An Evolutionary Multitasking Algorithm for Cloud Computing Service Composition

Liang Bao[1(✉)], Yutao Qi[2], Mengqing Shen[1], Xiaoxuan Bu[1], Jusheng Yu[2], Qian Li[3], and Ping Chen[1]

[1] School of Software, XiDian University, No. 2 South Taibai Road, Xi'an, China
{baoliang,chenping}@mail.xidian.edu.cn
[2] School of Computer Science and Technology, XiDian University, No. 2 South Taibai Road, Xi'an, China
ytqi@xidian.edu.cn
[3] School of Economics and Finance, Xi'an Jiaotong University, No. 74 Western Yanta Road, Xi'an, China
lqian@mail.xjtu.edu.cn

Abstract. Service composition is a convincing approach for rapidly constructing large-scale distributed applications in public clouds. With the rapid increase of the composite service requests from many concurrent clients in public clouds, it is critical to perform quality of service (QoS) aware cloud computing service composition (CCSC) efficiently. To address this issue, many approaches have been proposed. However, it remains a key challenge to improve the throughput and the solution quality of a CCSC solver. In this paper, we propose a novel algorithm, namely evolutionary multitasking algorithm for CCSC problem (EMA-CCSC), based on the evolutionary multitasking algorithm. Unlike existing CCSC solvers which have to pool the composite service requests in the waiting queue first and then solve them once a time, the proposed EMA-CCSC is able to optimize two CCSC tasks concurrently. As a result, it can deal with more requests at a fixed period of time. Based on the QWS data set including 2507 real Web services, experiments have been conducted by solving a sequence of 1188 randomly generated CCSC tasks with different sizes and structures. The results indicate that EMA-CCSC outperforms 7 out of 9 compared algorithms with different characteristics, even though it spends only half of their computing costs. We can draw the conclusion from the extensive experiments that the EMA-CCSC approach is competitive in both solution quality and time efficiency.

Keywords: QoS-aware cloud computing service composition
Evolutionary optimization · Multitasking algorithm

1 Introduction

Service composition has been adopted as a standard computing paradigm for rapidly constructing large-scale distributed applications within and across orga-

A. Yang et al. (Eds.): SERVICES 2018, LNCS 10975, pp. 130–144, 2018.
https://doi.org/10.1007/978-3-319-94472-2_10

nizational boundaries. Many modern enterprises have chosen to construct flexible applications by dynamically composing atomic services of wide variety and different quality of service (QoS) attributes.

Recently, with the bloom of cloud computing, the importance of affordable access to reliable high-performance hardware and software resources and avoiding maintenance costs has encouraged many decision makers to migrate enterprise applications partially or entirely from traditional distributed computing platforms (e.g., grid) to clouds.

Fast development in the utilization of cloud computing leads to publishing more cloud services on the worldwide service pool. Because of the prevalent presence of complex and diverse applications, it is essential to have a group of simple services that work with each other to meet the practical requirements for real-world cases. Therefore, there is a strong need to deploy a service composition system in cloud computing [14].

Figure 1 gives a system architecture for modeling and executing composite service in a public cloud. At design phase the provider of the composite service defines the set of required atomic services and their relations using a workflow-like language such as WS-BPEL[1] or YAWL[2] to fulfill the business goal. After that, service discovery is performed by exploiting the existing computing infrastructure (e.g. UDDI) to locate available atomic services for each task in the workflow. As a result, a collection of functionally equivalent atomic services (referred to as candidate services) is obtained for each task.

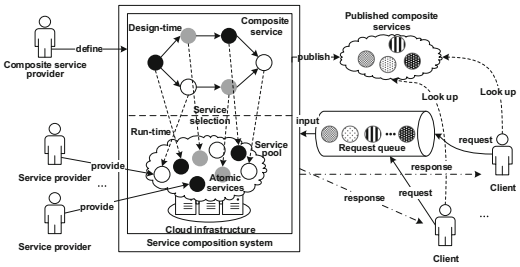

Fig. 1. Architecture of a cloud computing service composition system

At runtime phase, QoS-aware service selection is performed upon service request in order to select a set of appropriate atomic services from collections of candidate services such that the aggregated QoS values satisfy the client's QoS requirements.

As shown in Fig. 1, there are lots of similar services that are located in different places, implemented by different service providers, and have distinct values in terms of the QoS parameters. Therefore, the cloud computing service composition (CCSC) problem [14], i.e. selecting appropriate and optimal atomic services

[1] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
[2] http://www.yawlfoundation.org/.

to be combined together to provide composite services such that the obtained complex composite service satisfies both the functional and QoS requirements based on the end-user requirements, is one of the most important problems in providing cloud services. A substantial effort has been made to solve this problem, with a lot of research works produced.

More importantly, the open and shared natures of cloud computing bring a brand new challenge to the service composition system: it must has the ability to handle a lot of diverse requests from many concurrent clients. More concretely, users will look up and invoke their desired composite services independently, these requests are then received, queued, and processed by the service composition system. Since the response time is one of the most important QoS indicators for invoking composite services from the clients' perspective, an efficient selection algorithm with high-throughput, i.e. the algorithm that can handle with more requests within a fixed period of time, is critical for a service composition system to fulfill the performance requirements.

To deal with numbers of requests, administrators of service composition systems typically apply some simple and ad hoc dispatching algorithms, such as packing, striping, load-balance etc., to distribute these requests to different solvers. These solvers select appropriate atomic services for their own dispatched composite services from the global service pool independently. Although this approach has some advantages in terms of simplicity and scalability, it suffers from the bad performance and totally overlooks the possible similarities and connections among a bunch of requests, which have been proven to be very useful to improve the performance of algorithms and the quality of solutions according to our experiments. Such observation motivates us to solve multiple CCSC tasks simultaneously by using a unique optimizer in a single run.

Multifactorial optimization (MFO) is a newly developed algorithmic paradigm in the field of optimization and evolutionary computation. Instead of solving a pool of similar optimization problems singly, it handles multiple optimization problems at the same time by using a shared evolving population [13]. Due to the perfect match between the MFO paradigm and the CCSC problem, we propose an evolutionary multitasking algorithm (EMA) in this paper to solve CCSC problem, in which we aim to test the feasibility of the simultaneous optimizations of multiple CCSC tasks. The main contributions of this paper thus can be summarized as follows:

1. We propose a novel CCSC solver named CCSC-EMA based on the evolutionary multitasking algorithm, and explain its detailed process including the representation of solutions, quality assignment, reproducing and selection of new solutions.
2. We design new experimental and analytical methodologies to make comparisons between evolutionary multitasking algorithm and conventional single objective CCSC solvers.

2 Related Work

2.1 Evolutionary Multitasking Algorithm and Its Application

Evolutionary multitasking algorithm (EMA) is a new optimization paradigm proposed recently by Ong [19]. In contrast to traditional evolutionary optimization approaches, which focus on solving only a single optimization problem at a time, EMA was proposed to solve multiple optimization problems simultaneously. Ong and Gupta [20] presented a simple evolutionary methodology capable of cross-domain multitask optimization in a unified genotype space, and show that there exist many potential benefits of its application in practical domains. Zhou et al. [38] proposed a permutation-based EMA to improve the multi-tasking performance in the context of vehicle routing problem. Gupta et al. [13] developed a cross-domain optimization platform that allows one to solve diverse problems concurrently. Gupta et al. [12] showed that the practicality of population-based bi-level optimization can be considerably enhanced by simply incorporating the novel concept of evolutionary multitasking into the search process. Yuan et al. [33] focused on the evolutionary multitasking of PCOPs. Four kinds of well-known PCOPs, i.e., TSP, QAP, LOP and JSP are considered. Da et al. [7] suggested to solve a target single-objective optimization (SOO) task in conjunction with a closely related (but artificially generated) multiobjective optimization (MOO) task in the form of evolutionary multitasking. Sagarna and Ong [22] focused on branch testing and explore the capability of EMA to guide the search by exploiting inter-branch information.

2.2 Algorithms for CCSC Problem

Service composition techniques were first applied in cloud computing systems by Kofler et al. [17] and Zeng et al. [34] in 2009 [14]. The CCSC problem is often defined as an instance of the classic multiple-choice multidimensional knapsack problem (MMKP) [32] which searches for the composition that has the best composite QoS values when satisfying QoS constraints of atomic services. Such problem is NP-hard [16] and usually includes numerous constraints, on which the exact approaches can not perform well when large amounts of services are involved. Zeng et al. [34] presented a matching algorithm that considers the semantic similarity of multiple input and output parameters based on WordNet in the process of service matching. Cui et al. [6] proposed a service graph constructing algorithm to transform the CCSC problem into the optimal path finding problem in graph. Torkashvan and Haghighi [28] proposed a greedy approach to solve the CCSC problem by mapping workflows to composed services considering all flow structures. Zhang et al. [37] proposed a cluster-based DSP scheduling problem algorithm for service composition in multi-domain environment with time constraint. Wang et al. [29] presented a graph model that takes both QoS of Web services and QoS of network into consideration. Ba [3] proposed the automation of service composition that takes the abstract specification of a composition and the definition of concrete services. Syu et al. [26]

proposed an automatic composition approach through genetic algorithm (GA) to satisfy user's functional requirements, QoS criteria and transactional requirements automatically. Guidara et al. [11] presented a heuristic based time-aware service selection approach to select a close-to-optimal combination of services. Deng et al. [9] utilizes a constraints based service filtering process to reduce the searching space and adopts a differential evolutionary based algorithm to form a service combination. Rodriguez-Mier et al. [21] presented an A* algorithm which solves the problem of semantic input-output message structure matching for service composition. Mabrouk et al. [18] presented QASSA, a service selection algorithm that provides the appropriate ground for QoS-aware service composition in ubiquitous environments.

According to our previous investigation, one of the most important issues in CCSC problem is how to improve the throughput of the optimizers, because there are often a large number of independent composite service requests in public clouds. This observation motivates us to explore the possibilities and solutions of optimizing multiple CCSC tasks concurrently.

3 Problem Definition

3.1 Quality Criteria of an Atomic Service

There are lots of services available in public clouds, some have distinct functions while others have similar functions. Services having similar capabilities are distinguishable via their QoS values. QoS defines the overall performance of a service, the QoS values of a service indicate whether it is reliable, available, or efficient.

Services are usually advertised with multiple QoS values, each value represents a quality aspect of the service called a QoS criterion. QoS criteria are generally the most commonly used characteristics in measuring the quality of services, this is because they indicate whether a service is capable of measuring up to user's expectations [24]. Based on Web service benchmark [1,2] and previous studies [23,24], we identify six generic quality criteria [4], namely *price*$(q_p(s))$, *duration*$(q_d(s))$, *availability*$(q_a(s))$, *throughput*$(q_t(s))$, *successful rate*$(q_s(s))$, and *reliability*$(q_r(s))$, for an atomic service s in this paper.

3.2 Structures of a Composite Service

A composite service cs can be modeled as a directed acyclic graph (DAG) $cs = (S, E)$, where vertices $S = \{s_1, s_2, ..., s_n\}$ represent a set of atomic services and s_1, s_n represent the starting and ending services respectively. The dependency between a pair of adjacent services s_i, s_j is denoted by a directed edge $e_{ij} \in E$ between them.

To construct a composite service, atomic services need to be connected by different structures. In this paper, we consider four service composition structures: *Sequence*, *Concurrency*, *Condition* and *Loop*. For more details, see [4].

3.3 Service Selection

Given a composite service cs containing n atomic services $S = \{s_1, s_2, \cdots, s_n\}$, we must select a candidate for each $s_i \in S$ from a candidate service set to implement s_i. A candidate service set is the collection of atomic services having the same functionality but different QoS properties. The typical process of service selection for a composite service having three atomic services can be illustrated by Fig. 2. Note that we may choose candidates from the same candidate service set for different atomic services if they have the same function.

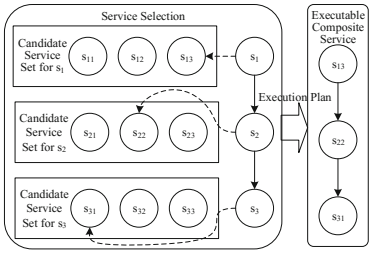

Fig. 2. Service selection for a composite service

Generally, let a composite service $cs = (S, E)$ has m candidate service sets $C = \{C_1, C_2, \cdots, C_m\}$, and for any $C_j \in C$, it contains a set of k candidate services $C_j = \{c_1^j, c_2^j, \cdots c_k^j\}$. We first define a function $\mathcal{M}(s_i) : S \to C$ to map each atomic service $s_i \in S$ to its corresponding candidate service set $C_j \in C$, i.e. $\mathcal{M}(s_i) = C_j$. Note that because the mapping between an atomic service to its corresponding candidate service set is often determined manually by the designer of a composite service, we assume $\mathcal{M}(\cdot)$ is given in this paper. Last, we define a matrix $\boldsymbol{x} = [x_{i,j}]_{|S| \times \max_{k=1}^{n}(|\mathcal{M}(s_k)|)}$ to represent the service selection scheme for each atomic service in S, where $\boldsymbol{x}$ has $|S|$ rows and $\max_{k=1}^{|S|}(|\mathcal{M}(s_k)|)$ columns, and the max function is applied to make the alignment on the columns of $\boldsymbol{x}$. For example, suppose we choose the candidate service $c_k^j \in C_j$ to implement an atomic service $s_i \in S$, then $x_{i,k} = 1$.

3.4 Execution of a Composite Service

As shown in Fig. 2, once we finish selecting the candidate for each atomic service in a composite service cs, we can generate an execution plan for cs. Given the execution plan, cs becomes executable and its QoS properties can be calculated by aggregating QoS properties of its component candidates. The detailed aggregation rules for four structures and six different QoS properties adopted in this work have been discussed in [4].

To represent the preference for QoS properties of different users, we define a utility function $\mathcal{F}(cs, \boldsymbol{x})$ for a composite service cs by a weighted sum of six quality criteria:

$$\begin{aligned}\mathcal{F}(cs, \boldsymbol{x}) = \alpha_1 \cdot -Q_p(cs, \boldsymbol{x}) + \alpha_2 \cdot -Q_d(cs, \boldsymbol{x}) \\ + \alpha_3 \cdot Q_a(cs, \boldsymbol{x}) + \alpha_4 \cdot Q_t(cs, \boldsymbol{x}) \\ + \alpha_5 \cdot Q_s(cs, \boldsymbol{x}) + \alpha_6 \cdot Q_r(cs, \boldsymbol{x})\end{aligned}$$

where α_i $(0 < \alpha_i < 1)$ is the weight for each QoS property and $\sum_{i=1}^{6} \alpha_i = 1$, $\boldsymbol{x}$ denotes the service selection scheme for cs, and we have:

$$Q_*(cs, \boldsymbol{x}) = \underset{s_i \in S \wedge x_{i,j}=1}{aggr} q_*(c_j^i),$$

the function $Q_*(\cdot)$ calculates the six QoS values of cs by aggregating corresponding QoS values of all candidates selected for its atomic services [4].

3.5 Problem Formulation

Given a set of composite service $CS = \{cs_1, cs_2, \cdots, cs_n\}$, the cloud computing service composition (CCSC) problem is to select service candidates from a group of candidate service sets to construct the execution plan for each $cs_i = (S_i, E_i) \in CS$, so that it can maximize the total QoS gain of CS.

Mathematically, the CCSC problem can be formulated as follows:

$$\text{Maximize} \sum_{cs_i \in CS} \mathcal{F}(cs_i, \boldsymbol{x^{(i)}}) \tag{1}$$

Subject to

$$\boldsymbol{x^{(i)}} = [x_{j,k}^{(i)}]_{|S_i| \times \max_{l=1}^{|S_i|}(|\mathcal{M}(s_l)|)} \tag{2}$$

$$\sum_{k=1}^{\max_{l=1}^{|S_i|}(|\mathcal{M}(s_l)|)} x_{j,k}^{(i)} = 1,$$

$$j = 1, 2, ..., |S_i|, \quad x_{j,k}^{(i)} \in \{0, 1\} \tag{3}$$

where Eq. (1) states that the goal of the CCSC problem is to maximize the sum of utility function $\mathcal{F}$ for each composite service $cs_i \in CS$. Equation (2) defines the service selection scheme, i.e. $\boldsymbol{x^{(i)}}$, for cs_i, which has been explained before. Equation (3) ensures that there is one and only one candidate can be selected for each atomic service in the cs_i.

According the definition, the CCSC problem aims to maximize the total QoS values of a set of composite services instead of a single composite service. The latter one is usually known as the classic QoS-aware service composition (QoS-SC) problem and has been well studied in previous work. Because [27] has proven that the QoS-SC is a NP-hard problem, our CCSC problem, consisting of a set of QoS-SC problems, is also considered to be NP-hard.

4 Evolutionary Multitasking Algorithm for CCSC Problem

Given an instance of CCSC problem consisting of n composite services, a conventional service composition system considers it as n independent optimization tasks (i.e. *CCSC tasks*), and focuses on solving only a single task at a time using a optimizer. In this paper, we propose a novel method, namely evolutionary multitasking algorithm for CCSC (EMA-CCSC), that considers two CCSC tasks at the same as suggested in [13]. The key motivation is to use evolutionary multitasking algorithm (EMA) for implicit knowledge transfer of useful traits across two optimization tasks, thereby enhancing the evolutionary search for problem-solving [38]. We first introduce some related terms, and then discuss the EMA-CCSC in detail.

Factorial Cost: For a given CCSC task $T_j (j = 1, 2)$ and a solution x^i in the evolving population of EMA, the factorial cost of x^i, denoted by Ψ_j^i, is defined as the QoS value of x^i with respect to T_j.

Factorial Rank: The factorial rank of x^i on task T_j, denoted by r_j^i, refers to the index of x^i in the list of populations sorted in ascending order with respect to Ψ_j^i.

Scalar Fitness: Given the list of factorial ranks $\{r_1^i, r_2^i\}$ of the solution x^i, its scalar fitness φ_i can be defined as $\varphi_i = 1/min\{r_1^i, r_2^i\}$.

Skill Factor: The skill factor τ_i of the solution x^i is one of the two CCSC tasks on which x^i achieves the best QoS value.

With these definitions above, the proposed EMA-CCSC shown in Algorithm 1 chooses two of the remainder CCSC tasks (randomly or with a certain pattern) at each run and works as follows.

Algorithm 1. The proposed EMA-CCSC

1: $\{\mathbf{x}^1, \cdots, \mathbf{x}^N\} \leftarrow InitPopulation(N)$;
2: $\{\varphi_i^i, \tau_i\} \leftarrow Evaluation(\mathbf{x}^i)$;
3: **while** the stopping criteria is not satisfied **do**
4: $\quad \{\mathbf{y}^1, \cdots, \mathbf{y}^N\} \leftarrow Reproduction(\{\mathbf{x}^1, \cdots, \mathbf{x}^N\})$;
5: $\quad \{\varphi_i^i, \tau_i\} \leftarrow SelectiveEvaluation(\mathbf{y}^i)$
6: $\quad P \leftarrow \{\mathbf{x}^1, \cdots, \mathbf{x}^N\} \cup \{\mathbf{y}^1, \cdots, \mathbf{y}^N\}$;
7: $\quad \{\mathbf{x}^1, \cdots, \mathbf{x}^N\} \leftarrow Selection(P)$;
8: **end while**
9: Output $\{\mathbf{x}^1, \cdots, \mathbf{x}^N\}$.

In the above pseudo-code of the EMA-CCSC algorithm, line 1 initializes a population of N individuals at random. Suppose that the two composite services cs_1 and cs_2 contain n_1 and n_2 atomic services respectively, then every single individual $\mathbf{x}^i = \{x_1^i, ..., x_n^i\}$ in the initial population is a n-dimensional real vector, where $n = \max\{n_1, n_2\}$. Each dimension of $\mathbf{x}^i$, denoted by $x_j^i (j = 1, 2, ..., n)$, is a real value between 0 and 1. If the j-th atomic service s_j can be selected from

a candidate collection $\mathcal{M}(s_i) = \{c_1, c_2...c_k\}$ with k available services, then the r-th candidate service will be chosen to execute, where r can be determined by $\lceil k \times \mathbf{x}_j^i \rceil$.

Line 2 of Algorithm 1 evaluates the individuals in the initial population by calculating their factorial costs, determining their factorial ranks, assigning their scalar finesses φ_i^i and skill factor τ_i in sequence. Figure 3 illustrates this evaluation process with two tasks T_1 and T_2 and four individuals p_1–p_4. The algorithm calculates the factorial rank and skill factors for these individuals first, and then sets the value of factorial cost as "infinity" to the task that doesn't have skill factor for each individual.

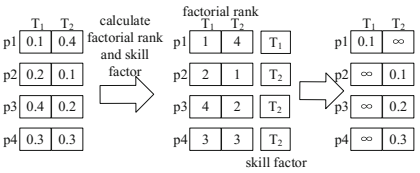

Fig. 3. Evaluate individuals by calculating their factorial costs

The reproduction step in line 4 of Algorithm 1 generates a offspring population of $\{\mathbf{y}^1, \cdots, \mathbf{y}^N\}$ according to the assortative mating rules provided in Algorithm 2. In which, *rand* is a random number between 0 and 1, *rmp* is a prescribed random mating probability. Here we employ the single-point crossover in line 4 and the random mutation with mutation rate of $1/n$ in lines 6 and 7 in Algorithm 2.

Algorithm 2. Assortative mating

1: Consider two parent candidates p_a and p_b randomly selected from $\{\mathbf{x}^1, \cdots, \mathbf{x}^N\}$.
2: $rand \leftarrow Rand(0, 1)$;
3: **if** $\tau_a == \tau_b$ or $rand < rmp$ **then**
4: $\quad \{c_a, c_b\} \leftarrow Crossover(p_a, p_b)$;
5: **else**
6: $\quad c_a \leftarrow Mutation(p_a)$;
7: $\quad c_b \leftarrow Mutation(p_b)$;
8: **end if**

In the selective evaluation step in line 5 of Algorithm 1, the offspring individuals are evaluated for only one selected task on which it is most likely to perform well. The selective evaluation step works as Algorithm 3. Figure 4 shows the reproduction and selective evaluation steps of T_1 and T_2 and their four individuals p_1–p_4, in which the algorithm performs random mutation on p_1 and p_4 and generates c_1 and c_4 respectively, and applies crossover operation on p_2 and p_3 and produces c_2 and c_3 separately.

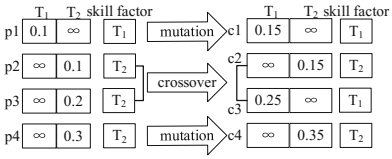

Fig. 4. Reproduction and selective evaluation

Algorithm 3. Selective evaluation

1: For each offspring $c \in \{\mathbf{y}^1, \cdots, \mathbf{y}^N\}$, it will either have two parents (p_a and p_b) or a single parent (p_a or p_b) - see Algorithm 2.
2: **if** c has two parents **then**
3: $rand \leftarrow Rand(0, 1)$;
4: **if** $rand < 0.5$ **then**
5: c is evaluated only for task τ_a (c imitates p_a);
6: **else**
7: c is evaluated only for task τ_b (c imitates p_b);
8: **end if**
9: **else**
10: c is evaluated only for that task which is its parent's skill factor (c imitates its single parent);
11: **end if**
12: Factorial costs of c with respect to all unevaluated tasks are artificially set to a very large number.

The selection step in line 7 of Algorithm 1 selects N individuals from the union set of the parent population of $\{\mathbf{x}^1, \cdots, \mathbf{x}^N\}$ and the offspring population of $\{\mathbf{y}^1, \cdots, \mathbf{y}^N\}$, giving rise to a new evolving population for the next generation. The selection step in Algorithm 1 follows an elitist strategy which ensures that the best individuals survive through the generations. Figure 5 illustrates the selection process with p_1–p_4 and their four offsprings c_1–c_4.

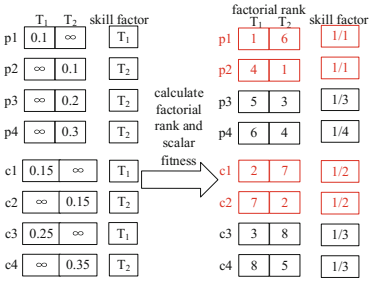

Fig. 5. Selection step

5 Experiments

5.1 Experiment Setup

All experiments are conducted on a Lenovo ThinkServer RD530 server equipped with 2 Xeon E5-2609 CPUs, 1 TB disk and 6 * 16 G RAMs, running on Windows Server 2008.

A sequence of 1188 CCSC tasks with various atomic service numbers from 10 to 100 are investigated [4]. These CCSC tasks are generated by using a synthetic workflow generator developed in [15]. The collection of candidate services are provided by an updated QWS data set [1,2] in which 2507 Web services with 233 categories are available.

5.2 Compared Algorithms and Control Parameters

Nine traditional CCSC solvers with different types are employed as the baseline algorithms. They are the genetic algorithms (GAs) including TGA [30], GAHS [31] and GASA [31], the particle swarm optimization (PSO) algorithms including ConstrictionPSO [5], FrankensteingPSO [10] and OLPSO [35], and the differential evolution (DE) algorithms including DE [25], DEGL [8] and JADE [36].

The parameter setting of the proposed EMA-CCSC algorithm are as follows. The population size is set to 30, which is the same in all the compared algorithms. The crossover probability and mutation probability are set to 0.3 and $1/n$ (n is the number of atomic services in the target CCSC) respectively. Except for the population size, all the other parameter settings are exactly the same as suggested by their authors. All the runs of the compared algorithms stop after 1000 iterations. When applying the EMA-CCSC algorithm, two CCSC tasks with the same number of atomic services are selected from the waiting queue, and submitted to the EMA-CCSC algorithm for scheduling.

5.3 Results and Comparison

Figures 6, 7 and 8 compared the performance of the baseline algorithms of GAs, PSOs and DEs respectively. In these figures, the symbol "++" indicates that EMA-CCSC improves the QoS values of both the two target CCSC tasks by optimizing them simultaneously, comparing with the performances of the compared algorithms which optimize them successively. In the pie charts, The symbol "+=" means one is improved while the other is unchanged. "==" means both the two instances are unchanged. "+−" is for one improved one degenerated. "−=" is for one degenerated one unchanged. "−−" means both the two are degenerated. The bar charts in these figures illustrate the average rate of improvement achieved by the proposed EMA-CCSC algorithm versus the compared algorithms on all the investigated CCSC tasks. All the experimental results in these figures are statistic values of five independent runs of the compared algorithms.

As shown in Fig. 6, when comparing with GAHS, GASA and TGA, EMA-CCSC achieves "++" on 89.90%, 80.14% and 32.83% of the investigated CCSC

tasks. On the other hand, EMA-CCSC achieves "−−" and thus degenerates the performances on both the two target CCSC tasks only on 0.17%, 0.84% and 14.65% of the investigated CCSC tasks. When looking at the bar charts, EMA-CCSC improves the QoS values of CCSC tasks on 1127, 1065 and 660 out of 1188 tasks respectively. As a result, EMA-CCSC achieves average rate of improvement of 36.6675%, 18.4813% and −1.7650% respectively versus GAHS, GASA and TGA. Based on the total improvement rate, we can come to the conclusion that EMA-CCSC achieves better performances than GAHS and GASA with only half of their computing costs.

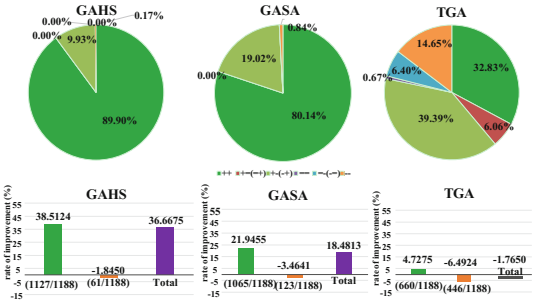

Fig. 6. Comparisons between EMA-CCSC and GAs

Similar conclusions can be drawn from Figs. 7 and 8 on the PSOs and DEs baseline algorithms. It can be seen that EMA-CCSC significantly outperforms the PSOs and achieves "++" on up to 92.76%, 90.74% and 88.55% of the investigated CCSC tasks. When comparing with the DEs baseline algorithms, EMA-CCSC outperforms two out of the three compared DEs algorithms in term of the total improvement rate.

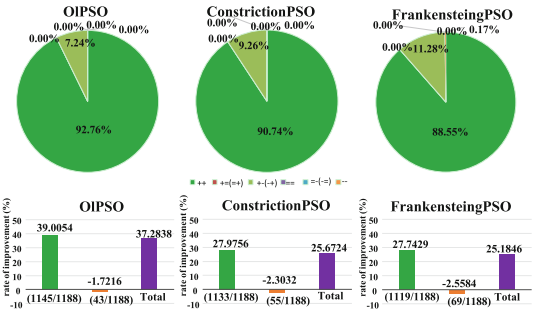

Fig. 7. Comparisons between EMA-CCSC and PSOs

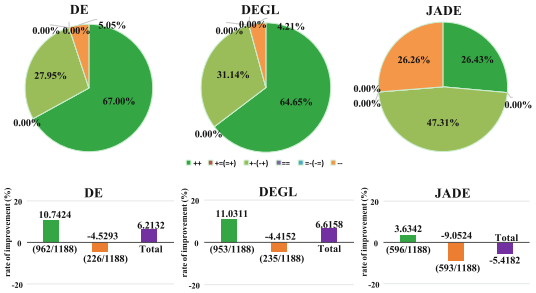

Fig. 8. Comparisons between EMA-CCSC and DEs

To conclude, our EMA-CCSC outperforms 7 out of 9 compared algorithms in term of the total improvement rate. It appears to be competitive against the baseline algorithms, even though it spends only half of their computing costs.

6 Conclusion

In this paper, we present an evolutionary multitasking algorithm based approach to efficiently solve CCSC problem. Unlike existing solvers, our EMA-CCSC algorithm can optimize two or more CCSC tasks concurrently. As a result, it has a greater throughput and is able to deal with more composite service requests given a fixed period of time, which is critical to improve the experiences of the cloud computing services. Our extensive experiments indicate that the proposed EMA-CCSC is competitive in both solution quality and time efficiency.

References

1. Al-Masri, E., Mahmoud, Q.H.: Discovering the best web service. In: Proceedings of the 16th International Conference on World Wide Web, pp. 1257–1258. ACM (2007)
2. Al Masri, E., Mahmoud, Q.H.: Investigating web services on the world wide web. In: Proceedings of the 17th International Conference on World Wide Web, pp. 795–804. ACM (2008)
3. Ba, C.: An exact cover-based approach for service composition. In: 2016 IEEE International Conference on Web Services (ICWS), pp. 631–636. IEEE (2016)
4. Bao, L., Zhao, F., Shen, M., Qi, Y., Chen, P.: An orthogonal genetic algorithm for QoS-aware service composition. Comput. J. **59**(12), 1857–1871 (2016)
5. Clerc, M., Kennedy, J.: The particle swarm: explosion, stability and convergence in a multi-dimensional complex space. IEEE Trans. Evol. Comput. **6**(1), 58–73 (2002)
6. Cui, L., Li, J., Zheng, Y.: A dynamic web service composition method based on viterbi algorithm. In: 2012 IEEE 19th International Conference on Web Services (ICWS), pp. 267–271. IEEE (2012)

7. Da, B., Gupta, A., Ong, Y.S., Feng, L.: Evolutionary multitasking across single and multi-objective formulations for improved problem solving. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 1695–1701. IEEE (2016)
8. Das, S., Abraham, A., Chakraborty, U., Konar, A.: Differential evolution using a neighborhood-based mutation operator. IEEE Trans. Evol. Comput. **13**(3), 526–553 (2009)
9. Deng, S., Huang, L., Wu, H., Wu, Z.: Constraints-driven service composition in mobile cloud computing. In: 2016 IEEE International Conference on Web Services (ICWS), pp. 228–235. IEEE (2016)
10. Deoca, M., Stutzle, T., Birattari, M., Dorigo, M.: Frankenstein's PSO: a composite particle swarm optimization algorithm. IEEE Trans. Evol. Comput. **13**(5), 1120–1132 (2009)
11. Guidara, I., Guermouche, N., Chaari, T., Tazi, S., Jmaiel, M.: Heuristic based time-aware service selection approach. In: 2015 IEEE International Conference on Web Services (ICWS), pp. 65–72. IEEE (2015)
12. Gupta, A., Mańdziuk, J., Ong, Y.S.: Evolutionary multitasking in bi-level optimization. Complex Intell. Syst. **1**(1–4), 83–95 (2015)
13. Gupta, A., Ong, Y.S., Feng, L.: Multifactorial evolution: toward evolutionary multitasking. IEEE Trans. Evol. Comput. **20**(3), 343–357 (2016)
14. Jula, A., Sundararajan, E., Othman, Z.: Cloud computing service composition: a systematic literature review. Expert Syst. Appl. **41**(8), 3809–3824 (2014)
15. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Future Gener. Comput. Syst. **29**(3), 682–692 (2013)
16. Kellerer, H., Pferschy, U., Pisinger, D.: Introduction to NP-completeness of knapsack problems. In: Knapsack Problems, pp. 483–493. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24777-7_16
17. Kofler, K., ul Haq, I., Schikuta, E.: A parallel branch and bound algorithm for workflow QoS optimization. In: 2009 International Conference on Parallel Processing, pp. 478–485. IEEE (2009)
18. Mabrouk, N.B., Georgantas, N., Issarny, V.: Set-based bi-level optimisation for QoS-aware service composition in ubiquitous environments. In: 2015 IEEE International Conference on Web Services (ICWS), pp. 25–32. IEEE (2015)
19. Ong, Y.S.: Towards evolutionary multitasking: a new paradigm. In: Proceedings of the Sixth International Symposium on Information and Communication Technology, p. 2. ACM (2015)
20. Ong, Y.S., Gupta, A.: Evolutionary multitasking: a computer science view of cognitive multitasking. Cogn. Comput. **8**(2), 125–142 (2016)
21. Rodriguez-Mier, P., Mucientes, M., Lama, M.: Automatic web service composition with a heuristic-based search algorithm. In: 2011 IEEE International Conference on Web Services (ICWS), pp. 81–88. IEEE (2011)
22. Sagarna, R., Ong, Y.S.: Concurrently searching branches in software tests generation through multitask evolution. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8. IEEE (2016)
23. Shehu, U., Epiphaniou, G., Safdar, G.A.: A survey of QoS-aware web service composition techniques. Int. J. Comput. Appl. **89**(12), 10–17 (2014)
24. Shi, Y., Chen, X.: A survey on QoS-aware web service composition. In: Third International Conference on Multimedia Information Networking and Security, Shanghai, China, 4–6 November 2011, pp. 283–287. IEEE Computer Society, Washington, DC (2011)

25. Storn, R., Price, K.: Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**(4), 341–359 (1997)
26. Syu, Y., FanJiang, Y.Y., Kuo, J.Y., Ma, S.P.: Towards a genetic algorithm approach to automating workflow composition for web services with transactional and QoS-awareness. In: 2011 IEEE World Congress on Services, pp. 295–302. IEEE (2011)
27. Tao, F., Zhao, D., Hu, Y., Zhou, Z.: Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system. IEEE Trans. Industr. Inf. **4**(4), 315–327 (2008)
28. Torkashvan, M., Haghighi, H.: A greedy approach for service composition. In: 2012 Sixth International Symposium on Telecommunications (IST), pp. 929–935. IEEE (2012)
29. Wang, D., Yang, Y., Mi, Z.: Qos-based and network-aware web service composition across cloud datacenters. TIIS **9**(3), 971–989 (2015)
30. Whitley, D.: A genetic algorithm tutorial. Stat. Comput. **4**(2), 65–85 (1994)
31. Yilmaz, A.E., Karagoz, P.: Improved genetic algorithm based approach for QoS aware web service composition. In: IEEE International Conference on Web Services, Alaska, USA, 6–7 November 2014, pp. 463–470. IEEE Computer Society, Washington, DC (2014)
32. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end QoS constraints. ACM Trans. Web (TWEB) **1**(1), 6 (2007)
33. Yuan, Y., Ong, Y.S., Gupta, A., Tan, P.S., Xu, H.: Evolutionary multitasking in permutation-based combinatorial optimization problems: realization with TSP, QAP, LOP, and JSP. In: 2016 IEEE Region 10 Conference (TENCON), pp. 3157–3164. IEEE (2016)
34. Zeng, C., Guo, X., Ou, W., Han, D.: Cloud computing service composition and search based on semantic. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) CloudCom 2009. LNCS, vol. 5931, pp. 290–300. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10665-1_26
35. Zhan, Z.H., Zhang, J., Li, Y., Shi, Y.H.: Orthogonal learning particle swarm optimization. IEEE Trans. Evol. Comput. **15**(6), 832–847 (2011)
36. Zhang, J., Sanderson, A.C.: JADE: adaptive differential evolution with optional external archive. IEEE Trans. Evol. Comput. **13**(5), 945–958 (2009)
37. Zhang, T., Ma, J., Sun, C., Li, Q., Xi, N.: Service composition in multi-domain environment under time constraint. In: 2013 IEEE 20th International Conference on Web Services (ICWS), pp. 227–234. IEEE (2013)
38. Zhou, L., Feng, L., Zhong, J., Ong, Y.S., Zhu, Z., Sha, E.: Evolutionary multitasking in combinatorial search spaces: a case study in capacitated vehicle routing problem. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8. IEEE (2016)