Expert Systems with Applications 42 (2015) 135–145

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Imperialist competitive algorithm with PROCLUS classifier for service time optimization in cloud computing service composition

Amin Jula [a],*, Zalinda Othman [a], Elankovan Sundararajan [b]

[a] Data Mining and Optimization Research Group, Centre for Artificial Intelligence, UKM Bangi, 43600 Selangor, Malaysia
[b] Centre of Software Technology and Management, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, UKM Bangi, 43600 Selangor, Malaysia

ARTICLE INFO

Article history:
Available online 4 August 2014

Keywords:
Cloud computing
Service composition
Service selection
Service time
Quality of service
QoS
Imperialist competition algorithm
Clustering
Proclus

ABSTRACT

Aiming to provide satisfying and value-added cloud composite services, suppliers put great effort into providing a large number of service providers. This goal, achieved by providing the "best" solutions, will not be guaranteed unless an efficient *composite service composer* is employed to choose an optimal set of required unique services (with respect to user-defined values for quality of service parameters) from the large number of provided services in the pool. Facing a wide service pool, user constraints, and a large number of required unique services in each request, the composer must solve an NP-hard problem. In this paper, *CSSICA* is proposed to make advances toward the lowest possible service time of composite service; in this approach, the PROCLUS classifier is used to divide cloud service providers into three categories based on total service time and assign a probability to each provider. An improved imperialist competitive algorithm is then employed to select more suitable service providers for the required unique services. Using a large real dataset, experimental and statistical studies are conducted to demonstrate that the use of clustering improved the results compared to other investigated approaches; thus, *CSSICA* should be considered by the *composer* as an efficient and scalable approach.

1. Introduction

In an era where computation complexity is growing dramatically and achieving desired results depends on the processing of big data, the computation world has no choice but to recognize the use of cloud computing (Armbrust et al., 2010; Hayes, 2008). Choosing each of the deployment models of cloud computing (public, community, private, and hybrid clouds (Dillon, Chen, & Chang, 2010; Peter Mell, 2011)) would provide required service models (Ellinger, 2013) with different security policies (Takabi, Joshi, & Gail-Joon, 2010; Wei et al., 2014; Zissis & Lekkas, 2012). According to a widely accepted classification, each service can belong to one of the three categories: software as a service (SaaS), platform as a service (PaaS), or infrastructure as a service (IaaS), which can provide more effective functionalities in cooperation and combination of other services.

Nevertheless, the increasing tendency of applicants to receive services from the cloud has led to an unprecedented increase in the number of providers who want to present their services in a cloud service pool. Hence, we are faced with a large number of unique services provided with similar functionality and different quality of service (QoS) (Jula, Sundararajan, & Othman, 2014).

On the other hand, due to the availability of complicated and varied services, a distinct simple service is unable to meet the prevailing functional prerequisites for several real-world cases. A set of simple atomic services that are able to work together is necessary to perform a complicated service. We also encounter hanging customer requirements from simple services into complicated services, along with a set of constraints, priorities, and QoS requirements (e.g., service time). Therefore, cloud suppliers must provide a package, referred to here as the *composite service composer (CSC)*, which is a set of components that searches for the best composition of pre-provided unique services in the service pool based on customer requirements and constraints. Because of the immense variety of unique services and large number of service providers, as well as the importance of customer-defined requirements and constraints, *CSC* is faced with an NP-hard problem referred to as cloud computing service composition *(CCSC)* (Fei, Yuanjun, Lida, & Lin, 2013; Li, Cheng, Ou, & Zhang, 2010; Wada, Suzuki, Yamano, & Oba, 2012) when seeking the optimal response to any request for a composite service.

Many studies have been conducted and many different heuristic and non-heuristic algorithms (Barney, 2012; Gutierrez-Garcia & Sim, 2010; Kofler, Haq, & Schikuta, 2010; Kofler, ul Haq, & Schikuta,

* Corresponding author.
E-mail addresses: amin.jula@gmail.com, aminjula@ftsm.ukm.my (A. Jula), zalinda@ftsm.ukm.my (Z. Othman), elan@ftsm.ukm.my (E. Sundararajan).

2009; Zhou & Mao, 2012) have been proposed to promote the quality of functionality of *CSC* and achieve an optimal solution for *CCSC*. The application of particle swarm optimization (Kennedy & Eberhart, 1995), genetic algorithms (Holland, 1992), game theory (Scutari, Palomar, & Barbarossa, 2008), and memetic algorithms (Jula & Naseri, 2011) has led to significant improvements in the optimization of proper service selection for service composition (Jula, Othman, & Sundararajan, 2013; Ludwig, 2012; Qi & Bouguettaya, 2013; Wang, Sun, Zou, & Yang, 2013; Xia, Wan, Dai, Luo, & Sun, 2012; Xiao & Zhang, 2012; Yang, Mi, & Sun, 2012; Ye, Zhou, & Bouguettaya, 2011; Zhu, Li, Luo, & Zheng, 2012); however, none of these approaches use the *clustering* of the service providers to achieve improved performance (Jula et al., 2014).

Despite all the research conducted in the domain, one of the main but almost neglected issues in facing very large search space problems is the large number of available options for being selected, and the limited number of solutions in the initialization phase of the algorithm. In other words, a very small percentage of potential solutions can be chosen as the first generation of solutions. Because selection of members of the initial generation of solutions is usually randomized, it is obvious that chance of selecting appropriate solutions is extremely low. Lack of providing proper initial solutions will lead to improper evolution process in execution of algorithm. Hence, it can be concluded that applying non-blind selection techniques can significantly enhance the performance of the algorithms. To reach this aim, two different approaches can be followed, including proposing intelligent initialization methods and operators, and search space clustering techniques. What previously has been employed to address this problem is restricted to the use of skyline operator to limit the number of service providers (Qi & Bouguettaya, 2013; Wang et al., 2013). What is the main challenge of using Skyline operator in real-world *CCSC* is high time-complexity that increases exponentially with the increase in the number of QoS parameters.

In this paper, PROCLUS (Aggarwal, Wolf, Yu, Procopiuc, & Park, 1999; Kurniawan et al., 1999) (as an appropriate clustering algorithm for high-dimensional data) is used to categorize service providers into three classes based on the time of their provided services. Afterwards, the probability of selection of single services from each category is calculated based on the average service time of all assigned service providers in each category. Using selection probability of the categories in initialization of first generation of solutions has altered the blind production of the first set of solutions and thus reduced the service time of the final solution.

But then imperialist competitive algorithm (ICA) as a very young evolutionary algorithm (Atashpaz-Gargari & Lucas, 2007) presents three attractive features to be used in solving *CCSC*. One of the most important concerns in solving optimization problems with very large search spaces is falling into trap of local optima. ICA searching process is designed in such a way that the probability of falling into the trap effectively reduced by frequent relocation of solutions in the search space using different well-designed operators. Novelty of utilizing sociopolitical behavior of countries in designing the algorithm, and extensive areas of innovation because of being newer than most of proposed evolutionary algorithms, are other two advantages of applying ICA.

In this study, for addressing Service Time Optimization in Cloud Computing Service Composition (STOCCSC), Classified Search Space Imperialist Competitive Algorithm (*CSSICA*) is proposed. In *CSSICA*, along with applying classified search space and using achieved probabilities in selection procedure, improvements in the structure and functioning methods of the ICA are also adopted to improve the algorithm's performance.

The remainder of this paper is organized as follows. After the introduction, descriptions of Service Time optimization in Cloud Computing Service Composition (*CCSC*) and Imperialist Competitive Algorithm (ICA) are presented in Section 2. A complete description of the proposed algorithm, *CSSICA*, is provided in Section 3. A detailed discussion of the experimental results obtained using *CSSICA* and two other algorithms, ICA and Niching PSO, including numerical and statistical investigations, is provided in Section 4. Finally, our conclusions and potential topics for future research are presented in Section 5.

2. Problem and algorithm description

2.1. Service time optimization in cloud computing service composition (STOCCSC)

An increase in the number of available services will lead to an increase in the number of similar operating services on various servers. Because these similar services are to be found in various places and have definite QoS parametric values, the *CSC* should employ suitable methods to select a unique service among the various similar services that are available on distinct servers. Applying the appropriate method will enable the best quality of service to be attained based on the requirements and priorities of the end user. In view of fundamental changes in cloud environments, accessible services, and end-user requirements, the *CSC* must be powerfully designed with automatic functional capabilities.

Thus, one of the most significant problems in service composition is the selection of suitable simple services to be merged together to produce an ideal combination that is able to meet the functional and QoS requirements of the end user.

This paper assumes that every *composite service (CS)* within a cloud is comprised of *n unique services (USs)*, all of which have *service time (ST)* as a QoS parameter. A combination of exclusive services must correspondingly act in an ordinal *workflow (wf)* to terminate a *CS*. However, if wf_k is the workflow of CS_k, then $ST(wf_k)$ can be defined to denote the *ST* value of the workflow *k*, whereupon the *ST* vector of the workflow can be expressed as (1).

$$ST(wf_k) = (ST_1(wf_k),\quad ST_2(wf_k), \ldots, ST_n(wf_k)) \tag{1}$$

The competency value (CV) of wf_k is the total service time of wf_k and is obtained using (2). The optimal solution for *STOCCSC* is the solution with the lowest *ST* and thus with the lowest *CV*.

$$CV(wf_k) = \sum_{i=1}^{n} ST_i(wf) \tag{2}$$

2.2. Imperialist competitive algorithm

In the field of evolutionary computation, the novel ICA is founded on human social and political advancements (Atashpaz-Gargari & Lucas, 2007; Bahrami, Faez, & Abdechiri, 2010; Zarandi, Zarinbal, Ghanbari, & Turksen, 2013), unlike other evolutionary algorithms, which are based on the natural behaviors of animals or physical events. ICA starts with an initial randomly generated population, in which the individuals are known as countries. Some of the best countries are considered imperialists, whereas the other countries represent the imperialist colonies. To solve an optimization problem with *n* dimensions, a country is formed as an $1 \times n$ array as follows:

$$Country = [p_1, p_2, \ldots, p_n] \tag{3}$$

The power of a country *i* is calculated using the objective function *f*, which is a function of the variables $(p_1, p_2, \ldots, p_n)$, yielding the following equation:

$$Power(Country_i) = f(Country_i) = f(p_{i1}, p_{i2}, \ldots, p_{in}) \tag{4}$$

The ICA begins with m countries, and the n_{imp} most powerful of these countries are chosen as the imperialists. The other countries are called imperialist colonies, and every non-imperialist country will belong to an empire. A simple procedure is used to disperse the non-imperialist countries among the imperialists. An imperialist country is randomly selected for each non-imperialist country, and the latter is then allotted to that empire.

During the execution of the algorithm, each imperialist country will attract its colonies according to the total power of the empire and that of the colonies. The total power of each imperialist is ascertained by the power of the empire added to a coefficient that is multiplied to obtain the average power of the imperialist colonies. The total power of each imperialist country is calculated as

$$TP_n = c_n + (\alpha \times Average\{power(colonies\ of\ empire_n)\}) \tag{5}$$

where TP_n is the total power of the nth empire, c_n is power of imperialist n and α is a positive random number less than one.

During the movement of a country, a colony moves a length of x units toward its imperialist country. As illustrated in Fig. 1, the direction of the movement can be regarded as a vector from the colony to its related imperialist, where d is the distance between the imperialist country and colony and x is a random value that can be determined using a uniform distribution as follows:

$$x \approx Uniform(0, \beta \times d) \tag{6}$$

where β is greater than one and is close to 2.

There is a very significant operator in the ICA known as the *imperialistic competition*; this operator is a function decreases the power and number of colonies of the weakest empire until it is destroyed; it can be applied in various forms depending on the design requirements of the algorithm.

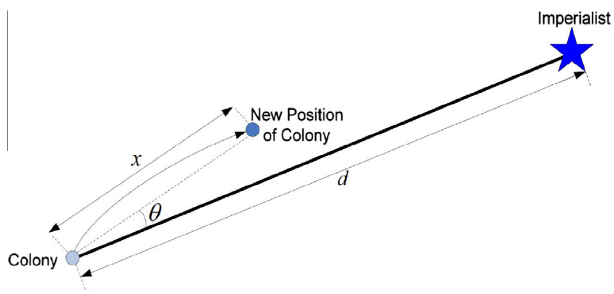


Fig. 1. Direction of colony movement to related imperialist (Atashpaz-Gargari and Lucas, 2007).

3. Proposed algorithm

To search large-scale spaces more efficiently, evolutionary algorithms can reduce the scale of the space by targeting more appropriate candidates in the selection phase. In the *STOCCSC*, eliminating blind service selection in the first step of the evolutionary algorithm is expected to lead to more effective results because there are very large-scale service pools in which thousands of unique services are presented by hundreds of service providers. Such improvements should not significantly increase in execution time of the algorithm and thus reduce its performance.

Clustering can be applied to the categorization of service pool service providers as a convenient and effective solution to this problem, with the aim of producing better solutions in the first phase of the algorithm. PROCLUS, which is the most appropriate clustering technique for large-scale search spaces in terms of calculation time (Widia Sembiring, Mohamad Zain, & Embong, 2010), is selected for use in *CSSICA* to solve *STOCCSC*.

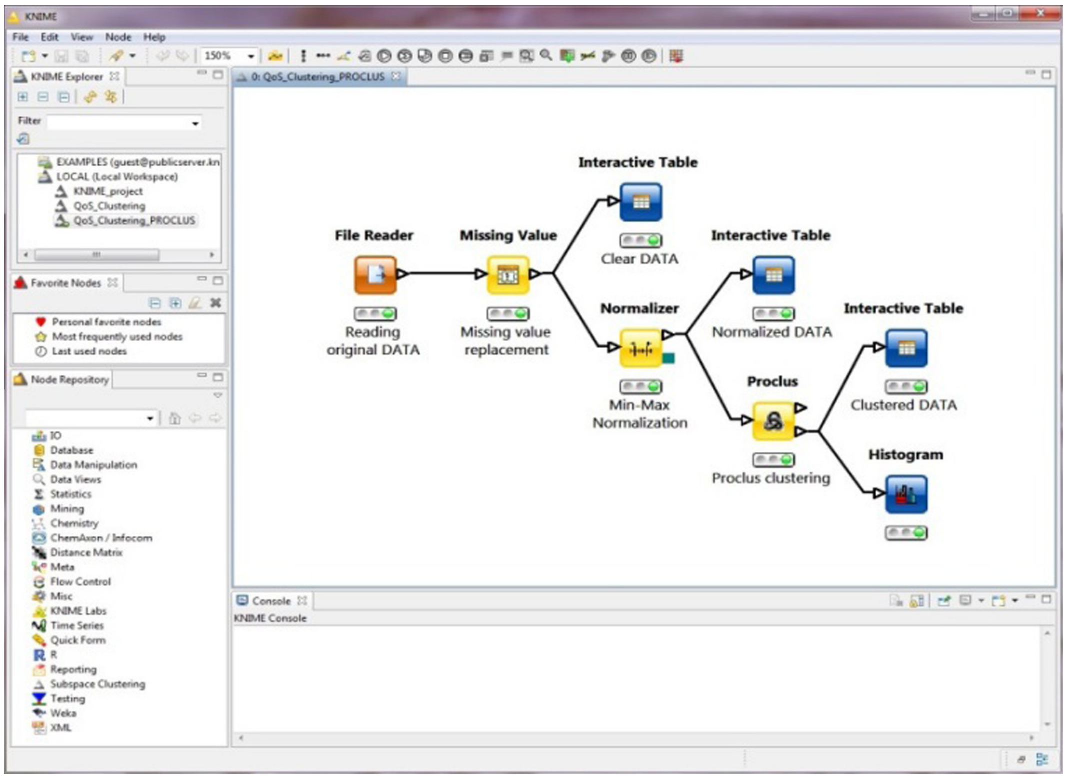


Fig. 2. KNIME 2.8.2 environment after PROCLUS is executed.

3.1. Missing value replacement and normalization

Missing value happens when there is no data to be stored for an attribute in an observation. Facing missing values is a prevalent incidence in using real-world datasets and can dramatically affect on results and achievements of processing the data. Validity of research may be undermined due to missing values (Allison, 2002; Di Nuovo, 2011). Hence, it is essential to prepare accurate normal data that do not contain missing values before starting the clustering procedure to ensure the accuracy and reliability of the results. *Mean Substitution* is selected to replace missing values because there are many values in the columns of the datasets and to avoid the destructive impacts of missing value replacement. Use of *Min–Max Normalization* could also help to preserve data relationships (Gan, 2007) and allow QoS parameters to be aggregated in the calculations (Jula et al., 2013; Wang et al., 2013).

3.2. Applying PROCLUS

To ensure data clearance, *CSSICA* applies PROCLUS to the data using KNIME 2.8.2 (Berthold et al., 2008) before starting the search process to divide the servers into three distinct categories. The first category is called *High-Recommended* and includes those servers that are most likely to achieve the minimum service time. *Recommended* is the second category and includes servers that are less likely to provide the minimum service time. The third category, called *Low-Recommended*, includes servers that are least likely to obtain the minimum service time. Thus, a server from the third category should only be considered with a low probability. A schematic of the designed KNIME workflow for missing value replacement, the normalizing procedure, and PROCLUS clustering is presented in Fig. 2, and the results are presented in Fig. 3.

Obtaining the selection probability of a category among the three categories for taking a service provider for each requires service is of utmost importance. To obtain this probability the probabilities should be calculated based on the direct proportion of the number of servers in each category and the inverse proportion of the average of all service times of all servers associated with each category. Table 1 presents the PROCLUS results and the calculated probabilities.

Table 1
PROCLUS results and calculated probabilities.

	Number of allocated servers	Average service time	Calculated probability
High-Recommended	176	3,971.6145	0.4455
Recommended	101	4,484.1619	0.3946
Low-Recommended	62	11,066.1211	0.1599

When calculating the probabilities, it is important to consider the fact that the highest probability value should be assigned to the category with the lowest average value, and vice versa. Thus, it is necessary to include the inverse of the average service times. Then, the demanded probabilities will be calculated by a simple first-order equation and three multiplication operations, as shown in (7), where *A*, *B*, and *C* are the average service times of the three categories; *A'*, *B'*, and *C'* are inverses of *A*, *B*, and *C*, respectively; and P(high-recom), P(recom), and P(low-recom) are the probabilities of the High-Recommended, Recommended, and Low-Recommended categories, respectively.

$$If\ A = 3,971.6145 \Rightarrow A' = \frac{1}{3,971.6145}, \tag{7}$$

$$if\ B = 4,484.1619 \Rightarrow B' = \frac{1}{4,484.1619},$$

$$If\ C = 11,066.1211 \Rightarrow C' = \frac{1}{11,066.1211},$$

$$(A' + B' + C')x = 1 \Rightarrow x = \frac{1}{5.6516e-4} \Rightarrow x \approx 1,769.4111,$$

$$P(high - recom) = A'x \approx 0.4455,$$

$$P(recom) = B'x \approx 0.3946,$$

$$P(low - recom) = C'x \approx 0.1599.$$

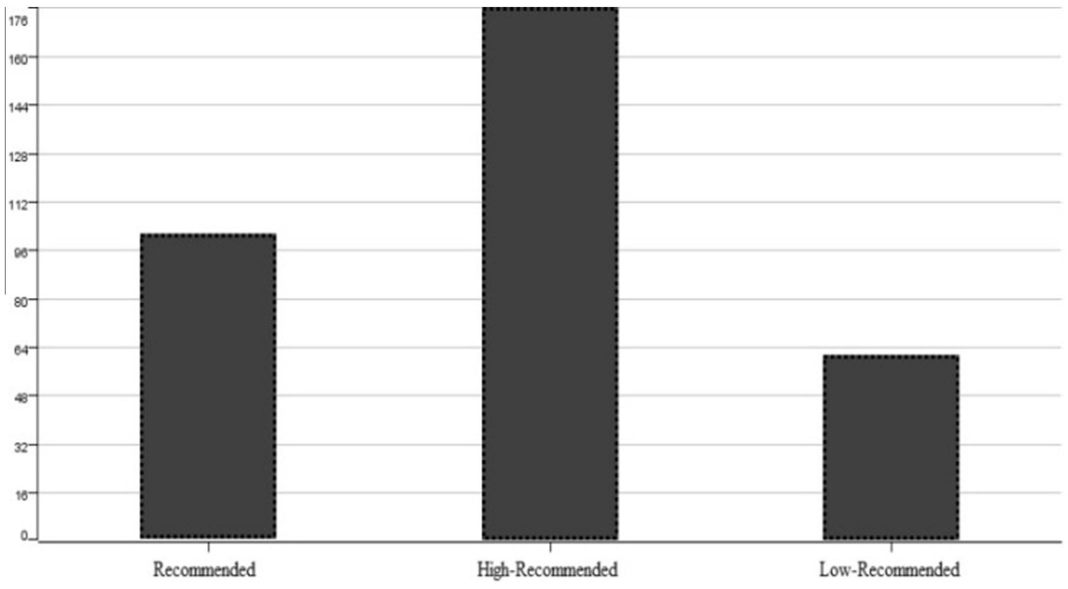


Fig. 3. KNIME results for PROCLUS clustering.

3.3. Applying the ICA

In the first phase, a set of countries called *World* (see Fig. 4) is generated randomly but with respect to the obtained probabilities and includes *CountryNo* countries in which each country represents a solution for the *STOCCSC*. A random generated number between zero and one is employed to determine which of the three categories should be used to select one of its servers. The pseudocode of this process is shown in Fig. 5, illustrating how a server is selected randomly from a probability-based selected category.

The structure of each of the countries, as shown in Fig. 3, consists of *d* distinct attributes that are used to save the assigned servers for *d* required unique services and a field for storing the total service time for the current generated solution. In the second phase, the total service time for the required composite service must be calculated based on the selected servers in each country to obtain a proper value for evaluating and comparing the initialized countries. The total service time of each country that is actually the power of the country is calculated using (8), where *Service_Time (i)* is the service time of required service *i* presented by the current country.

$$Total\_Service\_Time = \sum_{i=1}^{d} Service\_Time(i) \tag{8}$$

The third phase involves sorting the countries of the world based on their service time in ascending order. This sorting procedure helps the algorithm to reach the best-generated solutions at the top of the list of countries. The optimal solutions are then located at the top of the list, and the first *ImperialistNo* of countries would be selected as the imperialists and the remaining *CountryNo-ImperialistNo* countries would be known as their colonies. Colonies are assigned to the imperialists randomly so that an equal number of colonies are allocated to each of the imperialists. Hence, the number of colonial countries of each imperialist can be calculated by (9)

$$ColonialNo\ (IMP_i) = \frac{CountryNo - ImperialistNo}{ImperialistNo} \tag{9}$$

where IMP_i represents imperialist *i*.

The fourth phase, which involves moving colonies toward their related imperialist, plays an important role in the process of ICA. The movements in the *CSSICA* are designed as follows. The first step is to calculate the distance between the imperialist and colony in all dimensions separately using (10).

$$Dist^d(IMP_k, Country_i) = IMP_k.Service[d] - Country[i].Service[d] \tag{10}$$

where $Dist^d(IMP_k, Country_i)$ is the distance between country *i* and its imperialist *k* in dimension *d*. A random integer number between zero and the calculated distance is generated to determine the displacement of the country in each specific dimension. The index of the new assigned service in the specific dimension will be smaller

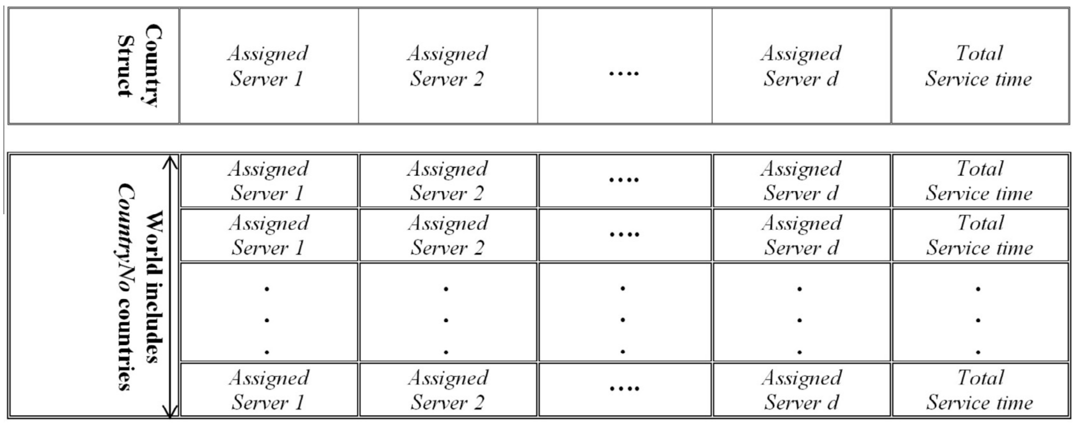
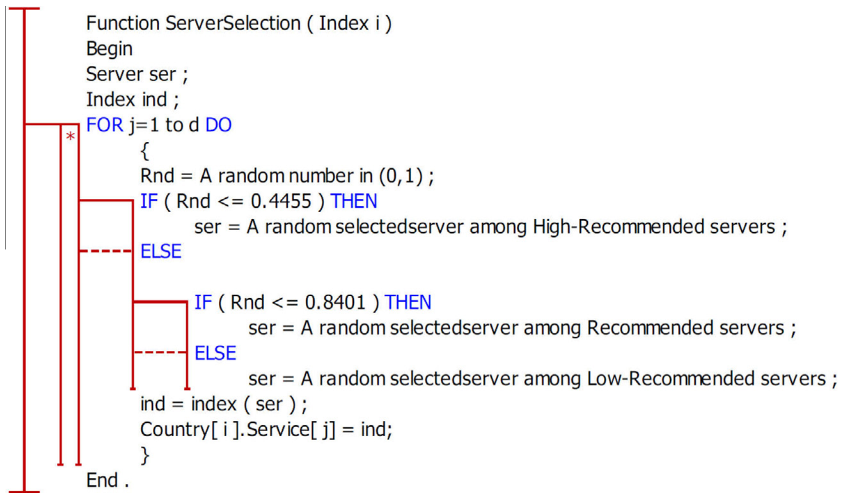


Fig. 4. Country structure and the World.



```
Function ServerSelection ( Index i )
Begin
Server ser ;
Index ind ;
FOR j=1 to d DO
    {
    Rnd = A random number in (0,1) ;
    IF ( Rnd <= 0.4455 ) THEN
        ser = A random selectedserver among High-Recommended servers ;
    ELSE
        IF ( Rnd <= 0.8401 ) THEN
            ser = A random selectedserver among Recommended servers ;
        ELSE
            ser = A random selectedserver among Low-Recommended servers ;
    ind = index ( ser ) ;
    Country[ i ].Service[ j] = ind;
    }
End .
```

Fig. 5. Pseudocode of the ServerSelection procedure.

or greater than the previous index if the distance is negative or positive, respectively, allowing the colony to move toward its related imperialist. The countries should be displaced in all dimensions.

In the fifth phase, the new total service time of the services must be calculated due to the change in assigned servers to some required unique services. Then, if the calculated total service time for the colony is smaller than the total service time of its related imperialist, their positions should be exchanged.

In the sixth phase, the imperialists compete with one another; namely, they attempt to overcome the other imperialists and oblige the weakest imperialist to surrender its weakest colony to the conqueror imperialist. To avoid hasty decisions, *CSSICA* provides all imperialists with 15 iterations to have sufficient opportunity to empower their empires before entering into the competition. In other words, imperialist competition takes place only once after each 15 iterations. If the weakest imperialist has no colony, the imperialist surrenders itself to the conqueror and its empire disappears. Thus, after a sufficient number of iterations, only one imperialist will remain; this situation can be considered a condition for the termination of the algorithm. After each imperialist competition, the *CSSICA* sorts current imperialists in ascending order instead of sorting all countries. Due to the lower number of imperialists comparing to the total number of countries, this sorting technique plays an important role in significant decrease in executing time of the algorithm. In this phase, the algorithm also saves the current best solution, and then returns to the fourth phase again to execute the next iteration.

At the end of sixth phase, the termination criteria must be investigated before continuing the next iteration. The algorithm can be terminated after a specified number of iterations, after reaching a predefined total service time, or after the disappearance of the penultimate imperialist.

4. Experimental results

4.1. Experiment setup

The *CSSICA* consists of two independent components. The first component is data clearance and clustering, which is executed via KNIME 2.8.2. The second component is the ICA; the applied changes in the ICA have been implemented in Visual C#.Net 2012 to solve different service time optimization problems of different sizes for *CCSC*. Due to the importance of using a reliable dataset to evaluate the proposed algorithm, WSDream-QoSDataset2 (Zibin, Yilei, & Lyu, 2010) was used because it is a large dataset in which real-world service times were collected from 339 servers for 5,825 web services. The improved ICA and Niching PSO, which are described in Section 3 and (Liao, Liu, Wang, & Zhu, 2012; Liao, Liu, Zhu, Xu, & Wang, 2011), respectively, are also implemented and executed to solve similar problems to evaluate and compare the performance of *CSSICA*.

The investigated problems were randomly generated once and solved by the three aforementioned algorithms. To ensure the accuracy and reliability of the obtained results, each algorithm was independently executed 40 times for every problem and the average of the results was used in the evaluation. In addition to the comparison of the results obtained from implementing the proposed algorithms and their related charts, well-known statistical

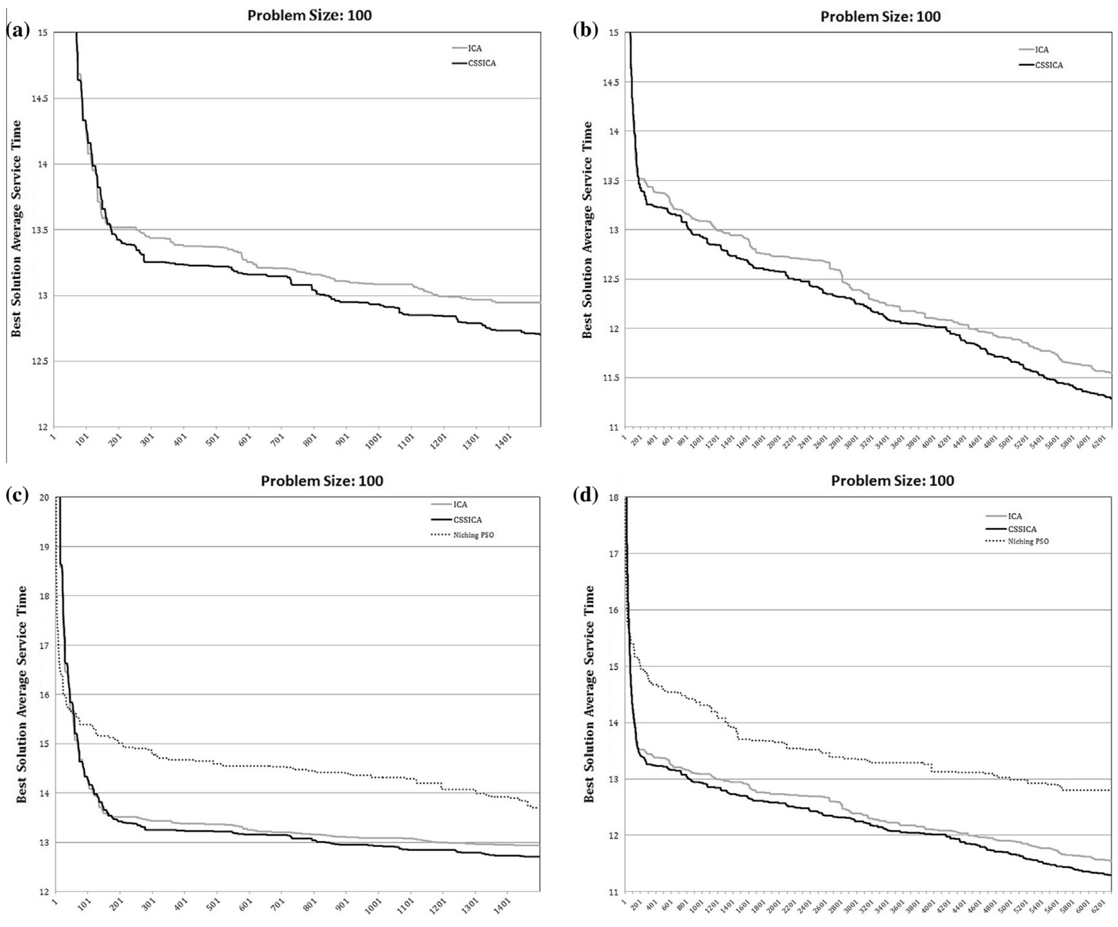


Fig. 6. Comparison of the total service time obtained for problem A, (a) CSSICA vs. ICA in 1,500 iterations, (b) CSSICA vs. ICA in 6,300 iterations, (c) CSSICA vs. ICA and Niching PSO for 1,500 iterations, (d) CSSICA vs. ICA and Niching PSO for 6,300 iterations.

tests are presented to provide further analysis and comparison of the algorithms. Due to the random nature of evolutionary algorithms, the use of such statistical assessments increases the reliability of the obtained results.

There are four randomly generated service time optimization problems for *CCSC*: Problems *A*, *B*, *C*, and *D*, in which the number of required unique services to be combined are 100, 200, 300, and 400, respectively. Five-hundred countries were considered for the ICA and *CSSICA*, and 10 imperialists were used as an initial setting. Thus, there were 50 countries that included an imperialist at the beginning of the run. The termination condition was the elimination of all but one imperialist. The algorithm required approximately 6,300 iterations to reach this condition. Thus, the number of the iterations was fixed at 6,300 for all 40 executions of each algorithm. To provide the same conditions for the execution of the algorithms, the number of particles and execution iterations considered for Niching PSO were also 500 and 6,300, respectively.

The three algorithms were executed on a PC with an Intel Core i7–3.40 GHz processor and 8 GB of RAM under identical conditions. The averages of the results obtained from each algorithm, as described before, were plotted to analyze and compare the results.

The first generated service time optimization problem of *CCSC* is a problem with 100 required unique services. Thus, to generate the problem, 100 distinct services were selected randomly among the 5,825 services in the dataset. The second, third, and fourth problems were generated in a similar manner and included 200, 300, and 400 required unique services, respectively.

4.2. Discussion

4.2.1. Comparison of the total service time

To more accurately analyze the results of the algorithms and obtain a better understanding of their executing process, two obtained solutions for each of the above problems were considered for comparison, the first one at iteration 1,500, called the *first proper solution (fps)*, and the other after termination of the algorithms at iteration 6,300, called the *final solution (fs)*. Using *fps*, it is possible to compare the speed and performance of the algorithms in exploration of the search space in a finite time-slot. The *fs* allows the capability of the algorithms in searching the search space over a longer period of time to be analyzed.

As mentioned in Section 4.1, four different-sized service time optimization problems were investigated using the ICA, *CSSICA*, and Niching PSO, applying WSDream-QoSDataset2 as a very large QoS dataset. From Fig. 6*a*, the best-obtained *fps* using *CSSICA* has a lower total service time than the *fps* obtained with the ICA. Part *a* of Figs. 6–9 demonstrate that *CSSICA* can always reach a better *fps* than the ICA. Thus, utilizing *CSSICA* in comparison with the ICA will be more useful for cloud suppliers who have pursued *Appropriate Solution at the Lowest Time (APLT)* policies.

Against *APLT*, *Optimal Solution at the Appropriate Time (OSAT)*, in which waiting is acceptable for only a brief window of time, with the goal of achieving near-optimal solutions, can be adopted. To compare the ICA and *CSSICA* in terms of satisfying this policy, the *fs* should be considered, shown in part *b* of Figs. 6–9. These figures illustrate that applying clustering in *CSSICA* obtains a solution that

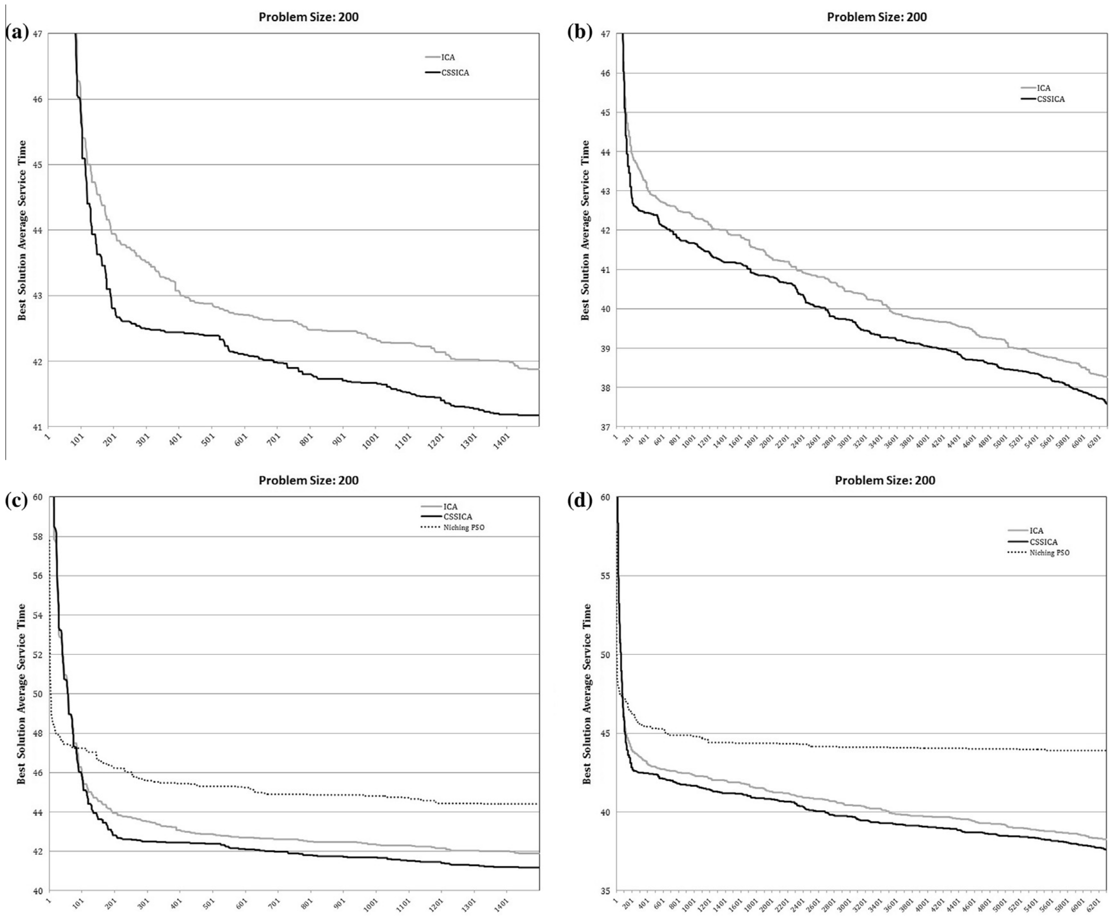


Fig. 7. Comparison of the total service time obtained for problem B, (a) CSSICA vs. ICA in 1,500 iterations, (b) CSSICA vs. ICA in 6,300 iterations, (c) CSSICA vs. ICA and Niching PSO for 1,500 iterations, (d) CSSICA vs. ICA and Niching PSO for 6,300 iterations.

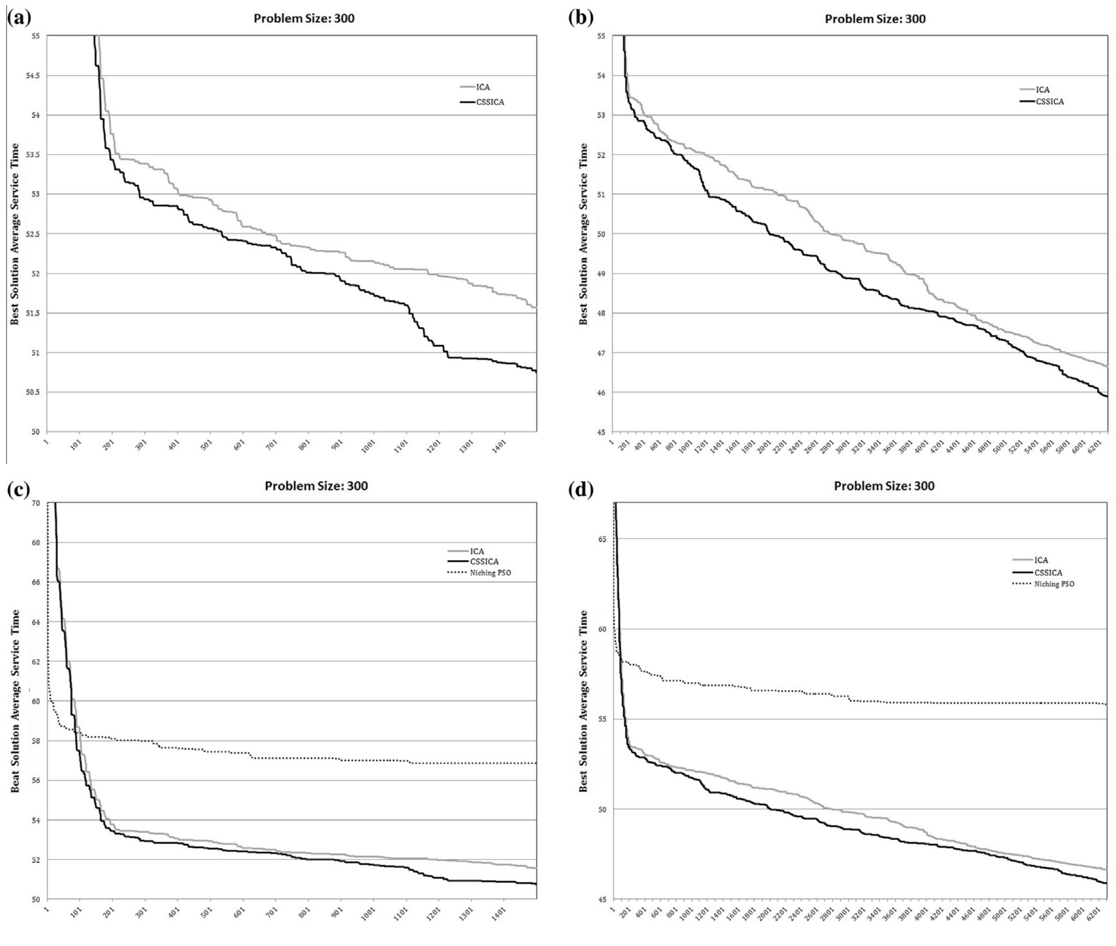


Fig. 8. Comparison of the total service time obtained for problem C, (a) CSSICA vs. ICA in 1,500 iterations, (b) CSSICA vs. ICA in 6,300 iterations, (c) CSSICA vs. ICA and Niching PSO for 1,500 iterations, (d) CSSICA vs. ICA and Niching PSO for 6,300 iterations.

is closer to the optimal than that obtained with the ICA for problems of all sizes.

The performance quality of *CSSICA* is also evident when compared with Niching PSO, which was one of the most effective approaches that have been used previously. Parts *c* and *d* of Figs. 6–9 illustrate this comparison. There are significant differences between the solutions obtained with *CSSICA* and Niching PSO when adopting either the *APLT* or *OSAT*. Although Niching PSO has a comparative advantage in the early iterations of all executions, it eventually falls into the trap of local minima due to the large size of the search space. Accordingly, *CSSICA* is superior in terms of exploring the search space in the later iterations. The results of the first few iterations are not shown in the figures because of the similarity of the results of the three algorithms in those iterations.

4.2.2. Comparison of the time-consumption percentage (TCP)

Another convenient method for evaluating and comparing the investigated algorithms is the *time-consumption check (TCC)*, in which one of the algorithms is chosen as the *basis* of the *time-consumption percentage (TCP)* and the other algorithms are compared with the *basis*. Based on the mathematical definition in (11), the *TCP* can be calculated by dividing the best-obtained total service time of one of the algorithms with the best-obtained total service time of the *basis* and multiplying by 100.

$$TCP(\%) = \frac{Best(A)}{Best(basis)} \times 100 \qquad (11)$$

where *Best(A)* and *Best(basis)* are the best-obtained total service time by algorithm *A* and the *basis*, respectively. A larger difference between *Best(A)* and *Best (basis)* indicates a higher quality of *A*. The *TCC* was used to compare the ICA, *CSSICA*, and Niching PSO results for the four above-mentioned problems using Niching PSO as the *basis*. Fig. 10 presents the optimality calculated in four separate categories, each related to one of the problems. *CSSICA* has achieved more optimality than the ICA and Niching PSO in all categories. Furthermore, the optimality rate of *CSSICA* increases with increasing problem size. The effect of applying clustering in generating the first generation of the countries is evident when using the ICA as the *basis* for *TCC*. Fig. 11 illustrates that applying clustering has led to an improvement in the results for all problem sizes. The improvements are more evident in the *fs* than *fps*. Furthermore, the very low time reduction (0.77%) should be neglected considering the similar impact of clustering on the *fps* of all problems. Regardless, the overall improvements will be significant for the supplier and permanent users due to the large number of requests for composed services.

4.2.3. Statistical performance comparison

Statistically evaluating and comparing the results that were obtained with the previous algorithms can provide more information regarding the algorithm functionality and performance. To this end, different statistical tests were performed using IBM SPSS STATISTICS version 22.

A repeated measures analysis of variance, with the Greenhouse-Geisser correction (Abdi, 2010), was conducted to investigate the difference in mean total service time obtained by the three algo-

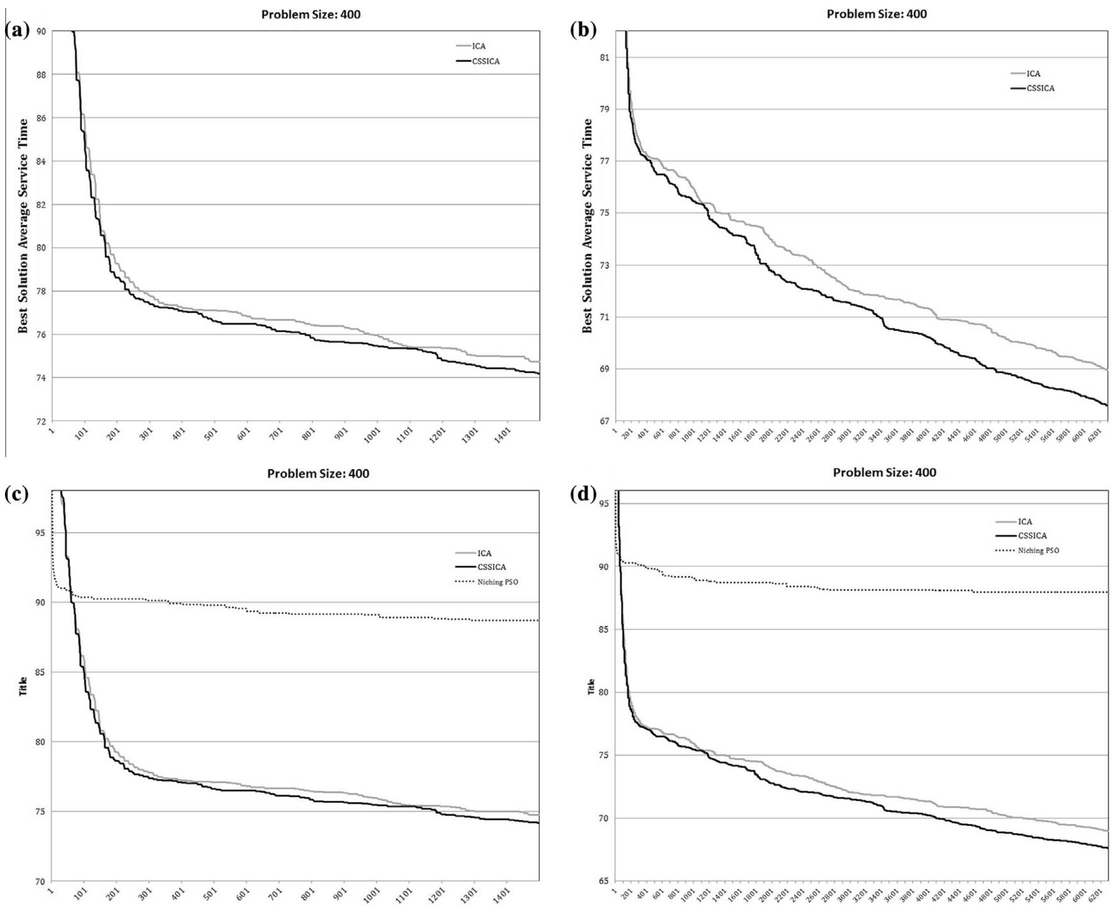
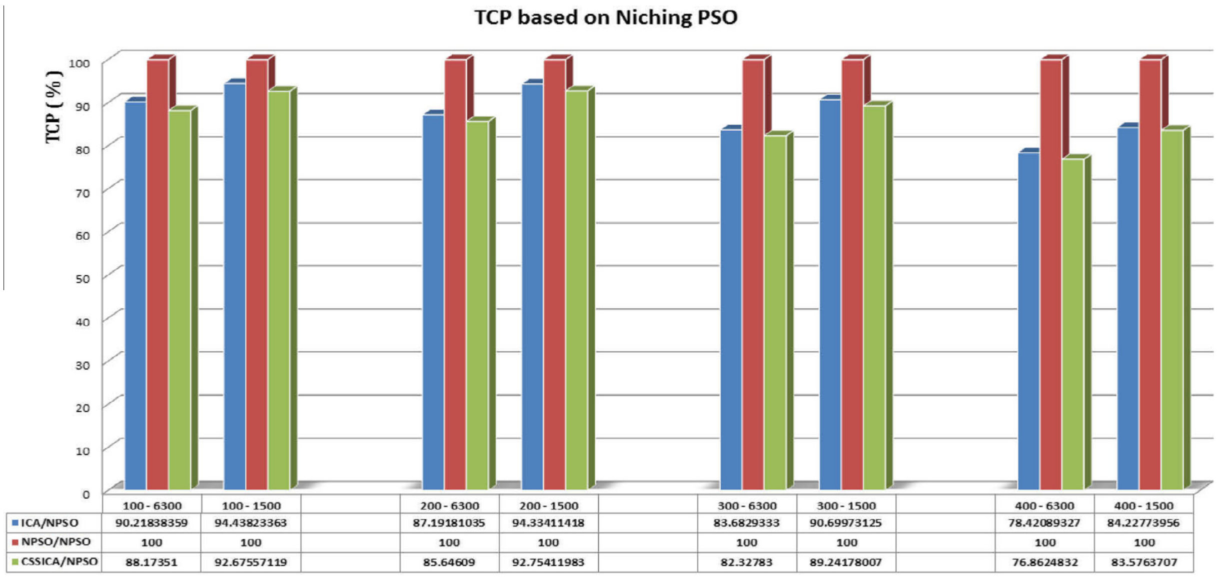


Fig. 9. Comparison of the total service time obtained for problem D, (a) CSSICA vs. ICA in 1,500 iterations, (b) CSSICA vs. ICA in 6,300 iterations, (c) CSSICA vs. ICA and Niching PSO for 1,500 iterations, (d) CSSICA vs. ICA and Niching PSO for 6,300 iterations.



	100 - 6300	100 - 1500	200 - 6300	200 - 1500	300 - 6300	300 - 1500	400 - 6300	400 - 1500
🟦 ICA/NPSO	90.21838359	94.43823363	87.19181035	94.33411418	83.6829333	90.69973125	78.42089327	84.22773956
🟥 NPSO/NPSO	100	100	100	100	100	100	100	100
🟩 CSSICA/NPSO	88.17351	92.67557119	85.64609	92.75411983	82.32783	89.24178007	76.8624832	83.5763707

Fig. 10. Time-consumption percentage of the three algorithms based on Niching PSO.

rithms. The result of repeated measures analysis of variance indicated that the mean service times of the three investigated algorithms were statistically significant for all four problems (see Table 2).

Pairwise comparisons with the Bonferroni correction (Cabin & Mitchell, 2000; Holm, 1979; Nakagawa, 2004) also revealed that *CSSICA* obtained a significantly lower mean total service time than that of ICA and Niching PSO in all four problems. Furthermore, the

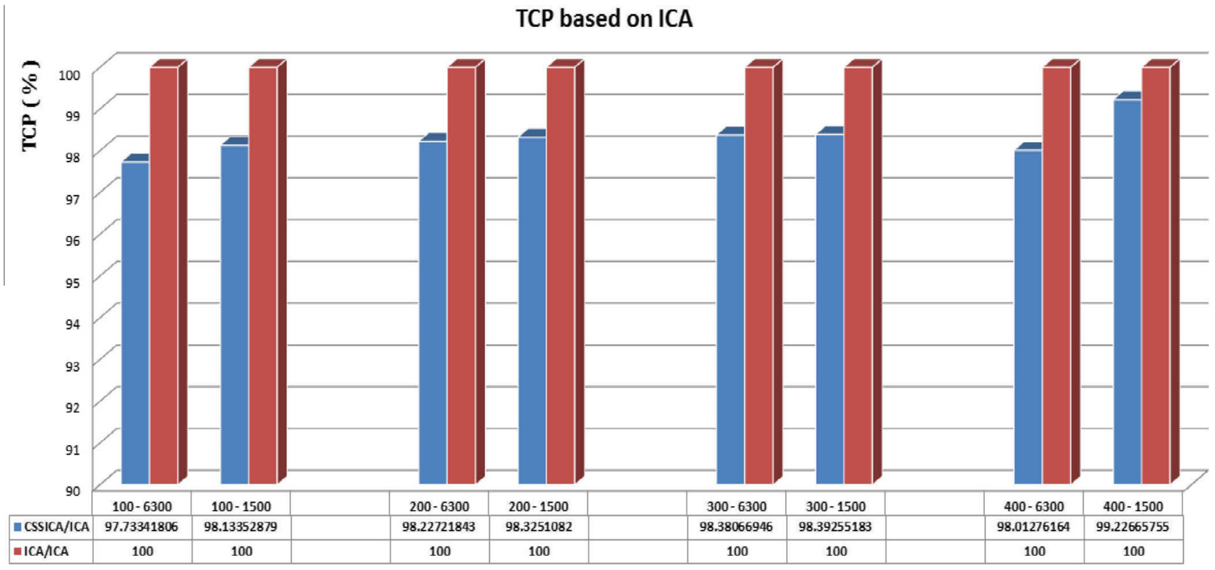


Fig. 11. Time-consumption percentage of CSSICA and the ICA based on the ICA.

Table 2
Results of repeated measures anova.

		df	Mean square	F	Sig.
Problem A	Between Groups	1.024	5300.729	23497.724	<0.001
	Error	6448.384	.226		
Problem B	Between Groups	1.009	66410.356	24353.190	<0.001
	Error	6354.242	2.727		
Problem C	Between Groups	1.017	186689.322	34227.041	<0.001
	Error	6406.163	5.454		
Problem D	Between Groups	1.009	1050096.130	89367.749	<0.001
	Error	6358.801	11.750		

Table 3
Results of Bonferroni pairwise comparisons.

	(I) Algorithm	(J) Algorithm	Mean difference (J - I)	Std. Error	Sig.
Problem A	CSSICA	ICA	0.175	.001	<0.001
	CSSICA	Niching PSO	1.214	.007	<0.001
	ICA	Niching PSO	1.039	.007	<0.001
Problem B	CSSICA	ICA	0.666	.002	<0.001
	CSSICA	Niching PSO	4.285	.026	<0.001
	ICA	Niching PSO	3.619	.025	<0.001
Problem C	CSSICA	ICA	.652	.004	<0.001
	CSSICA	Niching PSO	7.026	.036	<0.001
	ICA	Niching PSO	6.374	.036	<0.001
Problem D	CSSICA	ICA	0.965	.005	<0.001
	CSSICA	Niching PSO	16.348	.054	<0.001
	ICA	Niching PSO	15.382	.052	<0.001

ICA results are also significantly better than the results for Niching PSO (see Table 3).

5. Conclusion

Blindly initialization of first generation of solutions in evolutionary algorithms reduces the probability of providing a successful evolutionary process by which appropriate solutions are guaranteed. The wider the search space and the more dimensions the problem more prominent. Applying evolutionary algorithms in solving *CCSC* as a very large search space optimization problem is also faced such this problem. One of the strategies that can be proposed to solve this problem is search space classification using high-dimensional classification methods. In this paper, applying the PROCLUS algorithm for classifying cloud service providers has led to the adoption of a more realistic approach in generating the first generation of solutions of the ICA and creating the novel algorithm *CSSICA*. Based on service time values of all provided single services, the classifier has divided service providers into three categories called high-recommended, recommended and low-recommended providers for selecting a service of which *CSSICA* uses different probability values. The probability values are also calculated based on average service time values of all service providers that are assigned to each class. In ICA part of *CSSICA*, to avoid from hasty decision in determining the weakest empire, 15 iterations is intended as an opportunity time for empires to try to increase their power before entering into imperialist competition. Experimental and statistical evaluation of the results of *CSSICA*, the ICA, and Niching PSO demonstrated that clustering the cloud service pool plays an important role in reaching more appropriate solutions for *STOCCSC*. *CSSICA* is able to reduce composite service time by 2.27%, 1.78%, 1.62% and 1.99% compared to ICA for solving problems with 100, 200, 300 and 400 required single services, respectively. This is done while total service time for a composite service is reduced by 11.83%, 14.36%, 17.68% and 23.14% compared to Niching PSO in solving four mentioned problems. Furthermore, the fact that *CSSICA* achieved the best results in solving different-sized problems indicates that *CSSICA* is a scalable and efficient algorithm for finding optimal composite services.

With respect to the applied approaches and achievements of the paper, reaching specific goals can form the skeleton of the future research. Aiming to classify cloud computing service providers on the basis of 2 and more QoS parameters, it is indispensable to apply PROCLUS for each parameter separately and propose a model for making final decision about the class to which every service provider actually belongs. Another imperative research goal is to propose new techniques for generating first generation of ICA countries in such a way that whole parts of wide search space of the problem can be covered by the algorithm. Designing novel operators for ICA to enhance its ability in looking more efficiently for most proper solutions in the very large search spaces of CCSC, makes it easier to escape from traps of local optimum solutions. Eventually, admitted to the significant growth of mobile cloud computing and serious QoS differences among mobile devices, future research should be steered toward this progressive field.

References

Abdi, H. (2010). *Greenhouse-Geisser correction. Encyclopedia of research design*. Thousand Oaks, CA: SAGE Publications Inc.

Aggarwal, C. C., Wolf, J. L., Yu, P. S., Procopiuc, C., & Park, J. S. (1999). Fast algorithms for projected clustering. *SIGMOD Record, 28*, 61–72.

Allison, P. D. (2002). *Missing data*. SAGE Publications.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., et al. (2010). A view of cloud computing. *Communications of the ACM, 53*, 50–58.

Atashpaz-Gargari, E., & Lucas, C. (2007). Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In *IEEE congress on evolutionary computation. CEC 2007* (pp. 4661–4667).

Bahrami, H., Faez, K., & Abdechiri, M. (2010). Imperialist competitive algorithm using chaos theory for optimization (CICA). In *12th International conference on computer modelling and simulation (UKSim)* (pp. 98–103).

Barney, B. (2012). Message passing interface (MPI). In (Vol. 2013): Lawrence Livermore National Laboratory.

Berthold, M., Cebron, N., Dill, F., Gabriel, T., Kötter, T., Meinl, T., et al. (2008). KNIME: The Konstanz information miner. In C. Preisach, H. Burkhardt, L. Schmidt-Thieme, & R. Decker (Eds.), *Data analysis, machine learning and applications* (pp. 319–326). Berlin Heidelberg: Springer.

Cabin, R., & Mitchell, R. (2000). To Bonferroni or not to Bonferroni: When and how are the questions. *Bulletin of the Ecological Society of America, 81*, 246–248.

Dillon, T., Chen, W., & Chang, E. (2010). Cloud computing: Issues and challenges. In *24th IEEE international conference on advanced information networking and applications (AINA)* (pp. 27–33).

Ellinger, R. S. (2013). Governance in SOA Patterns (white paper). In The Northrop Grumman corporation for consideration by OASIS SOA reference architecture team (pp. 1–11).

Fei, T., Yuanjun, L., Lida, X., & Lin, Z. (2013). FC-PACO-RM: A parallel method for service composition optimal-selection in cloud manufacturing system. *IEEE Transactions on Industrial Informatics, 9*, 2023–2033.

Gan, W. (2007). *A statistical approach towards performance analysis of multimodal biometrics systems*. Canada: University of Windsor.

Gutierrez-Garcia, J. O., & Sim, K. M. (2010). Agent-based service composition in cloud computing. In T. H. Kim, S. S. Yau, O. Gervasi, B. H. Kang, A. Stoica, & D. Slezak (Eds.). *Grid and distributed computing, control and automation* (Vol. 121, pp. 1–10). Berlin: Springer-Verlag.

Hayes, B. (2008). Cloud computing. *Communications of the ACM, 51*, 9–11.

Holland, J. H. (1992). *Adaptation in natural and artificial systems*. MIT Press.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics, 6*, 65–70.

Jula, A., & Naseri, N. K. (2011). Using CMAC to obtain dynamic mutation rate in a metaheuristic memetic algorithm to solve university timetabling problem. *European Journal of Scientific Research, 63*, 172–181.

Jula, A., Othman, Z., & Sundararajan, E. (2013). A hybrid imperialist competitive-gravitational attraction search algorithm to optimize cloud service composition. In *IEEE workshop on memetic computing (MC)* (pp. 37–43).

Jula, A., Sundararajan, E., & Othman, Z. (2014). Cloud computing service composition: A systematic literature review. *Expert Systems with Applications, 41*, 3809–3824.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks* (Vol. 4, pp. 1942–1948, Vol. 1944).

Kofler, K., Haq, I. U., & Schikuta, E. (2010). User-centric, heuristic optimization of service composition in clouds. In LNCS (Vol. 6271, pp. 405–417).

Kofler, K., ul Haq, I., & Schikuta, E. (2009). A parallel branch and bound algorithm for workflow QoS optimization. In *International conference on parallel processing. ICPP '09* (pp. 478–485).

Kurniawan, A., Benech, N., Yufei, T., Feng, T., Jiying, W., & Malamatos, T. (1999). Towards high-dimensional clustering. In The Hong Kong University of Science and Technology (pp. 43).

Li, L., Cheng, P., Ou, L., & Zhang, Z. (2010). Applying multi-objective evolutionary algorithms to QoS-aware web service composition. In L. Cao, J. Zhong, & Y. Feng (Eds.). *Advanced data mining and applications* (Vol. 6441). Berlin Heidelberg: Springer, pp. 270–281.

Liao, J. X., Liu, Y., Wang, J. Y., & Zhu, X. M. (2012). Service composition based on niching particle swarm optimization in service overlay networks. *KSII Transactions on Internet and Information Systems, 6*, 1106–1127.

Liao, J. X., Liu, Y., Zhu, X. M., Xu, T., Wang, & J. Y. (2011). Niching particle swarm optimization algorithm for service composition. In *IEEE global telecommunications conference*. New York: IEEE.

Ludwig, S. A. (2012). Clonal selection based genetic algorithm for workflow service selection. In *IEEE congress on evolutionary computation (CEC)* (pp. 1–7).

Nakagawa, S. (2004). A farewell to Bonferroni: The problems of low statistical power and publication bias. *Behavioral Ecology, 15*, 1044–1045.

Di Nuovo, Alessandro G. (2011). Missing data analysis with fuzzy C-means: A study of its application in a psychological scenario. *Expert Systems with Applications, 38*, 6793–6797.

Peter Mell, T. G. (2011). The NIST definition of cloud computing. In *N. I. o. S. a. technology* (Ed.): US Department of Commerce.

Qi, Y., & Bouguettaya, A. (2013). Efficient service skyline computation for composite service selection. *IEEE Transactions on Knowledge and Data Engineering, 25*, 776–789.

Scutari, G., Palomar, D. P., & Barbarossa, S. (2008). Optimal linear precoding strategies for wideband noncooperative systems based on game theory – Part I: Nash equilibria. *IEEE Transactions on Signal Processing, 56*, 1230–1249.

Takabi, H., Joshi, J. B. D., & Gail-Joon, A. (2010). Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy, 8*, 24–31.

Wada, H., Suzuki, J., Yamano, Y., & Oba, K. (2012). E^3: A multiobjective optimization framework for SLA-aware service composition. *IEEE Transactions on Services Computing, 5*, 358–372.

Wang, S. G., Sun, Q. B., Zou, H., & Yang, F. C. (2013). Particle swarm optimization with skyline operator for fast cloud-based web service composition. *Mobile Networks & Applications, 18*, 116–121.

Wei, L., Zhu, H., Cao, Z., Dong, X., Jia, W., Chen, Y., et al. (2014). Security and privacy for storage and computation in cloud computing. *Information Sciences, 258*, 371–386.

Widia Sembiring, R., Mohamad Zain, J., & Embong, A. (2010). Clustering high dimensional data using subspace and projected clustering algorithms. *International Journal of Computer Science & Information Technology, 2*, 162–170.

Xia, Y., Wan, N., Dai, G., Luo, X., & Sun, T. (2012). A non-Markovian stochastic Petri net-based approach to performance evaluation of ontology-based service composition. *Concurrency and Computation: Practice and Experience, 24*, 2255–2267.

Xiao, P., & Zhang, Y. (2012). An evolution strategy based service composition algorithm in cloud computing systems. *International Review on Computers and Software, 7*, 996–1003.

Yang, Y., Mi, Z., & Sun, J. (2012). Game theory based IaaS services composition in cloud computing environment. *Advances in Information Sciences and Service Sciences, 4*, 238–246.

Ye, Z., Zhou, X., & Bouguettaya, A. (2011). Genetic algorithm based QoS-aware service compositions in cloud computing. In J. Yu, M. Kim, & R. Unland (Eds.). *Database systems for advanced applications* (Vol. 6588, pp. 321–334). Berlin Heidelberg: Springer.

Zarandi, M. H. F., Zarinbal, M., Ghanbari, N., & Turksen, I. B. (2013). A new fuzzy functions model tuned by hybridizing imperialist competitive algorithm and simulated annealing. Application: Stock price prediction. *Information Sciences, 222*, 213–228.

Zhou, X., & Mao, F. (2012). A semantics web service composition approach based on cloud computing. In (pp. 807–810).

Zhu, Y., Li, W., Luo, J., & Zheng, X. (2012). A novel two-phase approach for QoS-aware service composition based on history records. In *5th IEEE international conference on service-oriented computing and applications (SOCA)* (pp. 1–8).

Zibin, Z., Yilei, Z., & Lyu, M. R. (2010). Distributed QoS evaluation for real-world web services. In *IEEE international conference on web services (ICWS)* (pp. 83–90).

Zissis, D., & Lekkas, D. (2012). Addressing cloud computing security issues. *Future Generation Computer Systems, 28*, 583–592.