A Heuristic Algorithm For QoS (Non-Functional) Based Service Matching

Islam Harb[1], M. Ezz[2] and H. Farahat[3]

[1] Informatics Research Department, Electronic Research Institute

[2] Computers and systems Engineering Department, Faculty of Engineering, Al-Azhar University

[3] Computers and systems Engineering Department, Faculty of Engineering, Al-Azhar University

Abstract

The problem of QoS-based selecting a web service dynamically or composing a set of web services to conduct a business task has been investigated in this paper. It outlines the discovery of either atomic or composite services satisfying the request QoS requirements. After, the functional requirements matching is achieved where the request Inputs/Outputs syntactically and semantically match services signature, there may exist more than one matched service, then the QoS (nonfunctional) attributes best fit service is chosen. The QoS may be set by the service consumer or computed from past executions. If the chore cannot be satisfied by atomic process, it should be bind with a virtually composed service (set of atomic services).A collection of services sets may be selected where each set has a number of atomic services assembled together to result the desired business requirements. The best fit set of services based on QoS parameters is selected. This paper only concentrates on QoS parameters based selection. The problem is formulated as an optimization problem, so an optimization solving techniques such as integer programming can be applied. A heuristic technique is designed and evaluated.

Keywords: *Semantic Web Services, Non-Functional Requirements Matching, Quality of Service, Optimization.*

1. Introduction

The discovery of services consists of a semantic matching between the description of a service request and the description of published services. The service/request descriptions used for matching mainly include inputs, outputs, preconditions, effects (IOPEs). The inputs and outputs are the data channels where data flows between processes. The service preconditions are the world facts that must be asserted before its execution and the effects are the world facts that become asserted after its execution. The matching can also be based on goals to be achieved and nonfunctional requirements of the services. The degree of matching is based on some criteria. If more than one service matches the request, one is chosen (selected) based on non-functional attributes such as cost or quality. Services are classified into two main subclasses: Atomic and Composite service. An atomic service is a stateless service that receives a set of inputs to commence its execution and produces a set of outputs after it is executed. A composite service is a stateful service composed of atomic processes. In other words, it may accept inputs and produce output messages at different stages during its execution. A composite service model is a workflow describing the composition or orchestration of a service in terms of its constituent processes. A workflow expresses the composition of atomic services by using appropriate control constructs such as Sequence, Unordered, Choice, If-then-else, Iterate, Repeat-until, Repeat-while, Split, and Split+join. The problem of selecting a web service dynamically or composing a set of web services to conduct a requested service has been investigated by different researchers. This paper outlines the problem of the discovery of either atomic or composite processes where the service has to meet the request requirements. Firstly, the request Input/Output has to syntactically and semantically match the service signature. More than one matched process may exist, and then the QoS (nonfunctional) attributes best fit process is chosen. The QoS may be set by the service consumer or be computed from past executions. If the chore cannot be satisfied by atomic process, it should be bind with a virtually composed service. This paper only concentrates on QoS parameters, i.e. a Non-Functional Service Discovery.

The QoS properties may be End-to-end or Service-centered properties. Service-centered QoS properties focus only on the QoS properties of application services while end-to-end QoS considers all the factors having impact on the QoS delivered to users. The QoS concepts may be given in syntactic or semantic descriptions. In case of syntactic descriptions, users and service providers must use the same syntax to define the required and offered QoS while in case of semantically description they can use different syntax. Semantic QoS descriptions (e.g., QoS ontologies) explicitly define the semantics of QoS concepts and the relationship between these concepts allowing us to enable reasoning on QoS concepts and inferring matches between requested and offered services even at the run time. For this reason, semantic description is more reliable. Quality-based service description (QSD) is defined by embedding QoS specifications into the description of services. This can be carried out according to two approaches: the black-box approach where QoS

specifications (concern the whole service entity) are associated to services wrapped as black boxes and the white-box approach where QoS specifications are associated to the functional behavior of services (associated with elementary parts or operations). The black-box approach is generally used when addressing atomic services with simple behavior (i.e., a single operation). However, the white-box approach is more suitable for services with various operations, each characterized with a different QoS.

QoS-based service discovery determines service candidates that may fulfill the user's requirements. The way the matching is carried out impacts the spectrum of discovered services, notably the number of discovered services and the extent to which they fit the user's required task. QoS-based service discovery approaches may be either syntactic or semantic. Syntactic discovery approaches requires the same syntax to describe the required and offered QoS, so they constrain the number of discovered services as they disregard services that fit the user requirements but use a different QoS syntax. Semantic discovery approaches infer matches between heterogeneous QoS terms, hence determining all services which fulfill the required QoS.

Like all the algorithms dealing with combinatorial problems, QoS-based selection algorithms can be divided into two broad classes: brute-force-like algorithms and heuristic algorithms. The first class of algorithms (NP-hard [1]) aims at determining the optimal service composition (i.e., with the highest QoS) by exploring all possible compositions of services. A second class of algorithms is heuristic algorithms which do not explore all possible compositions and they find near-optimal compositions, i.e., compositions that meet global QoS constraints and sub-optimize the QoS delivered to the user. The goal of any QoS-based selection technique is to reduce the number of service compositions to be investigated. Therefore, the greedy technique is opted to be coupled with our selection approach to reduce the number of service compositions trials. Greedy selection is a technique that selects, for each abstract activity in the user task, the service candidate with the highest QoS. The selection is performed for each abstract activity individually and it is generally used under local QoS constraints.

2. Related Works

Recently, the QoS-based web service selection and composition in service-oriented applications has gained the attention of many researchers. Several selection algorithms have been proposed to select service compositions with different composition structures and various QoS constraints. Taxonomy of these solutions may be produced based on their objectives and the way they proceed. Approaches aim at determining the optimal service composition (i.e., composition with the highest QoS utility) using brute-force-like algorithms (e.g., Global Planning, BBLP, WS-IP [3]). These solutions have high computational cost and they cannot provide a solution in a satisfying amount of time, thus they are inappropriate to be used in the context of dynamic service environments. Other approaches propose heuristic-based solutions (e.g., WS-HEU [3] and DIST_HEU [11]), Genetic algorithm [4, 5, 6, 7, 8, 9] aiming to find near-optimal compositions, i.e., compositions that respect global QoS constraints and maximize a QoS utility function. Yu et al. [3] present two heuristics, WS-HEU and WFlow, for the service selection problem. WS-HEU is specific heuristic applied to sequential workflows (i.e., workflows structured as a sequence of activities), whereas WFlow is designed for general workflow structures (i.e., sequential, conditional, parallel). The main idea of WFLow is to decompose workflows into multiple execution routes. WFlow considers a probability of every route to be executed. Therefore, it focuses on the route with the highest probability. Bin Mubarak [10] adapts this solution giving feasible service compositions regardless of the way the workflow will be executed.

More recently, Alrifai et al. [11] presented a novel approach (DIST-HEU) that combines local and global optimization techniques. This approach starts from the global level and resolves the selection problem at the local level. It proceeds by decomposing global QoS constraints (i.e., imposed by the user on the whole composition) into a set of local constraints (i.e., for individual sub-tasks, part of the composition). To do so, it uses MILP (mixed integer linear programming) techniques [1] to find the best decomposition of QoS constraints. The main drawback of this approach is that it represents a greedy selection method, since it selects services at the local level and does not ensure that the global QoS constraints are respected.

The work of Zeng at al. [12] focuses on dynamic and quality-driven selection of services using global planning to find the best service components for the composition. They use (mixed) linear programming techniques [13] to find the optimal selection of component services. Linear programming methods suffer from poor scalability due to the exponential time complexity of the applied search algorithms [12].

3. The QoS Model

Web service may be evaluated based on many generic quality criteria such as: price, execution duration, reputation, reliability, availability,

documentation, and relevance ration. QoS attributes can be classified into four basic classes [14] as visualized in Figure 1. Only five QoS attributes are considered in this research: execution price, execution time (response time), reliability, availability, and reputation.

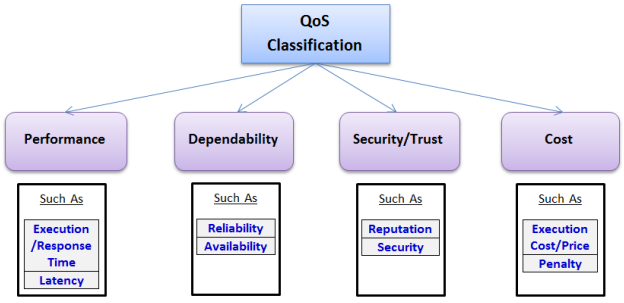

Fig. 1 The QoS taxonomy

3.1. The Composition model

Our QoS-aware service composition approach is initialized by taking an input as a user request R which is defined as a quadruple R = (T, C, Q, W), where T refers to the required task and C = <c_1, .., c_n> refers to global QoS constraints (e.g. cost doesn't exceed certain limit) imposed by the user on a set of QoS properties Q = <q_1, .., q_n >. The user has to specify the relative importance of its associated QoS property by giving a set of weights W = <w_1, ..,w_z >, where w_i is the weight of QoS property p_i. It is worth noting that the sum of all the weights must be equal to 1, i.e. $\sum_{i=1}^{z}(w_i) = 1$, where 0 <= w_i <= 1 and (z) is the number of QoS properties.

For a user task T, its structure is specified as a set of activities T = <A_1 … A_m > coordinated by composition patterns (Sequence, AND, XOR and Loop). There is an associated set of atomic service candidates S = {$s_{i,1}$, $s_{i,2}$, .., $s_{i,n}$} for each activity A_i in the user task T. Each candidate service $s_{i,k}$ (1 <= k <= n) in this set is able to realize/accomplish A_i and it is represented by its QoS vector $QoSs_{i,k}$ = <q_1, … , q_z >, where q_j is the advertised value of QoS property q_j (1 <= j <= z). Only one atomic service $s_{i,k}$ is enacted for each activity A_i, thus forming a candidate service composition CS = <$s_{1,k}$, ..., $s_{m,k}$ > realizing the user's task T, where 0 < k <= n. Figure 2 shows an example on "reserving the cheapest flight from Cairo to Austin" task with its associated activities and their candidate services.

The question to be asked is then which service should be enacted for each activity in the user task so that the overall composition meets the user's QoS requirements. The problem becomes even more complicated when we aim at selecting several alternative service compositions (CS_1, … , CS_v) in order to support dynamic binding of services. The first step to address this problem is to determine how to evaluate the QoS of a service composition QoS_C = <Q_1, ... , Q_n > based on the structure of the composition (i.e., its composition patterns) and the QoS of its constituent services.

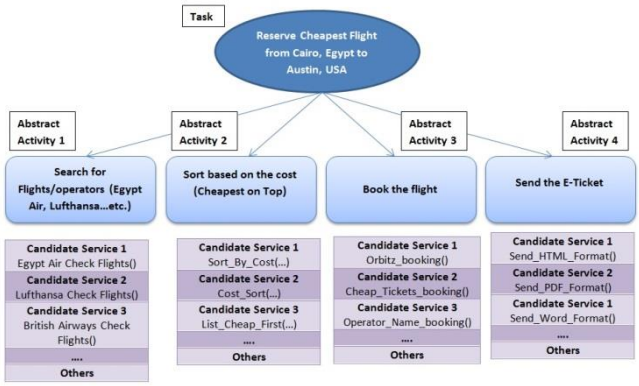

Fig. 2 Example on Task, activities and candidate services for each activity

3.2 The QoS Aggregation

Global service selection requires evaluating QoS of service compositions prior to their execution. The service composition is unforeseen during service selection, so the overall QoS of a composition cannot be assessed in an accurate manner. It is rather estimated with respect to possible execution scenarios of the composition. QoSCi = <$QoSs_1$, ..,$QoSs_m$> = <Q_1, ..,Q_m > is determined by aggregating QoS values of its constituent services. Each $QoSs_{i,k}$ = $Q_{i,k}$ = <q_1, .., q_z>, where 0 < k <= m corresponds to one of these atomic/constituent services from each activity, while taking into account the composition patterns which are the structuring elements used to build the composition. There is a set of common composition patterns such as sequence (sequential execution of activities), AND (parallel execution of activities), XOR (conditional execution of activities), and Loop (iterative execution of activities).Three QoS aggregation approaches may be considered: optimistic approach (i.e., considering the best QoS value), pessimistic approach (i.e., considering the worst QoS value), and mean-value approach (i.e., considering the average of services' QoS values). For instance, the response time of two services composed in exclusive choice (XOR) or AND can be pessimistically estimated considering the worst-case QoS values of the two services (the longest response time). A history-based estimation that considers the maximum number of loops may also be adapted for Loop.

In our model we assume that we have a universe of web services U which is defined as a union of abstract activities. Each abstract activity A_j Є U is used to describe

a set of candidate functionally-equivalent web services. A composite service (corresponds to the user task T) can be represented as CS = (A_1, … , A_m) where A_j refers to a requested task activity j. The $s_{i,j}$ is an atomic candidate service (i) as one of available atomic services which can be chosen of abstract activity j (assuming there are n_i concrete services for abstract activity j), $S_{i,j} \in A_j$. A QoS of the composition for each property is aggregation of the same QoS property for all constituent services. The QoS vector for a composite service C is defined as QoSc = { $Q_{rt}(C)$, $Q_{pr}(C)$, $Q_{rep}(C)$, Q_{rel} (C), $Q_{av}(C)$ }, where Q_i represents the aggregated i^{th} QoS property value which can be aggregated from the expected QoS values of its (n) component services (n = no. of all the atomic candidate services that constitute a solution/composite service = no. of activities) as given in Table 1. For example, the services "Lufthansa_Check_Flights()", "List_Cheap_First()", "Orbitz_booking()" and "Send_Word_Format()" in figure 2 are considered the component services for the solution of the user requirement/query "reserve cheapest flight from Cairo to Austin".

Table 1*: The Criterion Aggregation Formula for Sequence Composition*

Criteria	***Aggregate Function***
Price	$Q_{pr}(C) = \sum_{i=1}^{n}(q_{pr}(S_i))$
Response Time	$Q_{rt}(C) = \sum_{i=1}^{n}(q_{rt}(S_i))$
Reputation	$Q_{rep}(C) = \sum_{i=1}^{n}(q_{rep}(S_i))$
Reliability	$Q_{rel}(C) = \prod_{i=1}^{n}(q_{rel}(S_i))$
Availability	$Q_{av}(CS) = \prod_{i=1}^{n}(q_{av}(S_i))$

4. The Non-functional Matching Problem Formulation

This section formulates the problem as an objective function to be optimized and constraints to be satisfied. Optimization solving techniques can then be applied. As mentioned earlier, individual Web services are federated into composite one whose business logic is expressed as a process model. The selection of component services considers multiple QoS criteria taking into consideration global constraints and preferences set by the consumer (e.g., response time and budget constraints). The process model of a composite service can be specified as a state chart where the execution path is viewed as a sequence of states [t_1, t_2, … , t_m] with an initial and final states. Each state represents an atomic process/service. A whole path as a Directed Acyclic Graph corresponds to a composite service c composing of a set of pairs { $<t_1,q_1>$, $<t_2,q_2>$, … , $<t_m,q_m>$ } where m is the total number of states/activities (its constituent atomic processes) and q_k is the vector of QoS parameters for an atomic process k of the composite c. A path which optimizes the QoS and satisfies the constraints is chosen.

Once user requirements are specified, we proceed by automatically building executable service compositions with respect to user requirements and the dynamics of the service environment. Building executable compositions consists of selecting, and composing services at runtime. For every activity in the composition, there should be a preceded discovery phase that gives the set of service candidates, which are able to fulfill the activity (i.e., functional aspect) and to satisfy user QoS requirements (non-functional aspect). The non-functional service discovery uses advertised QoS of services to perform a preliminary filtering to ensure that QoS requirements are satisfied at the global level (i.e., for the whole composition) and at the level of individual service. Further, selection phase is done, so that we use the QoS attributes to determine the utility of service candidates regarding our objective, i.e., selecting optimal compositions. Once the global selection is fulfilled, the composition phase uses the selected services to define an executable service composition.

The problem can be treated as a single objective optimization problem to be solved by an optimization technique such as integer programming. By accumulating all the execution plans' quality vectors, we obtain matrix Q, where each column represents the quality vectors of all the atomic candidate services that achieve/correspond to one of the activity of the User task. On the other hand, each row represents an execution plan quality vector corresponds to a candidate solution (five services for the five activities) that constitutes the composite service which meets the user requirement/query. The "n" in the Q matrix denotes all the execution plans' quality vectors. In other words, "n" represents all the possible solution for the user query.

$$Q = \begin{bmatrix} Q_{1A1} & Q_{1,A2} & Q_{1,A3} & \cdots & Q_{1,Am} \\ Q_{2,A1} & Q_{2,A2} & Q_{2,A3} & \cdots & Q_{2,Am} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ Q_{n,A1} & Q_{n,A2} & Q_{n,A3} & \cdots & Q_{n,Am} \end{bmatrix}$$

Then Q matrix may be normalized by applying a scaling operation getting V to handle a variance in quality parameters values. The parameters/qualities that are needed to be maximized will take a positive value. On the other hands, qualities to be minimized will take negative values. In the scaling process each QoS attribute value is transformed into a value between 0 and 1, by comparing it

with the minimum and maximum possible values. Therefore, min and max aggregated values should be determined in each quality (i.e. price, time…etc.) in universe of web services U.

- $V_{i,j}$ = ($Q_{i,j}$ – Q_{min}) / (Q_{max} – Q_{min}), for reliability, availability, and reputation qualities to be maximized.
- $V_{i,j}$ = (Q_{max} – $Q_{i,j}$) / (Q_{max} – Q_{min}), for price and execution time to be minimized.

Where $Q_{i,j}$ is the quality vector of the service in the Matrix Q located in row i and column j,

- $Q_{i,j}$ = < q_{pr}, q_{rt}, q_{rel}, q_{rep},q_{av} > for each atomic service
- Q_{min} = Min Quality Vector = <min(q_{pr}), min(q_{rt}), … min(q_{av})>
- Q_{max} = Max Quality Vector = <max(q_{pr}), max(q_{rt}), … max(q_{av})>

$$V = \begin{bmatrix} V_{1,A1} & V_{1,A2} & V_{1,A3} & \cdots & V_{1,Am} \\ V_{2,A1} & V_{2,A2} & V_{2,A3} & \cdots & V_{2,Am} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ V_{n,A1} & V_{n,A3} & V_{n,A3} & \cdots & V_{n,Am} \end{bmatrix}$$

A weighting factor for each quality parameter is applied to get a path score. For example, the composite service score is given in equation 2.

$$\text{Score } (CS_i) = \sum_{i=1}^{5}(V_{i,j} * W_j) \quad \ldots\ldots\ldots\ldots\ldots \quad (2),$$

where W_j is the weight of criterion j, $0 <= W_j <= 1$, and $\sum_{i=1}^{5}(W_j) = 1$

The problem of finding the best service composition is an optimization problem, in which the overall objective function has to be optimized while satisfying all global constraints. Finding the solution by enumerating all possible combinations is NP-hard problem. Formally, the optimization problem can be stated as it follows.
For a given composite service CS, find an implementation <S_1, . . . , S_5 > by bounding it to set of QoS global constraints such that:

1. The overall objective function (given in 3) is optimized, and
2. The aggregated QoS values satisfy the following constraints:
   Q_{pr} (CS) <= C_{price}
   Q_{rt} (CS) <= $C_{responseTime}$
   Q_{rep} (CS) >= $C_{reputation}$
   Q_{rel} (CS) >= $C_{reliability}$
   Q_{av} (CS) >= $C_{availability}$

Note: Where C_j is the limit of the QoS property set by the requester.

The objective function to be optimized, is given in (3), if (Q_{jmax} – Q_{jmin}) is not equal to 0, otherwise it is given as 1.

Objective Function:

$$\text{Max } \{\sum_{j=1}^{2}(\left(q_{jmin} - q_{i,j}\right) / (q_{jmax} - q_{jmin}) * W_j) + \sum_{j=3}^{5}(\left(q_{i,j} - q_{jmin}\right) / (q_{jmax} - q_{jmin}) * W_j)\} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

The q_{jmax} (q_{jmin}) is the maximum (minimum) value of QoS property j. We only consider **five** QoS criteria. The five QoS criteria are enumerated in which "Price", "Response/Execution Time", Reliability", "Reputation" and "Availability" are assigned numbers starting from j=1 to j=5 respectively. A service class (i.e. abstract activity) is a set of services with same functionality (the same IOPEs), but different QoS. This leads to the optimization problem to select the best one in each class. In composite services, the objective function is used to evaluate a given set of alternative service compositions (alternative solutions). However, finding the best composition/solution requires enumerating all possible combinations of service candidates. For a composition request with m activities and n service candidates per activity, there are (n^m) possible combinations to be examined. Performing exhaustive search can be very expensive in terms of computation time and, therefore, inappropriate for applications with many services and dynamic needs, so a heuristic solution is necessary.

5. The Genetic Heuristic (G_HEU) algorithm

The linear programming solving methods suffer poor scalability, so their applicability is restricted to small-size problems [11]. These methods are also inappropriate for dynamic applications with run-time requirements since the selection problem to be optimized is NP-hard. In this research, we propose a heuristic algorithm, which is mainly based on greedy technique and genetic. This heuristic algorithm tries to find the best atomic service within each activity list. There is an assumption that all the atomic services are independent. So, the algorithm runs regardless of relations of a service in a certain activity with other services in other activities. It might be prone and stuck with local peaks/solutions. On the other, hand this sub-optimal solution could be reasonable and good enough with respect to the time

taken and complexity of the algorithm. Brute-force algorithms run in very long time especially when we have big number of services and activities to check for. Therefore, it can be considered as a good and sufficient alternative for the Brute-force algorithm when the problem size is really big.

```
Let's say,
   - "m" is the number of  the activities/service classes
   - "n" is the number of the atomic services in each activity. Assuming that all the activities have the same
     number of atomic services that meet their functionality.
   - We only have 5 quality attributes/criteria (Price, Execution time, Reputation, Reliability and
     Availability)
   - "V" is a matrix of the (m) activity column. Each column's entry represents a quality vector of an
     atomic service for the corresponding activity.
   - All the atomic services are independent
   - "G" is the number of Cross over operations that will be done.

G_Heuristic () begin
   i = 0
   G = 10;        // Could be passed from the User
   Matrix M;      // Same structure as V but empty
   // It will have the atomic services that construct the composite services that achieve the user query.
   solution_Service_ID_vector = []
   // Loop
   For each column "Va" in "V" do
      Evaluate (Va)              // Evaluate each entry in Va based on Eq. (3)
      sort_descending(Va)        // based on the Objective function values
   EndFor  // For column

   For each row "S" in "V" do
      Aggregate_QoS(S);          // Use Rules in table 1
      Save "S" in Matrix M;
      i = i + 1
      If (S Meets Constraints) do
         solution_Service_ID_vector = S;
         Break;
      EndIf
   EndFor  // For Row1

   Apply_Cross_Over(M, G);       // Apply genetic cross over technique to generate new populations

   For each row "S" in "M" do
      If (S Meets Constraints  &  (Aggregated_QoS(S) > Aggregated_QoS(solution_Service_ID_vector)) ) do
         solution_Service_ID_vector = S;
      EndIf
   EndFor  // For Row2
// solution_Service_ID_vector holds the best service from each activity, so that we got the sub-optimal solution
    Return  solution_Service_ID_vector;
End G_Heuristic
```

The algorithm runs in two stages. The first stage is done by applying Eq. (3) on each entry/quality vector $V_{i,j}$ in each column in Matrix V. Then apply Merge Sort on every column, so that all the services in each activity are sorted in a descending order based on the objective function value. The second stage starts with calculating the aggregated QoS values (as shown in table 1) for each row/solution (i.e. all the atomic services that represents all the activities of the Composite Service requested by the user). Iterate on each solution Aggregated QoS vector and check if meets all the global constraints; Stop when observing first solution that meets all the constraints. Keep it as the default solution to be returned. Discard all the below rows/solutions. For further improvement, a genetic cross over is applied on the accepted/kept rows/solutions. Number of cross over operations that are done might be set by the user. This genetic processing will produce new solutions that might have better QoS that abide by the constraints than the default one. Therefore, Apply on the new population equation (3) again, sort them descending and then check the constraints again. Select the best if solution with the highest QoS and abides by the constraints.

6. The Algorithm Analysis and Evaluation

Our algorithm is contrasted against the Linear Programming (LP) Brute-force and two other heuristic algorithms to be evaluated. We picked two famous heuristic algorithms which are WS HEU [3], and Alrifai Heuristic DIST HEU [11]. A simulation has been conducted to evaluate the proposed heuristic. The evaluation considers two criteria to be experimented amongst our algorithm and the other three. It considers the algorithm time taken to find the Optimal/Suboptimal solution and the optimality degree in case of suboptimal solution. This simulation runs many times while changing the size of the problem in terms of the number of activities (m) and the number of services per activity (n). For LP, the open source Linear Programming system lpsolve version 5.5 [18] was used. LP was experimented with varying the number of activities (m) and the number of service candidates per activity (n). Each unique combination of these parameters represents one instance of the composition problem. There is an available QoS real dataset QWS [19], but it only includes measurements of 3 QoS attributes for 364 real web services. To get a bigger dataset, we randomly generate activities and randomly generate component services (candidate services) per each activity. We also randomly generate input data between the minimum and maximum values of the QoS metrics associated with each component service.

Figure 3 shows the result of the experiments on the four algorithms. The graph on the top shows the Algorithms consumption time on y-axis meanwhile the x-axis represents the number of service candidates (n) is varied from 100 to 1000 per class. The number of activities is fixed to 20. Each experiment is executed 20 times and the mean value of the obtained results is computed. The obtained measurements also show that the execution time of our algorithm increases along with the number of services per activity and it also increases along with the number of QoS constraints. The results in both graphs show that our proposed heuristic has better computation time compared to all the others.

To evaluate the quality of the results of our approach, we measure the closeness of the returned results to the optimal results obtained by the LP method by calculating the optimality ration $R = U_h / U_{opt}$ as shown in the bottom graph in Figure 3. U_h is the objective value of the best composition returned by our approach according to Eq. (3) and U_{opt} is the optimal value of the composition returned by the LP method. The optimality of our algorithm increases along with the number of services per activity

which means that the more feasible compositions, the more probable to have a better utility as shown in Figure 3. It also indicates that our approach achieves good results with average 97.792% optimality. The biggest optimality degree difference between DIST_HEU and our approach is 1.7% and in average our approach is just 0.65% below the DIST_HEU. It is noticed that as the number of services increases per each activity, our approach shows better optimality that gets closer to that of the DIST_HEU.

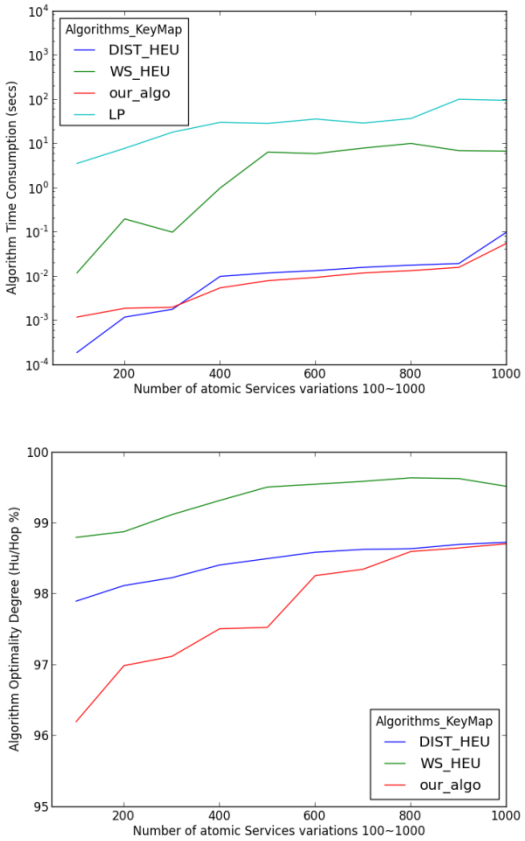

Fig. 3 Experiment the algorithm's time consumption and Optimality VS. Atomic Services Number

7. Conclusion

This paper has addressed the problem of the best fit discovery of either atomic or composite processes considering the request QoS requirements. After, the request Inputs/Outputs semantically match the service signature (functional requirements discovery), there may exist more than one matched service, then the QoS (nonfunctional) attributes best fit service is chosen. Usually, several component services are able to execute a given task, although with different levels of pricing and quality. In this paper, we advocated that the selection of component services should be carried out during the execution of a composite service, rather than at design-time. QoS-based service selection aims at finding the best component services that satisfy the end-to-end quality requirements.

The QoS may be set by the service consumer or computed from past experiments. The service registry has been checked for all services, so a collection of service sets may be selected where each set has a number of atomic services assembled together to result the desired business requirements. The best fit set of services based on QoS parameters is selected. This paper only concentrates on QoS parameters, i.e. a Non-Functional Service best matching. The problem is formulated as an optimization problem, so an optimization solving technique such as integer programming can be applied. Since it is a NP-hard problem, a heuristic technique is designed for shorter selection time (polynomial complexity) and suboptimal performance. Overall our approach showed a good optimality within very optimized and reduced time consumption.

References

[1] Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley-Interscience, New York, NY, USA (1988)

[2] D. Pisinger. Algorithms for Knapsack Problems. PhD thesis, University of Copenhagen, Dept. of Computer Science, February 1995

[3] Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end qos constraints. ACM Transactions on the Web (TWEB), Volume 1 Issue 1, May 2007

[4] Gerardo Canfora, Massimiliano Di Penta, Ra_aele Esposito, and Maria Luisa Villani. An approach for qos-aware service composition based on genetic algorithms. In GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, pages 1069-1075, New York, NY, USA, 2005. ACM.

[5] Chengwen Zhang, Sen Su, and Junliang Chen. A Novel Genetic Algorithm for QoS-Aware Web Services Selection. In Springer Berlin/Heidelberg, editor, Data Engineering Issues in E-Commerce and Services, pages 224-235, 2006.

[6] Michael C. Jaeger and Gero M□uhl. QoS-based Selection of Services: The Implementation of a Genetic Algorithm. In Torsten Braun, Georg Carle, and Burkhard Stiller, editors, Kommunikation in Verteilten Systemen (KiVS 2007) Industriebetr□age, Kurzbeitr□age und Workshops, pages 359{350, Bern, Switzerland, March 2007. VDE Verlag, Berlin und O_enbach.

[7] Ziad Kobti andWang Zhiyang. An Adaptive Approach for QoS-Aware Web Service Composition Using Cultural Algorithms. In Mehmet A. Orgun and John Thornton, editors, Australian Conference on Arti_cial Intelligence, volume 4830 of Lecture Notes in Computer Science, pages 140-149. Springer, 2007.

[8] Lei Cao, Minglu Li, and Jian Cao, "Using genetic algorithm to implement cost-driven web service selection ", Multiagent and Grid Systems - Special Issue on Nature inspired systems for parallel, asynchronous and decentralised environments, Volume 3 Issue 1, January 2007, Pages 9-17, 2007.

[9] Chunming Gao, Meiling Cai, and Huowang Chen. QoS-aware Service Composition Based on Tree-Coded Genetic Algorithm. In COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference, pages 361-367, Washington, DC, USA, 2007. IEEE Computer Society.

[10] Nebil BEN MABROUK, QoS-aware Service-Oriented Middleware for Pervasive Environments ,PhD thesis, UNIVERSITÉ PARIS 6, École doctorale informatique, télécommunications, électronique, 2012

[11] Mohammad Alrifai, Thomas Risse, Peter Dolog and Wolfgang Nejdl. A Scalable Approach for QoS-based Web Service Selection. Service-Oriented Computing – ICSOC 2008 Workshops. Lecture Notes in Computer Science, 2009, Volume 5472/2009, pages 190-199, DOI: 10.1007/978-3-642-01247-1_20

[12] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: WWW. (2003) 411–421

[13] Maros, I.: Computational Techniques of the Simplex Method. Springer (2003)

[14] Florian Rosenberg, QoS-Aware Composition of Adaptive Service-Oriented, PhD thesis, Technische Universität Wien, Austria, 2009

[15] W. Iverson. Real World Web Services. O'Reilly, 1. edition, 2005

[16] Boonserm Kulvatunyou , et. al. Integrated product and process data for business to business collaboration, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, v.17 n.3, p.253-270, June 2003

[17] OASIS: Web services business process execution language (April 2007) http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[18] Michel Berkelaar, Kjell Eikland, P.N.: Open source (mixed-integer) linear programming system. Sourceforge http://lpsolve.sourceforge.net/.

[19] Al-Masri, E.,Mahmoud, Q.H.: Investigating web services on the world wide web. In:WWW. (2008)