i concrete services $\{S_{i1}, \ldots, S_{in_i}\}$. Each concrete service S_{ij} may be executed by a set of actors $\{a_1, \ldots, a_k\}$. An actor can be an end-user or any other entity for which the service is rendered. We denote the concrete service S_{ij} executed by the actor a_k, by S_{ij}^k. Furthermore, in several situations, the same service instance can be executed more than once over time by the same actor. In this work, we assume that the time horizon can be divided in time steps. The concrete service S_{ij} executed by the actor a_k at time t is denoted by $S_{ij}^k(t)$ and is called *execution*. Example 1 shows a sample of a real case data-base for the services composition problem.

Example 1: Table I is an extracted line from the real data-set [49], [50] used in our experiments. According to the aforementioned notations, this line should be denoted as $S_{19994,3104}^{97}(5)$. It represents for the abstract service 19994, the response time and throughput values of the concrete service 3104 executed by user 97 at time step 5.

TABLE I
An entry from the real data-set [49], [50] used in our experiments.

User	time	service	con. service	time resp.	throughput
97	5	19994	3104	0.238	0.773

Each service performs functions that serve the actors. To evaluate the quality of a service at the application level, a set of criteria is used, such as response time, throughput, reliability, availability, price, etc. [43], [48].

In Service Oriented Architectures (SOA), the orchestration process composes the existing services in order to create a new service having central control over the whole process [19]. In our context, the main part of the orchestration process is the abstract service composition. This composition process is guided by one main constraint: an abstract service can have a set of prerequisite abstract services that must be executed in order to make its execution possible. All such precedence constraints can be represented with a suitable oriented graph (see Figure 1). A path in the precedence graph represents a possible orchestration of abstract services.

We denote as *orchestration* of abstract services a sequence of subsets of the set of possible abstract services S, one for each time step (one abstract service can be included in at most one time step). Moreover, we will consider only *feasible* abstract service orchestrations, i.e., orchestrations where an abstract service is included in a set at time steps t only if all its prerequisite abstract services are included in the set of precedent time steps (see Example 2).

Example 2: Let S_1, S_2, S_3 and S_4 be four abstract services. Services S_2 and S_3 need outputs of the service S_1 and the service S_4 can be executed only after S_2. The abstract services precedence graph is given in Figure 1. The possible orchestrations with respect to this graph are: $(S_1, \{S_2, S_3\}, S_4)$, (S_1, S_2, S_3, S_4), (S_1, S_2, S_4, S_3), etc.

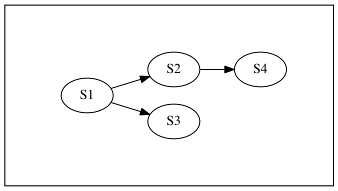

Fig. 1. The abstract services precedence graph of Example 2.

Each abstract service may be realized by several concrete services. When an abstract service orchestration is defined, we need to choose an appropriate concrete service that performs each corresponding service. In the context of the web of things, services composition can be seen as an invocation workflow or plan of the services, which are exposed by the objects through the web. In this paper, the possible concrete services organization for a given abstract service orchestration is represented by a concrete workflow. The concrete workflow must be coherent with the associated abstract service orchestration. Figure 2 gives an example of a concrete workflow for the orchestration given in Figure 1.

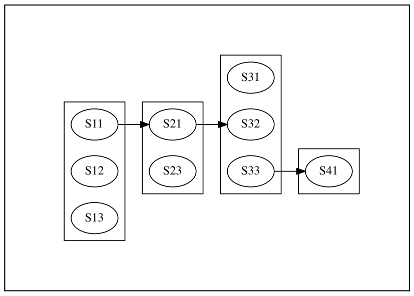

Fig. 2. The concrete workflow $(S_{11}, S_{21}, S_{32}, S_{41})$ is a possible organization to realize the orchestration given in Figure 1.

We define as *workflow* a concrete service concatenation of a given feasible abstract service orchestrations, i.e., a collection of subsets of concrete services obtained from a feasible abstract service orchestration after substituting each abstract service by one of its associated concrete services.

Let M be the number of all possible feasible abstract service orchestrations and ψ_k (for $k \in \{1, \ldots, M\}$) be the k-th possible orchestration. We are interested in defining an evaluation function $F(\psi_k)$ that associates a numerical value to all possible concrete service concatenations of ψ_k.

Let $\mathcal{Q} = \{q_1, \ldots, q_m\}$ be the set of all possible QoS criteria. The criteria can be applied both at the abstract and concrete levels. Without loss of generality, we limit the criteria to the concrete service level. In this case, $q_l(S_{ij})$ denotes the quality value of the q_l criteria for the S_{ij} concrete service. We suppose that the values of all criteria are normalized.

To obtain $F(\psi_k)$, it is necessary to select the best concrete service concatenation, according to the set of criteria in $\mathcal{Q}$, among the concrete services in the given ψ_k orchestration. In our context, the F function is assumed to be a linear combination of these criteria (see Example 3).

Without loss of generality, the service selection problem can be defined as follows:

$$Find\ the\ orchestration\ \psi_k\ with\ the\ best\ value\ of\ F(\psi_k).$$

Example 3: Let us consider a problem with three abstract services $\{S_1, S_2, S_3\}$. Assume that this problem has a single possible orchestration $\psi_1 = (S_1, S_2, S_3)$ such that ψ_1 represents a serial orchestration (we execute first concrete services of S_1, then S_2 and end with S_3). Let us consider a problem with two evaluation criteria $\mathcal{Q} = \{q_1, q_2\}$, for instance, throughput and service time response. To each abstract service S_i we associate a value for q_{i1} and a value for q_{i2}. The value of the orchestration is defined as a linear combination between these criteria: $\sum_{i=1}^{3}(w_1 q_{i1} + w_2 q_{i2})$ where the coefficients w_i (one for each abstract service S_i) are given by the actor (user).

From an operational point of view, the evaluation of the concrete services quality is estimated over their executions through the time. In the above example, the users' preferences are presented as weight values. In practice, the weights are not given *a priori*. In Section VI we propose an RL method based on MDPs for learning QoS attributes weights representing users' predefined preferences.

A. On the global search space size

The purpose of this section is to enumerate the number of end-to-end concrete service combinations (workflows) in order to evaluate the quality of service of each combination and then to select the best one according to a given set of user/actor preferences. To the best of our knowledge, there is no existing work in the literature dealing with this enumeration problem. In the rest of the section, we give the size of the search space for some special cases and for the generic case, where no constraints are imposed on the structure of the orchestration. Those sizes can be viewed respectively as lower and upper bounds on the search space for any service combination problem.

Let us consider the generic case of a services composition problem with n abstract services $S_1, \ldots, S_n$, such that each abstract service S_i can be realized by exactly m possible concrete services. In the following subsections, we analyze how the size of the search space changes according to the different constraints on the service order and concurrency.

1) Total order (sequential) abstract services: no explicit constraint on the order of execution of the services is imposed but at each time step it is possible to execute only one abstract service. Since each S_i can be realized by m possible concrete services, for a given fixed ordering of abstract services, the possible number of realizations for each order is m^n. If there is no additional constraint between abstract services, the number of abstracts services permutations is $n!$. In this case, the total number of workflows is $n!m^n$.

2) Parallel abstract services: all abstract services are executed at the same time. In this case, there exists only one possible combination between abstract services. The only degree of freedom is to choose which concrete service to associate to each abstract service. For this reason, there exists m^n possibles realizations.

3) General case: for each time step it is possible to execute simultaneously any number (up to n) of abstract services. We recall that $S(n, k)$, the Stirling number of the second kind, counts the number of ways for partitioning an n-element set into k subsets[1] [16]. On the other hand, k subsets can be ordered in $k!$ different ways. This leads to a total of $S(n, k)k!$ different orchestrations of time length exactly k. This means that all abstract services in each set should be executed at the same time. Therefore, the total number of orchestrations is:

$$\sum_{k=1}^{n} S(n,k)k! = \sum_{k=1}^{n}\sum_{l=0}^{k}(-1)^l \binom{k}{l}(k-l)^n. \quad (1)$$

For any given orchestration with n abstract services and m concrete services, we have m^n total number of workflows.

[1]Several equivalent formulas are available to compute the Stirling number of the second kind, in this case we use $S(n,k) = \frac{1}{k!}\sum_{l=0}^{k}(-1)^l\binom{k}{l}(k-l)^n$.

Notice that each abstract service is associated to one and only one executed concrete service at each time step. Consequently, to obtain the total number of workflows, we need to multiply the quantities given by the Equation (1) with the total number of workflows for each orchestration:

$$m^n \sum_{k=1}^{n} S(n,k)k!. \tag{2}$$

We finally notice that the situation is slightly different if we impose to associate the same concrete abstract to all the abstract services at the same time step. If this is the case we have a total number of workflows equal to:

$$\sum_{k=1}^{n} S(n,k)k!m^k. \tag{3}$$

The results showed in this section give an idea of the combinatorial explosion one could expect if no precedence is imposed on the orchestrations. For this reason, we restrict our study to sequential orchestrations.

V. Services composition as an MDP

Markov Decision Processes (MDPs) are often used to model sequential decision problems [27].

In this section, we describe how to solve the services composition problem via MDP models. We are looking for an optimal QoS-selection strategy satisfying the user's requirements in terms of QoS attributes. Since the user's preferences with respect to QoS attributes are unknown, we use a partially known MDP model, and more particularly, a Vector-valued MDP (VMDP) model [2].

A. Vector-valued Markov Decision Process

The following definitions are necessary to define our VMDP model.

Definition 1: A *discrete-time Markov Decision Process* (MDP) [1] is defined by a tuple $(T, S, A, P_t(.|s,a), r_t)$ where:

- $T = 0, \dots, N$ are the decision time steps at which the decisions are made[2].
- S is a finite set of states.
- $A(s)$ is a finite set of actions that the agent can select in a state $s \in S$.
- $P_t(s'|s,a)$ is the State Transition Probability Distribution, it encodes the probability at time t of going to state s' when the agent is in state s and executes action a.
- $r_t : S \times A \longrightarrow \mathbb{R}$ is the Reward Function, $r_t(s,a)$ quantifies the value gained when executing action a in state s at time step t.

Definition 2: A *Decision rule* D_t is a function depending on time step t that defines what action $D_t(s) \in A(s)$ at time t the agent should select. By assuming N number of time steps, we define *policy* $\pi = (D_1, \dots, D_{N-1})$ as a sequence of $N-1$ decision rules.

[2]Time steps can be days, hours, minutes or any time interval.

A policy is *stationary* if the decision rule for all time steps are the same i.e. : $\forall\, t \in \{1, \dots, T\}\ D_t = D$. A solution for an MDP is a policy $\pi : S \longrightarrow A$ that associates an action to each state. Normally, policies are evaluated by a *value function* $v^\pi : S \longrightarrow \mathbb{R}$. The value function is computed recursively as follows:

$$v_N^\pi(s) = r_N(s, \pi(s))\ \ \forall s \in S_T \tag{4}$$

where S_T is the set of terminal states as a subset of all states. For the rest of time steps $t < T$, the value function is defined as:

$$v_t^\pi(s) = r_t(s,\pi(s)) + \gamma \sum_{s' \in S} P_t(s'|s,\pi(s)) v_{t+1}^\pi(s') \tag{5}$$

where γ is a discount factor $(0 < \gamma \leq 1)$. $v_t^\pi(s)$ represents the expected gain in the future from state s, at time step t if the policy π is executed. Therefore, the final value of a policy is given by $v^\pi = v_0^\pi$. Let π and π' be two policies, in the following, we indicate $\pi \succeq \pi'$ if the policy π is better than policy π':

$$\pi \succeq \pi' \Leftrightarrow \forall s \in S\ v^\pi(s) \geq v^{\pi'}(s)\ . \tag{6}$$

Equation 6 indicates that policy π is better than policy π' if its value is higher than the value of π' for all states. π^* is an *optimal policy* if $\pi^* \succeq \pi$, for all feasible policies π.

What remains to show is how to find an optimal policy. To find the value of the optimal policy, we can use a dynamic programming, namely the *Bellman equation* [27]:

$$v_t^*(s) = \max_{a \in A(s)} \left\{ r_t(s,a) + \gamma \sum_{s' \in S} P_t(s'|s,a) v_{t+1}^*(s') \right\}. \tag{7}$$

For extracting the optimal policy, we need to define a *Q-value function* on state s and action a at time step t as:

$$Q_t(s,a) = r_t(s,a) + \gamma \sum_{s' \in S} P_t(s'|s,a) v_{t+1}^*(s'). \tag{8}$$

The optimal policy selects action a^* at state s and stage t as follows:

$$a_t^* \in \text{argmax}_{a \in A(s)} \{Q_t(s,a)\} \quad \text{for } t = 1 \dots N-1. \tag{9}$$

In the following, we present a generalisation of MDPs where the rewards are vectors instead of scalars:

Definition 3: [40] A *discrete-time Vector-valued MDP* (VMDP) is defined by a tuple $(T, S, A, P_t(.|s,a), \bar{r}_t)$ where $T, S, A, P_t(.|s,a)$ are the same as in Definition 1.

- $\bar{r}_t : S \times A \longrightarrow \mathbb{R}^d$ is a vector valued reward function defined as $\bar{r}_t(s,a) = (r_{1t}(s,a), \dots, r_{dt}(s,a))$

In the following section, we will see that VMDPs are particularly suited for services composition context, because each executed service has several quality values, such as response time, throughput, price, availability, etc.

B. Services composition as a discrete-time VMDP

In this section, we adapt VMDPs in the context of services composition problems. The solution of such MDPs gives the optimal service composition. there are basically two reasons for using an MDP:

- MDPs make it easy and natural to model services composition problems: every salient aspect present in the decision-making process relating to the services composition problem has an evident counterpart in an MDP context. The division of an MDP into states and actions is particularly suitable for representing the division of a composition service problem into abstract services and concrete services. Furthermore, an optimal policy in an MDP reduces to chose which actions to take at each state, while the optimal workflow reduces to find the best concrete service for each abstract service.
- MDPs can be easily integrated into a dynamic environment with uncertainty. In this paper we use part of the results presented in [2], [41] on how to efficiently learn the users' prefrences. These results are based on the combined use of MDP and reinforcement learning techniques.

From now on, to be coherent with the services composition terminology, we will use the term *workflow* as a synonym of policy, *abstract service* as a synonym of state, *concrete service* as a synonym of action and quality of service as a reward value.

Definition 4: A *concrete service* S_{ij} can be described by several *functional* and *non-functional* properties.

- Functional properties.
  Each functional property is described through a transaction function, $\texttt{Action}(S_{ij})$ that takes an input data vector $\texttt{InputData}(S_{ij})$ to produce an output data vector $\texttt{OutputData}(S_{ij})$.
- Non-functional properties.
  Each non-functional property consists of a vector of QoS attributes $Q(S_{ij})$, a set of quality of experience criteria (QoE), and other aspects about the service such as energy consumption and the context of use.

Definition 5: An *abstract service* $S_i = \{S_{i1}, \dots, S_{in_i}\}$ is a class of n_i concrete services with similar functional properties, i.e., they have the same input and output data vectors, but their non-functional properties can be different.

In the rest of this section, we will explain how various classes of abstract services, each one associated to several concretes services, can be modeled as a VMDP.

Definition 6: A *VMDP-Service Composition* (VMDP-SC) is defined by a tuple $(T, AS, CS, P_t(.|as, cs), \bar{r}_t)$ similar to Definitions 1 and 3:

- $T = 0, \dots, N$ are the decision time steps.
- AS is a finite set of abstract services.
- $CS(S_i)$ is a finite set of executable concrete services that the agent can use for a chosen abstract service $S_i \in AS$. We have that $CS(S_i) = \{S_{i1}, \dots, S_{in_i}\}$
- $P_t(S_j|S_i, S_{ik})$ is the probability of executing concrete service S_{ik} for abstract service S_i and entering to abstract service S_j.
- $\overline{QoS}_t : AS \times CS \longrightarrow \mathbb{R}^d$ is the vector valued reward function. $\overline{QoS}_t(S_i, S_{ik})$ represents a vector of QoS attribute values after invoking S_{ik} in S_i at time step t. For d attributes, we obtain $\overline{QoS}_t(S_i, S_{ik}) = (qos_{1t}(S_i, S_{ik}), \dots, qos_{dt}(S_i, S_{ik}))$.

The terminal states, defined as AS_T, do not have any executable services.

Definition 7: A *workflow* $\pi : AS \longrightarrow CS$ is a function that defines which concrete service should be invoked for each abstract service in order to give the best trade-off among multiple QoS attributes.

Since the rewards used in an MDP-SC are vectorial, also the value function $\bar{v}$ and the Q-value function $\bar{Q}$ are vectorial:

$$\bar{v}_t^{\pi}(S_i) = \overline{QoS}_t(S_i, \pi(S_i)) + \gamma \sum_{S_j \in AS} P_t(S_j|\pi(S_i), S_i)\bar{v}_{t+1}^{\pi}(S_j). \tag{10}$$

Thus, comparing two workflows boils down to comparing two vectors. The optimal workflow can differ from user to user according to their preference with respect to the d value qualities of services. We make the assumption that each user assign a different weight $\bar{w_i}$ to each QoS attribute. The Simple Additive Weighting (SAW) technique [28] is used to obtain a single QoS value as a weighted linear combination of the QoS attributes. Let $\overline{W} = (w_0, \dots, w_d)$ be the set of normalized weights (i,e., $\sum_{i=1}^{d} w_i = 1$) given by a user, the aggregated QoS becomes:

$$QoS_t(S_i, S_{ij}) = \overline{W} \cdot \overline{QoS}(S_i, S_{ij}) = \sum_{k=1}^{d} w_k qos_{kt}. \tag{11}$$

If the user preferences on QoS attributes are given, the optimal workflow can be therefore computed using SAW techniques. However, determining appropriate weights for QoS attributes requires the knowledge of the user preferences, which is often not obvious to obtain in practice. Even if the user preferences have been obtained, setting accurately these weights remains a problem. For instance, it is hard to decide the weight of an attribute such as response time as 0.2 or 0.21, which appears no big difference yet it can affect the result of the QoS optimal composition [10]. We assume that $\overline{W}$ is unknown, and we try to find the best workflow by querying the users when it is necessary.

The basic idea is to deduce the user weights by comparison: we present to the users two alternative Qos vectors, and we assume that he is always able to find which of the two he prefers. In order to compute the optimal workflow in the service composition, it is required to compare the vector value function of two different workflows.

To compare workflow vector values with each other, we consider first the $d-1$ dimensional polytope $\mathcal{W}$ that represents the set of all possible values for $\overline{W}$:

$$\mathcal{W} = \{(w_1, \dots, w_d) \in \mathbb{R}_+^d \mid \sum_{i=2}^{d} w_i \leq 1 \text{ , } w_1 = 1 - \sum_{i=2}^{d} w_i\}. \tag{12}$$

Three methods can be used to compare vector values associated to the workflows [41]. Assume $\bar{v}^{\pi_a} = (a_1, \dots, a_d)$ and $\bar{v}^{\pi_b} = (b_1, \dots, b_d)$ are two vector values associated to two workflows π_a and π_b:

- *Pareto comparison*:

$$\bar{v}^{\pi_a} \succeq_P \bar{v}^{\pi_b} \Leftrightarrow \forall\, i\ a_i \geq b_i \tag{13}$$

- *K-dominance comparison*: $\bar{v}^{\pi_a}$ is preferred than $\bar{v}^{\pi_b}$ if it is better for any $\bar{w}$ in the polytope $\mathcal{W}$:

$$\bar{v}^{\pi_a} \succeq_K \bar{v}^{\pi_b} \Leftrightarrow \forall\, \overline{W} \in \mathcal{W},\ \overline{W} \cdot \bar{v}^{\pi_a} \geq \overline{W} \cdot \bar{v}^{\pi_b} \tag{14}$$

- *Query to the user*: $\bar{v}^{\pi_a} \succeq_q \bar{v}^{\pi_b}$ if the user prefers workflow $\bar{v}^{\pi_a}$ over $\bar{v}^{\pi_b}$. This check is a sort of "last resort": if none of the previous methods allow to eliminate one of the two vectors, the only option is to ask directly to the user.

It is possible to show that checking if a K-dominance exists between two workflow vector values can be done by solving an ad hoc linear programming problem [41]: $\bar{v}^{\pi_a} \succeq_K \bar{v}^{\pi_b}$ is true if there is a non-negative solution to the following linear program:

$$\begin{array}{ll} \min & \overline{W} \cdot (\bar{v}^{\pi_a} - \bar{v}^{\pi_b}) \\ \text{s.t.} & \overline{W} \in \mathcal{W}. \end{array} \tag{15}$$

We noticed that

$$\bar{v}^{\pi_a} \succeq_K \bar{v}^{\pi_b} \not\Longleftrightarrow \bar{v}^{\pi_b} \succeq_K \bar{v}^{\pi_a}$$

therefore, to check if a dominance exists between the two vectors we need to solve two separate linear programs. In the remaining of this paper, we explain how to find the optimal workflow that gives the best trade-off among multiple QoS criteria, satisfying the user requirements in terms of QoS only querying a few times.

VI. Interactive Reinforcement Learning algorithms for the services composition problem

After modeling the services composition environment as a VMDP-SC, we are interested in finding a solution for our model, i.e., how to compute the optimal workflow respecting the users' preferences on the QoS attributes. In this section, we introduce an algorithm with this aim, namely *Interactive Value Iteration for Services Composition* (IVI-SC). It is possible to use interactive value iteration methods for finding the optimal workflow by learning user's preferences weights dynamically [2], [41].

We assume that a VMDP-SC with finite discrete-time is given. The services can be invoked in $T+1$ number of discrete time steps: $\{0, \dots, T-1\} \cup \{T\}$ where T is a final empty time step. Since the MDP-SC objective is finding the workflow that maximizes a measure of long-run expected $\overline{Q}$ vector values, we propose a backward induction method to solve the Bellman equation given in Equation 7 to obtain the optimal policy/workflow, see Equation 9. Our approach is formalized in Algorithm 1.

In the iterative Algorithm 1, first we assign a zero vector to the set of states (abstract services) at time step T. For

```
Data: VMDP-SC($T, AS, CS, P_t, \bar{r}_t$), a $\mathcal{W}$ polytope of
      user weights.
Result: The optimal service selection workflow for the
        given user.
$t \longleftarrow T$
$\pi_{\text{best}} \longleftarrow$ choose random policy
$\bar{v}_T(s_T) \longleftarrow (0, \dots, 0)\ \forall s_T$ at time $T$[3]
$\mathcal{K} \longleftarrow$ set of constraints on $\mathcal{W}$
while $t \geq 0$ do
    $t \longleftarrow t-1$
    for each $S_i(t) \in AS_t$ do
        best $\longleftarrow (0, \dots, 0)$
        for each $S_{ij}(t) \in CS(S_i(t))$ do
            $\bar{v}_t(S_i(t)) \longleftarrow$
            $\overline{QoS}(S_i(t), S_{ij}(t)) + \sum_{S_k(t+1)} P_t(S_k(t+1)|S_i(t), S_{ij}(t))\bar{v}_{t+1}(S_k(t+1))$
            ( best , $\mathcal{K}$) $\longleftarrow$ getBest(best, $\bar{v}_t, \mathcal{K}$)
            $\bar{v}_t(S_i(t)) \longleftarrow$ best
        end
        if best = $\bar{v}_t(S_i(t))$ then
            $\pi_{\text{best}(S_i(t))} \longleftarrow S_{ij}(t)$
        end
    end
end
return $\pi_{\text{best}}$
```

Algorithm 1: *Interactive Value Iteration for Services Composition*. The algorithm computes the optimal workflow respecting the user preferences on QoS attributes.

```
Data: Two vectors $\bar{v}$ and $\bar{v}'$, polytope constraints $\mathcal{K}$.
Result: Finds the more preferred between $\bar{v}$ and $\bar{v}'$ for the
        given user.
if paretodominates($\bar{v}, \bar{v}'$) then
    return $(\bar{v}, \mathcal{K})$
end
if paretodominates($\bar{v}', \bar{v}$) then
    return $(\bar{v}', \mathcal{K})$
end
if Kdominates($\bar{v}, \bar{v}', \mathcal{K}$) then
    return $(\bar{v}, \mathcal{K})$
end
if Kdominates($\bar{v}', \bar{v}, \mathcal{K}$) then
    return $(\bar{v}', \mathcal{K})$
end
$(\bar{v}_{\text{best}}, \mathcal{K}) \longleftarrow$ query($\bar{v}, \bar{v}', \mathcal{K}$)
return $(\bar{v}_{\text{best}}, \mathcal{K})$
```

Algorithm 2: *getBest*.

each abstract service $S_i(t)$ (at time t) given in the MDP-SC, the algorithm selects the best concrete service among the all available ones. These actions (concrete services) are dependent on various time steps, for instance, the possible actions of the $S_i(t)$ service at times step t can be different from the possible actions for time step $t+1$. In the finite horizon time (our case), the iteration continues until either this difference becomes small enough or the horizon time steps finish.

Since the quality of services is the d dimensional vectors, solving Equation 7 and finding the maximum among the vectors

```
Data: $\bar{v}, \bar{v}', \mathcal{K}$.
Result: Queries the comparison between $\bar{v}$ and $\bar{v}'$, to the
        user and modifies $\mathcal{K}$ according to his response.
Build query $q$ for the comparison between $\bar{v}$ and $\bar{v}'$
if the user prefers $\bar{v}$ to $\bar{v}'$ then
    return $(\bar{v}, \mathcal{K} \cup \{(\bar{v} - \bar{v}') \cdot \overline{W} \geq 0\})$
end
else
    return $(\bar{v}', \mathcal{K} \cup \{(\bar{v}' - \bar{v}) \cdot \overline{W} \geq 0\})$
end
```

Algorithm 3: *query*: It asks to the user about his preferences on the existed QoS.

is not obvious. For this reason, we use the Algorithm 2 *getBest* that applies the three comparison methods presented at the end of Section V. The `getBest` function receives two d dimensional vectors with the $\mathcal{W}$ polytope constraining the user weight preferences on the quality of services. If the Pareto comparison can not find the greater vector, The K-dominance comparison is used to find the most preferred vectors. Otherwise, the `query` function should be called (given in Algorithm 3). The user's response to the comparison between the two given vectors, adds a new constraint to the $\mathcal{W}$ polytope.

Algorithm 1 finally finds the optimal policy/workflow or services composition for the given system VMDP-SC and returns back the optimal policy π_{best}. Notice that the condition best $= \bar{v}_t(S_i(t))$ in Algorithm 1 checks if the best selected concrete service for $S_i(t)$ has been changed regarding the previous iteration. If it was, the optimal concrete service should be replaced by the concrete service $S_{ij}(t)$ which generates a better vector value for $S_i(t)$.

The proposed algorithm is exact, and its complexity is polynomial w.r.t three parameters: the number of abstracts services forming the composition, the number of candidates concretes services per each abstract services, and the number of QoS attributes. Assuming $|AS|$ is the set of all abstract services and $M = \max_{i,t} CS(S_i(t))$ is the maximum number of abstract services in each time step t and each abstract service S_i. In order to compute the best QoS vector in each inner iteration, the *Best* algorithm tests the Pareto dominance and K-dominance comparison twice in the worst case which are polynomial w.r.t d. Where d is the number of attributes for quality of services and any K-dominance (LP) can be solved in polynomial time. Finally, we suppose that the time necessary to ask a preference to the user is constant. Therefore, the complexity of the algorithm is $O(Md|AS|)$.

VII. Performance evaluation

We evaluate our methods on a public available data-set containing two QoS attributes: throughput and response time. These are the records between 339 users and 5825 web services distributed worldwide [49], [50]. The data-set also includes some information about user features and service features such as countries, autonomous systems, IP dresses, latitude and longitude. In the studied data-base, 142 users execute various web services in different time steps. The information is available only for 64 time steps. In this section, we first explain how to model the data-set as a VMDP-SC and then we examine the performances of our algorithm on the data-set.

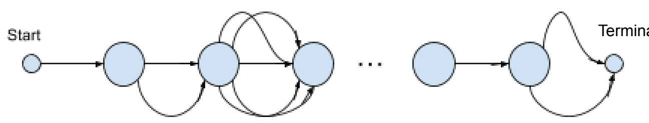

Fig. 3. A sequential form of abstract service connection. The start state is an empty state and it is connected to the first selected abstract service in the model. While the terminal state is an empty state to indicate that we have a finite horizon VMDP-SC.

A. Data-set as a VMDP-SC

The main issue in implementing IVI-SC (Algorithm 1) on any data-base is how to model the given data-set as a vector-valued MDP. In the supported data-set [49], [50], there are several text files, including wslist.txt, userlist.txt, rtdata.txt and tpdata.txt. The wslist.txt is our source file for extracting a list of web services and their related abstract services. In our data-set, we consider the web services as the concrete services. The userlist.txt file includes some information about users of different web services. The two other tpdata.txt and rtdata.txt files include the throughput and response time values respectively on various web-services executed by 142 users. This means that any web service invoked by a user has two parameters for measuring the service quality: throughput (mega bits per second, Mbps) and response time (second, s).

The studied data-base [49], [50] is generated in practice by observing various users utilizing enormous number of web services. After cleaning the data-base and extracting all web services and their related abstract services from the wslist.tx file, and getting the web services qualities from two files tp.txt and rt.txt, we have a VMDP-SC with the following parameters (see Definition 6) :

- 64 time steps.
- 744 abstract services.
- 3551 total number of concrete services (in our case web services).
- The transition function is not given directly in the data-base. Several models can be used to define the possible orders and concurrences among the abstract services (see Section IV. In this case, we adopt a sequential structure (see Figure 3).
- The $\overline{QoS}_t$ function comes from the extracted data on web services and their two qualities: response time and throughput.

To demonstrate the efficiency of our approach in calculating the optimal workflow, we study our method on a common state of the art model: the sequential model.

For the sequential model (see Figure 3) the start state is an empty state connected to the first selected abstract service in the model. On the other hand, the terminal state is an empty state that indicates that the MDP has a finite horizon. The sequential order on abstract services can be defined in any order. In our model, the order is randomly selected once, to fix the MDP model. For any time step $t \in \{0 \ldots 63\}$, the

TABLE II

AN EXAMPLE OF 2 DIMENSIONAL WEIGHT VECTORS (THROUGHPUT AND RESPONSE TIME) FOR 5 USERS WITH PREFERENCES ON THE ATTRIBUTES. THE GOAL OF ALGORITHM 1 IS TO LEARN SUCH A VECTORS FOR EACH OF THE FIVE USERS.

user	weight vector
	[throughput, response time]
$\overline{W}_0$	$[0.320, 0.680]$
$\overline{W}_1$	$[0.857, 0.143]$
$\overline{W}_2$	$[0.170, 0.830]$
$\overline{W}_3$	$[0.645, 0.355]$
$\overline{W}_4$	$[0.472, 0.528]$

transition probability $P_t(S_j(t+1)|S_i(t), S_{ik}(t))$ is 1, if web service S_{ik} is invocable for a given abstract service $S_i(t)$ according to our data-base and the abstract service $S_j(t+1)$ is the next demanded service in our selected sequential MDP model, otherwise $P_t(S_j(t+1)|S_i(t), S_{ik}(t)) = 0$.

B. *Model the web service data-set as VMDP*

To evaluate Algorithm 1 we consider the complete data-set, i.e., we keep all the quality services executed by all 142 users. Modelling such a huge size data-base as a VMDP-SC and implementing the IVI-SC algorithm on it is a challenging task.

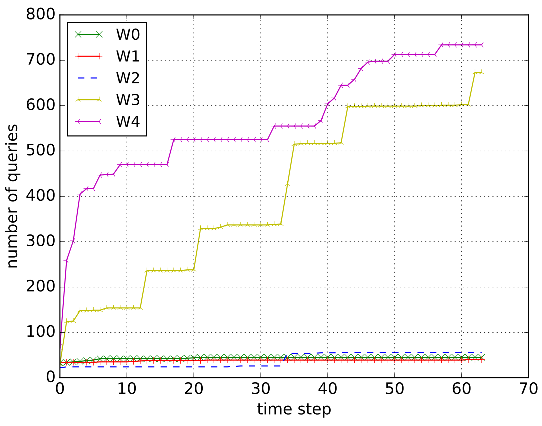

Fig. 4. This figure shows the number of queries proposed to the user during each time step. The weight preferences are based on Table II.

In order to evaluate our algorithm performance, we analyze the results for 5 different users with various preferences on the service qualities (response time and throughput). Our tested user weights vectors ($\overline{W}$) on service qualities are given in Table II. Notice that the weight preferences on the quality of services are "unknown" to our algorithm. They are used to simulate the user's behavior.

Figure 4 shows how the interactive value iteration algorithm communicates with users during the 64 time steps. Since the user weight preferences are unknown to the algorithm, it is needed to query them in the required situations. According to the figure, the Algorithm 1 finds the optimal workflows after asking less than 56 queries to the users with preferences weights $\overline{W}_0$, $\overline{W}_1$ and $\overline{W}_2$. On the other hand, for the users with weight preferences $\overline{W}_3$ and $\overline{W}_4$, the algorithm queries a

TABLE III

IT DEMONSTRATES HOW THE COMPUTED WORKFLOW FROM ALGORITHM 1 IS DIFFERENT FROM THE IDEAL WORKFLOW FOR $\overline{W}_3$.

Services	number of WS	IVI	Exact	Services	number of WS	IVI	Exact
AS14280	2	566	567	AS9829	2	1559	1565
AS6360	3	3723	3725	AS29650	2	1583	1586
AS4766	9	2226	2237	AS4768	2	2053	2054
AS3288	2	1622	1623	AS42695	2	2503	2504
AS20284	8	1547	1554	AS42915	3	1588	1590
AS852	7	533	4126	AS22047	2	612	613
AS14415	13	4120	4119	AS50517	3	2304	2306
AS16265	2	2018	2665	AS5050	4	3584	3586
AS13301	6	1382	1487	AS13041	24	2375	2368
AS15806	7	1566	1576	AS47720	3	1569	1571
AS11955	12	3882	3888	AS8075	11	12	4183
AS33821	2	2256	2258	AS23148	5	583	4233
AS29944	3	4223	4224	AS35041	2	2514	2533
AS20773	6	1348	2548	AS6830	2	1579	1584
AS17819	4	2325	2329	AS7050	4	4198	4201
AS27437	3	4047	4050	AS48347	3	2271	2273
AS19855	3	3652	3653	AS760	3	90	92
AS23650	22	625	853	AS8737	13	1932	1940
AS87	5	4111	4112	AS9116	2	1615	1630
AS8551	4	1620	1634	AS19875	5	547	549
AS15366	4	1244	1246	AS55481	4	44	47
AS5603	2	2345	2349	AS39418	10	959	970
AS31727	4	1582	2922	AS5786	3	2212	2214
AS9848	4	2232	2233	AS29097	3	2538	2540
AS29076	5	2275	2293	AS29951	8	3856	3860
AS4808	10	670	770	AS32577	10	3640	3638
AS17431	2	752	753	AS32475	3	3728	3729
AS2819	5	918	946	AS3786	13	2215	2225
AS32613	3	93	94	AS27030	3	4467	4469
AS11305	3	3688	4046	AS24969	2	999	1004
AS15670	3	1982	1984	AS1251	10	175	194
AS156	2	4179	4180	AS12714	2	2279	2280
AS16095	3	998	1016	AS16245	2	1000	1018
AS15290	6	536	574	AS8542	6	2081	2090
AS31815	11	4236	4414	AS34235	4	1141	1208
AS8151	3	1918	1919	AS9308	5	748	801
AS4812	11	760	862	AS5409	2	1440	1441
AS3389	4	4109	4106	AS8928	2	1202	3045
AS81	5	509	510	AS3561	25	3647	3579
AS6785	2	955	956	AS8659	4	2543	2545
AS45061	12	648	649	AS44249	10	2171	2173
AS702	16	156	1147	AS701	19	3493	3491
AS22070	2	3661	3662	AS11426	50	3207	3163
AS1128	5	1964	1965	AS2611	10	132	140
AS25518	2	1774	1775	AS14584	4	3894	3896
AS224	7	2115	2075	AS16339	2	2911	2912
AS15348	3	3731	3732	AS16237	4	1959	1960
AS31827	3	4341	4342	AS32	9	4387	4388
AS30190	2	4408	4409	AS9143	8	1974	1991
AS8220	15	160	1182	AS9318	6	2229	2243
AS4230	8	167	199	AS22489	3	210	211
AS43200	3	2087	2089	AS3307	3	2091	2120
AS3301	6	2506	2522	AS9811	3	695	697
AS34779	9	2332	2335	AS73	4	3911	3914
AS553	2	1448	1466	AS12859	7	125	2024
AS15756	3	2283	2298	AS15879	4	1961	2029
AS10929	4	4065	4066	AS15085	2	554	555
AS21844	13	4163	4161	AS12731	4	1361	1378
AS237	6	3523	816	AS195	13	4188	4027
AS1226	3	4010	4012	AS1221	2	43	79
AS8358	2	1521	1522	AS680	44	1398	1401
AS8	3	3667	3668	AS3316	4	2260	2261
AS4837	29	680	791	AS27617	3	4445	4446
AS8190	3	4067	4068	AS8904	5	2264	2266
AS43541	3	913	915	AS2519	4	1857	1859
AS42949	2	1994	1995	AS18125	5	1861	1862
AS2200	28	1120	1108	AS26228	6	2077	2080
AS19024	8	3924	3929	AS11343	3	4444	4442
AS40142	7	3740	3767	AS1103	3	1989	1992
AS33491	3	3681	3682	AS33494	9	3700	3707
AS12695	3	2277	2291	AS9120	2	1011	1028
AS41635	3	2250	2252	AS21309	3	1792	1794
AS8342	5	950	2269	AS12306	9	736	4317
AS12301	3	1512	1514	AS9338	9	2040	2046
AS10694	9	4123	4354	AS3778	7	3839	3838
AS6983	50	3214	3234	AS43220	2	980	981
AS39111	3	964	1319	AS3614	4	3539	3541
AS17477	9	18	2062	AS29134	2	939	940
AS43892	2	1532	1533	AS14335	2	3596	3597
AS8972	3	1343	1344	AS3292	36	2068	2478
AS8803	3	2554	2555	AS12874	23	1711	1821
AS1930	2	2204	2208	AS210	13	3709	3712
AS24958	3	947	948	AS15083	28	863	864
AS6400	4	1030	1031	AS1205	4	108	111
AS8508	3	2168	2169	AS11798	18	3822	3824
AS36017	3	4473	4474	AS55454	3	2031	2032
AS36850	4	3655	3657	AS40619	4	3526	3525
AS16276	8	1191	1201	AS5537	2	2288	2295
AS13367	10	3750	3738	AS12312	4	1365	1367

TABLE IV
IT DEMONSTRATES HOW THE COMPUTED WORKFLOW FROM $\overline{\text{ALGORITHM}}$ 1 IS DIFFERENT FROM THE IDEAL WORKFLOW FOR $\overline{W}_4$.

Services	number of WS	IVI	Exact	Services	number of WS	IVI	Exact
AS42695	2	2503	2504	AS50517	3	2304	2306
AS16265	2	2018	2665	AS15806	7	1566	1576
AS8075	11	3830	4183	AS23148	5	583	4233
AS35041	2	2514	2533	AS48347	3	2271	2273
AS760	3	90	92	AS39418	10	963	970
AS4808	10	670	770	AS32475	3	3728	3729
AS32613	13	99	94	AS156	2	4179	4180
AS25260	5	1268	1270	AS16245	2	1000	1018
AS8542	6	2081	2090	AS5409	2	1440	1441
AS3561	25	3647	3579	AS2611	10	132	135
AS14584	4	3894	3896	AS16237	4	1959	1960
AS10929	4	4065	4066	AS12731	4	1361	1378
AS4837	29	739	791	AS2519	4	1857	1859
AS18125	5	1861	1862	AS2200	28	1120	1052
AS26228	6	2077	2080	AS41635	3	2250	2252
AS21309	3	1792	1794	AS10694	9	4123	4124
AS3778	7	3836	3838	AS39111	3	964	1319
AS3614	4	3539	3541	AS1930	2	2204	2208
AS8508	3	2168	2169	AS36850	4	3656	3657

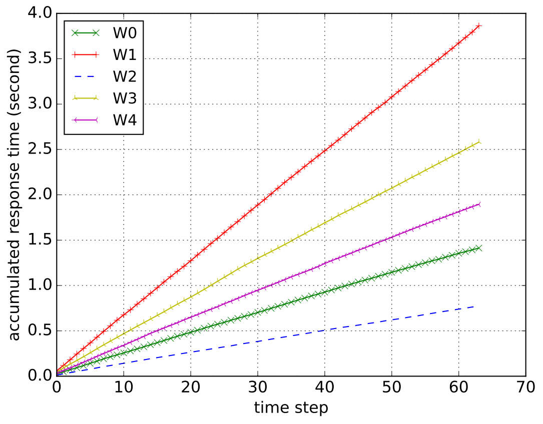

Fig. 5. The figure shows how the accumulated response time increases during each time step. The weight preferences are based on table II.

more significant amount of questions to the user. One possible explanation for the different behavior from one user to the other is that, if the weights are more balanced, comparing vectors using Pareto dominant and K-dominant methods is less informative and therefore the only option to eliminate one of the two vectors is to ask directly to the user. This problem can be mitigated by approximating a workflow for the system instead of calculating the exact one. Since IVI-SC is an exact algorithm, the initial sequence of data-set presentation have an important effect on the number of required queries in order to find the optimal workflow.

Figure 5 and Figure 6 shows how the service qualities change with respect to the time step for the 5 given weight preferences. The optimal workflow should maximize the total sum of throughputs while minimizing the total sum of response times. The two figures show how throughput and response time increase linearly w.r.t the time steps. Figure 5 shows that the accumulated response time for all weight preferences does not exceed 4 seconds. On the other hand, Figure 6 maximizes the accumulated throughput until a value of around 9000 Mbps.

If the users weight vectors are available, the exact workflow is computed by taking the weights into account and using a classical approach on MDPs such as Value iteration or Policy

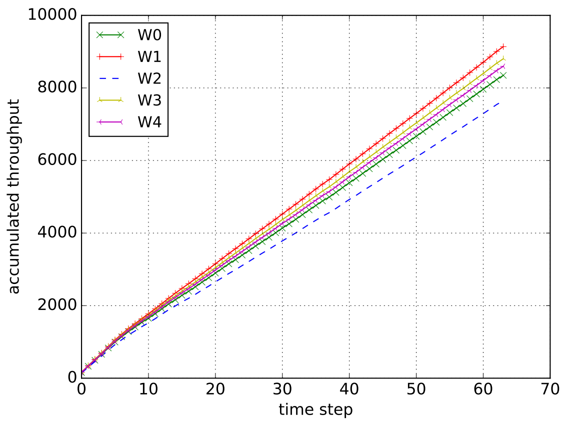

Fig. 6. The figure shows how the accumulated throughput increases during each time step. The weight preferences are based on table II.

Iteration method [5], [34]. In order to compare the workflows obtained with our method with the ideal ones, we transfer the quality vectors to the quality values: $QoS(S_i, S_{ij}) = \overline{W} \cdot \overline{Qos}(S_i, S_{ij})$ by doing the assumption of knowing *a priori* the user weights $\overline{W}$. In this way, we can compute the exact services compositions for each user and compare them with the computed workflows from the IVI-SC method. Our experiments confirm that the optimal workflow computed by the IVI-SC algorithm is exactly the same as the exact computed workflow for the 3 users with weights preferences $\overline{W}_0, \overline{W}_1$ and $\overline{W}_2$. For the two other weight preferences $\overline{W}_3$ and $\overline{W}_4$, the IVI-SC and the exact approach are different in a few number of abstract services: 178 out of 744 abstract services for $\overline{W}_3$ and 38 out of 744 abstract services for $\overline{W}_4$. In other words, our method computes 76% and 95% of concrete services correctly for users $\overline{W}_3$ and $\overline{W}_4$ respectively. Table III presents the list of abstract services where our approach (IVI-SC) and the exact approach propose different concrete services for the services composition problem under $\overline{W}_4$. Table IV shows the differences between these two approaches for $\overline{W}_4$. The first column contains a service abstract name with its possible number of concrete services to execute, and the second column shows the selected concrete service number by the IVI and the exact methods.

VIII. CONCLUSIONS AND FUTURE WORKS

In this paper, a Reinforcement Learning-based approach is proposed to solve the services composition problem in a web of things environments and without knowing the user preferences on the QoS attributes. The services composition problem is formulated as a discrete-time Vector-valued Markov Decision Process and solved using an interactive value iteration method. The registered qualities of the web services in our studied data-set [49], [50] are executed by selecting 142 users from the data-set. The experiments show that our algorithm finds the optimal services composition by learning the user preferences weights with high accuracy. The optimal services composition is obtained by maximizing the accumulated throughputs and minimizing the accumulated response times. As potential future work, different models of MDPs should be tested, such as parallel MDP, sequential parallel MDP, etc. Finally, it could also be interesting to classify and observe the users types w.r.t. their service qualities and their execution information on the web services before modeling the data-set into an MDP.

References

[1] O. Alagoz, H. Hsu, A. J. Schaefer, and M. S. Roberts. Markov decision processes: A tool for sequential decision making under uncertainty. *Medical Decision Making*, 30(4):474–483, 2009.

[2] P. Alizadeh, Y. Chevaleyre, and F. Lévy. Advantage Based Value Iteration for Markov Decision Processes with Unknown Rewards. In *International Joint Conference on Neural Networks (IJCNN 2016)*, Vancouver, Canada, July 2016.

[3] M. Alrifai, T. Risse, and W. Nejdl. A hybrid approach for efficient web service composition with end-to-end qos constraints. *ACM Transactions on the Web (TWEB)*, 6(2):7, 2012.

[4] M. Alrifai, D. Skoutas, and T. Risse. Selecting skyline services for qos-based web service composition. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 11–20. ACM, 2010.

[5] F. E. M. Arasi, S. Govindarajan, and A. Subbarayan. Weighted quality of service based ranking of web services. In *Indian Journal of Science and Technology*, Jul 2017.

[6] J. Cao, J. Huang, G. Wang, and J. Gu. Qos and preference based web service evaluation approach. In *2009 Eighth International Conference on Grid and Cooperative Computing*, pages 420–426, 2009.

[7] M. Chandra, A. Agrawal, A. Kishor, and R. Niyogi. Web service selection with global constraints using modified gray wolf optimizer. In *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2016.

[8] N. Chen, N. Cardozo, and S. Clarke. Goal-driven service composition in mobile and pervasive computing. *IEEE Transactions on Services Computing*, PP(99):1–1, 2017.

[9] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu. Web service recommendation via exploiting location and qos information. *IEEE Transactions on Parallel and Distributed Systems*, 25(7):1913–1924, 2014.

[10] Y. Chen, J. Huang, C. Lin, and J. Hu. A partial selection methodology for efficient qos-aware service composition. *IEEE Transactions on Services Computing*, 8(3):384–397, 2015.

[11] A. Chibani. *ITEA 3 Web of Objects European Project*, 2019 (accessed October 8, 2019). http://www.lissi.fr/woo-en/.

[12] S. Deng, L. Huang, W. Tan, and Z. Wu. Top-k automatic service composition: A parallel method for large-scale service sets. *IEEE Transactions on Automation Science and Engineering*, 11:891–905, July 2014.

[13] S. Deng, H. Wu, D. Hu, and J. L. Zhao. Service selection for composition with qos correlations. *IEEE Transactions on Services Computing*, 9:291–303, 2016.

[14] A. M. Ejaz, H. Mukhtar, D. Belaid, and J. B. Song. Qos-aware device selection using user preferences for tasks in ubiquitous environments. In IEEE, editor, *ICET'11 : IEEE International Conference on Emerging Technologies*, pages 1 – 6, 2011.

[15] V. Gabrel, M. Manouvrier, and C. Murat. Web services composition: Complexity and models. *Discrete Applied Mathematics*, 196(Supplement C):100 – 114, 2015.

[16] R. L. Graham, D. E. Knuth, and O. Patashnik. Concrete mathematics, addison wesley. *Reading, MA*, 1989.

[17] M. Graiet, I. Abbassi, M. Kmimech, and W. Gaaloul. A genetic-based adaptive approach for reliable and efficient service composition. *IEEE Systems Journal*, 12(2):1644–1654, 2018.

[18] M. S. Hossain, M. Moniruzzaman, G. Muhammad, A. Ghoneim, and A. Alamri. Big data-driven service composition using parallel clustered particle swarm optimization in mobile environment. *IEEE Transactions on Services Computing*, 9(5):806–817, 2016.

[19] N. M. Josuttis. *SOA in practice: the art of distributed system design.* " O'Reilly Media, Inc.", 2007.

[20] Y. Lei, Z. Jiantao, W. Fengqi, G. Yongqiang, and Y. Bo. Web service composition based on reinforcement learning. In *Web Services (ICWS), 2015 IEEE International Conference on*, pages 731–734. IEEE, 2015.

[21] J. W. Liu, L. Q. Hu, Z. Q. Cai, L. N. Xing, and X. Tan. Large-scale and adaptive service composition based on deep reinforcement learning. *Journal of Visual Communication and Image Representation*, 65:102687, 2019.

[22] W. Liu. Trustworthy service selection and composition - reducing the entropy of service-oriented web. In *INDIN 2005. 2005 3rd IEEE International Conference on Industrial Informatics, 2005.*, pages 104–109, Aug 2005.

[23] N. B. Mabrouk, N. Georgantas, and V. Issarny. Set-based bi-level optimisation for qos-aware service composition in ubiquitous environments. In *Web Services (ICWS), 2015 IEEE International Conference on*, pages 25–32. IEEE, 2015.

[24] D. Mallayya, B. Ramachandran, and S. Viswanathan. An automatic web service composition framework using qos-based web service ranking algorithm. *The Scientific World Journal*, 2015.

[25] U. S. Manikrao and T. V. Prabhakar. Dynamic selection of web services with recommendation system. In *International Conference on Next Generation Web Services Practices (NWeSP'05)*, pages 5 pp.–, Aug 2005.

[26] A. Mostafa and M. Zhang. Multi-objective service composition in uncertain environments. *IEEE Transactions on Services Computing*, 2015.

[27] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[28] L. Qi, Y. Tang, W. Dou, and J. Chen. Combining local optimization and enumeration for qos-aware web service composition. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 34–41. IEEE, 2010.

[29] R.V. Rao, V.J. Savsani, and D.P. Vakharia. Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 2011.

[30] L. Ren, W. Wang, and H. Xu. A reinforcement learning method for constraint-satisfied services composition. *IEEE Transactions on Services Computing*, 2017.

[31] P. Rodriguez-Mier, C. Pedrinaci, M. Lama, and M. Mucientes. An integrated semantic web service discovery and composition framework. *IEEE Transactions on Services Computing*, 9(4):537–550, July 2016.

[32] Q. Z. Sheng, X. Qiao, A. Vasilakos, C. Szabo, S. Bourne, and X. Xu. Web services composition: A decade's overview. *Information Sciences*, 280:218–238, 2014.

[33] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis. Ranking and clustering web services using multicriteria dominance relationships. *IEEE Transactions on Services Computing*, 3(3):163–177, 2010.

[34] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[35] N. Temglit, A. Chibani, K. Djouani, and M. A. Nacer. A distributed agent-based approach for optimal qos selection in web of object choreography. *IEEE Systems Journal*, 12:1655–1666, 2018.

[36] W. Tsai, P. Zhong, X. Bai, and J. Elston. Dependence-guided service composition for user-centric soa. *IEEE Systems Journal*, pages 889–899, 2014.

[37] H. Wang, G. Huang, and Q. Yu. Automatic hierarchical reinforcement learning for efficient large-scale service composition. In *Web Services (ICWS), 2016 IEEE International Conference on*, pages 57–64. IEEE, 2016.

[38] H. Wang, J. Li, Q. Yu, T. Hong, J. Yan, and W. Zhao. Integrating recurrent neural networks and reinforcement learning for dynamic service composition. *Future Generation Computer Systems*, 107:551–563, 2020.

[39] H. Wang, P. Ma, Q. Yu, D. Yang, J. Li, and H. Fei. Combining quantitative constraints with qualitative preferences for effective non-functional properties-aware service composition. *Journal of Parallel and Distributed Computing*, 100:71–84, 2017.

[40] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, and A. Bouguettaya. *Adaptive Service Composition Based on Reinforcement Learning*, pages 92–107. Springer Berlin Heidelberg, 2010.

[41] P. Weng and B. Zanuttini. Interactive Value Iteration for Markov Decision Processes with Unknown Rewards. In *IJCAI '13 - Twenty-Third international joint conference on Artificial Intelligence*, pages 2415–2421, Beijing, China, August 2013. AAAI Press.

[42] Q. Wu, F. Ishikawa, Q. Zhu, and D.-H. Shin. Qos-aware multigranularity service composition: Modeling and optimization. *IEEE Trans. Systems, Man, and Cybernetics: Systems*, 2016.

[43] P. Xiong, Y. Fan, and M. Zhou. Web service configuration under multiple quality-of-service attributes. *IEEE Transactions on Automation Science and Engineering*, 6(2):311–321, 2009.

[44] M. E. Khanouche Y., Amirat, A. Chibani, M. Kerkar, and A. Yachir. Energy-centered and qos-aware services selection for internet of things. *IEEE Transactions on Automation Science and Engineering*, 13(3):1256–1269, 2016.

[45] H. Yang and X. Xie. An actor-critic deep reinforcement learning approach for transmission scheduling in cognitive internet of things systems. *IEEE Systems Journal*, 14(1):51–60, 2020.

[46] Q. Yu and A. Bouguettaya. Efficient service skyline computation for composite service selection. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):776–789, 2013.

[47] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web*, pages 411–421. ACM, 2003.

[48] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Transactions on software engineering*, 30(5):311–327, 2004.

[49] Y. Zhang, G. Cui, S. Deng, and Q. He. Alliance-aware service composition based on quotient space. In *2016 IEEE International Conference on Web Services (ICWS)*, pages 340–347, June 2016.

[50] Z. Zheng, Y. Zhang, and M. R. Lyu. Investigating qos of real-world web services. *IEEE Transactions on Services Computing*, 7:32–39, 2014.

[51] G. Zou, Q. Lu, Y. Chen, R. Huang, Y. Xu, and Y. Xiang. Qos-aware dynamic composition of web services using numerical temporal planning. *IEEE Trans. Serv. Comput.*, 7(1):18–31, January 2014.