A Hybrid Local-Global Optimization Strategy for QoS-Aware Service Composition

Pablo Rodriguez-Mier*, Manuel Mucientes*, Manuel Lama*
*Centro de Investigación en Tecnologías de la Información (CITIUS)
Universidad de Santiago de Compostela, E-15782 Spain
{pablo.rodriguez.mier,manuel.mucientes,manuel.lama}@usc.es

***Abstract*—This paper presents a hybrid approach for automatic composition of Web services that generates semantic input-output matching compositions minimizing the number of services and optimizing the global QoS. The proposed approach has four main steps: 1) generation of the composition graph for a request; 2) computation of the optimal QoS of the composition graph; 3) multi-step optimizations of the graph to identify equivalent and dominated services; and 4) hybrid local-global search to extract the optimal QoS with the minimum number of services. A validation with the datasets of the Web Service Challenge 2009-2010 is also provided.**

Keywords*—*Service Composition; Hybrid Optimization; QoS

I. Introduction

As the number of available services on the Internet grows, the need to develop efficient composition algorithms that can 1) select and combine different services to build rich functionalities and 2) deal with a large number of services ensuring optimal Quality-of-Service (QoS) becomes a challenging task. This has motivated researchers to explore different strategies to efficiently generate QoS-aware Web service compositions from different perspectives [1]. One common approach is to consider the QoS composition problem as a Knapsack Problem [1], where the goal is to select a combination of services for a predefined workflow that optimizes some global QoS utility function subject to different constraints [2]. These approaches do not contemplate how to automatically generate the composition workflow, so the total number of services in the composition is fixed. Other approaches [3], [4], [5] consider a broader problem by not only optimizing the total QoS but also building the entire composition in an automatic fashion. However, the problem of building the composition with the minimum number of services that guarantee the optimal end-to-end QoS is rarely considered. For example, in [3] the authors analyze the problem of generating top *K* query compositions by relaxing the optimality of the QoS in order to introduce service variability. The problem is that the alternatives are generated at the expense of worsening the optimal QoS instead of looking first for other composition alternatives that guarantee the optimal QoS. Other authors, such as the top-3 of the Web Service Challenge 2009-2010 [6], [4], [5], proposed efficient techniques to optimize a single QoS parameter. However, these approaches do not find optimal compositions in terms of number of services and also do not contemplate pruning techniques to improve the scalability.

This paper focuses on the automatic generation of semantic input-output based compositions with optimal QoS, minimizing the total number of services. We present a hybrid optimization algorithm for QoS-aware semantic service composition. It combines a near-optimal heuristic local search with a global combinatorial search in order to retrieve the optimal composition in terms of the number of services and the global QoS for a given semantic input-output composition request. The approach works as follows: given a composition request, a directed graph with the relevant services for the request is generated. Once the graph is built, an optimal label-correcting forward search is performed in polynomial time in order to compute the global optimal QoS. This information is used later in a multi-step pruning phase to remove sub-optimal services. Finally, a hybrid local/global search is performed within a fixed time limit to extract the optimal solution from the graph. The main contributions of this paper are threefold:

- Multi-step optimizations based on the analysis of non-relevant, equivalent and dominated services in terms of interface functionality and QoS, which is an extension of our previous work [7].
- A fast local search strategy to obtain a near-optimal number of services while satisfying the optimal end-to-end QoS for an input-output based composition request.
- An heuristic guided combinatorial search that combines global QoS bound propagation with sub-optimal service pruning to improve the solution obtained by the local search strategy.

Experimental results with the Web Service Challenge 2009-2010 show that the proposed approach is able to solve effectively all the datasets. The rest of the paper is organized as follows: Sec. II introduces the composition problem, Sec. III describes the proposed approach, Sec. IV presents the results obtained, and Sec. V gives some final remarks.

II. Problem Description & Motivating Example

In a nutshell, the composition problem tackled in this paper can be informally described as follows: Given a semantic description of the composition problem $R = \{I_R, O_R\}$, where I_R are the inputs provided and O_R the outputs expected, and a set of available services with different quality properties (e.g., response time, throughput...), a composition graph is computed layer by layer. This graph contains a set of services $w = \{I_w, O_w, q_w\}$ –where I_w is the set of inputs required by the service, O_w is the set of outputs returned, and q_w is the QoS property associated to the service– and all the possible matches between outputs and inputs of services. A

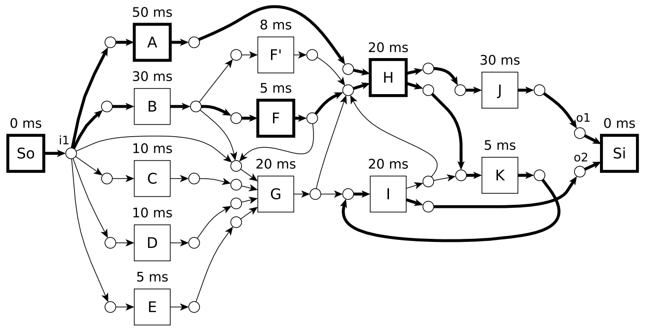

Fig. 1. Composition graph example with the solution with optimal QoS and minimum number of services highlighted.

match between a semantic output O_{w_1} of a service w_1 and a semantic input I_{w_2} of a service w_2 ($match(O_{w_1}, I_{w_2})$) exists iff $(O_{w_1} \equiv I_{w_2}) \vee (O_{w_1} \sqsubseteq I_{w_2})$, i.e., the output concept is equivalent to the input concept (exact match) or the output concept is a subclass of the input concept (plugin match). The graph also contains two dummy services $So = \{\emptyset, I_R\}$ and $Si = \{O_R, \emptyset\}$, where So is the source dummy service and Si is the sink dummy service.

Fig. 1 shows an example of a composition graph for a request $R = \{\{i1\}, \{o1, o2\}\}$ where the services are associated with response time values. Each service is represented by squares, whereas inputs and outputs are represented by circles. As can be seen, this graph contains many different compositions since there are inputs in the graph that can be matched by the outputs of different services. The total response time of the example is 100 ms, because the optimal cost for $o1$ is 100 ms and for $o2$ is 30 ms. This can be easily computed by applying the adequate aggregation functions depending on the type of the QoS properties [8] and computing the best QoS of the graph, which can be calculated in polynomial time [9]. However, there can exist multiple combinations of services that satisfy the composition request subject to the optimal QoS constraint, which in this case is 100 ms. For example, in Fig. 1, the optimal solution (highlighted) is composed of 7 services, but there is for example another one with 8 services and the same QoS. Selecting the minimum number of services for the optimal QoS is a hard combinatorial problem. The goal of this paper is to find the composition with the minimum number of services that satisfy the request with an optimal QoS.

III. Algorithm for QoS-aware Service Composition

The proposed approach has four steps: 1) calculation of the composition graph for a request; 2) computation of the optimal QoS of the composition graph; 3) optimization of the composition graph; 4) hybrid search to extract the optimal combination of services.

A. Generation of the composition graph

Given a composition request, the generation of a composition graph with the relevant services for the request is calculated by selecting all invocable services layer by layer, starting with So in the first layer and terminating with Si in the last layer [7]. Once all possible services are selected, a match between all outputs and inputs of services is computed (thus, the resulting graph can contain cycles). The output of this step is a graph such as the one represented in Fig. 1.

B. Optimal end-to-end QoS

The optimal QoS for a composition graph can be computed in polynomial time using a shortest path algorithm to calculate the best accumulated QoS cost for each input and output of the graph. In order to compute the optimal QoS, we use a Dijkstra-based label-setting algorithm from Si to So [10]. The output of this step is the best accumulated QoS cost for each input and output in the graph.

C. Graph optimizations

Once the optimal QoS for each input and output of the graph is known, we perform a multi-step optimization to reduce the size of the graph. At each step, the algorithm analyzes different criteria to identify services that are redundant or can be substituted by better ones, so the size of the graph decreases monotonically in each step. The steps that are sequentially applied are: 1) pruning of services that lead to suboptimal QoS; 2) elimination of services that do not contribute to the outputs of the request; 3) combination of interface / QoS equivalent services; and 4) replacement of interface / QoS dominated services. The first step uses the information of the optimal QoS for each input and output, which has been previously calculated. The algorithm is applied backwards, from Si to So, by propagating the optimal QoS bounds for each input. This allows to detect and remove the services from the graph that cannot be part of the final composition. The remaining optimization steps are described in detail in [11].

D. Hybrid search

Each input in the graph may be matched by many outputs from different services. Thus, there may be multiple combinations of services that satisfy the composition request with same or different QoS. The goal of the hybrid search is to extract good solutions from the composition graph, optimizing the total number of the services involved while keeping the optimal QoS. The hybrid search performs a local search to extract a good solution and then it tries to improve the solution by running a global search in the remaining time.

1) Local search with backtracking: Fig. 2 shows the pseudocode of the local search strategy. The algorithm starts with a composition graph, the unresolved inputs of the service Si (the expected outputs of the request) and the service Si selected to be part of the solution. Using the list of the unresolved inputs to be matched, the method RANK-RESOLVERS returns a list of services that match any of the unresolved inputs. Services are ranked according to the number of unresolved inputs that can match, so the service whose outputs match more inputs is considered first to be part of the solution. Then, for each input that the selected service can match, the method CYCLE performs a forward search to check if the selected service can safely resolve the input, without generating cyclic dependencies. For example, in Fig. 1, if we select the service K to match the input of I after having decided to resolve the input of K with the service I, we end up with an invalid composition, so K is an invalid resolver for I and must be

```
function LOCAL-SEARCH($graph$)
    return LS-BACKTRACK($graph$, INPUTS($Si$), $\{Si\}$)

function LS-BACKTRACK($graph$, $unresolved$, $services$)
    if $unresolved = \emptyset$ then return $graph$
    $services \leftarrow$ RANK-RESOLVERS($services$,$unresolved$)
    for each $w \in services$ do
        $resolved \leftarrow \{\}$
        $matched \leftarrow$ MATCH($w$, $unresolved$)
        for each $input \in matched$ do
            if $\neg$CYCLE($w$, $input$) then
                $resolved \leftarrow resolved \cup input$
        if $resolved \neq \emptyset$ then
            $unresolved \leftarrow unresolved \setminus resolved$
            if $w \notin services$ then
                $unresolved \leftarrow unresolved \cup$ INPUTS($w$)
            $g \leftarrow$ RESOLVE($graph$, $w$, $resolved$)
            $result \leftarrow$ LS-BACKTRACK($g$, $unresolved$, $services \cup w$)
            if $result \neq fail$ then return $result$
    return $fail$
```

Fig. 2. Local search algorithm to extract a composition from a graph

```
function GLOBAL-SEARCH($graph$)
    $Q_{opt} \leftarrow$ QOS-UPDATE($graph$)
    $selected \leftarrow \{Si\}$
    $max \leftarrow$ MAX-QOS(INPUTS($Si$), $Q_{opt}$)
    for $i_{Si} \in$ INPUTS($Si$) do
        $unresolved[i_{Si}] \leftarrow [Q_{opt}[i_{Si}], max]$
    $queue \leftarrow$ INSERT($\langle graph, unresolved, Q_{opt}, selected\rangle$,$queue$)
    while $queue \neq \emptyset$ do
        $\langle graph, unresolved, Q_{opt}, selected\rangle \leftarrow$ POP($queue$)
        if $unresolved = \emptyset$ then return $graph$
        $input \leftarrow$ SELECT-UNRESOLVED($graph$)
        $[Q_{lmax}, Q_{gmax}] \leftarrow unresolved[input]$
        for $w \in$ RESOLVERS($input$, $[Q_{lmax}, Q_{gmax}]$) do
            if $\neg$CYCLE($w$,$input$) then
                $successor \leftarrow$ RESOLVE($graph$, $w$, $\{input\}$)
                $unresolved \leftarrow$ REMOVE($i$, $unresolved$)
                $Q_{out_w} \leftarrow$ QOS-OUTPUT($w$, $Q_{opt}$)
                if $Q_{out_w} > Q_{lmax}$ then
                    $Q_{opt} \leftarrow$ QOS-UPDATE($successor$)
                if $w \notin selected$ then
                    $[Q^w_{lmax}, Q^w_{gmax}] \leftarrow$ BOUNDS($w$,$Q_{gmax}$)
                    for $i_w \in$ INPUTS($w$) do
                        $unresolved[i_w] \leftarrow [Q^w_{lmax}, Q^w_{gmax}]$
                $u \leftarrow selected \cup w$
                $queue \leftarrow$ INSERT($\langle successor, unresolved, Q_{opt}, u, queue$)
    return $fail$
```

Fig. 3. Global search algorithm to extract the optimal composition

discarded. Once all resolvable inputs are collected in $resolved$, the method RESOLVE creates a copy of the current graph where the inputs in $unresolved$ are matched only by the selected service, so any other match to that each resolved input is removed from the graph. If the selected service has not already selected, then all its inputs are marked as unresolved and a recursive call to LS-BACKTRACK is performed to select a new service to resolve the remaining inputs, until a solution is found. If a dead end is reached (a solution that has no services to resolve the remaining inputs without cycles) the algorithm backtracks to a previous state to try a different service.

2) Global search: The goal of this algorithm is to extract the composition with the minimum number of services and optimal QoS. This search is based on the observation that each input of the graph can have different QoS bound constraints. In order to discover which are the valid and invalid candidates to resolve an input, the algorithm needs to propagate a global QoS bound constraint for each input $i_w \in w$. This interval is defined as $Q_{i_w} = [Q_{lmax}, Q_{gmax}]$. The lower bound Q_{lmax} represents the maximum local cost of the QoS that does not affect the output cost of service w, i.e., any output of a service with an accumulated cost Q such that $Q = Q_{lmax}$ can be used to match the input without affecting the optimal output cost of the service. If this bound is exceeded, the output cost of the service increases, and an optimal QoS update should be performed to recompute the costs of each input/output in the graph. The upper bound Q_{gmax} represents the maximum global cost for the input that cannot be exceeded without worsening the QoS cost of the overall composition. These bounds are used to detect prune sub-optimal candidates during the global search.

Fig. 3 shows the pseudocode of the global search strategy. The algorithm starts computing the optimal QoS cost for each input and output by calling QOS-UPDATE. This method returns a hash table Q_{opt} where the keys are the inputs and outputs and the values are the optimal QoS cost for each input/output. Then, the dummy service Si is selected to be part of the composition and its inputs are stored in the hash table $unresolved$. At this time, we compute the QoS bound constraints for the unresolved inputs. For example, in Fig. 1, the bounds of the input $o1$ of Si are $[100, 100]$ and the bounds of $o2$ are $[30, 100]$ ms. All this information is inserted in a priority queue sorted by the total number of services selected, so the solutions with less services are always extracted first. Then, for each solution in the queue, we heuristically select one of the unresolved inputs (method SELECT-UNRESOLVED) using a minimum-remaining-values heuristic. This heuristic selects always the input with less resolvers (services). Then, for each possible service that resolves the selected input, a new composition graph with the selected input resolved is created, and the resolved input is removed from the list. If the output of the selected service exceeds the value of Q_{lmax} of the resolved input, then a QoS update is performed to recompute the new QoS costs. Finally, if the selected service was not selected before (is not in the $selected$ list), then its inputs are added to $unresolved$ and the new bounds for these inputs are computed with the method BOUNDS. This method *propagates* the bounds of the QoS constraints by subtracting $Q_{gmax} - q_w$, where q_w is the QoS property of the selected service. The subtraction function, as in the case of the QoS aggregation function, depends on the type of the QoS, which in the case of the response time is a standard subtraction for real numbers and in the case of the throughput is the min function. For example, if we resolve the input $o2$ (with bounds $[30, 100]$ ms) of Si with the service I, whose response is 20 ms, the bounds for the input of I would be $[30, 100 - 20]$ ms. What this basically means is that, although the max QoS cost of I is 30 ms, any service whose output cost is between 30 ms and 80 ms can be used to resolve the inputs of I without affecting the optimal QoS of the composition. The new partial solution generated is finally inserted in the queue to expand later. The algorithm stops when it extracts a solution with no unresolved inputs. Since the algorithm always extracts the partial solution with less services, it always finds the optimal solution first.

IV. Evaluation

In order to evaluate the performance of the proposed approach, we used the datasets of the Web Service Challenge 2009-2010 [12]. These datasets range from 572 to 15,211 services with two different QoS properties: response time and throughput. Tests were executed with a time limit of 5 min.

Tables I and II show the results obtained for each dataset (D-01 to D-05), for the response time and throughput respectively. The best global response time (measured in milliseconds) and throughput (measured in invocations/second) are shown in the first row of each table. Row *#Graph services* shows the number of services of the composition graph and *#Graph services (opt)* the number of services after applying the graph optimizations. As can be seen, the optimizations reduce, on average, the number of services in the composition graph by 64%. This indicates that equivalence and dominance analysis of QoS, and functionality of services is a powerful technique to reduce the search space in scenarios where the number of available services and the possible matches between inputs and outputs is very large. Row *Local Search* and row *Global Search* show the number of services of the solution obtained by the corresponding method as well as the total time spent on the search. The total time reflects the time elapsed between the composition query being submitted by the end-user and the results being obtained by the search algorithm.

The global search strategy obtained the best solution in terms of number of services (*#Services*) for the optimal QoS in every dataset, except for the dataset 04 due to combinatorial explosion (Tables I and II). However, in this case, the local search strategy extracts a good solution in 7.767 s (Table I) and 8.571 s (Table II). Solutions highlighted are those that improved the solutions obtained by the 1st place winners of the Web Service Challenge 2009-2010 [6]. The rest of the solutions are as good as the ones obtained by the winners. Execution times are slightly worse since our algorithm improves also the number of services of the solutions. Even so, most of the solutions were obtained in a few seconds.

Note also that the local search in many cases obtains the best solution, comparing it with the global search, except for the throughput in D-03 and D-05. As far as we know, the results obtained are the best achieved so far in terms of number of services for the optimal response time and throughput. We have compared these results not only with the participants of the challenge but also with other recent approaches [13].

TABLE I. WSC Validation with Response Time

		D-01	D-02	D-03	D-04	D-05
Response Time (ms)		500	1,690	760	1,470	4,070
#Graph services		81	141	154	331	238
#Graph services (opt)		21	57	15	160	126
Local Search	#Services	5	20	10	**40**	32
	Time (s)	0.613	0.998	2.608	7.767	2.920
Global Search	#Services	5	20	10	-	32
	Time (s)	0.617	1.580	2.613	-	24.971

V. Conclusions

In this paper we have presented a hybrid algorithm to automatically build semantic input-output based compositions minimizing the total number of services while guaranteeing the optimal QoS. Results obtained with the Web Service Challenge 2009-2010 datasets show that the combination of graph optimizations with a local-global search strategy performs better than other state-of-the-art approaches.

TABLE II. WSC Validation with Throughput

		D-01	D-02	D-03	D-04	D-05
Throughput (inv/s)		15,000	6,000	4,000	4,000	4,000
#Graph services		81	141	154	331	238
#Graph services (opt)		10	43	90	156	69
Local Search	#Services	5	20	15	**62**	**31**
	Time (s)	0.343	1.173	1.933	8.571	2.562
Global Search	#Services	5	20	10	-	**30**
	Time (s)	0.345	1.246	2.085	-	119.322

Acknowledgment

This work was supported by the Spanish Ministry of Economy and Competitiveness (MEC) under grant TIN2014-56633-C3-1-R and the Galician Ministry of Education under the project CN2012/151. Pablo Rodríguez-Mier is supported by an FPU Grant from the MEC (ref. AP2010-1078).

References

[1] A. Strunk, “QoS-Aware Service Composition: A Survey,” *IEEE European Conference on Web Services*, 2010.

[2] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, “Qos-Aware Middleware for Web Services Composition,” *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[3] W. Jiang, S. Hu, and Z. Liu, “Top K Query for QoS-Aware Automatic Service Composition,” *IEEE Transactions on Services Computing*, vol. 7, no. 4, pp. 681–695, Oct. 2014.

[4] Y. Yan, B. Xu, Z. Gu, and S. Luo, “A QoS-Driven Approach for Semantic Service Composition,” in *IEEE International Conference on Commerce and Enterprise Computing*, 2009.

[5] M. Aiello, E. E. Khoury, A. Lazovik, and P. Ratelband, “Optimal QoS-Aware Web Service Composition,” in *IEEE International Conference on Commerce and Enterprise Computing*, 2009.

[6] W. Jiang, S. Hu, Z. Huang, Z. Liu, and Q. D. Handler, “Two-Phase Graph Search Algorithm for QoS-Aware Automatic Service Composition,” in *International Conference on Service-Oriented Computing and Applications*, 2010.

[7] P. Rodríguez-Mier, M. Mucientes, J. C. Vidal, and M. Lama, “An Optimal and Complete Algorithm for Automatic Web Service Composition,” *International Journal of Web Service Research*, vol. 9, no. 2, pp. 1–20, 2012.

[8] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, “Quality of service for workflows and web service processes,” *Web Semantics*, vol. 1, pp. 281–308, 2004.

[9] W. Jiang, T. Wu, and S.-l. Hu, “QoS-Aware Automatic Service Composition: A Graph View,” *Science And Technology*, vol. 26, no. 5, pp. 837–853, 2011.

[10] P. Rodriguez-Mier, M. Mucientes, and M. Lama, “A Dynamic QoS-Aware Semantic Web Service Composition Algorithm,” in *International Conference on Service-Oriented Computing*, 2012.

[11] P. Rodriguez Mier, C. Pedrinaci, M. Lama, and M. Mucientes, “An Integrated Semantic Web Service Discovery and Composition Framework,” *IEEE Transactions on Services Computing*, 2015 (DOI 10.1109/TSC.2015.2402679).

[12] S. Kona, A. Bansal, M. B. Blake, S. Bleul, and T. Weise, “WSC-2009: a quality of service-oriented web services challenge,” in *IEEE International Conference on Commerce and Enterprise Computing*, 2009.

[13] M. Chen and Y. Yan, “Redundant Service Removal in QoS-Aware Service Composition,” *IEEE International Conference on Web Services (ICWS)*, 2012.