A User Constraint Awareness Approach for QoS-Based Service Composition

Zhihui Wu, Piyuan Lin(✉), Peijie Huang, Huachong Peng, Yihui He, and Junan Chen

South China Agricultural University, Guangzhou, China
pyuanlin@scau.edu.cn

Abstract. Web service composition adopts functional features including the inputs and outputs, and non-functional features including quality of service (QoS), conditional structure constraints, user preferences, and trusts to compose homogeneous or heterogeneous services together in order to create value-added services. However, in some complex practical application scenarios, the web services with the same function can provide the generous differentiated contents, and there is no approach to focus on the user's constraints on the content provided by the web services. In this paper, we focus on handling three composition dimensions simultaneously including functional features, QoS and the user's constraints on the contents provided by the web services. Therefore, an improved genetic algorithm to obtain an optimal solution for this task is applied. In addition, we also take it into consideration that the over-constrained problem caused by implicit conflicting constraints and improve a constraint correction approach to solve this problem with less cost of consistency checks. Experimental results using the real datasets about travel demonstrate the effectiveness of our approach in creating the fully functional and quality-optimized solutions, on the premise that the users constraints on the content are satisfied.

Keywords: User constraint awareness · Web service composition · Over-constrained problems · Genetic algorithm · Constraint correction

1 Introduction

Nowadays, Service-Oriented Computing (SOC) has been widely employed in many fields and plays an important role in practical applications. Considering the complexity of user requirements, web service composition is an effective and available solution by composing homogenous or heterogeneous services together in order to create value-added services that meet user requirements [1]. Thus, promoting automated web service composition by considering functional and non-functional features has attracted the attention of researchers [2].

The complexity of web service composition lies in the number of distinct dimensions it must simultaneously account for [3]. On the first dimension, services must be combined so that their inputs and outputs are properly linked. In other words, the output produced by a given service can be used as an input to the next service in the combination, ultimately resulting in the desired overall output. On the second dimension, the composition must

J. Miller et al. (Eds.): ICWS 2019, LNCS 11512, pp. 48–62, 2019.
https://doi.org/10.1007/978-3-030-23499-7_4

meet any specified user constraint or preference. A constraint is defined as the user restriction that must be met to make the composited solution valid. On the third dimension, considering the quality of service (QoS) and service trust, the resulting combination must achieve the best overall QoS in terms of time, cost, reliability, etc.

Thus, several techniques have been proposed to address this composition problem, which are QoS-based approaches [4–8], constraint/preference-based approaches [1, 3, 9, 10], and trust-based approaches [1, 11] in the literature. Initially, QoS-based approaches are committed to building functional and quality-optimized applications based on non-functional features. Secondly, considering the degree of user preference for non-functional features (QoS) and user constraint, constraint/preference-based approaches focus on providing personalized web service compositions to users. Thirdly, trust-based approaches further recognize the importance of involving trust in web service composition in generating trustworthy web service compositions. These above works can effectively meet user complex requirements and generate the most suitable service compositions. Despite these advantages, one limitation of existing works is they only consider structure constraints (e.g. the user can specify logical branching) and neglect the user's constraints on the content provided by the web service. For example, a typical composite service is illustrated in Fig. 1 where a user in Canton plans a trip to attend an international conference held in Beijing. It consists of three parts: (1) train/flight services from Canton to Beijing; (2) booking a hotel closer to conference venue; (3) transportation used during the conference period.

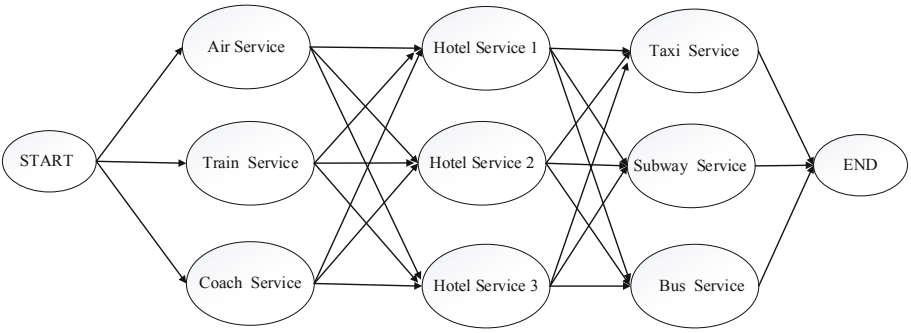

Fig. 1. A typical composite service for a user's trip to attend a conference

We assume that the tuple <Air Service, Hotel Service 1, Taxi Service> is the best solution whose overall quality of service is maximum. However, the user's expected cost does not exceed 1,500 RMB. Obviously, the price of round-trip tickets from Canton to Beijing exceeds the budget. Additionally, Users can express more constraints on the content, such as train type, seat type, hotel price, hotel grade, etc. Hence, implicit conflicting constraints given by the user lead to no solution in this problem. For example, the user's expected cost is less than 1,500 RMB and the expected transportation is by airplane. In this paper, we aim to resolve this problem by integrating the user's constraints on the content provided by the web service into an optimization model.

We apply the genetic algorithm to obtain an optimal solution in which each service can return some contents that meet the user's corresponding constraints. To address the over-constrained problem, the interactive constraint satisfaction problem (ICSP) is used [12]. The experimental results on the real data set demonstrate the efficiency and effectiveness of our method in comparison with other counterparts. In short, we shade light on a new way to promote web service composition by integrating the user's constraints on the content provided by the web service.

In summary, the major contributions of this paper are:

(1) We take it into consideration that the user's constraints on the content provided by the web services in order to express user requirement more completely.
(2) We formulate the service composition as a combined optimization problem that aims to satisfy the user's constraints and achieve the best overall QoS.
(3) We improve a constraint correction approach for solving over-constrained on user requirements.

The remainder of this paper is organized as follows: Sect. 2 gives an overview of related research. Section 3 defines the combined optimization problem addressed in this paper. Section 4 presents the improved genetic algorithm and constraint correction approach. Section 5 showed that where our method can achieve superior performance. Finally, Sect. 6 concludes our present work and outlines future research directions.

2 Related Work

In this section, we discuss previous work related to the two aspects of our approach.

Web Service Composition. In the existing researches, many research works on web service discovery and composition rely on syntax and semantics-based service matchmaking [13]. The effective method is to formulate the web service composition problems as network optimization problems [14]. Service network (SN) approaches have been put forward to deal with this issue in a cost-effective and agile way [15]. And various AI planning methods are used to solve these network problems for discovering optimal solutions with different goals. Wang et al. combined both qualitative and quantitative preferences as well as service trust together in the process of service composition and discovered the best solution by genetic algorithm and simulated annealing algorithm [1]; Rodriguez-Mier et al. used the hybrid local-global search to extract the optimal QoS with the minimum number of services [4]; Wang et al. proposed a dynamic web service composition method based on automatic hierarchical reinforcement learning (HRL) that addresses the problem of low efficiency in web service composition based on traditional reinforcement learning (RL) when applied to large scale services [5]. And Wang et al. proposed a new service composition solution based on deep reinforcement learning (DRL) for adaptive and large-scale service composition problems [6]; Labbaci et al. proposed a deep learning approach for long-term QoS-based service composition [8]; Zhao et al. used the multi-objective reinforcement learning (MORL) algorithm to make trade-offs among user preferences and recommend a collection of services to achieve the highest objective [9]. Broadly, the

above approaches can be classified into three categories, namely QoS-, constraints/preference- and trust-based approaches [1], which choose the optimal web service combination according to QoS, user preference and service trust. Different from the researches discussed above, we pay attention to the service composition problem in application scenarios such as in a tour where the user's constraints on the content provided by the web service are important and non-ignorable.

User Constraints Problem. User constraints that determine the complexity of the problem are the root cause for cumbersome burdens of users. Silva et al. presented a genetic programming approach that addresses the web service composition problems including conditional constraints. But these conditional constraints are branching constraints [3]. Najar et al. focused on predicting the user's future intention based on his/her context, in order to offer him the most suitable service considering his/her incoming constraints [16]. But this approach is based on the assumption that, even in a dynamic and frequently changing pervasive information system, common situations can be found. Zhao et al. mined user intents using natural language processing techniques [17]. However, in practical application scenarios, users constraints sometimes are too tight. Therefore, there are maybe over-constrained problems that lead to no solution. ICSP corrects over-constrained problems by computing maximal satisfiable subset (MSS) or minimal correction subset (MCS) of users' constraints, which ensure that the original constraints are retained as much as possible under the circumstance that there is at least one solution [18]. Alessandro et al. proposed a novel approach to speeding up MCS enumeration over conjunctive normal form propositional formulas by caching of so-called premise sets seen during the enumeration process [19]. And Nina et al. proposed a new algorithm named FLINT for extracting minimal unsatisfiable cores and correction sets simultaneously [20]. But these works are not applicable to our task by considering that in our context different user preferences for constraints need the optimal MSS/MCS, not all MSS/MCS enumeration. Therefore, the algorithms proposed by Li et al. are more suitable, who generate the optimal MSS by taking into account different user preferences for constraints [18].

3 Problem Description

The objective of web service composition is to create a new, composite service that accomplishes a given task. A user sends a composition request that is automatically processed by a system that then returns an application assembled using a set of atomic services. In the web service composition, an atomic web service can be represented as $S = (\text{input}, \text{output}, QoS(q_1,\ldots,q_n))$, where n is the number of QoS attributes considered. Services are selected from a service repository $SR = \{S_1, \ldots, S_m\}$(where m is the number of services in the repository) and combined according to the specifications of a composition request $R = (\text{input}, \text{output})$ in order to produce a solution with the desired overall outputs. In certain cases, however, the customer may have specific constraints $C = \{C_1, \ldots, C_g\}$ on the content returned by the web service (where g is the number of constraints). For example, the customer's preferred class of airline ticket is likely to depend on his/her current budget: if the customer has enough money to pay for the

book, then he/she would like to book the first-class cabin. However, if no first class tickets available in the flight service A. In this case, the flight service A is not a good choice. Specifically, we divide the constraints into 2 parts: the local constraints and the global constraints. The local constraints apply only to a single service, like the hotel price. And the global constraints work on the web services combination, such as the expected total cost. On the premise that the user's constraints on the content returned by the web service are satisfied, the objective is to produce a composition solution with the highest possible quality, which is optimized by calculating the weighted sum of a set of objective functions $f_1, \ldots, f_n$, corresponding to the different QoS attributes considered.

In our work, three popularly considered attributes [21] have been considered: availability (A), the probability of a service immediately responding to a request; reliability (R), the probability that the response produced by the service is accurate; time (T), the overall execution time for responding a request. Besides, we apply the number of solutions provided by a composition which satisfies the user constraints as a new QoS index, named N.

The overall QoS for a composition is determined by two aspects: the QoS values of the single services within it and the structure of the workflow, which are based on existing composition languages [22]. In our work, two constructs, sequence and parallel [21] are considered.

(1) Sequence construct. The services in this construct are chained together so that the outputs of the preceding service can fulfill the inputs of the next one, as shown in Fig. 2. The overall A and R probabilities are calculated by multiplying the individual values associated with each service in the construct, and the overall T is calculated by adding up the individual values. And N is calculated by the following methods. Firstly, considering the local constraints, the overall N is calculated by multiplying the number of the content provided by each service in the construct. Therefore, n_k must be greater than 0. Next, we will remove some solutions that do not meet the global constraints. Finally, we will normalize N to [0, 1] by the maximum and minimum normalization.
(2) Parallel construct. As seen in Fig. 3, the parallel constructs allow the services to be executed in parallel, which means that their inputs are independent, so their outputs are also independent. While A, R, and N are still calculated in the same way as the sequence construct, T is obtained by the longest service execution time in the construct.

4 Design of Proposed Approach

4.1 Overview of the Framework

Figure 4 shows the overview of our framework. At a high level, the three components of our proposed approach are the service network graph, the genetic programming for generating an optimal solution, and the over-constrained problems solver.

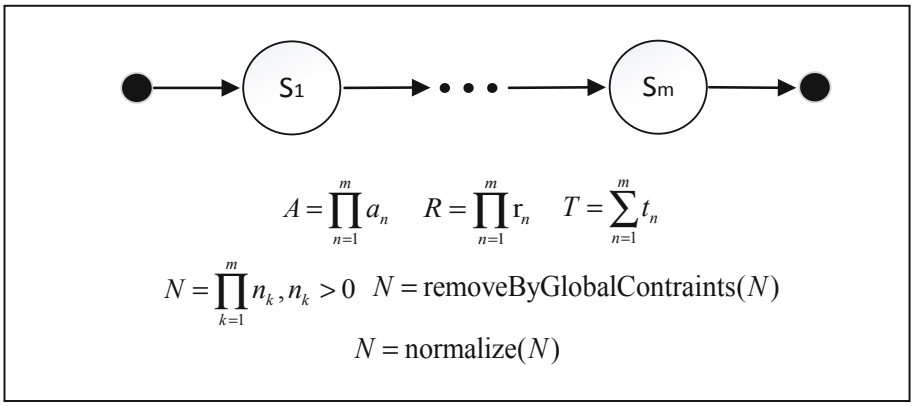

Fig. 2. Sequence construct and formulae for calculating its traditional QoS properties [21].

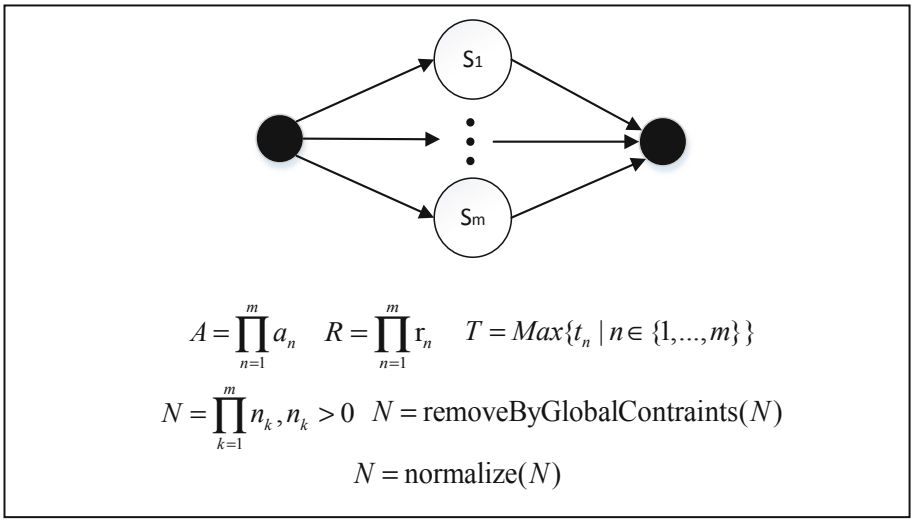

Fig. 3. Parallel construct and formulae for calculating its traditional QoS properties [21].

The process begins when the user provides a composition request R and the expression of user constraints. First, we use the composition request R to identify the service layers and extract related service network graph. In this graph, the single web service is represented by the big orange circle when the small cyan circle represents the input/output of a service or the parameters of composition request R. Then, the expression of user constraints is considered when generating the solution with the optimal overall QoS for the service composition task. However, if there is no solution due to too tight constraints, the over-constrained problems solver will detect conflicts among user constraints and computing maximal relaxation of user original constraints based on user's preference on constraints. Thus, the user original constraints are

updated by the new constraints and applied when generating the optimal solution for the service composition task again. Finally, the optimal solution that meets the user original/new constraints is returned to users.

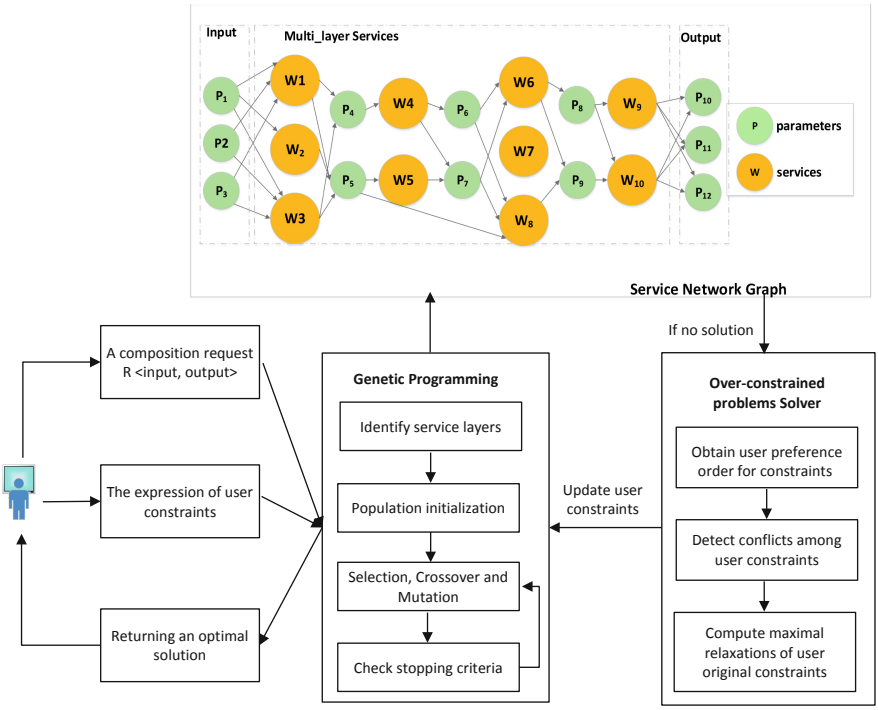

Fig. 4. An overview of our proposed framework.

4.2 Genetic Programming to Generate an Optimal Solution

Our proposed approach employs Genetic Programming to evolve solutions according to their overall QoS while maintaining their functional correctness. Specifically, it consists of three important parts: the identification of service layers, the population initialization and the genetic operators.

Identification of Service Layers. Before the composition process takes place, it is necessary to determine which layer each candidate service belongs to. This is done to prevent cycles from forming during the solution decoding process when using a backward graph-building algorithm. The identification process is detailed in [23]. The overall idea is to iteratively classify services in sets, each time finding a new set of services (i.e. new layer) whose inputs can be entirely fulfilled by the previously found services.

Fitness Function. The fitness function is used to measure the overall quality of a candidate composition. The traditional function is to calculating the weighted sum of the several overall composition QoS attributes [24]. In this paper, we apply the number of solutions provided by a composition satisfied by the user constraints as a new QoS attributes, named S. Therefore, our fitness function must take this new QoS attribute into account, as the following equation shown:

$$fitness_i = W_1(1 - A_i) + W_2(1 - R_i) + W_3 T_i + W_4(1 - N_i) \tag{1}$$

where $\sum_{i=0}^{4} w_i = 1$. The value of the function is in the range of 0 to 1, and the smaller the better. The A, R, T and S values for each candidate composition are computed by the formula shown in Figs. 2 and 3. And S describes the number of solutions and are normalized by using the smallest (Min) and largest (Max) respective values, as the Eq. 2 shown. N_{orig} is the number of solutions provided by the service composition under the user constraints before normalization. Especially, the smallest value of N is 0 corresponding to over-constraints problems and the largest value of N is corresponding to the unconstrained problem and determined by the dataset.

$$N = normalize(N) = \frac{N_{orig} - N_{\min}}{N_{\max} - N_{\min}} \tag{2}$$

Population Initialization. As opposed to generating a composition candidate purely based on a set of available inputs and another of desired outputs, when handing user constraints on the content returned by the web service it is necessary to ensure that selected web services can return content that meets user constraints. Thus, when facing a web service, we first check the consistency between user constraints and the web service. If false, remove it directly. Otherwise, the fitness value will be considered. When choosing the web service in the zeroth layer, a random strategy is adapted to select the service from a set of services that satisfy user constraints. Then, we use a greedy strategy to select services on the remaining layers. Specifically, we traverse all the available services in this layer and then calculate the fitness of the current service composition. After selecting n service combinations with the lowest fitness as candidates, the roulette strategy is adopted to randomly select one from candidates.

Crossover and Mutation. The crossover operator employed in the evolutionary process exchanges node fragments from two individuals with common service nodes. If the generated offsprings are satisfied with the user constraints, they are retained, otherwise, the parents are retained. In the mutation operation, we first traverse all the service nodes on the current service composition and check the consistency between user constraints and the web service and calculates the fitness of each available service on the layer where the service node is located. If the user constraints are satisfied and the fitness is higher than the original fitness, the original service node is replaced with the new node.

4.3 Solving Over-Constrained Problems

The main cause of the over-constrained problems is that the user constraints are too tight in the actual application scenarios. An effective way to solve this problems is that the user should modify or remove some constraints. However, user is not familiar with the main cause of the over-constraint problem. In other word, users do not know which constraints resulted to the over-constrained problems. Therefore, it may occur that the over-constrained problems still exist even though the users have modified or removed certain irrelevant constraints. In order to ensure that the original constraints of users are not changed as much as possible and accelerate user planning by reducing retrieval burdens, the interactive constraint satisfaction problem (ICSP) is adopted to describe and solve the over-constrained problems in this paper.

ICSP has many applications in Artificial Intelligence. Its interactive applications often require suggestions from a system to help a user solve the problem. It was developed from a constraint satisfaction problem (CSP) [25]. ICSP consists of four fundamental components denoted as (X, D, U ∪ B). X is a set of n variables X = {x_1, x_2,…, x_n}, D is a set of domains D = {dom(x_1), dom(x_2),…, dom(x_n)} where dom(x_i) is a finite set of possible values for variable x_i, B is the set of background constraints which are generated from the technical characteristics of the problem and cannot be modified and U is the set of user constraints which represent users constraints and can be modified [18]. There are two main strategies for ICSP to solve over-constrained problems: Junker U computed the minimal and preferred conflicts for solving over-constrained problems [26] and Li et al. proposed two algorithms to generate Corrective Explanations by computing maximal relaxations called CORRECTIVERELAXREDUCED (CER) and CORRECTIVERELAXDC (CEDC) respectively [18]. And the CEDC algorithm performs best in reducing consistency checks when generating corrective explanations. In this paper, we have improved the algorithm CEDC to make it outperformed than other algorithms.

Compared to CEDC, the improved CEDC, named Imp_CEDC which uses an extra space to record the consistency status of previously checked constraints. This strategy effectively reduces duplicate checks and therefore the number of consistency checks is less than CER. We used the Imp_CEDC algorithm to compute maximal relaxations of user constraints when facing the over-constrained problems. At the same time, Imp_CEDC takes it into account that different user preferences for constraints. Therefore, Imp_CEDC will preserve the user preferred constraints as much as possible.

The method Imp_CEDC is depicted in Fig. 5. Users give a set of user constraints U which result in over-constrained problems. The algorithm first checks if the RECORD contains the consistency state of the current constraint U. If true, then skip the method Consistent. U is divided into Δ1 and Δ2 by Imp_CEDC. Then, it calculates the relaxation of Δ1 and Δ2 respectively, and ultimately combines them to obtain the relaxation of U. The method named Split is to divide the user's constraint set into two equal parts. And the method named Consistent is aimed to check the consistency of a set C of constraints. Based on the current constraints, it returns true if at least one solution can be generated.

```
global RECORD
Algorithm: Imp_CEDC (R, U, B)
Input: The current relaxation R; the ordered set of constraints being tested in this
invocation U; and the set of background constraints B.
Output: A maximal relaxation of U.
if R ∪ U in RECORD.keys then
      status = RECORD [(R ∪ U)]
else
      RECORD [(R ∪ U)] = Consistent (R ∪ U ∪ B)
      status = RECORD [(R ∪ U)]
end if
if status then
     return R ∪ U
else if len(U) > 1 then
      Let(Δ1,Δ2)← Split (U) ,
      R = Imp_CEDC (R, Δ1, B),
      R = Imp_CEDC (R, Δ2, B)
end if
return R
```

Fig. 5. Imp_CEDC Algorithm.

5 Experiments

5.1 Datasets

To evaluate the performance of our proposed approach, we applied it to real datasets about travel crawled from Ctirp, 12306, Nuomi, Elong, etc. This datasets are composed of the information of transportation, restaurants, hotels, and attractions. In this experiment, we assume that a user in Guangzhou is planning a day trip to Hangzhou. Therefore, there are three types of transport tool from Guangzhou to Hangzhou: coach, trains and airplanes. In addition, based on some attributes of the content like seat type, a user can have dozens of choices for transportation. Then, we have collected 635 restaurants information, 2164 hotels information and 436 attractions information. They have a variety of different attributes, including price, user rating, location, style, tags, etc.

5.2 Experiment Results

The QoS-based Composition Tasks. In order to show the efficiency of the proposed algorithm, we compare our proposed algorithm with the state-of-the-art algorithms named GP, proposed by Silva et al. [3]. In this experiments, we generate K web services with the inputs, outputs, and additional QoS attributes automatically whose type belongs to one of the transportation, restaurant, hotel, and interest. And we randomly assign the contents to these services. The range of K is 100–1000. Experiment results are shown in Figs. 6 and 7. Figures 6 and 7 display the mean best fitness values and the mean best numbers of solutions separately when running the QoS-based composition tasks over 20 runs with the horizontal ordinate showing the number of web services.

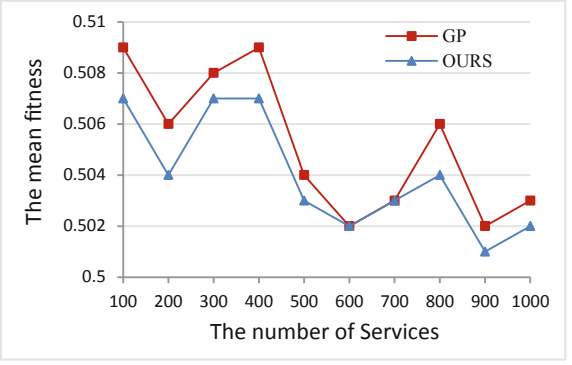

Fig. 6. Mean for the fitness of each approach.

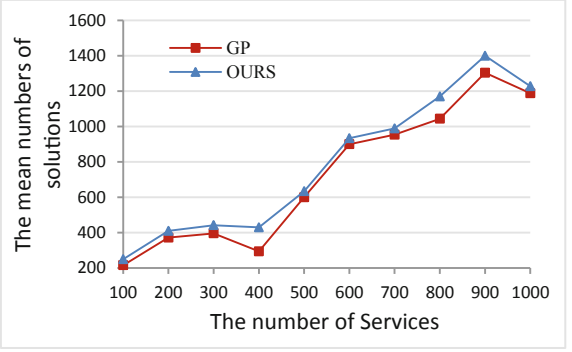

Fig. 7. Mean for the numbers of solutions of each approach.

Solving Over-constrained Problems. In this experiment, to verify the effectiveness of the approach to solve over-constrained problems, firstly, we created dozens of user constraints sets, each of which leads to over-constrained problems. Specifically, M user

constraints on the contents are generated randomly, and the range of M is 3-10 in our experiment. And we assume that, users will correct some of the constraints by cumulatively removing constraints just based on their own preference for the constraints when facing the over-constraint problem in practice. Therefore, we simulated user's actual operation only using the user's preference for constraints as the strategy of removing the requirements, called User_TRY approach. The Figs. 8 and 9 contain the results for running the over-constrained tasks, and the effectiveness is compared among our algorithm Imp_CEDC, the algorithm CEDC and User_TRY. The average number of remaining constraints is shown in Fig. 7 while Fig. 8 showing the mean number of consistency checks and their abscissas both represent the number of user original constraints.

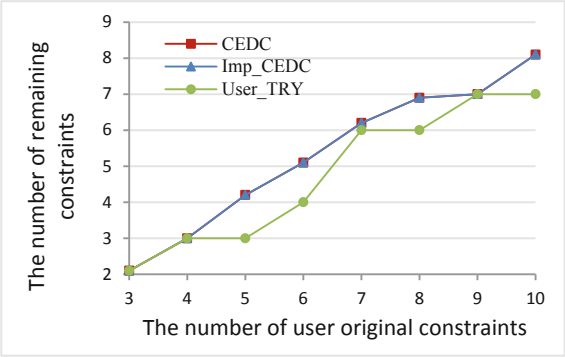

Fig. 8. Mean for the number of remaining constraints of each approach.

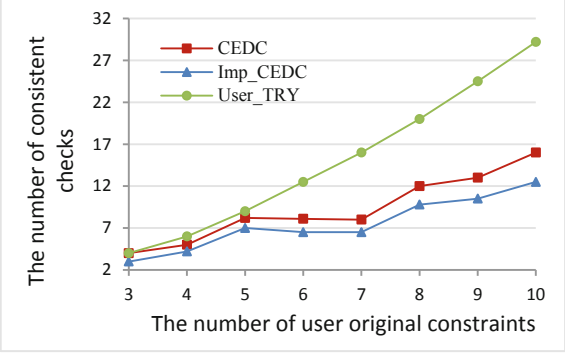

Fig. 9. Mean for the number of consistency checks of each approach.

Discussions. For the QoS-based composition tasks, we assume that these fitness values are quite close to the global optimal solution for each task, though the exact optimal values are not known. The fitness of GP and ours are calculated using the same function, and the results shown in Fig. 6 indicate that our algorithm performs better with the better fitness than GP on each dataset. In terms of the mean numbers of

solutions, we can see that more solutions are generated by our algorithm compared to GP from Fig. 7.

When solving the over-constrained problems, it is observed that as the number of user original constraints grows, the number of user constraints to be remained are equal between CEDC and Imp_CEDC from the Fig. 8. However, Fig. 9 shows that the number of consistency checks of Imp_CEDC is less than CEDC. In other words, Imp_CEDC can give users the same suggestions at less cost and faster speed than CEDC. In addition, we can observe that when the number of user original constraints is small, the user can find the conflicts among their constraints with less number of trial. But with the number of user original constraints grows, the cost for users to correct their original constraints without any system suggestion became increasingly expensive, that shown by the higher number of consistency checks and the lower number of remained user constraints.

6 Conclusion

In this paper, we presented an improved genetic algorithm to QoS-aware automated Web service composition. The novelty of this approach is that it addresses three composition dimensions simultaneously: functional features, QoS and the user's constraints on the content provided by the web services. In addition, we also took it into consideration which the over-constrained problem caused by implicit conflicting constraints and an improved constraint correction approach with less cost of consistency checks was proposed to address this problem. No dataset with conditional tasks was available to test this approach, therefore the real datasets about travel crawled from the Internet were extended and used for this purpose. Results showed that the proposed approach was capable of identifying solutions that are fully functional and quality-optimized, on the premise that the user's constraints on the content are satisfied. While facing the over-constrained problem, the experimental results show that the efficiency of our method in reducing the cost of consistency checks, compared with other counterparts.

Acknowledgments. The research work was supported by National Natural Science Foundation of China under Grant No. 71472068.

References

1. Wang, H., Zou, B., Guo, G., Zhang, J., Yang, Z.: Optimal and effective web service composition with trust and user preference. In: Proceedings of the 22th IEEE International Conference on Web Services (ICWS 2015), pp. 329–336 (2015)
2. Lamparter, S., Ankolekar, A., Studer, R., Grimm, S.: Preference-based selection of highly configurable web services. In: Proceedings of the 16th International Conference on World Wide Web, pp. 1013–1022 (2007)
3. da Silva, A., Ma, H., Zhang, M.: A GP approach to QoS-aware web service composition including conditional constraints. In: Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC 2015), pp. 2113–2120 (2015)

4. Rodriguez-Mier, P., Mucientes, M., Lama, M.: A hybrid local-global optimization strategy for QoS-aware service composition. In: Proceedings of the 22th IEEE International Conference on Web Services (ICWS 2015), pp. 735–738 (2015)
5. Wang, H., Huang, G., Yu, Q.: Automatic hierarchical reinforcement learning for efficient large-scale service composition. In: Proceedings of the 23th IEEE International Conference on Web Services (ICWS 2016), pp. 57–64 (2016)
6. Wang, H., Gu, M., Yu, Q., Fei, H., Li, J., Tao, Y.: Large-scale and adaptive service composition using deep reinforcement learning. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 383–391. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_27
7. Wang, H., Chen, X., Wu, Q., Yu, Q., Zheng, Z., Bouguettaya, A.: Integrating on-policy reinforcement learning with multi-agent techniques for adaptive service composition. In: Franch, X., Ghose, Aditya K., Lewis, Grace A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 154–168. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45391-9_11
8. Labbaci, H., Medjahed, B., Aklouf, Y.: A deep learning approach for long term qos-compliant service composition. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 287–294. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_20
9. Zhao, Y., Wang, S., Zou, Y., Ng, J., Ng, T.: Automatically learning user preferences for personalized service composition. In: Proceedings of the 24th IEEE International Conference on Web Services (ICWS 2017), pp. 776–783 (2017)
10. Mistry, S., Bouguettaya, A., Dong, H., Erradi, A.: Probabilistic qualitative preference matching in long-term iaas composition. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 256–271. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_18
11. Paradesi, S., Doshi, P., Swaika, S.: Integrating behavioral trust in web service compositions. In: Proceedings of the 16th IEEE International Conference on Web Services (ICWS 2009), pp. 453–460 (2009)
12. Freuder, E., Mackworth, A.: Constraint satisfaction: an emerging paradigm. Found. Artif. Intell. **2**, 13–27 (2006)
13. Zhang, J., et al.: A bloom filter-powered technique supporting scalable semantic service discovery in service networks. In: Proceedings of the 23th IEEE International Conference on Web Services (ICWS 2016), pp. 81–90 (2016)
14. Oh, S., Lee, D., Kumara, S.: Effective web service composition in diverse and large-scale service networks. IEEE Trans. Serv. Comput. **1**, 15–32 (2008)
15. Wang, S., Wang, Z., Xu, X.: Mining bilateral patterns as priori knowledge for efficient service composition. In: Proceedings of the 23th IEEE International Conference on Web Services (ICWS 2016), pp. 65–72 (2016)
16. Najar, S., Pinheiro, M.K., Souveyet, C.: A context-aware intentional service prediction mechanism in PIS. In: Proceedings of the 21th IEEE International Conference on Web Services (ICWS 2014), pp. 662–669 (2014)
17. Zhao, Y., Wang, S., Zou, Y., Ng, J., Ng, T.: Mining user intents to compose services for end-users. In: Proceedings of the 23th IEEE International Conference on Web Services (ICWS 2016), pp. 348–355 (2016)
18. Li, H., Shen, H., Li, Z., Guo, J.: Reducing consistency checks in generating corrective explanations for interactive constraint satisfaction. Knowl.-Based Syst. **43**, 103–111 (2013)
19. Alessandro, P., Carlos, M., Matti, J., Joao, M.: Premise set caching for enumerating minimal correction subsets. In: Proceedings of the 32nd National Conference on Artificial Intelligence (AAAI 2018), pp. 6633–6640 (2018)

20. Nina, N., Nikolaj, B., Maria-Cristina, M., Mooly S.: Core-guided minimal correction set and core enumeration. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018), pp. 1353–1361 (2018)
21. Al-Masri, E., Mahmoud, Q.H.: QoS-based discovery and ranking of web services. In: Proceedings of the 16th International Conference on Computer Communications and Networks (ICCCN 2007), pp. 529–534 (2007)
22. Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A.: Analysis of web services composition languages: the case of BPEL4WS. In: Proceedings of the 22th International Conference on Conceptual Modeling (ER 2003), pp. 200–215 (2003)
23. da Silva, A., Mei, Y., Ma, H., Zhang, M.: A memetic algorithm-based indirect approach to web service composition. In: Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC 2016), pp. 3385–3392 (2016)
24. da Silva, A., Hui, M., Zhang, M.: A graph-based particle swarm optimisation approach to QoS-aware web service composition and selection. In: Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC 2014), pp. 3127–3134 (2014)
25. Jannach, D., Zanker, M., Fuchs, M.: Constraint-based recommendation in tourism: a multiperspective case study. Inf. Technol. Tourism **11**, 139–155 (2009)
26. Junker, U.: QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI 2004), pp. 167–172 (2004)