Finding Preferred Skyline Solutions for SLA-constrained Service Composition

Xin Zhao, LiWei Shen, Xin Peng&Wenyun Zhao
School of Computer Science, Fudan University
Shanghai, China
{x_zhao,shenliwei,pengxin,wyzhao}@fudan.edu.cn

***Abstract*—in this paper, we address the optimization problem of SLA-constrained service composition. Focusing on the main drawbacks of traditional approaches surveyed:1) the difficulties in preference definition and weight assignment; 2) the limitation of linear utility function for identifying preferred skyline solutions; and 3) the poor efficiency and scalability of algorithms, we present a systematic approach of combining the weighted Tchebycheff distance with skyline computation to cope with this optimization problem. More specifically, we first propose a fuzzy linguistic preference model that can help service composer elicit, represent and establish consistent preference relations upon QoS dimensions. Then we present a weighting procedure to transform the preference relations into numeric weights that are used in the Tchebycheff distance as quantified measurement of preference for skyline solutions. Finally we propose a hybrid evolutionary algorithm to heuristically find preferred skyline solutions in an efficient way. The algorithm is further evaluated by a set of experimental studies.**

Keywords-SLA, service composition, evolutionary algorithm, multi-objective optimization,skyline coputation

I. INTRODUCTION

The service-oriented architecture provides a loose-coupled computing paradigm, in which the software entities such as business functions, data accessing APIs can be described, located and accessed as web services using standardized techniques. Meanwhile, web services often need to be composed as work flows to achieve more complex tasks or to mash up data from different data resources by using business process description languages such as BPEL [1] and ORC [2]. The interoperations between service providers and service consumers enable wide range integration of applications across domain boundaries, hence encouraging the formation of a market-oriented computing environment [4].Considering the dynamic and loose-coupled characteristics of service computing, the service level agreement (SLA) provides a contract basis for fast establishing and stabilizing of the business relationship between service providers and service consumers. As a declarative contract, an SLA clarifies the QoS levels for which service provider should be responsible, as well as the penalty when violation occurs [5].

In the scenario of SLA management, service providers are used to predefine a set of QoS levels and prices, to allocate computing resources supporting these QoS levels, and to publish them as SLA offers on some kinds of broker systems or market-place systems [6].Therefore the SLA offers can be discovered, be selected or be negotiated to achieve final SLA contract. The situation is more complex for service composition, i.e. multiple service providers may be available for one function accessed by the composition while each provider may supply different back-end SLA offers. Since selecting different providers as well as different back-end SLA offers may lead to different overall composition QoS properties, service composers are often required to select "optimal" from possible solutions. Moreover, considering the fore-end SLA contracting with end users, the optimal selection should be carried out under certain fore-end SLA constraints. In this paper, we call this *optimization problem* as *SLA-constrained service composition*.

There are two main challenges the optimization of SLA-constrained service composition confronts.

First, since multiple QoS dimensions are involved in SLA contracting and conflicts may exist among dimensions when performing optimization (e.g. lower response time can result in higher price), it is a *Multi-objective optimization problem*. Furthermore, without considering the relative importance (i.e. preference) between different dimensions, there is no single optimal solution but a set of *Pareto optimal* solutions that can also be called *skyline solutions* [19].Therefore the service composer needs to present their preference to identify the optimal from a set of skyline solutions. Lots of researches have leveraged the linear utility function (i.e. weighted sum) to define the most preferred skyline solutions [11-16], where the numeric weights are required and used as quantified representation of preferences. However, few of them have mentioned how to elicit and represent the preference, and how to assign numeric weights according to preference information. On the other hand, in the multi-objective optimization theory it is well known that the weights used in linear utility function as quantified measurement of preference intensity is effective only for convex problem [7]. There are skyline solutions which can never be obtained by turning weights and by optimizing the utility function in non-convex case. Unfortunately, we cannot simply assume the convexity exists for the discrete SLA-constrained service composition problem.

Second, since all possible solutions for service composition are constructed by combining different back-end SLA offers from different providers, it is a *combinatorial optimization problem*. As all we know, it is NP-hard in the strong sense [14]. Meanwhile, many researchers have indicated that a fast growth of published web services, the pay-per-use model promoted by the cloud computing paradigm will also enable service providers to offer their services with different QoS levels [3][15]. Therefore, the potential huge number of candidate solutions makes the efficiency and scalability of optimization algorithm more important and more challenging.

To cope with these challenges and the underlying drawbacks of traditional approaches, we propose a

systematic approach for the optimization problem of SLA-constrained service composition. In the approach, we first introduce a fuzzy linguistic preference model [10] to help service composer establish consistent preference relations upon QoS dimensions, in which an intuitive preference representation method based on fuzzy linguistic terms such as "important than", "more important than" are used to define preference relations between different QoS dimensions. Then we put forward a novel weighting procedure to interpret and transform the fuzzy preference relations into a set of numeric weights. Second, we leverage the weighted Tchebycheff distance rather than linear utility function to incorporate the preference information into the optimization process [8][9]. Third, we propose a hybrid evolutionary algorithm called hybrid_EA. In this algorithm the Pareto-dominance and the weighted Tchebycheff distance are combined to heuristically find a set of preferred skyline solutions in an efficient way. Finally, the effectiveness and efficiency of the algorithm is evaluated through a set of comprehensive experimental studies.

The rest of the paper is organized as follows. Section II discusses related works. Section III provides the definition of the optimization problem of SLA-constrained service composition. Section IV presents the fuzzy preference model, the weighting procedure as well as the definition of weighted Tchebycheff distance. In section V, we describe the hybrid evolutionary algorithm in detail, and section VI presents our experimental study. Finally, section VII concludes the paper.

II. Related Works

For nearly ten years, many researchers have addressed QoS-based service composition problem from different perspectives. In [13], Zeng et al. focus on the runtime management of service composition under dynamically changing QoS environment. They use linear utility function to define the optimization problem, and then solve it by linear programming techniques to find single optimal solution. Similarly, in [11] Ardagna et al. exploit utility-based optimization for runtime adaptive service composition. While modeling the problem as mixed integer linear programming, they consider not only global constraints but also local constraints. Concerning about the poor efficiency and scalability of linear programming methods, many different heuristic approaches have been proposed. In [12], Yu et al. model the problem in two ways that are the combinatorial model and the graph model. Then, they propose efficient heuristic algorithms for each model, i.e. WS_HEU and MCSP_K respectively, to find single near-to-optimal solution based on application-specific utility function. Alrifai et al. [14] present a hybrid approach to find single close-to-optimal solution in terms of utility function, in which the mixed integer programming is used to decompose the global constraints into local constraints. In addition, the local selections are performed for each service class independently. In their following work [15], the skyline computation is used as a pre-processing step to prune non-interesting candidate services. Since the number of candidate services may still be too large after skyline computation, they present a hierarchical clustering-based method for further reducing the search space. Recently, some kinds of utility-based meta-heuristic algorithms have also been considered [16]. Different with the above works, we consider the limitations of traditional linear utility based approaches, thus propose the weighted Tchebycheff distance for identifying preferred skyline solutions. Furthermore, we present a comprehensive procedure to cope with the preference representation and weight assignment.

On the other hand, Yu et al. [18] present a Dual Progressive Algorithm (DPA) that can progressively enumerate the candidate solutions and progressively report an entire set of skyline solutions. In [22], Wada et al. leverages Evolutionary Multi-Objective Optimization (EMOO) techniques. They present two multi-objective genetic algorithms, E^3-MOGA and X-E^3. The former is designed to find a subset of skyline solutions that are uniformly distributed and diverse over the entire skyline set, while the latter is designed to find the extreme solutions that have extreme value on certain dimensions even if they are dominated by other solutions. Different with these two works, our hybrid evolutionary algorithm is designed to simultaneously find a set of skyline solutions that are optimal and close-to-optimal based on preferences.

III. Problem Definition

A. The optimization problem of SLA-constrained service composition

In the SLA-oriented QoS management scenario, a service composition consists of a set of abstract service classes, and each service class corresponds to a certain function. A service composition is denoted as $CS = \{S_1, S_2, \dots, S_m\}$. The function of S_i can be provided by different service providers, where each provider can release a set of SLA offers with different QoS levels. In this paper, we call the alternative providers and respective SLA offers as *SLA bindings*. Assuming the number of available service providers for service class S_i is n, and each provider releases k SLA offers, we have $n \times k$ different alternatives of SLA binding for S_i, denoted as $B(S_i) = \{b_{i1}, b_{i2}, \dots, b_{ij}\}$. Meanwhile, we have $n \times k$ available QoS levels at most since same QoS level for different SLA binding is allowed. In addition, a QoS level is organized by quantitative dimensions such as response time, throughput, price and availability. We then represent it as a vector $Q(b) = \{q^1(b), q^2(b), \dots, q^r(b)\}$, where r is the dimension number and $q^i(b)$ refers to the QoS value on dimension i that can be guaranteed by service provider for SLA binding b.

There are a set of candidate solutions for combining different SLA bindings for service composition. Each of them requires to calculate the overall QoS vector by aggregating its back-end QoS levels according to the commonly used QoS aggregation model (details see[15]), denoted as $Q(cs) = \{q^1(cs), q^2(cs), \dots, q^r(cs)\}$.Without the loss of generality, we assume the QoS value of each dimensionis scaled [13] into range [0, 1] in the rest of this paper, such that the smaller values they have, the better they

are. As for optimization, the solutions with optimal QoS vector should be selected under certain QoS constraint of fore-end SLA contracting. The constraint is represented as a vector $C = \{c_1, \dots, c_k \dots, c_r\}$ where $k \in [1, r]$ and c_k indicates the QoS value constraint of the dimension. Thus a selected solution should firstly satisfy $q^k(cs) \leq c_k$.

B. Pareto-dominance and Skyline solutions

Without considering preferences, the optimization of SLA-constrained service composition can be carried out based on skyline computation, in which the pair-wise comparison of candidate solutions is required in terms of the Pareto-dominance relationship.

Definition 1. (Pareto-dominance) For any two candidate solutions cs_i and cs_j, if and only if solution cs_i is better than (smaller thanin our case) or equal to solution cs_j in terms of QoS values on all dimensions, and better on at least one dimension, cs_i is Pareto dominating cs_j. It is represented as follows.

$$cs_i \prec cs_j \Leftrightarrow \left(\forall k \in [1, r], q^k(cs_i) \leq q^k\left(cs_j\right)\right) \wedge \left(\exists t \in [1, r], q^t(cs_i) < q^t\left(cs_j\right)\right)$$

Definition 2. (Skyline Solutions) In the set of candidate solutions, those cannot be Pareto-dominated by any other solutions are called skyline solutions, i.e.

$$sky^{CS} = \left\{cs_i \in CS \middle| \nexists cs_j \in CS, cs_j \prec cs_i\right\}$$

Since the Pareto-dominance relationship does not provide any discrimination for skyline solutions themselves, the service composers should make further selection according to their preference. However, the size of skyline set is uncontrollable [17]. For large scale or high dimensional problems, the skyline set will become too large to provide any insight for service composer's final selection. Therefore, the optimization framework used for SLA-constrained service composition should satisfy the following requirements: 1) effectively elicit and represent the preference intents of service composer; 2) efficiently find a single or a set of preferred skyline solutions according to preference information provided by service composer; 3) by tuning preference information all skyline solutions can be reached.

Since preference is subjective and vague, theoretically said, it is impossible for service composers to precisely define the optimal solutions before the distribution of skyline solutions are exactly known, no matter what methods are used to define the preference or to incorporate the preference into optimizing process. Therefore, finding a set of skyline solutions rather than a single one that conforms to the provided preference information is more useful and practical. It can give more freedom and support to service composers for their final selection. Moreover, satisfaction of requirement 3) is also very important and valuable since it can help the service composer explore different parts of skyline set by providing different preference information.

IV. Representation of Preference

For skyline solutions, improvement of QoS value on some dimensions can be attained only by allowing impairment on some other dimensions. Thus, the skyline solutions represent different kinds of tradeoff on QoS dimensions. Preferences are used by service composers to express their intents of which QoS dimensions should be decreased, which can be increased and how much the increase/decrease should be, so as to identify desired skyline solutions.

A. Fuzzy Preference Relations

Two kinds of information are required for the definition of preference relations. One is a consistent preference order (priority) of all involved QoS dimensions. The other is the preference intensity. Here we introduce a set of predicates proposed in [10] that can be used to specify different kinds (intensities) of preference relations between QoS dimensions. In addition, a set of properties with respect to those predicates are also introduced to build consistent preference order upon the dimensions. Table I presents the predicates, their semantic meanings and respective properties.

TABLE I. Fuzzy Preference Relations

Predicates	Intended meaning	Properties
$\gg$	is much more important than	irreflexive , transitive
$>$	is more important than	irreflexive , transitive
$\approx$	is equally important	symmetrc, reflexive, transitive
$\neg$	is not important	unary

Note that the last unary predicate is a special case. We use its absolute meaning rather than its relative meaning. A QoS dimension is set by ¬ means that we can deteriorate the QoS value on this dimension to the worst for improving other important QoS dimensions. Non-important (¬) dimensions are all equally important amongst themselves.We present only four levels of preference intensities, more levels can be easily extended. However more preference levels will bring out more cognitive burden to service composers.

The formal definitions of important properties are presented as following:

1)Relation $\gg$ and $>$.

$$x \gg y \wedge y \gg z \Rightarrow x \gg z \quad (1)$$

$$x > y \wedge y > z \Rightarrow x > z \quad (2)$$

2)Relation $\approx$.

$$x \approx y \wedge y \approx z \Rightarrow x \approx z \quad (3)$$

3)Miscellaneousproperties

$$x \gg y \wedge y \approx z \Rightarrow x \gg z \quad (4)$$

$$x > y \wedge y \approx z \Rightarrow x > z \quad (5)$$

$$x \approx y \wedge y > z \Rightarrow y > z \quad (6)$$

$$x \approx y \wedge y \gg z \Rightarrow y \gg z \quad (7)$$

$$\neg x \wedge \neg y \Rightarrow x \approx y \quad (8)$$

$$x \gg y \wedge y > z \Rightarrow x \gg z \quad (9)$$

$$x > y \wedge y \gg z \Rightarrow x > z \quad (10)$$

We use an example to explain how to use these properties to establish consistent fuzzy preference relations upon QoS dimensions.

Example 1: Suppose that there are seven QoS dimensions involved in the service composition, i.e. $\{q^1, \dots, q^7\}$, the service composer would be asked to specify preference relations as follows.

$$\neg q^3, \neg q^4 \quad (11)$$

$$q^5 \approx q^6 \quad (12)$$
$$q^1 > q^2, q^2 > q^5, q^2 > q^7 \quad (13)$$
$$q^5 \gg q^7 \quad (14)$$

According to relations (13), using properties (2) we get $q^1 > q^2 > q^5$ and $q^1 > q^2 > q^7$.According to (14) and using (10), we have $q^1 > q^2 > q^5 \gg q^7$.Finally, according to (12), (5) and (7), the consistent preference order can be established as:

$$q^1 > q^2 > q^5 \approx q^6 \gg q^7, \neg q^3 \wedge \neg q^4 \quad (15)$$

The non-import ant dimension (11) will be handled independently.

It should be noticed that each involved QoS dimension should appear at least once in the specified preference relations. If there are not enough relations to build complete preference order, further relations should be specified by service composer. If contradictions exist between preference relations, modification should be made. The transitivity of predicates plays an important role in the establishment of final consistent preference order. It can reduce the number of questions required for constructing the order. On the other hand, it can also help service composer identify the preference contradictions.

B. *Interpretation of Fuzzy Preference Relationsand the Weighting Procedure*

In order to incorporate preference into optimizing process, we have to quantify the fuzzy preference order and transform the relations into numeric weights. There are other pure quantitative approaches for calculation of weights, where an additional consistence checking mechanism is required (e.g. consistency ratio in AHP [10]). In our approach, since a complete, consistent preference order has been established before quantitative weighting, we can calculate weights in a more straightforward way.

Without the loss of generality, a set of normalized weights for a set of QoS dimensions $\{q^1, \dots, q^r\}$ can be defined in the formof $\{w_1, \dots, w_r\}$, where $\sum_{k=1}^{r} w_k = 1$and$w_k \in [0,1]$. In the geometrical sense, the weights can be treated as a point $W = (w_1, \dots, w_r)$ on an r dimensional hyper plane defined by $w_1 + w_2 + \cdots + w_r = 1$, where the value of each weight w_k is thecoordinate on dimension k. When we move point W on that hyper plane, different weights will be obtained. The important thing here is that if we want to increase the value on one dimension, at least one of other dimensions has to be decreased. For example, on 3-dimensional plane $w_1 + w_2 + w_3 = 1$, if $w_1 = 0.7$, the valuation of the other two dimension has to satisfy the condition $w_2 + w_3 = 0.3$. This can be explained aswe want to improve the value dimension w_1 up to 70% of its value range by deterioratingthe values of other two dimensions to their 30%. When the preference relation between w_2 and w_3 is defined, all three weights can be determined. For example, if we further want to improve the value of w_2 to its 70%, the final weights are $(0.7,0.21,0.09)$. This is an important reason why weights can be used to quantitatively express the tradeoff intentions of service composers according to their preference on different QoS dimensions. By tuning weights, all possible consistent preference order can be expressed, and vice versa.

According to the above explanation, we present the following weighting procedure to transform consistent fuzzy preference order into a set of numeric weights.

1) Classify involved QoS dimensions into a set of equally important classes according to the preference order.

For example, according to the preference order (15) in example 1, we get following five equivalent classes (note that all non-important dimensions are equally important):

$c_1 = \{q^1\}, c_2 = \{q^2\}, c_3 = \{q^5, q^6\}, c_4 = \{q^7\}, c_5 = \{q^3, q^4\}$

2) Consider only important classes and their preference order while ignoretemporary non-important classes.If there is only one important class, assign weight to this class by 1, and skip to step 5).

Continuing the example in step 1), we get preference order for four important classes:

$$c_1 > c_2 > c_3 \gg c_4 \quad (16)$$

3) Quantify different kinds ofpredicates(i.e.different fuzzy prefence relations) as different real number ranging from 0 to 1.Note that it is unnecessary here to quantify the predicate $\approx$ after classification in step 1).

For example, we can quantify two predicates proposed in this paper as $v(\gg) = 0.8$ and $v(>) = 0.6$.According to the semantic meaning of predicate $\gg$ and $>$ (see Table I), following constraints should be satisfied, i.e.$0.5 < v(>) < v(\gg) < 1$.Otherwise we will get inverse preference order. By presenting different valuation for predicates, different weights will be resulted.

4) Calculate weight for each class according to their preference order. Assuming there are totally m important classes after performing step *2)*, p_i^{pre} is the i_{th} predicates before k_{th} classc_k in the $preference$ order andp^{after} is the first predicator after c_k , the weight of class c_k can be caculated as

$$w(c_k) = \begin{cases} v(p^{after}) & k = 1 \\ v(p^{after}) \prod_i \left(1 - v(p_i^{pre})\right) & 1 < k < m \\ \prod_i \left(1 - v(p_i^{pre})\right) & k = m \end{cases} \quad (17)$$

For example, we can calculate weights for each class in preference order (16): $w(c_1) = 0.6, w(c_2) = 0.24, w(c_3) = 0.128, w(c_4) = 0.032$.

5) Assign weight toeachQoS dimension in class c_k as $w(c_k)$, assign zeroto non-important dimensions, andperform normalization(i.e. adjusting weights so that their sum is 1 by recaculating $w_i = w_i / \sum w_j$, 1<i, j<r)for all weights.

Therefore, the normalized weight of each QoS dimension (approximate value)of example 1 is $w_1 = 0.53, w_2 = 0.21, w_3 = 0, w_4 = 0, w_5 = 0.12, w_6 = 0.12, w_7 = 0.03$.

C. *The Weighted Tchebycheff Distance*

After the weighting procedure, we leverage the weighted Tchebycheff distance to each skyline solution, so that the skyline solutions can be ranked and preferred skyline

solutions can be identified. The weighted Tchebycheff distance (WTD) for solution cs is defined as following:

$$d^T = \max\{w_k(q^k(cs) - q_*^k)\}$$

where$k \in [1, r]$, $q^k$and w_k are the QoS value and weight on k_{th} dimension respectively,q_*^kisthe best (minimal) value on k_{th} dimension with respect to all possible solutions $cs \in CS$. Since$d^T$is calculated after scaling, the best QoS value on each dimension must be zero, i.e. $\forall k \in [1, r], q_*^k = 0$.

For certain candidate solution cs , the weighted Tchebycheff distance deal with its multiple QoS dimension separately, where each weighted QoS value term, i.e.$w_k(q^k(cs) - q_*^k)$, correspond to one QoS dimension in the QoS vector, and the biggest one is taken to be the measurement of the overall degree of cs achieving to the best QoS value. This is a way different from linear utility function, where the summation of all weighted QoS value terms is taken as quality measurement for each candidate solution in the multi-objective optimization problem.

From the optimization viewpoint, if one solution in the candidate set has smaller value of d^T, it is better. For skyline solutions, since no single solution can have better QoS value on all dimensions at the same time, the weighted Tchebycheff distance is the measurement of preference degree, where the weights represent tradeoff rate on different dimensions. If skyline solution incs^{sky} set has smaller value of d^T, it is more preferred.

The main reason we introduce the weighted Tchebycheff distance into SLA-constrained service composition problem is to avoid the limitation of traditional utility-based approaches. The capability of the weighted Tchebycheff distance for identifying preferred skyline solutions has been proved since 1976, now it is one of the most common scalarization methods in multi-objective optimization [8].Its main advantage lies that every skyline solution can be reached in terms of minimal value of d^T by appropriately tuning the weights regardless of whether the problem is convex or non-convex, or discrete, where the utility function is effective only for convex problem.

However, the weighted Tchebycheff distance used alone cannot differentiate non-dominated (skyline) solution and weakly non-dominated solution[8][9]. For optimization, the best solution in terms of weighted Tchebycheff distance may not be skyline solution but weakly non-dominated solution. This is the reason we propose to combine the weighted Tchebycheff method with skyline computation for solving the optimization problem of SLA-constrained service composition.

V. THE HYBRID EVOLUTIONARY ALGORITHM

The multi-objective evolutionary algorithm (MOEA) is an important and widely used type of meta-heuristic algorithm for solving large scale multi-objective optimization problems. Although many traditional MOEAs (e.g. NSGAII [20]) can be used to efficiently find a set of skyline solutions or near-to skyline solutions that are uniformly distributed and diverse over the entire skyline set, they do not meet our goal. In this section, we present our hybrid evolutionary algorithm called hybrid_EA. The algorithm is designed on the basis of Pareto-dominance and the weighted Tchebycheff distance.

A. Design of Gene and Populations

In hybrid_EA, populations are maintained for a set of individuals, of which each individual represents a certain candidate solution as well as its scaled QoS vector. We use sequence of integer gene to encode candidate solutions according to the structure of service composition. Each gene represents one service used in service composition and its values encode the alternative back-end SLA bindings for this service. Figure 1 illustrates a gene sequence representing candidate solutions for a service composition that consists of seven services with 20 alternative SLA bindings each. During the evolutionary process, different solutions are given by generating different gene sequences. For each solution, the QoS vector is calculated and scaled using approaches mentioned in Section III and Section IV.

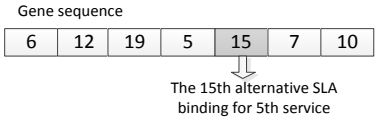

Figure 1. Example of gene sequence

In the process, if elitist individuals (i.e. solutions with better QoS) in current generation are preserved and incorporated into the evolutionary operations together with individuals of next generation, it is called elitism. Elitism has been widely used in different kinds of MOEAs, which can improve the performance of EAs significantly [21].To implement elitism, we maintain three populations in hybrid_EA, i.e. current population P, next population Q and combined population R. The number of individuals in P and Q is$N$, while the number of individuals in R is$2 * N$. In each generation, individuals in Q are generated by evolution of individuals in P. The individuals in P and Q are then combined to construct R. Ranking is performed for all individuals in R and elitist individuals will be copied to Pin order to start a new generation. In this way, since elitist individuals (P) are always ranked together with offspring individuals, they are preserved naturally during the evolutionary process.

B. Two-Phase Ranking Procedure

The elitist individuals in the combined population R are determined through ranking. The proposed ranking includes two phases. First, we leverage an efficient Pareto ranking procedure called *non-dominated sorting* [20] to rank the individuals in R, and then get a set $\{r^1, \dots, r^i, \dots, r^n\}$ in which r^i records a set of individuals that have the same rank value i (the detail see [20]). By non-dominated sorting, the individuals are enabled to evolve towards skyline solutions efficiently. Second, the weighted Tchebycheff distanced^T is calculated for each individual in R as the secondary rank value. It enables the individuals evolving towards preferred region in a skyline set.

Considering SLA constraints, the individuals should firstly be encouraged to converge toward feasible solutions. By feasible we mean those solutions that satisfy given SLA

constraints. For this purpose, when comparing two individuals in the first phase of ranking, we replace the original Pareto dominance by a new comparison operator called *SLA-constrained dominance*. For any two individuals p_i and p_j, we say p_i SLA constraint dominates p_j when p_i is feasible while p_j is not, or both of them are feasible or infeasible while p_i Pareto dominates p_j.

C. Main Loop

The main loop defining the complete evolutionary process of the hybrid evolutionary algorithm is shown from line 4 to 19.At first, three evolutionary operations, i.e. selection, crossover and mutation, are performed upon all individuals in current population P to generate offspring individuals (line 5). We use *binary tournament selection*, *single point crossover* and *bit wise mutation* to implement the evolution respectively. Among them, binary tournament selection is used to select individuals that will be operated by crossover and mutation to reproduce next generation individuals. In the operation, two individuals are first picked from P randomly. The better one will be selected according to their rank values. For any two picked individuals p_i and p_j, if following condition is satisfied we say p_i is better than p_j:

$$p_i \prec_r p_j \Leftrightarrow (p_i.r < p_j.r)$$
$$or\ \big((p_i.r = p_j.r) and (p_i.d^T < p_j.d^T)\big)$$

The generated offspring individuals are inserted into next population Q until it is filled. Meanwhile, QoS vector for each individual of Q is calculated and scaled (line 6). Both individuals in P and Q are copied into combined population R (line 7), and then the two-phase ranking procedure is performed upon R (line 8-9).

After ranking, the top N (N is the population scale of P) elitist individuals in R should be copied back to population P so that the elitism can be implemented. First the individuals in rank list r^1 are copied, and then the rank list r^2, and so forth, until population P is filled (line 11, 15-18). If the number of individuals in last copied rank list r^{last} exceeds the number required for filling P, individuals in r^{last} are sorted in ascending order firstly according to their weighted Tchebycheff distance d^T, then the top-k individuals will be chosen to fill the population P (line 12-14).

In addition, the duplicated individuals may be generated by crossover or mutation. If the duplicated individuals are just elitists, they are always preserved. Moreover, the elitism can enhance duplication and decrease the diversity of populations when resulting in more and more duplicated individuals. This behavior will deteriorate the performance seriously. To eliminate duplication, we propose to check duplication when copying elitists. Before one elitist individual p_i is copied from population R to P, check if there is duplication in the individuals that have been copied in P. If duplication does exist, discards p_i and pick next elitist individual from R, otherwise p_i is copied into P (line 16-17).

Outside the main loop, there are no elitist individuals that can be evolved at the beginning of the algorithm. Therefore, in our algorithm a set of initial individuals of P are generated randomly (line 1-3). On the other hand, the stop condition can be stopping after fixed number of iterations e.g. 200 iterations. We can also maintain a variable to record the average d^T value in terms of all individuals in P for each iteration. If this variable does not change for fixed number (e.g. 50) of consecutive iterations, the algorithm can be stopped. Individuals in P are returned as the final result.

Hybrid evolutionary algorithm

1. P←randomly generate N individuals
2. Calculate scaled QoS vecor for each individual in P
3. P←two-phase rank individuals in P
4. Repeate until stop condition is satisfied{
5. Q←selection, crossover and mutation operate on P
6. Calculate scaled QoS vecor for each individual in Q
7. R←P ∪ Q
8. R←Pareto-rank individuals in R, generate{$r^1, r^2, \dots, r^n$}
9. R←calculate d^T for each individual of R
10. P←∅
11. for each r^i in {$r^1, r^2, \dots, r^n$} and P is not filled {
12. if r^i can fill P {
13. sort r^i in ascending order according to d^T
14. }
15. for each individual p_i in r^i {
16. if duplication does not exist for p_i in P{
17. P←p_i
18. }
19. }}}

D. Complexity analysis

Consider the complexity of one loop in the hybrid evolutionary algorithm, following operations and their worst-case complexity is analyzed. First, the complexity of non-dominated sorting is $O(M(2N)^2)$ [20].Second, the sorting on Tchebycheff distance is only required for top-k individuals in last copied rank list, in worst case $k = N$, then its complexity is $O(Nlog(N))$. Third, in worse case for duplication checking, the last individual copied to fill population P will be discarded for N times, thus its complexity is $O(2N^2)$. Since the complexity of three evolutionary operations is $O(N)$, the overall complexity is $O(M(2N)^2)$ where M is the number of QoS dimensions and N is the population size of P.

VI. Experimental Study

In this section, we present a set of experimental studies to evaluate the performance of our hybrid evolutionary algorithm (hybrid_EA). In the first set of experiments, we focus on result quality and convergence behavior. We apply hybrid_EA to optimize a middle scale service composition problem which has 10^{10} candidate solutions, and then investigate how the solutions produced in population P converge to the most preferred skyline solutions during evolutionary process and what quality of the final result set is after fixed number of generations. In the second set of experiments, we evaluate the efficiency and scalability of hybrid_EA when confronting different problem scales.

As a random search algorithm, the hybrid_EA converge to desired solutions through continuous evolutionary process. However, it cannot be guaranteed that all top-k preferred skyline solutions must be found after certain number of iterations. For the purpose of evaluation and comparison, we develop a base line algorithm called hybrid_BNL, which is

based on BNL [19] (block nested loop) to produce exact optimal solutions. Both the algorithms hybrid_EA and hybrid_BNL are implemented in Java. All experiments are conducted on a machine with 2.1GHz Intel dual cores CPU and 2 GB of RAM, running windows.

A. ExperimentSetup

We define the following experiment settings for hybrid_EA. The sizes of the three different populations i.e. P, Q and R are set to 100, 100 and 200 respectively. Crossover probability of 0.9 and mutation probability of $1/l$ are used for all experiments, where l is the number of genes in an individual. The algorithm is run for maximum of 500 generations.

Before evaluating hybrid_EA, we apply hybrid_BNL to produce the exact results under the same parameter settings of the SLA-constrained service composition problem. We then compare the solutions produced by each generation of hybrid_EA with the solutions in exact results. Based on the comparison, we calculate the percentage of exact optimal solutions found by hybrid_EA so as to investigate its optimizing behavior. The related parameters include the number of services in composition (m), the number of candidate SLA bindings for each service (n) and the number of QoS dimensions (d). We present the results along with specific parameter settings in the next sub-sections. All experiments are conducted independently for 50 times and the average results are calculated.

In addition, we adopt synthetically generated data sets due to the limited availability of real data. The QoS values of candidate SLA bindings are generated in three different ways[17]: independent (ind) which means the QoS values are randomly set; correlated (cor) which means all dimensions are positively correlated, i.e. good values on some QoS dimensions will lead to good values on others; anti-correlated (anti-cor) which means the QoS dimensions are negatively correlated, i.e. the SLA binding have good QoS value on some dimensions but bad on others. Besides, the SLA constraints and weights used in experiments are generated by random.

B. Convergence BehaviorandResult Quality

In this set of experiments, we focus on the convergence behavior of hybrid_EA. In each independent running we evaluate the quality of solutions in population P for each generation in terms of different kinds of optimization objectives, i.e. feasible, constraint skyline and top-k most preferred skyline solutions, where k=100, 50, 20.As the number of generation increases, we calculate the percentage of certain kind of exact optimal solutions found in P.

Figure 2(a) illustrates the convergence behavior of hybrid_EA for finding feasible, constraint skyline and top-100 preferred skyline solutions. Resultsof per 50 generations are presented. As we can see, after less than 50 generations (actually around 15 generations) all solutions in P have been feasible. At around the 100th generation, all solutions in P have been constraint skyline solutions. Nearly 100% (98.6% of average) top-100 most preferred skyline solutions have been found after 300 generations. This behavior indicates that hybrid_EA can converge progressively to feasible, then to skyline, finally to preferred skyline solutions. Figure 2(b) presents the results of per 50 generations for finding top-20, top-50 and top-100 preferred skyline solutions, where the progressively converging behavior can also be found.

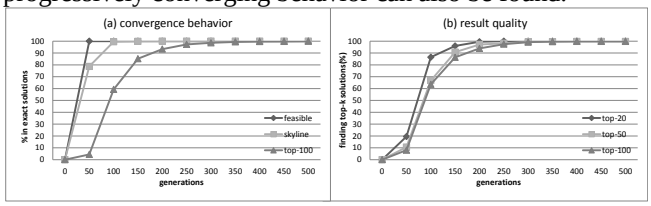

Figure 2. Convergence behavior and result quality, default setting: n=10,m=10,d=7, independent data

As a stochastic searching algorithm, although it is a very hard task to find exactly top-100 preferred skyline solutions from 10^{10} candidates, hybrid_EA can converge fast toward desired solutions in the early stage of searching (more than 90% are found before 200th generation in Figure 2). Thenthe convergence of hybrid_EA becomes slower due to the difficulty of generating certain solutions by stochastic evolutionary operations. Even so, all the top-100 preferred solutions are found successfully in less than 500 generations. In particular, the elitism mechanism guarantees the optimal solution to be preserved until the end of the algorithm once it is found.

C. Efficiency and Scalability

In the second set of experiments, we apply hybrid_EA to find top-100 most preferred skyline solutions under different parameter settings. In such case, we adjust the value of one parameter while keeping the others'. The convergence behavior of hybrid_EA is investigated using the same approach mentioned in the previous sub-section.

Figure 3 presents the results for different values of m (plot-a) and n (plot-b). Note that bigger value implies larger number of candidate solutions. Therefore it is expected that more generations and more time cost are required to find the same number of desired solutions for larger problems. As showed in (a), for all different values of m, hybrid_EA has similar convergence behavior and more than 95% top-100 solutions are found in less than 500 generations. The similar results can be found for different values of n in (b).On the other hand, when problem scale increases, the convergence of hybrid_EA becomes a little slower. The respective time cost under different parameter settings when finding 95% of top-100 preferred solutions are presented in table II and table III.As we can see, compared to exact algorithm such as hybrid_BNL, the time cost of hybrid_EA is fairly proportional to the problem scale with a small constant proportionality while the time cost of exact algorithm grows exponentially. It means that hybrid_EA is significantly more efficient than exact algorithms, especially for large scale problems.

Figure 4 shows the experiment results for different number of QoS dimensions (a) and for different QoS correlations (b). In less than 500 generations hybrid_EA can successfully find more than 95% desired solutions for all

different QoS settings, where similar convergence behavior are illustrated. Meanwhile, the convergence of hybrid_EA becomes slightly slower when the number of QoS dimensions increases. The reason lies that more QoS dimensions leads to bigger skyline size, thus the searching space for finding preferred skyline solutions is increased. The same situation exists for different QoS correlations. In it the anti-correlated QoS results in bigger skyline set than independent QoS, while the correlated QoS results in smaller skyline set.

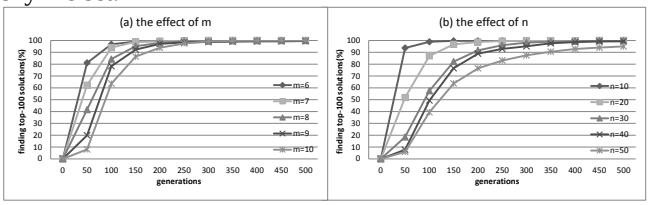

Figure 3. Effect of propblem scale, default setting(a): n=7,d=7, independent data.default setting(b):m=5,d=7,independent data

TABLE II. TIME COST FOR DIFFERENT PROBLEM SCALE

Algorithm	Time cost for finding 95% of top-100 solutions				
	m=6	***m=7***	***m=8***	***m=9***	***m=10***
Hybrid_EA	0.9(s)	1.2(s)	1.7(s)	2(s)	2.5(s)
Hybrid_BNL	0.9(s)	8.5(s)	83(s)	906(s)	9735(s)

n=7, d=7, independent data.

TABLE III. TIME COST FOR DIFFERENT PROBLEM SCALE

Algorithm	Time cost for finding 95% of top-100 solutions				
	n=10	***n=20***	***n=30***	***n=40***	***n=50***
Hybrid_EA	0.7(s)	1.6(s)	2.9(s)	3.6(s)	5.5(s)
Hybrid_BNL	0.3(s)	2.3(s)	16(s)	65(s)	205(s)

m=5, d=7, independent data.

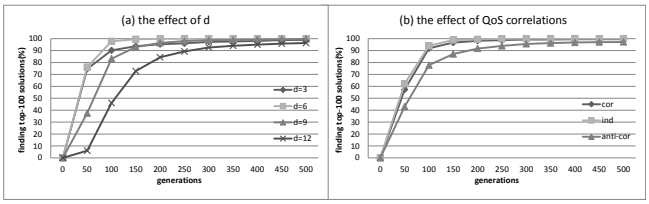

Figure 4. Effect of QoS number and QoS correlations, default setting: n=7,m=10,d=7,independent data

VII. CONCLUSION

In this paper, we present a systematic approach to cope with the optimization problem of SLA-constrained service composition. In the approach the preference on QoS dimensions can be elicited, represented and established as consistent preference order based on a set of fuzzy linguistic predicates. Then a novel weighting procedure to transforming the preference relations into numeric weights is proposed, which is further used in the weighted Tchebycheff distance to identify preferred skyline solutions. Finally we present a hybrid evolutionary algorithm to find a set of preferred skyline solutions heuristically. As for the future work, we plan to integrate our proposed approach in general SLA-oriented service composition management framework.

ACKNOWLEDGEMENT

This work is supported by National High Technology Development 863 Program of China under Grant No.2013AA01A605.

REFERENCES

[1] J. Evdemon, A. Arkin, A. Barreto, B. Curbera, F. Goland, G.Kartha, et al., "Web Services Business Process Execution Language Version 2.0,"BPEL4WS Specification, 2007.

[2] J. Misra and W. R. Cook, "Computation Orchestration: A Basis for Wide-area Computing," Springer J. of Software and Systems Modeling, vol. 6, no. 1, pp.83 – 110, Mar. 2007.

[3] Z. Zheng, Y. Zhang, M. R. Lyu, "Distributed QoS Evaluation for Real-World Web Services," ICWS 2010: 83-90

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,"Future Generation Comp. Syst. 25(6): 599-616 .2009

[5] A. Dan, D. Davis, R. Kearney, et al, "Web services on demand: WSLA-driven automated management,"IBM Systems Journal 43(1): 136-158, 2004

[6] M. Comuzzi, B. Pernici: A framework for QoS-based Web service contracting. TWEB 3(3) (2009)

[7] S. Schenkerman, "Use and abuse of weights in multiple objective decision support models," Decision Sciences 1991;22:369-378.

[8] K. D ̈achert, J. Gorski,K. Klamroth, "An augmented weighted Tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems,"Computers & OR 39(12): 2929-2943 , 2012.

[9] W. Chen, M. Wiecek, and J. Zhang, "Quality utility: a compromise programming approach to robust design," Journal of Mechanical Design, vol. 121, no.2, pp. 179-187, 1999.

[10] I. C. Parmee, D. Cvetkovic, "Preferences and their application in evolutionary multiobjectiveoptimization,"IEEE Trans. Evolutionary Computation 6(1): 42-57 (2002)

[11] D. Ardagna, B. Pernici, "Adaptive Service Composition in Flexible Processes,"IEEE Trans. Software Eng. 33(6): 369-384 (2007)

[12] T. Yu, Y. Zhang, K.J. Lin,"Efficient algorithms for Web services selection with end-to-end QoS constraints,"TWEB 1(1) (2007)

[13] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, Q. Z. Sheng, "Quality driven webservices composition,"WWW 2003: 411-421

[14] M. Alrifai, T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," WWW 2009: 881-890.

[15] M. Alrifai, D. Skoutas, T. Risse, "Selecting skyline services for QoS-based web service composition,"WWW 2010: 11-20.

[16] F. Rosenberg, M. B. Mu ̈ller, P. Leitner, et al, "Metaheuristic Optimization of Large-Scale QoS-aware Service Compositions," IEEE SCC 2010: 97-104.

[17] C. Y. Chan, H. V. Jagadish, K.L. Tan, A. K. H. Tung, Z. Zhang, "Finding k-dominant skylines in high dimensional space,"SIGMOD Conference 2006: 503-514.

[18] Q. Yu, A. Bouguettaya, "Computing Service Skylines over Sets of Services,"ICWS 2010: 481-488

[19] D. Papadias, Y. Tao, G. Fu, B. Seeger, "Progressive skyline computation in database systems,"ACM Trans. Database Syst. 30(1): 41-82 (2005).

[20] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE Trans. Evolutionary Computation 6(2): 182-197 (2002)

[21] A. Konak, D. W. Coit, A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," Rel. Eng. & Sys. Safety 91(9): 992-1007 (2006)

[22] H. Wada, J. Suzuki, Y. Yamano, K. Oba, "E3: A Multiobjective Optimization Framework for SLA-Aware Service Composition," IEEE T. Services Computing 5(3): 358-372 (2012)