Article

An Intelligent Web Service Composition and Resource-Optimization Method Using K-Means Clustering and Knapsack Algorithms

Issam Alhadid ¹ , Sufian Khwaldeh ¹, Mohammad Al Rawajbeh ², Evon Abu-Taieh ¹ , Ra'ed Masa'deh ³
and Ibrahim Aljarah ^{4,*}

¹ Faculty of Information Technology and Systems, University of Jordan, Aqaba 77111, Jordan; i.alhadid@ju.edu.jo (I.A.); Skhwaldeh@gmail.com (S.K.); abutaieh@gmail.com (E.A.-T.)

² Faculty of Science and Information Technology, AL Zaytoonah University of Jordan, Amman 11942, Jordan; m.rawajbeh@zuj.edu.jo

³ School of Business, University of Jordan, Amman 11942, Jordan; r.masadeh@ju.edu.jo

⁴ King Abdulla II School for Information Technology, University of Jordan, Amman 11942, Jordan

* Correspondence: i.aljarah@ju.edu.jo

Abstract: Service-oriented architecture (SOA) has emerged as a flexible software design style. SOA focuses on the development, use, and reuse of small, self-contained, independent blocks of code called web services that communicate over the network to perform a certain set of simple tasks. Web services are integrated as composite services to offer complex tasks and to provide the expected services and behavior in addition to fulfilling the clients' requests according to the service-level agreement (SLA). Web service selection and composition problems have been a significant area of research to provide the expected quality of service (QoS) and to meet the clients' expectations. This research paper presents a hybrid web service composition model to solve web service selection and composition problems and to optimize web services' resource utilization using k-means clustering and knapsack algorithms. The proposed model aims to maximize the service compositions' QoS and minimize the number of web services integrated within the service composition using the knapsack algorithm. Additionally, this paper aims to track the service compositions' QoS attributes by evaluating and tracking the web services' QoS using the reward function and, accordingly, use the k-means algorithm to decide to which cluster the web service belongs. The experimental results on a real dataset show the superiority and effectiveness of the proposed algorithm in comparison with the results of the state-action-reward-state-action (SARSA) and multistage forward search (MFS) algorithms. The experimental results show that the proposed model reduces the average time of the web service selection and composition processes to 37.02 s in comparison to 47.03 s for the SARSA algorithm and 42.72 s for the MFS algorithm. Furthermore, the average of web services' resource utilization results increased by 4.68% using the proposed model in comparison to the resource utilization by the SARSA and MFS algorithms. In addition, the experimental results showed that the average number of service compositions using the proposed model improved by 26.04% compared with the SARSA and MFS algorithms.

Keywords: service-oriented architecture; web services; composition; k-means clustering; knapsack; optimization; quality of service

Citation: Alhadid, I.; Khwaldeh, S.; Al Rawajbeh, M.; Abu-Taieh, E.; Masa'deh, R.; Aljarah, I. An Intelligent Web Service Composition and Resource-Optimization Method Using K-Means Clustering and Knapsack Algorithms. *Mathematics* **2021**, *9*, 2023. <https://doi.org/10.3390/math9172023>

Received: 1 July 2021

Accepted: 19 August 2021

Published: 24 August 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Web services are distributed, loosely coupled software components designed to perform simple tasks and to support interoperable machine-to-machine interaction and exchange data over a network. Web services use the Web Service Definition Language (WSDL) as an interface described in a machine-processable manner to expose the web service functions using Extensible Markup Language (XML) [1]. There are two types of web services:

elementary web services, and service compositions. The elementary web services are used to perform one self-dependent task; meanwhile, the service compositions combine multiple web services connected in sequential order to perform an end-to-end complex task and to respond to users' requests that cannot be satisfied by single elementary web services using different heuristic and non-heuristic approaches and techniques [2–7]. Integrated web services that construct the service composition can be managed, replaced, and updated and rearranged at runtime without disrupting the ongoing business processes to provide the expected services and behavior in addition to fulfilling the clients' requests according to the service-level agreement (SLA) [2,3,8,9].

The authors of [8] stated that the Internet is the host environment of web services, and because the Internet is a highly dynamic environment, the web services' quality of service (QoS) might be affected by various aspects and factors, such as external changes related to the host environment and network, or internal software or hardware evolution. Moreover, the authors of [4,10,11] claimed that web services' performance and scalability might be affected by denial-of-service (DoS) or distributed denial-of-service (DDoS) attacks by flooding the service composition of the web service with a massive number of synchronous requests to overload web services' resource capacity and prevent the clients' SLA or the expected functionality from being fulfilled. The authors of [2,10–13] studied web services' QoS changes, and built and reconstructed the service composition in addition to the problem of web services' resource utilization. The authors of [4] stated that resources of the integrated web services in the service compositions are controlled by the web service with the minimum acceptable request capacity per second in the composition. While the resources of the other web services in the same service composition will be wasted and unused, the authors of [9] stated that is not a straightforward process to find the optimal number of web services required to construct the service composition in real time and provide a large-scale web service repository, especially with identical and similar web services functionally. The authors of [3,9] argue that minimizing the number of integrated web services improves the service composition maintenance and management, while also increasing the success rate of responses, minimizing costs, and saving service-composition-utilized resources. The authors of [9–11,13] argue that minimizing the number of integrated web services improves the service composition management and maintenance, as well as reducing costs and increasing the success of the rate of service compositions' responses. The authors of [2] claimed that the complexity of web service selection and composition processes involves three main factors: the vast available number of web services with similar functionality, the different possibilities of integrating web services into a service composition, and the various QoS requirements of the service composition.

According to the above discussions and related work, there are five major research perspectives to be analyzed:

1. Design a dynamic approach to improve the process in terms of determining the optimal number of web services required to construct the service composition to satisfy a set of predefined QoS constraints specified in the clients' SLA;
2. Upgrade the service composition; replace similar web service(s) functionality with the acceptable QoS in the service compositions to substitute the unavailable or unacceptable web service;
3. Enhance the service composition scalability;
4. Optimize the utilization of the web service resources in the service composition construction process;
5. Track the service composition requests to make sure that the clients' SLA QoS requirements are satisfied.

To achieve our goal using the proposed model, a classification of the web services specified by the services' action that presents the web services' functionality is needed. Then, using the k-means clustering algorithm, the classified web services will be grouped in clusters according to the web service reward calculated using the composition reward function. The classification and the clustering processes are used to avoid exhaustive

search in the large-scale web service repository, and to speed up the search process. Later, the knapsack algorithm is used to construct the service composition, and the participating web service rewards will be used to find the service composition reward, which represents the QoS of service composition. After the construction of the service composition, if some of the participating web service resources are still available, as represented by the web service capacity, such web services can be used by other service compositions to enhance the resource utilization and provide additional service compositions that satisfy the clients' requests and the SLA requirements. Finally, each time the service composition is invoked, the composition reward function will be used to track the service composition requests and compare the results with the clients' SLA QoS requirements, updating the service composition knapsack whenever the minimum requirements of the clients' SLA QoS attributes are not met.

This research presents a hybrid technique using k-means clustering and knapsack algorithms to solve the service composition construction process problem efficiently. The main contributions are:

- The proposed model has been tested on a real benchmarks dataset (QWS) to demonstrate its efficiency for optimizing the web service resource utilization, tracking the service composition QoS using the composition reward function, and in comparing the results with the requirements of clients' SLA QoS attributes;
- The experimental results show that the proposed model reduces the average time of the web service selection and composition processes to 37.02 s in comparison to 47.03 s for the state-action-reward-state-action (SARSA) algorithm and 42.72 s for the multistage forward search (MFS) algorithm. Furthermore, the average of web services' resource utilization results increased by 4.68% using the proposed model in comparison to the resource utilization by the SARSA and MFS algorithms. In addition, the experimental results show that the average number of the service compositions using the proposed model improved by 26.04% in comparison to the SARSA and MFS algorithms.

The proposed web service composition model using k-means clustering and knapsack algorithms is used to enhance the service composition selection and construction processes and to optimize web services' resource utilization by maximizing the service compositions' QoS and minimizing the number of web services integrated in the service composition knapsack.

The proposed model uses the unsupervised machine learning k-means algorithm to discover underlying patterns and to group convergent data together, where the k-means algorithm creates clusters that hold the web services with similar QoS characteristics [14,15], while the knapsack algorithm is used to determine the objects to be placed in the knapsack—where the goal is to maximize the profit of the placed objects without exceeding the knapsack capacity [9]. The proposed model uses the knapsack algorithm to construct the service compositions by choosing the web services to be integrated in the service composition knapsack to maximize the QoS value of the web services without exceeding the capacity that represents the minimum acceptable number of requests by the service composition.

The experimental results of the proposed model will be compared with the composition results of the multistage forward search (MFS) algorithm and SARSA reinforcement learning to prove the robustness and effectiveness of the proposed model.

This research paper is organized as follows: Section 2 presents published web services selection and composition research; in Section 3, the proposed model is introduced, which includes the web service selection and composition QoS attributes, as well as the reward functions, the processes of classification and clustering, and the knapsack algorithm; Section 4 discusses the experiments' dataset, results, and evaluation; finally, Section 5 presents the research conclusions and suggestions for future work.

2. Related Work

Web service selection and composition has been addressed as one of the active research areas to improve the performance and reliability of service composition construction [3–6]. Different approaches and mechanisms have been suggested to solve the web service selection and service composition, such as ignoring the clients' requests that exceed the service composition maximum capacity to avoid denial-of-service (DoS) and distributed denial of service (DDoS) attacks [10,11]. The authors of [4] proposed a mechanism to prioritize the clients' requests according to their class in order to avoid DoS and DDoS attacks. Many researchers have suggested artificial intelligence (AI) algorithms to improve web service selection and service composition efficiency and effectiveness. The authors of [7] proposed a selection and composition technique using a genetic-based algorithm for combining cloud services that ensures that the services provided are working efficiently. The authors of [16] proposed an algorithm using simulated annealing and a genetic algorithm to optimize web services' selection and composition ability. Moreover, Ref. [13] used a Bayesian learning algorithm to estimate the true probability models involved in the Markov decision processes, where the workflow composition is modeled as a decision to find the solution that represents the best service composition. Furthermore, Ref. [2] proposed a model based on the Markov decision process to solve the problem of service composition by employing artificial and real datasets to validate the proposed approach using the iterative policy evaluation, value iteration, and policy iteration algorithms. The results of the proposed model were compared with two popular reinforcement learning algorithms: SARSA, and Q-learning. Other researchers proposed approaches focused on different graph-based algorithms to solve the service composition problem [12,17]. The authors of [18] suggested an approach using dynamic programming to generate a weighted multistage graph and to find the longest path as a solution for the web service selection problem. The authors of [19] proposed a method based on the integrated ant colony and artificial bee colony optimization algorithms to find the optimal solution for web service selection and composition using the directed workflow modeled as a directed acyclic graph used to determine the optimal feasible path that represents the best service composition. The authors of [20] presented a method to optimize the web service composition by modeling the semantic relationships between all of the web services into a directed graph to find all of the shortest paths. Moreover, Ref. [9] proposed a mechanism based on the knapsack-variant algorithm by mapping the relevant web services into items with changeable volume and cost to generate a service dependency graph. Using the generated graph, they transformed each search step on the graph into a dynamic knapsack problem and calculated the minimum composition that satisfies the client's SLA requirements using the knapsack-variant algorithm. Furthermore, Ref. [21] introduced the smart multistage forward search (SMFS) technique to enhance the web service selection and composition in addition to optimizing the WS resource utilization. The SMFS technique is a modified version of the SMF where the web services' resources can be shared by different client classes to utilize the unused web services' resources and to construct new service compositions.

Other researchers suggested using reinforcement learning methods in service composition. The authors of [22,23] proposed an adaptive and dynamic service composition using a Q-learning algorithm, while [24] suggested the use of SARSA reinforcement learning to solve the service composition problem. The authors of [25] proposed a multi-agent reinforcement learning model for web service composition by modeling the service composition as a Markov decision process using a distributed Q-learning algorithm.

According to the previous discussion, researchers are working hard to optimize the service composition process—especially during the selection—and creating procedures to minimize and utilize the web service composition and its resources effectively and efficiently. This paper introduces an efficient model using k-means clustering, knapsack, and composition reward function to optimize the web services' selection and service composition in addition to reusing and utilizing the available web services' resources optimally in large-scale web service repositories.

3. Proposed Web Service Composition Model

This research paper proposes a web service composition model to improve the web service selection and service composition process and to optimize the utilization of the available web services' resources using k-means clustering and knapsack algorithms.

The proposed model consists of four processes: the first process classifies the web services into classes based on each web service action; the second process creates clusters inside each class based on the web services' QoS attributes and the reward function; the third process constructs the services' compositions using the knapsack algorithm; and the final process tracks and evaluates the QoS attributes related to the integrated web services that construct the service composition using the reward function, where web services' and compositions' QoS attributes are used to find the reward function of the service composition to evaluate and track the provided services and to upgrade the composition knapsacks according to the service-level agreement requirements.

3.1. Web Service QoS Attributes

Web service response time is the time needed by a service to successfully respond to a request. Response time is one of the important QoS attributes that is used to classify the web services and to achieve the clients' SLA requirements. A web service's response time can be calculated using the following equation [26]:

$$\text{WSResTime} = \text{WSResTime}(\text{in}) + \text{WSResTime}(\text{out}) \quad (1)$$

where

$\text{WSResTime}(\text{in})$ and $\text{WSResTime}(\text{out})$ are the web service's invocation round trip transmission time; $\text{WSResTime}(\text{in})$ represents the request time from the service invoker to the web service, and can be calculated using Equation (2):

$$\text{WSResTime}(\text{in}) = \frac{\text{packet size}}{\text{available bandwidth} - \text{in bytes}} \quad (2)$$

while $\text{WSResTime}(\text{out})$ is the time required by the response of the web service to reach the invoker; Equation (3) is used to find $\text{WSResTime}(\text{out})$:

$$\text{WSResTime}(\text{out}) = \frac{\text{packet size}}{\text{available bandwidth} - \text{out bytes}} \quad (3)$$

Web service Availability is the probability that the web service is accessible and ready for immediate use; it can be calculated using the following equation [26]:

$$\text{WSava} = \frac{\text{uptime}}{\text{Total Time}} * 100\% \quad (4)$$

where:

- The up time denotes the web service's operational time between the last web service failure to the next failure, which is known as the mean time between failure (MTBF);
- The Total Time is the overall period of the web service's operation being investigated.

Web service throughput represents the number of requests the web service can process successfully within a defined period.

$$\text{WSThr} = \frac{\text{Cs}}{\text{Time}} \quad (5)$$

where:

- The Cs is the number of web service requests served successfully;
- Time is the period of the web service's operation being investigated.

3.2. Service Composition QoS Attributes and Composition Reward Function

Reward function is used to find the web service's QoS reward (R), where the R value is measures using the web service's QoS attributes; R is calculated in the following equation proposed in [2]:

$$R_{(WSi)} = \alpha - \beta + \gamma \quad (6)$$

where:

- $R_{(WSi)}$ is the reward value gained by the web service (i);
- WSi is the web service with state I , where $i \in \{t, t+1, t+2, \dots, n\}$;
- α is the web service availability reward;
- β is the web service response time reward;
- γ is the web service throughput reward.

The web service availability reward (α) can be calculated using:

$$\alpha = \frac{AV^s - AV^{min}}{AV^{max} - AV^{min}} \quad (7)$$

where:

- AV^s : The availability value of the web service with state (i) calculated using Equation (4);
- AV^{min} : The minimum value of the web service availability in the web services' cluster;
- AV^{max} : The maximum value of the web service availability in the web services' cluster.

While the web service response time reward (β) can be calculated using:

$$\beta = \frac{RespTime^s - RespTime^{min}}{RespTime^{max} - RespTime^{min}} \quad (8)$$

where:

- $RespTime^s$: The response time value of the web service with state (i) calculated using Equation (1);
- $RespTime^{min}$: The minimum value of the web service response time in the web services' cluster;
- $RespTime^{max}$: the maximum value of the web service response time in the web services' cluster.

Finally, the web service throughput reward (γ) can be calculated using:

$$\gamma = \frac{Thr^s - Thr^{min}}{Thr^{max} - Thr^{min}} \quad (9)$$

where:

- Thr^s : The throughput value of the web service with state (i) calculated using Equation (5);
- Thr^{min} : The minimum value of the web service throughput in the web services' cluster;
- Thr^{max} : The maximum value of the web service throughput in the web services' cluster.

The sum of all integrated web services' rewards is the service composition reward value, which can be calculated using:

$$SCr = \sum_{i=1}^n R(WSi) \quad (10)$$

where

- $R(WSi)$ is the reward value gained by the web service (i) using Equation (6).

3.3. Proposed Model Processes

The proposed model uses several algorithms to achieve the research goals. The following steps illustrate how the processes in the proposed model work and execute:

1. Classify the web services into classes based on the web service action. Web services with the same functionality will be grouped in one class together. Moreover, if a composite service provides a certain task and also contributes the required functionality, such composite services will be treated as one service and grouped within the class according to the composite service's action;
2. Use the classified web services to create clusters inside each class based on the web services' QoS attributes calculated using Equations (1), (4), (5) mentioned in Section 3.1. The following steps represent the k-means clustering algorithm:

Step 1: Randomly generate (k) random points as initial clusters' centers;

Step 2: Assign each point to the nearest cluster center using the Euclidean distance between web service x_i and cluster center μ_k (Equation (11)):

$$j(\mu_k) = \sum_{i=1}^m \sum_{k=1}^K |x_i - \mu_k|^2 \quad (11)$$

where:

- $j(\mu_k)$: squared error function;
- k : index of the web services' clusters;
- i : index of the web services;
- x_i : web service;
- μ_k : center of the web services' cluster.

Step 3: Re-compute the new cluster centers using Equation (12):

$$\mu_k = (1/m) \sum_{i=1}^m x_i \quad (12)$$

where:

- m represents the number of web services in the k th cluster;
- x_i : web services in the cluster.

Repetition step: Repeat steps 3 and 4 until convergence; the assignment of points to clusters becomes stable;

Cluster Selection step: Select the web service clusters that meet the clients' SLA, which will be used in the service composition process.

After the algorithm's completion, the results are the number of clusters where each cluster contains all web services with the same action and similar QoS attributes. Figure 1 demonstrates the web service classification and k-means clustering algorithms.

3. Using the knapsack algorithm, construct the service compositions according to clients' classes and to satisfy the requirements of the SLA. Step 1: Define the knapsack volume {}, cost, capacity, and weight for each client's class, where the knapsack cost is the sufficient QoS value of the web services, and the knapsack volume set comprises the web services integrated in the service composition, while the knapsack capacity represents the minimum acceptable number of requests by the service composition, and the weight is the minimum QoS value of the client's class SLA; Step 2: Generate two random numbers (R_1, R_2) where $R_1 \in \{1, 2, 3, \dots, N\}$ and $R_2 \in \{1, 2, 3, \dots, M\}$, N is the number of web services' actions classes, R_2 is the number of clusters inside the R_1 class, and R_1 and R_2 are not in the knapsack volume set; Step 3: Search for a web service with an action that matches the random number R_1 from the cluster that matches the random number R_2 , where the selected web service's maximum capacity value is the minimum value in the R_2 cluster, and the QoS reward (Equation (6)) value is the highest value in the cluster R_2 ; Step 4: After the selection process, insert the selected web service in the knapsack volume set, and then use the composition reward function equation (Equation (10)) to find the service composition QoS reward value, which represents the knapsack cost; Step 5: repeat steps 3 and 4 until the knapsack is full or no more R_1, R_2 can be generated (there are no more available web services in the cluster); Step 6: After constructing the service compositions in

case one of the web services' maximum capacity is less than that of the other web services participating in the service composition, the web services' available capacity must be set to the difference between the maximum capacity of the web service and the minimum value of the maximum capacity related to the web service in the service composition. Therefore, all of the web services participating in the construction of the service composition might have available resources according to the web service with minimum value of the maximum capacity. This value is called the available capacity; Step 7: Define the composition threshold minimum (WSAVA.cap), which is the minimum number of requests that can be accepted and executed by the web services with resources that can be used to construct other service compositions; then, go to step 1.

```

//Classification
initialize class1, class2, ... class j Web service classes, where each class represents a
web service action // input from the SLA contracts between the service provider and the
clients
for each web service(i) ∈ web service repository // web services located in the repository
either built by the service provider or rented services. Also web services might be free of
charge.
    Find  $R_{(WSi)}$  //Calculate the Web service  $R(WSi)$  using eq. 6
    find the class (j) where web service(i) action ∈ class(j)action
Loop// Next Web Service
//all the web services are classified according to the QoS attributes
// K-Means Clustering
for each Class (j) ∈ web service classes
    Generate  $\mu_1, \mu_2, \mu_3, \dots, \mu_k$  random points as initial cluster's centers
    Repeat Until No More Web Services in the Class (j)
    {
        for each web service (i) ∈ web service class (j)
             $j(\mu_k) = \sum_{i=1}^m \sum_{k=1}^K |x_i - \mu_k|^2$ 
        Loop// Next Web Service
        for each cluster (k) ∈ web service clusters
             $\mu_k = (1/m) \sum_{i=1}^m x_i$ 
        Loop// Next Web Service cluster
    }
Loop// Next Web Service class
//the output of the stage: clusters of web services contain web services that share the
same QoS attributes

```

Figure 1. Web service classification and k-means clustering algorithm.

- After the composition process, all of the service composition requests will be tracked, and the value of service composition reward will be compared with integrating web services to make sure that all of the clients' SLA QoS attributes are satisfied. If one of the of the web service QoS rewards (α , β , γ) is less than zero (negative value), this means that the web service's QoS attributes are less than the acceptable minimum value in its clusters; accordingly, it must be replaced with another web service and moved to another cluster; otherwise, the service composition will not satisfy the clients' SLA requirements. Figure 2 illustrates the knapsack algorithm implemented in the service composition process.

```

var composition threshold = 0;
Repeat until no more service composition can be constructed OR composition threshold > 0 {
  For each client_class ∈ clients_SLA_classes // client classes in the SLA contracts
  {
    var Ser_Com_Knapsack [volume { }, cost, capacity, weight];
    // cost is the QoS value of the service composition.
    // volume is the integrated web services in the service composition.
    // capacity is the minimum acceptable number of requests by the service composition
    // weight is the minimum QoS value of the client's class SLA
    Repeat until no more R1, R2 to be generated OR Ser_Com_Knapsack [volume { }] is Full
    {
      Generate random number R1 ∈ [1, N]; // N is the number of the web services action class
      R2 = Generate random number R2 ∈ [1, M]; // M is the number of the web services clusters
      // in the web service action class R1.
      Search for a web service WSi ∈ R1 class and WSi ∈ R2 cluster AND (WSi max. capacity is
      min. value in R2 AND max(R(WSi)) in R2)
      {
        Insert WSi into Ser_Com_Knapsack[volume { }];
        Calculate Composition reward (SCr) and update
        Ser_Com_Knapsack[cost];
      }
    }
    // Repeat until no more R1, R2 OR Knapsack is Full
    For All web services in the service composition (Ser_Com_Knapsack[volume { }])
      set WS(available capacity) = WS(max. capacity) - min. (service composition(max.
      capacity));
    SCr =  $\sum_{i=1}^n R(WS_i)$  // calculate the service composition Reward Function using Eq. 10
  } // Next client class
  If no more web service can be integrated in service composition
    Set composition threshold = 100; // or 200 requests per Sec.
  } Repeat until no more service compositions

```

Figure 2. Service composition using the knapsack algorithm.

After the algorithm's completion, the results are the number of clusters where each cluster contains all web services with the same action and similar QoS attributes. Figure 1 demonstrates the web service classification and k-means clustering algorithms.

4. Experimental Simulation, Results, and Discussion

A modified version of the hybrid service-oriented architecture simulator for web service selection and composition created by [27] was used to implement the suggested service composition paradigm. The simulator uses the QWS web service dataset proposed by [28]. QWS is a labeled dataset that contains 2508 web services. The dataset describes real-world QoS evaluation using response time, availability, throughput, and other QoS attributes, where the web services in the QWS dataset are divided into 830 excellent web services, 831 good web services, and 847 poor web services. The simulator supports the service classification and composition, in addition to the SLA gap used to simulate the behavior of the real web services and the changes in the QoS values during the service execution engine running. Hence, a modification to the simulator applies the k-means clustering and knapsack algorithms to evaluate the proposed technique. All simulation experiments were performed using an i7-3632QM processor 2.20 GHz computer with 8 GB DDR3 RAM. The initial simulator setup parameters are shown in Table 1.

Table 1. Hybrid service-oriented architecture simulator setup parameters.

Parameter	Value
Number of runs	20
Number of clients/runs	1000
Minimum number of requests/client	50
Maximum number of requests/client	5000
Number of web services	2509
Number of clients	200
Minimum value of web services' maximum capacity level	600
Maximum value of web services' maximum capacity level	1800
Minimum number of web services in service composition	5
Maximum number of web services in service composition	10

Researchers presented several algorithms and techniques using AI algorithms to handle the web service selection and composition challenge, as indicated in the related work section. This section presents results of the time required to select and construct service composition using the proposed model compared to results of the time required by the reinforcement learning algorithms SARSA and multistage forward search (MFS), as well as the comparison between the number of constructed service compositions using the proposed model, SARSA, and MFS; it also presents the comparison between the results of the web service resource utilization for the proposed model, SARSA, and MFS.

The multistage forward search algorithm [29] is one of the dynamic programming algorithms used to find the minimum cost from source to sink. The multistage forward search algorithm starts with graph construction using the relevant web services and divides the web services into a set of stages based on their actions; then, the MFS algorithm is used to find the service composition that represents the best path in the graph.

SARSA is proposed by [30] as one of the reinforcement learning algorithms used for learning a Markov decision process policy. SARSA depends on the agent interactions with the environment (states) and the rewards for the chosen actions related to states to update the Q-value.

This section shows the comparison between the proposed model and the multistage forward search and SARSA in web service selection and service composition. The QWS Dataset contains 2508 web services, which are grouped into three clusters—excellent, good, and poor—where the excellent cluster contains 830 web services, the good cluster includes 831 web services and, finally, the poor cluster holds 847 web services.

Figure 3 shows the time needed to construct the web services using the proposed model, MFS algorithm, and SARSA algorithm with different numbers of web services in each service composition, where the proposed model composition threshold minimum (WSAVA.cap) = 100 requests. Meanwhile, Figure 4 illustrates the service composition's construction time where the composition threshold minimum (WSAVA.cap) = 200 requests. In both Figures 3 and 4, the SARSA algorithm learning rate = 0.7. Figure 3 confirms that the experimental results of the proposed model reduce the average time of web service selection and composition processes to 39.8 s, compared to 47.03 and 42.72 seconds for the SARSA and MFS algorithms, respectively.

Figure 3. Service composition construction time when the composition threshold = 100.

Furthermore, Figure 4 shows that the average selection and composition time of the proposed model when the composition threshold minimum ($WSAVA_{cap}$) = 200 is reduced to 34.24 s, compared to the average SARSA and MFS selection and composition times, which are 47.03 and 42.72 s, respectively.

Figure 4. Service composition construction time when the composition threshold = 200.

Figures 3 and 4 confirm that the time required by the proposed model for the selection and construction processes is less than the time needed by the SARSA and MFS algorithms, even with the extra time needed to construct the additional service compositions and share the unused and available web service resources, where the complexity time of the MFS dynamic programming algorithm is $O(n^2)$, which is less than the time needed by the SARSA algorithm to discover the state space and to build the optimal service compositions, which might require more than one iteration to find.

Figure 5 shows the number of constructed service compositions based on the number of web services integrated in the composition using the proposed model, the MFS

algorithm, and the SARSA algorithm, with different numbers of web services in each service composition, and where the proposed model composition threshold minimum (WSAVA.cap) = 100 requests. Meanwhile, Figure 6 illustrates the number of service compositions where the composition threshold minimum (WSAVA.cap) = 200 requests. In both Figures 5 and 6, the SARSA algorithm learning rate = 0.7.

Figure 5 confirms that the number of constructed service compositions using the proposed model exceeded the number created by the SARSA and MFS algorithms, where the results show that using the proposed model increased the number of service compositions by 41.61% compared to the SARSA and multistage forward search algorithms.

Figure 5. Number of constructed service compositions with composition threshold = 100.

Moreover, Figure 6 shows that using the proposed model with a composition threshold minimum (WSAVA.cap) = 200 increased the number of the constructed service compositions by 10.47% compared to the SARSA and multistage forward search algorithms. The results illustrated in Figures 7 and 8 illustrate that the proposed model raises the number of service compositions, especially when the threshold minimum (WSAVA.cap) = 100; because the minimum number of acceptable requests by web services participating in the service composition equals 100 requests, this allows the building of more service compositions by sharing additional resources between different integrated web services.

Figure 7 presents the number of used and unused web services utilized in the service compositions using the proposed model, and the MFS and SARSA algorithms. Figure 7 also shows that when the composition threshold minimum (WSAVA.cap) = 100 the average improvement of the web services' resource utilization is around 5.6% using the proposed model compared to the other algorithms.

Figure 8 presents the number of used and unused web service resources in the services composition process using the proposed model, and the MFS and SARSA algorithms, where the web services' resource utilization using the proposed model increased by up to 3.75% compared to resource utilization by the SARSA and MFS algorithms when the composition threshold minimum (WSAVA.cap) = 200.

Figure 6. Number of constructed service compositions when the composition threshold = 200.

Figure 7. The number of used and the unused web services when the composition threshold = 100.

As noted, the percentage of the web services' resource utilization when the composition threshold equals 100 is better than the results when the composition threshold equals 200. This is related to the minimum number of requests allowed in the service compositions, which means that the unused web services and the web services with available resources will be utilized and reused to construct a greater number of service compositions and to provide services to clients and consumers that satisfy the SLA requirements.

Figure 8. The number of used and unused web services when the composition threshold = 200.

5. Conclusions

This research paper has proposed a novel model to solve the web service selection and composition problem and to enhance web service resource utilization using k-means and knapsack algorithms and composition reward function. A comparison of the results of the proposed model with the SARSA and MFS algorithms shows the reliability, soundness, and validity of our model, where the experiments were conducted using the hybrid service-oriented architecture simulator and the QWS dataset. The experiments on the benchmark QWS dataset showed significant improvement of the proposed model over the SARSA and MFS algorithms in terms of the number of constructed service compositions, where the average of the additional number of service compositions constructed by the proposed model using the composition thresholds 100 and 200 is 26.04% compared to the number of created service compositions by the SARSA and MFS algorithms. Furthermore, the experimental results show that the proposed model enhances the web services' resource utilization by 4.68% compared to the SARSA and MFS algorithms, where 4.675% is the average number of integrated web services using web service composition thresholds of 100 and 200 requests. The results also show that the proposed model's selection and construction process time is less than the time required by the SARSA and MFS algorithms in both cases when the composition threshold equals 100 and 200 requests. The complexity time of the proposed model is $O(n)$, where 'n' is the number of web services. Additionally, the classification and the clustering processes in the proposed model are used to avoid exhaustive search in the large-scale web service repository, and to speed up the search process.

On the other hand, the complexity time of the MFS algorithm is $O(n^2)$, which is less than the time for more than one iteration required by SARSA to discover the state space of the web services dataset and to construct the optimal collection of service compositions. Moreover, SARSA might stick to suboptimal solutions or end up with no solution at all. The experimental results confirm that even with the extra time needed by the proposed model to utilize the available web service resources after the first composition phase—and to create additional service compositions, which will offer more service composition to be used by clients—the proposed model reduces the average time of the web services' selection and composition processes to 37.02 s, compared to 47.03 and 42.72 s for the SARSA and MFS algorithms, respectively, where the proposed model's average time is calculated using

the processing time for composition thresholds of 100 and 200. For future perspectives on the subject of web service composition, it would be valuable to conduct experiments that address the process of real web service compositions that simulate the behavior of real servers, networks, and web services. Moreover, the effects of the other QoS attributes on the service compositions' rewards or penalties should be investigated.

One of the limitations of the proposed model is the availability of web services with different functionality and diversity of QoS attributes that are required to construct an end-to-end service composition according to the SLA requirements, which is one of the major challenges related to service compositions in general. This challenge affects the process of service composition construction; when the web service with the required functionality is unavailable, it will be hard to complete the service composition, and the required SLA might not be achieved for all clients. Another limitation considered to be a challenge is the unavailability of the web service either when the integrated web service is down, or when the web service cannot either provide the service with the minimum SLA requirements or provide a substitutional web service. This will also directly affect the execution of the service composition, and will stop the provided service.

Author Contributions: Conceptualization, I.A. (Issam Alhadid), E.A.-T. and R.M.; Investigation, I.A. (Issam Alhadid), S.K., M.A.R., E.A.-T. and R.M.; Methodology, I.A. (Issam Alhadid), S.K., M.A.R. and I.A. (Ibrahim Aljarah); Resources, E.A.-T.; Supervision, R.M. and I.A. (Issam Alhadid); Writing—review & editing, I.A. (Issam Alhadid), S.K., M.A.R., E.A.-T., R.M. and I.A. (Ibrahim Aljarah). All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data can be downloaded from <https://qwsdata.github.io/> (accessed on 25 February 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. W3C Working Group. Web Services Architecture. 2004. Available online: <http://www.w3.org/TR/ws-arch/> (accessed on 25 February 2021).
2. Uc-Cetina, V.; Moo-Mena, F.; Hernandez-Ucan, R. Composition of web services using Markov decision processes and dynamic programming. *Sci. World J.* **2015**, *2015*. [[CrossRef](#)] [[PubMed](#)]
3. Khwaldeh, S.; Abu-Taieh, E.; Alhadid, I.; Alkhawaldeh, R.; Masa'deh, R. DyOrch: Dynamic orchestrator for Improving web services composition. In Proceedings of the 33rd International Business Information Management Association Conference, IBIMA 2019, Granada, Spain, 10–11 April 2019; pp. 6030–6047.
4. AlHadid, I.; Abu-Taieh, E. Web services composition using dynamic classification and simulated annealing. *Mod. Appl. Sci.* **2018**, *12*, 395–405. [[CrossRef](#)]
5. Jung, J.Y.; Bae, J.; Liu, L. Hierarchical clustering of business process models. *Int. J. Innov. Comput. Inf. Control* **2009**, *5*, 1349–1498.
6. Mirzayi, S.; Rafe, V. A hybrid heuristic workflow scheduling algorithm for cloud computing environments. *J. Exp. Theor. Artif. Intell.* **2015**, *27*, 721–735. [[CrossRef](#)]
7. Shirvani, M.H.; Gorji, A.B. Optimisation of automatic web services composition using genetic algorithm. *Int. J. Cloud Comput.* **2020**, *9*, 397–411. [[CrossRef](#)]
8. Zeng, L.; Benatallah, B.; Dumas, M.; Kalagnanam, J.; Sheng, Q.Z. Quality driven web services composition. In Proceedings of the 12th International Conference on World Wide Web, Budapest, Hungary, 20–24 May 2003; pp. 411–421.
9. Fan, S.; Yang, Y. Efficient web service composition via knapsack-variant algorithm. In Proceedings of the International Conference on Services Computing, Seattle, WA, USA, 25–30 June 2018; Springer: Berlin/Heidelberg, Germany, 2018.
10. Jung, J.; Krishnamurthy, B.; Rabinovich, M. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In Proceedings of the 11th International Conference on World Wide Web, Honolulu, HI, USA, 7–11 May 2002; pp. 293–304. [[CrossRef](#)]
11. Yau, D.K.; Lui, J.; Liang, F.; Yam, Y. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. *IEEE ACM Trans. Netw.* **2005**, *13*, 29–42. [[CrossRef](#)]
12. Yan, Y.; Chen, M.; Yang, Y. Anytime QoS optimization over the PlanGraph for web service composition. In Proceedings of the 27th Annual ACM Symposium on Applied Computing, Trento, Italy, 26–30 March 2012; pp. 1968–1975.
13. Doshi, P.; Goodwin, R.; Akkiraju, R.; Verma, K. Dynamic workflow composition: Using Markov decision processes. *Int. J. Web Serv. Res. IJWSR* **2005**, *2*, 1–17. [[CrossRef](#)]
14. Likas, A.; Vlassis, N.; Verbeek, J.J. The global k-means clustering algorithm. *Pattern Recognit.* **2003**, *36*, 451–461. [[CrossRef](#)]

15. Krishna, K.; Murty, M.N. Genetic K-means algorithm. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* **1999**, *29*, 433–439. [[CrossRef](#)] [[PubMed](#)]
16. Gao, Z.-p.; Chen, J.; Qiu, X.-S.; Meng, L.-M. QoS/QoE driven simulated annealing-based genetic algorithm for web services selection. *J. China Univ. Telecommun.* **2009**, *16*, 102–107. [[CrossRef](#)]
17. Hwang, S.Y.; Lim, E.O.; Lee, C.H.; Chen, C.H. Dynamic web service selection for reliable web service composition. *IEEE Trans. Serv. Comput.* **2008**, *1*, 104–116. [[CrossRef](#)]
18. Gao, Y.N.J.; Zhang, B.; Yang, L.; Gong, Q. Optimal web services selection using dynamic programming. In Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06), Cagliari, Italy, 26–29 June 2006.
19. Shree, S.; Amuthan, A.; Suresh, J.K. Integrated ant colony and artificial bee colony optimization meta heuristic mechanism for quality of service based web service composition. *J. Comput. Theor. Nanosci.* **2019**, *16*, 1444–1453. [[CrossRef](#)]
20. Elmaghraoui, H.; Zaoui, I.; Chiadmi, D.; Benhlila, L. Graph based e-government web service composition. *IJCSI Int. J. Comput. Sci.* **2011**, *8*, 103–110.
21. Alhadid, I.; Tarawneh, H.; Kaabneh, K.; Masa'deh, R.E.; Hamadneh, N.N.; Tahir, M.; Khwaldeh, S. Optimizing Service Composition (SC) Using Smart Multistage Forward Search (SMFS). *Intell. Autom. Soft Comput.* **2021**, *28*, 321–336. [[CrossRef](#)]
22. Wang, H.; Zhou, X.; Zhou, X.; Liu, X.W.; Li, W. Adaptive and dynamic service composition using q-learning. In Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence, Arras, France, 27–29 October 2010; Volume 1, pp. 145–152.
23. Yu, L.; Zhili, W.; Lingli, M.; Jiang, W.; Meng, L.; Xue-song, Q. Adaptive web services composition using q-learning in cloud. In Proceedings of the 9th World Congress on Services, Santa Clara, CA, USA, 28 June–3 July 2013; pp. 393–396.
24. Wang, H.; Zhang, X.; Yu, Q. Integrating POMDP and SARSA for service composition with incomplete information. In Proceedings of the International Conference on Service-Oriented Computing, Banff, AB, Canada, 10–13 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 677–684.
25. Wang, H.; Wang, X.; Zhang, X.; Yu, Q.; Hu, X. Effective service composition using multi-agent reinforcement learning. *Knowl. Based Syst.* **2016**, *92*, 151–168. [[CrossRef](#)]
26. Emeakaroha, V.C.; Brandic, I.; Maurer, M.; Dustdar, S. Low level metrics to high level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In Proceedings of the 2010 International Conference on High Performance Computing and Simulation (HPCS), Caen, France, 1 July 2010; pp. 48–54.
27. Al-Tarawneh, H.; AlHadid, I.; Kaabneh, K.; Alhroob, A. Hybrid Service Oriented architecture simulator. In Proceedings of the 3rd International Computer Sciences and Informatics Conference (ICSIC 2019), Amman, Jordan, 2–4 April 2019.
28. Al-Masri, E.; Mahmoud, Q. Toward quality-driven web service discovery. *IT Prof.* **2008**, *10*, 24–28. [[CrossRef](#)]
29. Sitinjak, A.A.; Pasaribu, E.; Simarmata, J.E.; Putra, T.; Mawengkang, H. The analysis of forward and backward dynamic programming for multistage graph. *Mater. Sci. Eng.* **2018**, *300*, 1–6. [[CrossRef](#)]
30. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.