symmetry

MDPI

Article

Color Revolution: A Novel Operator for Imperialist Competitive Algorithm in Solving Cloud Computing Service Composition Problem

Amin Jula [1,*], Elankovan A. Sundararajan [2], Zalinda Othman [1] and Narjes Khatoon Naseri [2]

[1] Centre for Artificial Intelligent (CAIT), Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi 43600, Selangor, Malaysia; zalinda@ukm.edu.my
[2] Centre for Software Technology and Management, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi 43600, Selangor, Malaysia; elan@ukm.edu.my (E.A.S.); narges.naseri85@gmail.com (N.K.N.)
* Correspondence: amin.jula@gmail.com

Abstract: In this paper, a novel high-performance and low-cost operator is proposed for the imperialist competitive algorithm (ICA). The operator, inspired by a sociopolitical movement called the color revolution that has recently arisen in some countries, is referred to as the color revolution operator (CRO). The improved ICA with CRO, denoted as ICACRO, is significantly more efficient than the ICA. On the other hand, cloud computing service composition is a high-dimensional optimization problem that has become more prominent in recent years due to the unprecedented increase in both the number of services in the service pool and the number of service providers. In this study, two different types of ICACRO, one that applies the CRO to all countries of the world (ICACRO-C) and one that applies the CRO solely to imperialist countries (ICACRO-I), were used for service time-cost optimization in cloud computing service composition. The ICACRO was evaluated using a large-scale dataset and five service time-cost optimization problems with different difficulty levels. Compared to the basic ICA and niching PSO, the experimental and statistical tests demonstrate that the ability of the ICACRO to approach an optimal solution is considerably higher and that the ICACRO can be considered an efficient and scalable approach. Furthermore, the ICACRO-C is stronger than the ICACRO-I in terms of the solution quality with respect to execution time. However, the differences are negligible when solving large-scale problems.

Keywords: cloud computing; color revolution operator; imperialist competitive algorithm; quality of service; service composition; service time-cost

Citation: Jula, A.; Sundararajan, E.A.; Othman, Z.; Naseri, N.K. Color Revolution: A Novel Operator for Imperialist Competitive Algorithm in Solving Cloud Computing Service Composition Problem. *Symmetry* **2021**, *13*, 177. https://doi.org/10.3390/sym13020177

Academic Editors: Hari M. Srivastava and José Carlos R. Alcantud
Received: 26 November 2020
Accepted: 18 January 2021
Published: 22 January 2021

1. Introduction

Providing enhanced processing facilities and appropriate on-demand self-service systems has always been the major concern of web service suppliers [1]. Recently, these concerns were largely resolved by the introduction and development of cloud computing [2,3]. Its appealing financial and technical characteristics [4] and the increasing complexity of the requested services have led to a growing trend among service providers and customers toward cloud computing.

Although clouds and their applications are growing rapidly, service providers are not able to prepare all the complex combinations of required services. Hence, clouds suppliers need a mechanism for composing the required composite services using the unique services already provided in the service pool (see Figure 1).

The dramatic increase in cloud computing customers has made the environment a lucrative commercial space that encourages facility owners to provide services. Hence, a considerable number of service providers are currently offering a large set of different unique services in the pool, allowing one to find a large number of instances of the same

service, each with different functional and quality of service (QoS) specifications [5–8]. In the current situation, cloud suppliers are confronted with a difficult optimization problem [4,9–12]: providing the optimal compositions of unique services that will satisfy composite service customers and persuade them to make their next requests. This is the problem that network function virtualization (NFV) [13] systems are also confronting [14,15]. Due to the undeniable importance of the cloud computing service composition (CCSC) problem, which is an NP-hard problem [9,16–18], an increasing amount of research on the development of cloud computing has been conducted, which will be studied in Section 1.1. Despite the research conducted and progress made thus far, considerable effort is still needed to close the gaps between the solutions obtained to date and the optimal CCSC solutions.

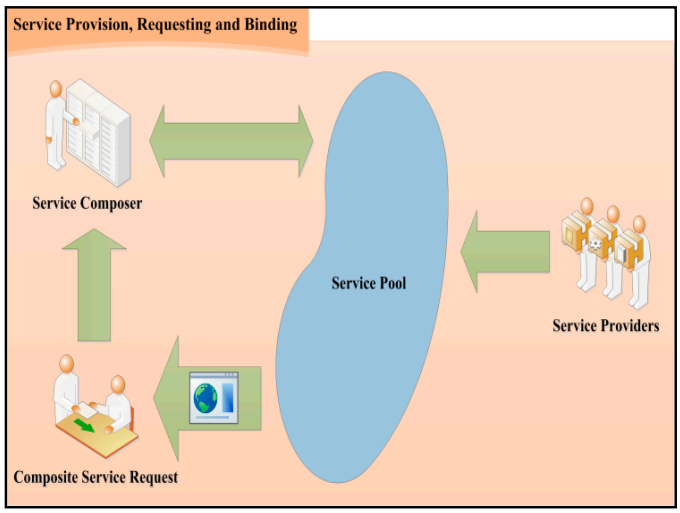


Figure 1. Service provision, requesting and binding in cloud computing.

These gaps, which have mostly arisen because of the very large problem search space, necessitate the customization of more efficient evolutionary algorithms (EAs) to solve the CCSC problem with higher accuracy and optimality. In addition, service time and service cost have been identified as the two most important QoS parameters in cloud service composition that should be further considered in optimizing the QoS [4]. Hence, in this paper, the researchers are strongly motivated to focus on service time and cost optimization for CCSC by improving the imperialist competitive algorithm (ICA) [18–20].

In this study, a new operator is designed based on the sociopolitical movements known as the color revolutions, which have largely been observed in newly independent states of the former Soviet Union and the Balkans and applied to the ICA. The operator is denoted as the color revolution operator (CRO) and attempts to replace the worst part of a country (solution) with a better option within an acceptable amount of time. The algorithm, obtained by applying the *CRO* to the ICA, is called the imperialist competitive algorithm with color revolution operator (ICACRO), two forms of which are used in the service time-cost optimization for CCSC.

To provide a more explicit description of the subject, a complete explanation of the research motivations is presented. The strategy employed by many EAs in finding the optimal solutions to a given optimization problem is to perform a random search in the specified search space and impose a variety of information to guide the search process such that it is carried out more efficiently. Accordingly, the applied algorithm is expected to reach the optimal solution of a given problem within a reasonable amount of time provided that the problem does not include a very large search space and a large number of dimensions.

When facing a problem with a vast search space and numerous dimensions, the number of space points that are actually potential solutions increases to the extent that, in practice, very few of them can be investigated using the algorithm.

Example 1. *In this paper, WSDream-QoSDataset2 [21] is used as a real-world dataset to evaluate the ability of the proposed algorithms to solve the service time-cost optimization problem in CCSC (STCOCCSC) (please refer to part 2.1 for the problem description). A total of 339 service providers exist in the dataset, each of which provides 5825 different simple services. On the one hand, suppose that 100 different simple services are required to represent a composite cloud service. Each of the required simple services can be provided by any of the 339 service providers. In this STCOCCSC problem, the algorithm faces a 100-dimensional problem in which each dimension involves 339 options. Hence, there are* 339^{100} *distinct potential solutions in the search space. On the other hand, assume that 1000 members exist in the first generation of solutions of the applied evolutionary algorithm and that the algorithm is executed 6000 times. Accordingly, in the most optimistic situation, 6,000,000 different potential solutions are examined by the algorithm provided that repeated investigated solutions are neglected. Hence,* $5.73e-245\%$ *of all available points in the search space are examined during the execution time. In other words, many potential solutions are neglected by the algorithm.*

Example 2. *A visual description of the issue is illustrated in Figure 2, in which Nh is the radius of the neighborhood circle of an investigated potential solution. As mentioned above, many potential solutions located in the neighborhood circle are neglected by the algorithm. Although ICA designers have made commendable efforts to prepare a more successful search using irregular but purposive movement of the solutions in the search space, the large search space of some problems such as the STCOCCSC greatly reduces the efficiency of the algorithm. In this study, a new operator is designed and proposed for the ICA to improve its efficiency in investigating large search spaces. The remainder of the paper is organized as follows. A motivation section is provided as the last part of the introduction. Related works are briefly studied in Section 1.1. Descriptions of the STCOCCSC and the ICA are presented in Section 2. Complete descriptions of the proposed operator (CRO) and algorithm (ICACRO) are provided in Section 3. The experimental design and test results obtained using two different types of the ICACRO as well as two other algorithms, i.e., the ICA and niching particle swarm optimization (PSO), including numerical and statistical investigations, are discussed in detail in Sections 4 and 5, respectively. Finally, our conclusions and potential topics for future research are presented in Section 6.*

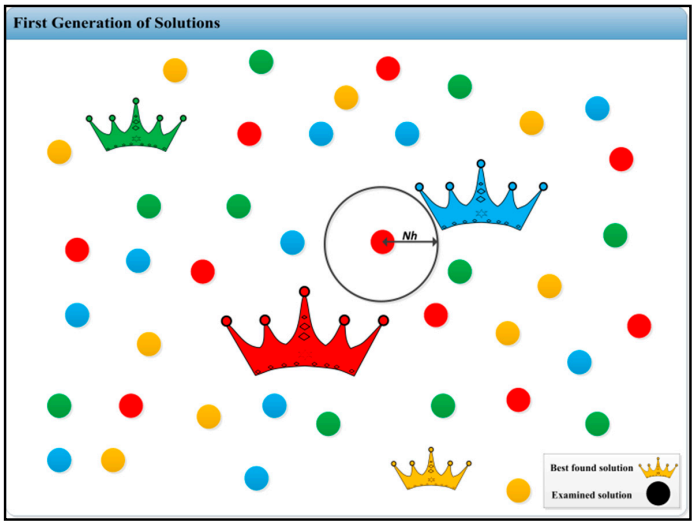


Figure 2. Countries of the world and their neighborhoods.

1.1. Literature Review

Different goals have been pursued by different researchers in solving the CCSC problem, but a factor common to most of these studies is the consideration of QoS parameters, particularly service time and service cost. The objectives pursued in the literature, the QoS parameters investigated, and their importance percentages were detailed by Jula et al. in [4]. The previous studies conducted to solve the CCSC as an optimization problem can be divided into two main categories.

The first category consists of studies that used different classic algorithms and designed customized structures or workflows. Koflet et al. mapped the problem onto the knapsack problem and solved it by applying a parallel form of the branch-and-bound method [22]. Backtracking and dynamic programming are other classic algorithms used to solve the CCSC problem [23,24]. Mixed integer programming (MIP) and linear programming (LP) were employed by Shangguang et al. and Zhu et al. [25–27] and Hossain et al. [28], respectively, to find appropriate unique services. CCSC is also often mapped onto graph structures to employ graph-based algorithms.

A weighted directed graph with an improved fast-EP algorithm was used in [29], and other examples of applying graph structures in the field include the directed acyclic graph (DAG) [30], discovery graph with Markov chain [31], and the graphical construction of the analytic hierarchy process (AHP) [32]. Petri nets are another non-heuristic approach that has been used to solve the CCSC problem [33].

Artificially intelligent, combinatorial, and evolutionary algorithms [34–39] constitute the second category of studies conducted to solve the CCSC problem. Although these algorithms have been used less frequently than the algorithms included in the first category, the importance of achieving closer-to-optimal solutions and shorter execution times and increases in the size of CCSC problems have forced researchers to prioritize the application of EAs [40,41]. Different types of artificial neural networks (ANNs) and fuzzy logic have been used in several studies [33,42–44]. A genetic algorithm (GA) including a roulette wheel selection was applied by Ye et al. in 2011 [45]. Two improved GAs were also proposed by Klein et al. and Ludwig [46,47].

The most successful proposed approaches are based on PSO [48,49], the chaos optimization algorithm (COA) [50], game theory [51], and hybrid algorithms. In addition, unlike almost all conducted studies focusing on applying different search approaches to find the best combination of single services, PROCLUS, as a high-dimensional clustering algorithm, was utilized in [52] to categorize service providers. The categorized search space enhanced the ability of the ICA to select the best possible services by optimizing the service time in CCSC. To accomplish a comprehensive review of the literature of the domain, including the proposed methods, used datasets and tools, and different research objectives, a systematic literature review was presented in [4].

2. Problem and Algorithm Description

2.1. Service Time-Cost Optimization in Cloud Computing Service Composition (STCOCCSC)

With the development of computer-based system procedures, process execution and the required services have become more complex. Due to the growing complexity and variety of systems, a simple independent service is incapable of satisfying functional prerequisites of various real-world requests. Hence, it is necessary to prepare a set of simple atomic services that can effectively work together to perform a complex service. Therefore, a cloud service composer system (CSC) must be incorporated into cloud computing. Increasing the number of service providers will increase the number of similar unique services. Because the similar services are found in different parts of the network and have absolute QoS values, the CSC should efficiently choose a unique service for each requirement among the many similar services provided by different service providers. Application of the most suitable method yields the highest QoS based on customers' requirements and priorities.

Due to fundamental changes in cloud environments, accessible services, and service customer needs, automatic functional capabilities are mandatory in designing the CSC. Therefore, choosing appropriate unique services for merging to provide an optimal combination that satisfies the functional and QoS requirements of a customer can be considered one of the foremost critical problems of service composition. This problem is called Cloud Computing Service Composition (*CCSC*) that is defined in detail and discussed in [4].

This paper assumes that within a cloud, each composite service (CS) consists of n unique services (*USs*), each of which has service time (*ST*) and service cost (*SC*) as its QoS parameters. A combination of *USs* must correspondingly act in an ordinal workflow (*wf*) to provide a required CS. However, if wf_k is the workflow of CS_k, then $ST(wf_k)$ and $SC(wf_k)$ will be defined as the *ST* and *SC* of the workflow k, where the *ST* and *SC* vectors of the workflow can be expressed as (1) and (2), respectively.

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

$$SC(wf_k) = (SC_1(wf_k), SC_2(wf_k), \ldots, SC_n(wf_k)) \quad (2)$$

The merit value (*MV*) of wf_k is the sum of the total *STs* and total *SCs* of all elements of wf_k and is calculated using (3), where TW and CW are the weights of time and cost, which should be determined by the user from [0,1] such that $TW + CW = 1$. Hence, the optimal solution to *STCOCCSC* is the solution with the minimum *MV* value. Note that the *ST* and cost values should be normalized between 1 and 10 via min-max normalization [53] to be used in (3). Section 4 shows that the proposed operator uses the *MV* of solutions. Hence, different structures of the *STCOCCSC* have no effect on the application and efficiency of the operator and are therefore neglected here.

$$MV(wf_k) = \sum_{i=1}^{n} (TW \times ST_i(wf_k) + CW \times SC_i(wf_k)) \quad (3)$$

2.2. Imperialist Competitive Algorithm (ICA)

In evolutionary computation, the recently proposed algorithm, ICA, was formed based on social and political activities [20,54,55], in contrast to other EAs, which are based on the physical events or animals' natural behaviors. The ICA starts with an initial population created at random, in which members of the population are regarded as countries. A few of the most powerful countries are considered imperialist, whereas the rest represent colonies of the imperialists. Assuming that there are n dimensions in the given optimization problem, a country is treated as an $1 \times n$ array as in (4):

$$Country = [p_1, p_2, \ldots, p_n] \quad (4)$$

The power of 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:

$$Power(Country_i) = f(Country_i) = f(p_{i1}, p_{i2}, \ldots, p_{in}) \quad (5)$$

The ICA commences with m countries, and n_{imp} of the most powerful ones are labeled as the imperialists. The other countries are called colonies each of which belongs to an empire. A simple procedure is used to disperse the non-imperialists among the imperialists. An imperialist is randomly selected for each non-imperialist country, which is dedicated to that empire. During the execution of the algorithm, every imperialist attracts their colonies according to the total power of the empire and the colonies. As Equation (6) states, the total power of each empire is the sum of the corresponding imperialist's power and average of power of all the colonies of the empire multiplied to a coefficient.

$$TP_n = c_n + (\alpha \times Average\{power(coloniesofempire_n)\}) \quad (6)$$

where TP_n is the total power of the nth empire, c_n is the power of the imperialist country and α is a positive decimal number between 0 and 1.

In the course of movement stage, a colony moves a x units toward its imperialist. As illustrated in Figure 3, the direction of the movement can be represented by a vector from the colony to its associated imperialist, where d is the distance between the colony and the imperialist, and x is a value taken at random that can be obtained following a uniform distribution like what is shown in (7):

$$x \approx Uniform(0, \beta \times d) \tag{7}$$

where β is greater than one and close to 2.

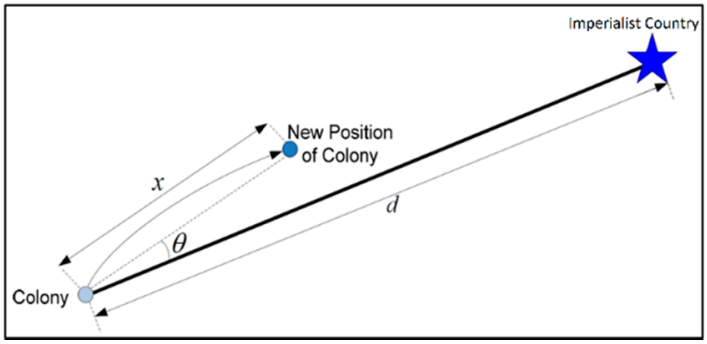


Figure 3. Country movement [20].

In ICA, the imperialistic competition plays an important role in the convergence of the algorithm. This operator decreases the power of the weakest empire by taking some of its colonies and assigning them to other empires. The process will be continued until the weakest empire is destroyed. Imperialistic competition can be applied in various forms based on the requirements of the problem. To facilitate the description of the proposed algorithm, a mapping description of the key terms of the *STCOCCSC*, ICA and optimization is presented in Table 1.

Table 1. Mapping description of key terms of *service time-cost optimization problem in CCSC (STCOCCSC)* and imperialist competitive algorithm (ICA).

STCOCCSC	ICA	Optimization
Composite Service (CS)	Country	Solution
Merit Value (MV)	Power of Country	Objective Function

3. Imperialist Competitive Algorithm with the Color Revolution Operator (ICACRO)

Manipulation of the routine process may enhance the ability to navigate the search space using heuristic algorithms if it is performed intelligently. Changes must be made such that the optimal solution can be reached without changing the nature of the algorithm. For this purpose, the CRO is applied to the ICA to obtain closer-to-optimal solutions for the *STCOCCSC*. As described in Section 4.1 and shown in Figure 4, by making a heuristic change in a selected part of a solution structure that does not yield an evolution of the solution structure, the CRO increases the likelihood of achieving more suitable solutions within a smaller execution time. The CRO can be imposed on either all existing countries of the ICA world or only selected ones. The functional details and use conditions of the CRO are discussed in the following sections.

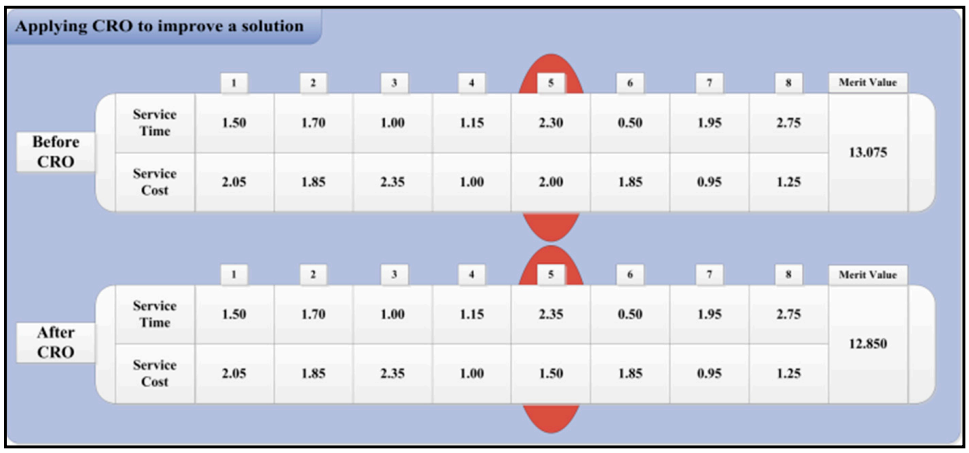


Figure 4. A simple example of applying the color revolution operator (CRO).

3.1. Color Revolution Operator (CRO)

New operators designed for an algorithm should ideally be similar to the algorithm in terms of nature and structure. Hence, the operator designed to improve the ICA is similar to the ICA and rooted in the sociopolitical attitudes of society. The main concept of the CRO is inspired by the color revolutions that occurred in some newly independent states of the former Soviet Union and Balkans during the early 21st century.

The color revolutions are movements in which activists protesting against the government or ruling party under a common symbol carry out civil disobedience and nonviolent resistance to exert strong pressure to enact change. The desired changes occur in the laws governing the country, with the structures remaining intact, increasing hope for social and political reform in a society. The quality of life is expected to be enhanced after a color revolution; however, the targeted improvements are uncertain [56,57].

Based on the characteristics of color revolutions described above, the CRO must impose changes in the selected countries without changing either the structure of the countries or the process through which the algorithm is executed. The operator should try to replace one of the less optimal features of the target solution with a more suitable feature with the ultimate goal of improving the solution. To reach this goal, in this study, the CRO examines the USs that compose the existing CS to identify the unique service with the greatest time-cost value (TCV), where the TCV is the sum of normalized ST and SC values for the service. To do this, the CRO should traverse the solution to find its maximum TCV.

If the number of service providers is m, then the process of finding the maximum value of the list using a linear time operation has a time complexity of $O(m)$. In the next step, the CRO randomly seeks a better alternative among service providers to replace a service with one that provides a shorter TCV. Replacing the service that requires the greatest TCV with an equivalent service that terminates for a smaller TCV will decrease the MV of the CS. An important factor of the performance and time complexity of the CRO is the number of service providers considered by the operator when searching for a smaller TCV. How many attempts should the CRO make to find an alternative service provider? This question has no single answer.

The service provider that provides a smaller TCV and is sought by the CRO could be located anywhere on the list of providers considered; it might be the first one considered or may not even be included in the list because there is no better provider. Therefore, it is best to choose the number of attempts randomly. To provide a reasonable likelihood of finding a better provider while preventing an excessive search time, the random number should be neither too small nor too large. Hence, one should determine a limited range from which the number of attempts should be chosen. In this paper, $\left[\log_2 m, \log_2^2 m\right]$ is applied to obtain the number of attempts needed to find a better service provider, where m indicates the number of service providers in the list. As a simple example, if the

number of service providers is 1024, then the number of attempts will be a random number between 10 and 100. Clearly, $\left(\log_2 m + \log_2^2 m\right)/2$ attempts will result in better choices being found. After the replacing process, the CRO can calculate the new MV of the solution simply and quickly by finding the difference between the old TCV and the new one and subtracting it from the current MV. Accordingly, the CRO time complexity, as shown in (8), will be $O\left(\log_2^2 m\right)$.

Figure 4 provides an example of applying the CRO to improve a solution that includes 8 USs, in which the worst selected service (located in the 5th field) is replaced with a new one, yielding a time-cost reduction in the solution. In this example, the weights of time and cost are 0.5. The pseudocode for the CRO is presented in Figure 5. Another factor to consider is the number of countries that would be affected by the color revolution. This factor has a considerable impact on the overall performance of the algorithm and will be discussed in part 6.

$$T(CRO) = O(\log_2 m) + O\left(\log_2^2 m\right) \Rightarrow T(CRO) = O\left(\log_2^2 m\right) \quad (8)$$

Algorithm 1. Function CRO
Input: Country ID i
Output: Updated Country ID i

```
BEGIN
m = Number of available service providers;
ReqServiceNo = Number of required unique services;
WorstServiceID = 1;
WorstProviderID = i.AssignedProvider[ 1 ];
SelectedProviderID = 1;
WorstServiceTimeCost = i.ServiceTimeCost[WorstServiceID];
for all assigned services j ∈ [2, ReqServiceNo] to country i do
Begin
    if ( i.ServiceTimeCost[ j ] > WorstServiceTimeCost ) then
        WorstServiceID = j;
        WorstServiceTimeCost = i.ServiceTimeCost[ j ] ;
        WorstProviderID = i.AssignedProvider[ j ];
    end if;
end;
Rnd = A random number in [log2 m, log2^2 m];
for k =1 to rnd do
Begin
    TmpServPro = A random number in [1, m];
    if ( Provider [TmpServPro, WorstServiceID].TimeCost <
        Provider[WorstProviderID, WorstServiceID].TimeCost )
    then
        i.AssignedProvider[WorstServiceID] = TmpServPro;
        SelectedProviderID = TmpServPro;
        WorstProviderID = SelectedProviderID;
    end if;
MeritDecreaseAmount =
    WorstServiceTimeCost –
    Provider[SelectedProviderID, WorstServiceID].TimeCost;
i.MeritValue = i.MeritValue – (MeritDecreaseAmount);
return i;
END;
```

Figure 5. Pseudocode of applied *CRO* in imperialist competitive algorithm with color revolution operator (ICACRO).

3.2. ICA with CRO (ICACRO)

Designing an efficient and simple structure for solutions is important for enhancing the implementation performance. Hence, the structure of a country in the ICACRO consists of two separate parts. The first part is an array of d elements, representing the index of service providers that are selected to provide the *USs* required, where d is the number of required *USs* that should be composed to provide the required *CS* and is thus identified as the size of the problem. The second part of the country structure is a decimal variable for storing the total *ST*, which is the *MV* of the solution.

The execution of the algorithm begins with the generation of a set of countries called the 'world'. The world consists of *CountryNo* countries, in which each country represents a solution to the intended problem. Each country is generated by randomly selecting a service provider for each of the required *USs*. The *MV* of each country is calculated immediately following its generation. The *MV* of country i is equal to the sum of the *STs* and *SCs* of all its constituent services and is calculated using (9).

$$Country\_TST(i) = -\frac{1}{2} \sum_{j \in Country_i} (ST(\ j\ ) + SC(\ j\ )) \tag{9}$$

where *Country_TST(i)* is the *MV* of country i and *ST(j)* and *SC(j)* are, respectively, the *ST* and *SC* of the required service j obtained from the selected service provider; the *TW* and *CW* values are assumed to both equal 0.5. Because the ICA was originally designed for maximization and the objective of the *STCOCCSC* is to achieve the minimum total service time, the negative sign reflects a smaller *MV* and thus a larger quantity. The countries are sorted in ascending order at the end of the generation process, which helps the algorithm by placing the best solution at the top of the world list. In this phase, the first *imperialistNo* countries of the world list are selected as imperialist countries. The remaining countries are considered colonies. The number of colonies can be obtained using (10).

$$ColonyNo = CountryNo - imperialistNo \tag{10}$$

The policy applied by the ICACRO to distribute the colonies among the imperialist countries is an equal division. Hence, the number of colonies or all imperialist countries is equal in the first iteration of the algorithm and can be easily obtained using (11).

$$ColonialNo = \frac{ColonyNo}{imperialistNo} \tag{11}$$

Colony displacement is the next phase of the ICACRO, in which each colony finds and moves to a new location closer to its imperialist country. Displacements of various sizes are made in distinct dimensions such that the displacement in each dimension is the distance between the colony and imperialist country in the dimension multiplied by a random coefficient. The distance between colony i and its imperialist country (imp_k) and the displacement of colony i in dimension d can be calculated using (12) and (13), respectively.

$$dist^d(imp_k, colony_i) = imp_k.AssignedServer(d) - colony_i.AssignedServer(d) \tag{12}$$

$$disp^d(colony_i) = rnd \times dist^d(imp_k, colony_i) \tag{13}$$

where *AssignedServer(d)* is the dedicated service provider for the required service d and *rnd* is a random decimal in $(0, 1]$.

The obtained distance is a positive number provided that the index of the imperialist's assigned server is greater than the index of the server dedicated to the colony and vice versa. Similarly, the size of the displacement may also be positive or negative. Hence, the index of the next assigned service provider can be less or greater than the index of the current service provider. The index of the next assigned service provider of colony i for dimension d can be obtained using (14).

$$colony_i.AssignedServer^{t+1}(d) = colony_i.AssignedServer^{t}(d) + disp^{d}(colony_i) \quad (14)$$

The *MV* of each country is calculated after every displacement. The newly calculated *MV* of the colony may be greater than that of its imperialist country, in which case the power of the colony is greater than that of the imperialist country and their positions should be exchanged. After the position exchange, the previous colony and imperialist country are now an imperialist country and colony, respectively.

After the displacement and position exchange of the countries, the *CRO* is introduced to enhance the performance of the ICA. Excessive usage of the *CRO* may affect the convergence of the ICA and disturb its evolutionary process, although it is not permitted to make basic changes according to the color revolution concept. Excessive usage may also lead to an unacceptable increase in the execution time of the ICACRO. Hence, one must apply a *CRO* rate by which the effects of the *CRO* are controlled. For this purpose, a *CRO* rate of 0.05 is used in the ICACRO, i.e., 5 out of 100 iterations of the algorithm will execute the *CRO*. The *MV* of the colonies should be recalculated after the *CRO* is executed for every colony. Accordingly, the colony and its imperialist country should exchange their positions if the newly obtained *MV* of the colony is better than that of the imperialist country.

In the last phase of the ICACRO, the imperialist countries enter an imperialistic competition. In this competition, each imperialist country attempts to overcome the other imperialist countries by taking the weakest colony of the weakest empire. The victorious imperialist country separates the colony from its empire and adds it to its own colony set. An imperialist country with no colonies in its empire will itself be taken by the victorious country. To ensure that there is an adequate opportunity for each imperialist country to increase its number of colonies and thus strengthen its empire, the imperialist competition function is called at a rate of 0.05, i.e., the imperialist competition is executed in 5 out of every 100 iterations, as described for the *CRO*.

The ICACRO is designed such that except during the first initialization, it does not require all the countries to be sorted to find the optimal solution. Therefore, it is sufficient to sort the imperialist countries in descending order at the end of each iteration of the algorithm. Because there are fewer imperialist countries than non-imperialist countries, a clear reduction in the ICACRO execution time compared to that of the ICA occurs. The ICACRO can be terminated after a specified number of iterations, the achievement of a specific *MV*, or the disappearance of all but one empire in the world. If the termination criterion is not satisfied, the next iteration will start via displacement. The ICACRO flowchart and pseudocode are shown in Figures 6 and 7, respectively.

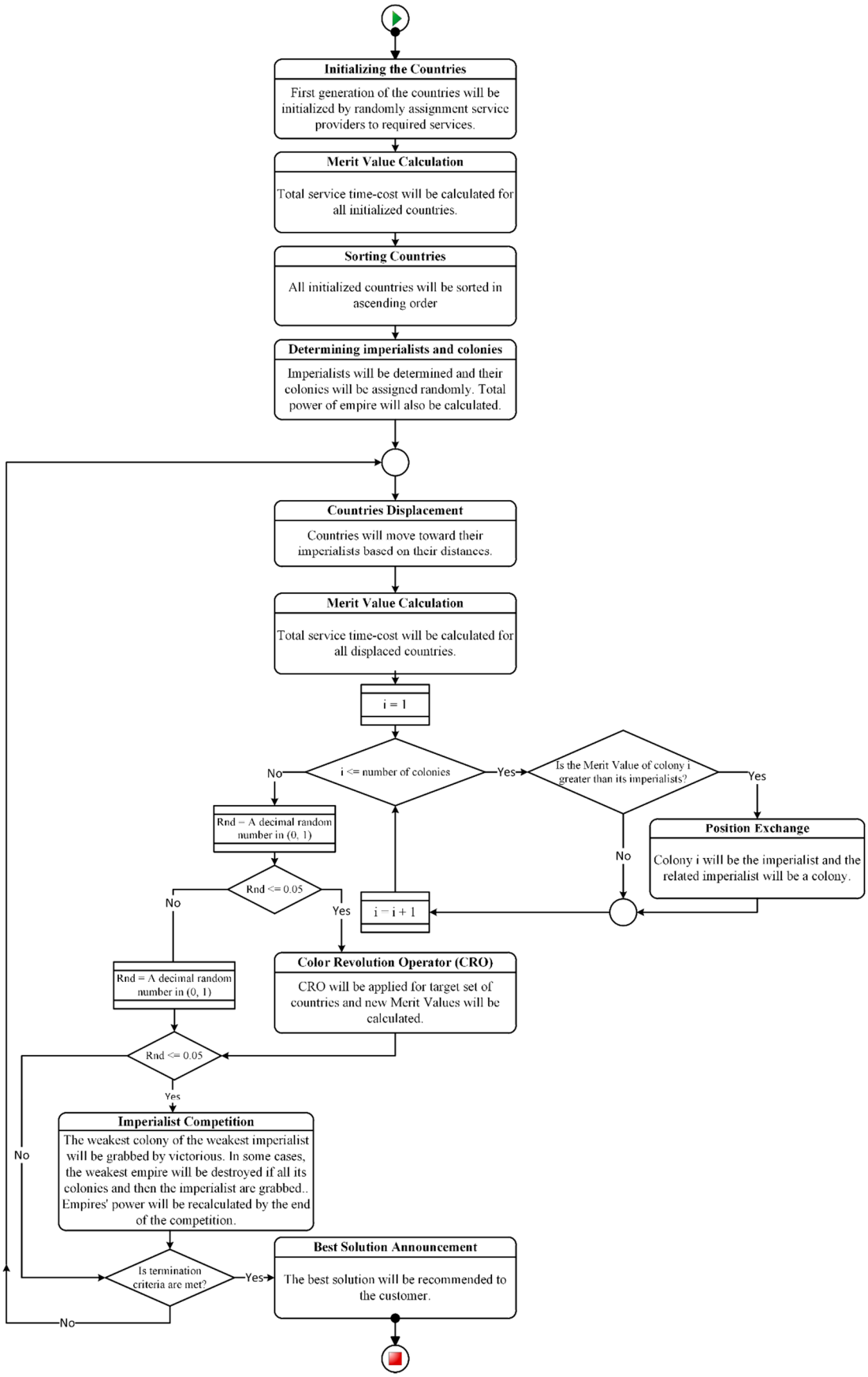


Figure 6. ICACRO flowchart.

Algorithm 2. Function ICACRO
Input: Termination criteria
Output: The best solution

```
BEGIN
Initialize the first generation of countries ;
Calculate total service time-cost (Merit Value) of all countries ;
Sort the countries ascendingly according to their Merit Value ;
Determine the Imperialists and assign their colonies randomly ;
Calculate total power of every empire ;
While the termination criteria are not met do
begin
    Move the countries toward their imperialist ;
    Calculate Merit Value of the moved countries ;
    for every colony i do
    begin
        if Merit Value of i > Merit Value of imperialist of i
        then
            i is the new imperialist of the empire and
            current imperialist will be a colony for i ;
        end if;
    end;
    Rnd = a decimal random number in (0, 1) ;
    if Rnd <= 0.5 then
        Apply CRO operator for the target set ;
        Calculate Merit Value for the CRO target set ;
    end if;
    Rnd = a decimal random number in (0, 1) ;
    if Rnd <= 0.5 then
        Apply Imperialist Competition ;
        Calculate total power of all the empires ;
    end if;
Sort the countries ascendingly according to their Merit Value ;
end;
Return the best imperialist as the best solution ;
END;
```

Figure 7. Pseudocode of ICACRO.

4. Experimental Design

4.1. ICACRO Implementation and Execution

To apply the ICACRO to the *STCOCCSC*, the algorithm is implemented in Microsoft Visual Studio C#.NET 2012. Five *STCOCCSCs* of different sizes, denoted as problems A, B, C, D and E, are generated randomly based on the WSDream-QoSDataset2 [21,58], which is a large real-world QoS dataset that includes *STs* collected from 339 service providers

and 5825 *USs*. Problems A, B, C, D and E consist of 100, 200, 300, 400 and 500 unrepeated required *USs*, respectively. In other words, the 100 required *USs* in A should be combined to prepare a requested *CS*. With respect to the description of the *STCOCCSC*, the composition algorithm should select a service among 339 similar services for each required unique service such that the sum of the service time-costs of all selected services is minimized. The classic ICA and niching PSO [49,59], which is one of the most efficient methods proposed to date, are also implemented in the same environment for comparison with the ICACRO for the five described problems. The ICACRO, the classic ICA, and niching PSO are executed 40 times independently on a PC with an Intel Core i7—3.40 GHz processor and 8 GB of RAM under identical conditions. The average value of the results of each method is employed for comparison. Because WSDream-QoSDataset2 does not provide *SCs*, these values are generated randomly by the authors such that their statistical distribution and value range follow the *ST* distribution and value range, respectively.

For a more accurate evaluation of the ICACRO and more useful comparisons between the ICACRO and the ICA and niching PSO, the algorithms are compared for each problem at two fixed points of the execution process. The first point is at iteration 1500, and the second one is at iteration 6000, which is the intended endpoint of the run. The importance of these two points can be considered from the perspective of the abilities of the algorithms to reach better solutions within a limited number of iterations and their achievements after the termination of the evolutionary process. Because 1500 is 25% of 6000, the execution time of the algorithm after 1500 iterations should be 25% of that after 6000 iterations. In this study, the solutions obtained after 1500 and 6000 iterations are termed the first proper solution (*fps*) and final solution (*fs*), respectively.

4.2. Definitions

In this section, some new analysis factors are introduced and defined, with which a more meticulous comparison can be carried out among the different algorithms.

Definition 1. *The appropriate solution at the lowest time (APLT) policy can be applied by cloud suppliers to select the best obtained solution as the final solution after a limited number of iterations, although there is still a high probability that better solutions could be achieved. This policy is useful for suppliers who prefer to respond to requests as rapidly as possible. To satisfy the APLT policy, the fps should be considered by cloud suppliers.*

Definition 2. *The optimal solution at the appropriate time (OSAT) policy is an appropriate policy for cloud suppliers who prefer to wait a reasonable amount of time to achieve closer-to-optimal solutions. Application of this policy causes slightly more delay in responding to service requests than APLT but yields more QoS-satisfying solutions. Utilization of the fs will satisfy the OSAT policy.*

Definition 3. *To assess the optimal method of applying the CRO to empower the ICA for obtaining better solutions, the time-cost consumption check (TCC) can be defined, using (15), and used for the consumption check. The TCC allows the optimality of all algorithms to be evaluated in comparison to that of one of the algorithms under consideration. For this purpose, the weakest algorithm in the experimental test is chosen as the basis, and the other investigated algorithms are compared to the basis. In (15), Best(A) and Best(basis) are the best MVs obtained using algorithm A and the basis, respectively. A larger difference between Best(A) and Best(basis) indicates a higher quality of A.*

$$TCC(\%) = \left(1 - \frac{Best(A)}{Best(basis)}\right) \times 100 \tag{15}$$

In this study, the *TCC* is employed to compare the results obtained using niching PSO, the ICA, the ICACRO-I, and the ICACRO-C for problems A to E, where niching PSO is selected as the basis.

Definition 4. *The ratio of the MV of the final obtained solution to the execution time of an algorithm is called the time utilization (TU) and can be a suitable and reliable factor for comparing the efficiencies of algorithms in solving the same problem. A larger/smaller TU value indicates the greater efficiency of an algorithm compared to that of others when solving maximization/minimization problems.*

Definition 5. *Let us define TU-based optimality (TUO) as the ratio of the TU value of an algorithm ALG1 to the TU value of another algorithm ALG2. TUO can be used to show the efficiency of ALG1 compared to that of ALG2 by taking their execution times as effective parameters.*

Definition 6. *Merit value variation (MVV) can be studied from the 1500th iteration to 6000th iteration to identify the improvement of the best solution obtained by the algorithms between these two points of execution. The MVV value can easily be calculated by finding the difference between the best MVs obtained at the two points. The MVV values of the ICACRO-I and ICACRO-C are obtained using (16) and (17), respectively. In the same way, execution time variation (ETV) can also be defined as the difference between execution times of an algorithm in the 1500th iteration and 6000th iteration. According to this definition, the ETVs of the two algorithms are calculated using (18) and (19), respectively.*

$$\Delta MC = MVV(ICACRO - C) = Cfps - Cfs \tag{16}$$

$$\Delta MI = MVV(ICACRO - I) = Ifps - Ifs \tag{17}$$

$$\Delta TC = ETV(ICACRO - C) = CfsT - CfpsT \tag{18}$$

$$\Delta TI = ETV(ICACRO - I) = IfsT - IfpsT \tag{19}$$

Definition 7. *The merit value gradient (MVG), which is defined in (20) and (21) for the two algorithms, can be considered a meaningful factor that includes the MVV and ETV of an algorithm and is used to evaluate how well an algorithm can avoid premature convergence by looking for better solutions. For minimization problems, the greater the MVG, the more convincing the evidence for allowing the algorithm to more thoroughly search through a search space. The very low MVG for small problems may indicate that a solution very close to the optimal solution is obtained.*

$$MVG(ICACRO - C) = \frac{\Delta MC}{\Delta TC} \tag{20}$$

$$MVG(ICACRO - I) = \frac{\Delta MI}{\Delta TI} \tag{21}$$

5. Comparison of Results and Discussion

As previously stated, the number of countries and their positions in the ranking are expected to play a decisive role in the efficiency and effectiveness of the CRO. To assess this role, the CRO is applied in the algorithm in two different ways. The first is the ICACRO-C, in which the CRO is utilized for all countries. The second is the ICACRO-I, in which the CRO is applied only to the imperialist countries.

As also discussed earlier, five different-sized problems with different degrees of difficulty are generated and addressed using the ICACRO-C, ICACRO-I, ICA, and niching PSO. For the first three algorithms, the number of countries initialized in the first iteration is 500. Similarly, 500 particles are generated for niching PSO to provide an equivalent comparison for all five algorithms.

Part a of Figure 8 indicates that the solutions obtained using the ICACRO-C and ICACRO-I are significantly better than those generated using the ICA and niching PSO in terms of the APLT policy. Similarly, part *b* of the same figure indicates the superiority of the ICACRO-C and ICACRO-I over the ICA and niching PSO in terms of the OSAT policy. The execution results, obtained using the two different types of the ICACRO, and their trends shown in Figure 8 also reveal that the ICACRO-C achieved a better solution than that of the ICACRO-I for problem A. Hence, based on the first evaluation, the ICACRO-C is the most suitable algorithm for solving problem A due to the better solution achieved. Despite this conclusion, the following paragraphs will show that other factors can affect the decision.

The improvements achieved by applying the *CRO* to the ICA are demonstrated by comparing the quality of the solutions in parts a and b of Figures 9–12. The total service time-cost of the best solutions, obtained using the ICACRO-C and ICACRO-I, are consistently lower than the service time-cost of the solutions generated using the ICA and niching PSO for problems B, C, D and E. The ICACRO-C and ICACRO-I obtain solutions that are significantly closer to optimal considering that these two algorithms, when evaluating the CRO, achieved optimality for different problems in which different levels of difficulty arose because of the different numbers of *USs* required. Since the ICA demonstrated better results than those of niching PSO, and because the ICA and the ICACRO differ only in the application of the CRO, it can be concluded that the high-performance level achieved resulted from the CRO efficiency and its appropriate embedding in the ICA.

On the other hand, Figure 13 presents the optimality calculated for the five investigated problems in five separate categories, each in two parts reflecting the checkpoints after 1500 and 6000 iterations. Based on the TCCs calculated based on niching PSO for problems A to E, the minimum and maximum optimality values obtained by applying the *CRO* are 39.05% and 47.86% for the *fs* and 38.68% and 46.66% for the *fps*, respectively. The averages of the optimality values achieved using the ICACRO-C, ICACRO-I, and ICA are 42.89%, 40.51%, and 21.13% for the *fs* and 46.12%, 44.11%, and 22.88% for the *fps*, respectively. The optimality achieved using the CRO in the ICA compared to that achieved using the classic ICA can be calculated by considering the ICA as the basis. Figure 14 shows that the minimum and maximum optimality values obtained for problems A to E are 20.48% and 32.80% for the *fs* and 22.04% and 31.63% for the *fps*, respectively. Hence, the averages of the optimality values obtained using the ICACRO-C and ICACRO-I are 30.30% and 27.49% for the *fs* and 27.81% and 24.72% for the *fps*, respectively. The optimality values of the ICACRO-C based on the ICACRO-I for problems A to E are presented in Figure 15.

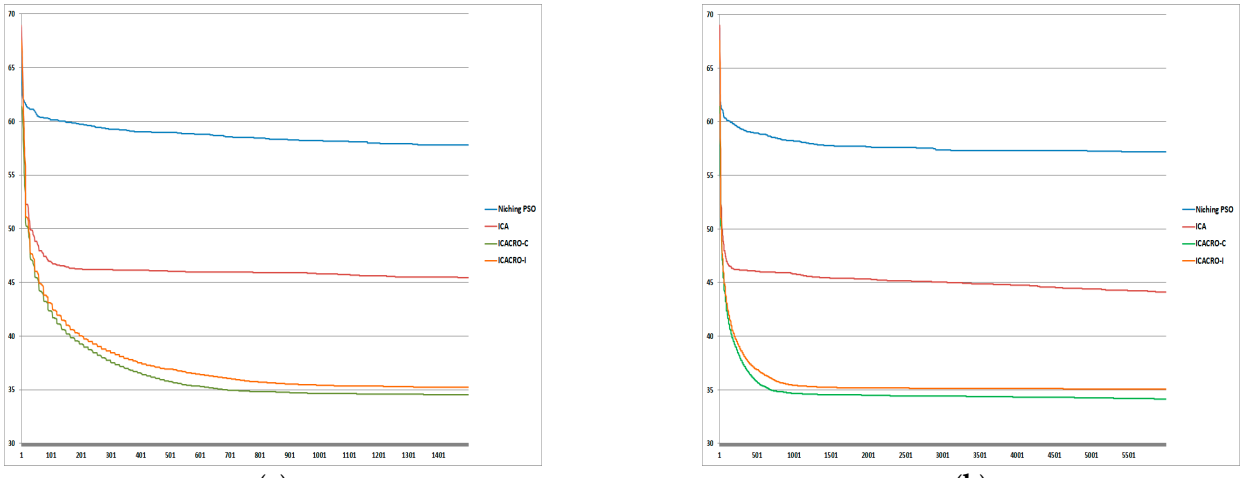
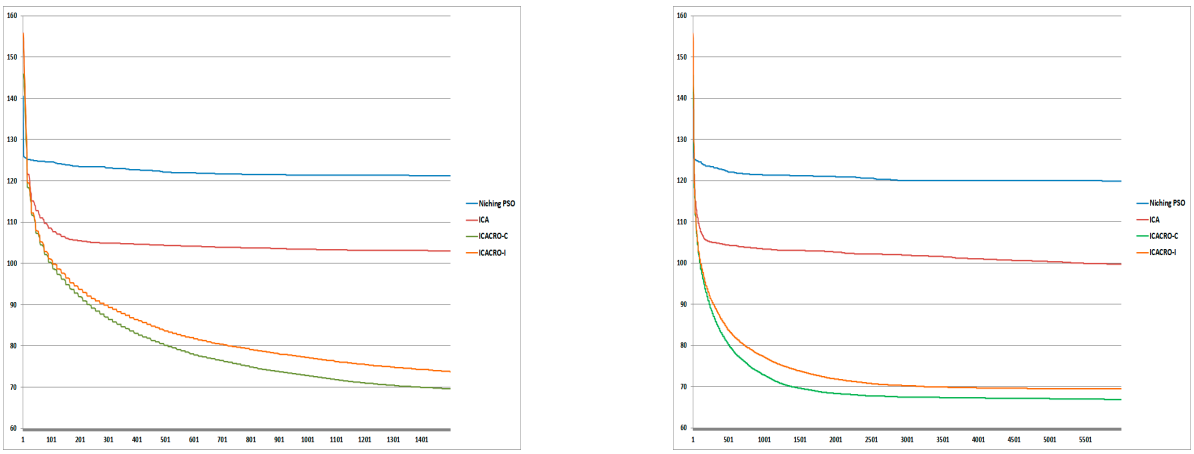
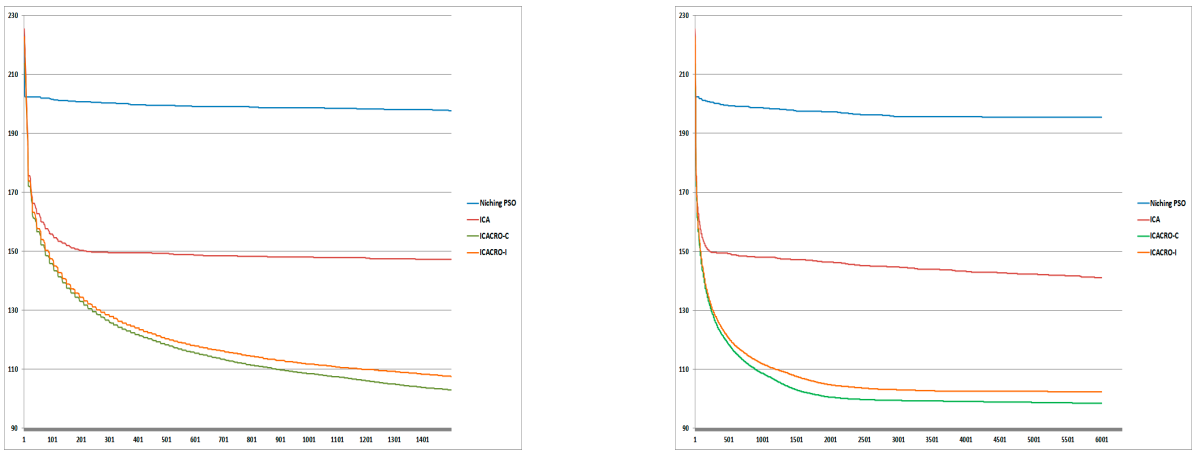


Figure 8. Comparison of the total service time-costs obtained using the four algorithms for problem A: (**a**) after 1500 iterations and (**b**) after 6000 iterations.



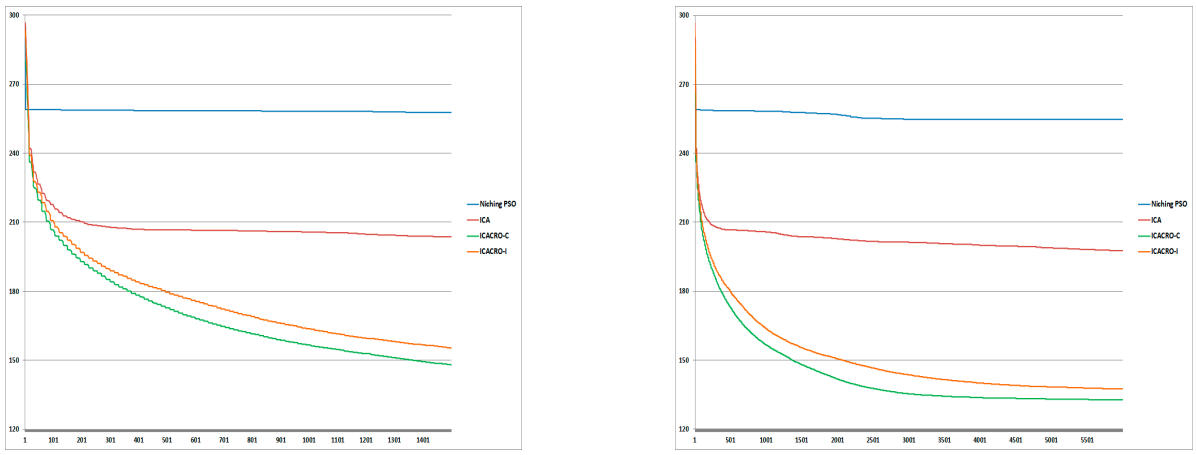
(**a**) (**b**)

Figure 9. Comparison of the total service time-costs obtained using the four algorithms for problem B: (**a**) after 1500 iterations and (**b**) after 6000 iterations.



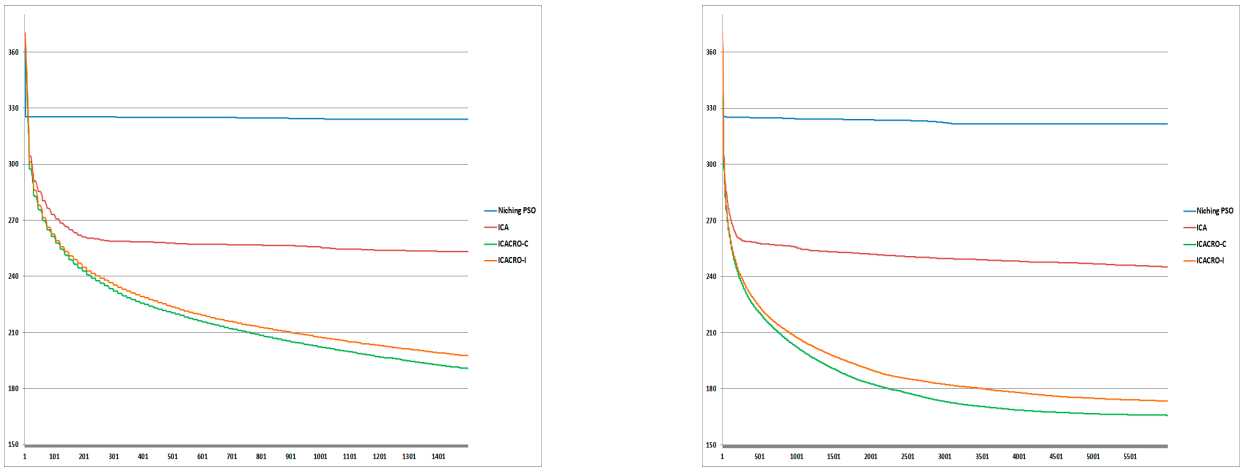
(**a**) (**b**)

Figure 10. Comparison of the total service time-costs obtained using the four algorithms for problem C: (**a**) after 1500 iterations and (**b**) after 6000 iterations.



(**a**) (**b**)

Figure 11. Comparison of the total service time-costs obtained using the four algorithms for problem D: (**a**) after 1500 iterations and (**b**) after 6000 iterations.



(**a**) (**b**)

Figure 12. Comparison of the total service time-costs obtained using the four algorithms for problem E: (**a**) after 1500 iterations and (**b**) after 6000 iterations.

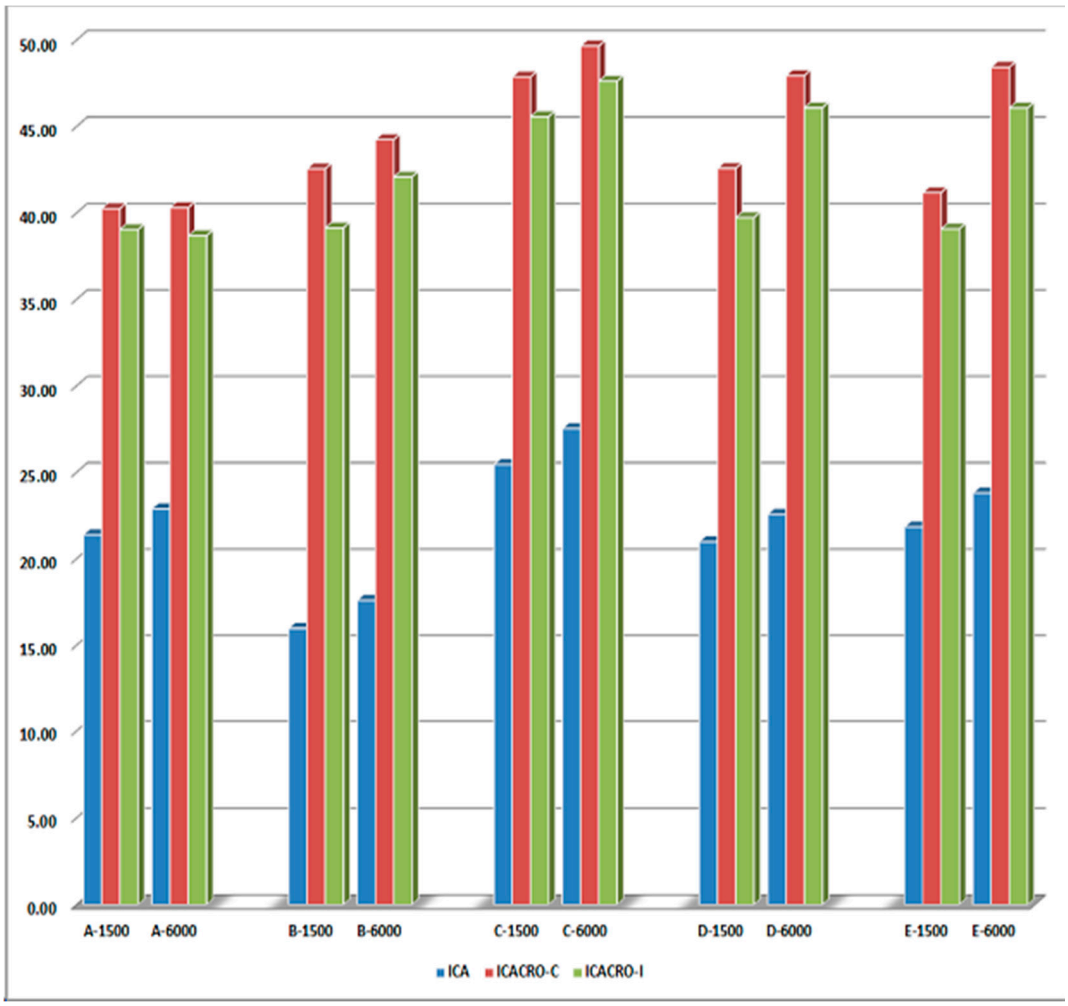


Figure 13. Optimality values of the four algorithms based on niching particle swarm optimization (PSO) for problems A–E.

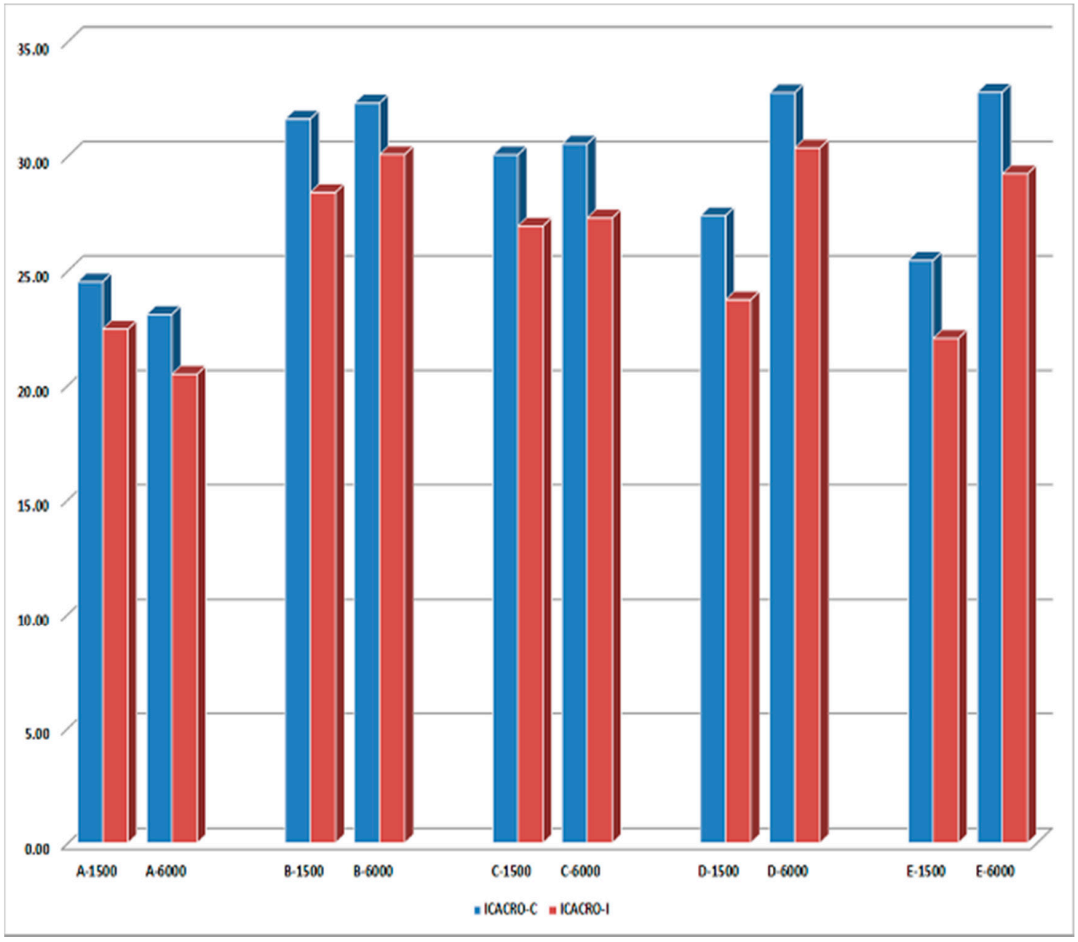


Figure 14. Optimality values of the ICACRO-C, ICACRO-I and ICA based on the ICA for problems A–E.

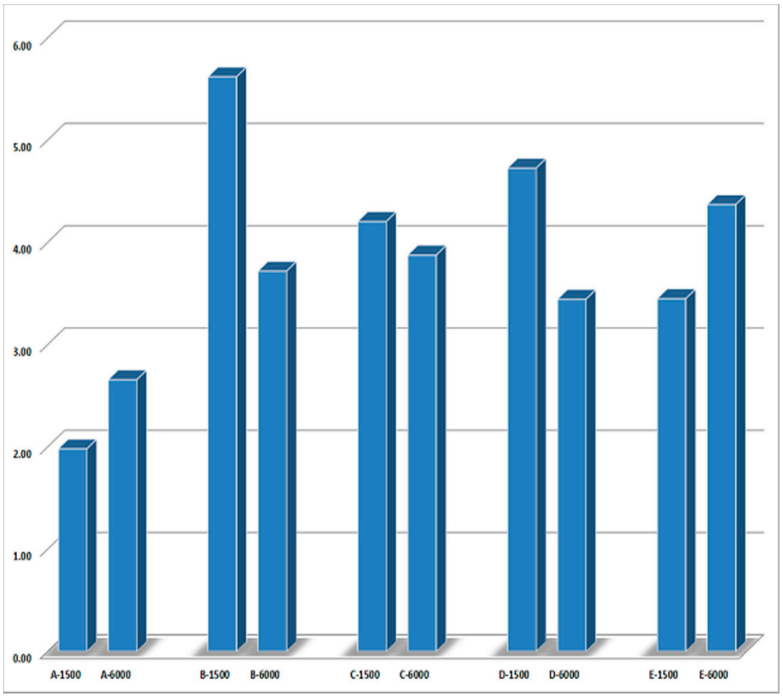


Figure 15. Optimality values of the ICACRO-C based on the ICACRO-I for problems A–E.

5.1. ICACRO-C or ICACRO-I?

The performances of the ICACRO-C and ICACRO-I are examined more closely to identify the differences between the two algorithms (see Table 2). Comparison of the best solutions obtained using the ICACRO-C and ICACRO-I in solving the different problems demonstrates that although the ICACRO-C performs better than the ICACRO-I for all investigated problems and for both APLT and OSAT policies, investigation of the execution times of the algorithms and the effect of the problem size can help us to study the performances of the two algorithms more precisely.

Table 2. *fs* and *fst* results for ICACRO-I and ICACRO-C.

	Problem A	Problem B	Problem C	Problem D	Problem E
ICACRO-C Best result after 1500 iterations (Cfps)	34.53	69.63	103.02	147.96	190.64
ICACRO-C Best result after 6000 iterations (Cfs)	34.15	66.94	98.49	132.70	165.87
ICACRO-C 1500-iteration execution time (CfpsT)	6.5 s	13.4 s	20.2 s	26.8 s	34.4 s
ICACRO-C 6000-iteration execution time (CfsT)	25.7 s	53.5 s	80.4 s	107 s	136.6 s
ICACRO-I Best result after 1500 iterations (Ifps)	35.23	73.78	107.54	155.29	197.45
ICACRO-I Best result after 6000 iterations (Ifs)	35.08	69.50	102.39	137.41	173.46
ICACRO-I 1500-iteration execution time (IfpsT)	5.9 s	12.2 s	18.7 s	24.5 s	31.1 s
ICACRO-I 6000-iteration execution time (IfsT)	23.5 s	49.1 s	74.6 s	97.5 s	123.8 s
Cfps—Cfs	0.38	2.69	4.53	15.26	24.77
Ifps—Ifs	0.15	4.28	5.15	17.88	23.99
CfsT—CfpsT	19.2 s	40.1 s	60.2 s	80.2 s	102.2 s
IfsT—IfpsT	17.6 s	36.9 s	55.9 s	73 s	92.7 s

To study the execution times of the ICACRO-C and ICACRO-I, denoted as *CfsT* and *IfsT*, respectively, the times, shown in Table 2, are recorded after execution of the algorithms for problems A to E. Figure 16 demonstrates that *CfsT* and *IfsT* increased in linear fashion according to the problem size. It also shows that the larger the problem size, the larger the difference between *CfsT* and *IfsT*. With the execution times and information regarding their trends, it is possible to compare the ICACRO-C and ICACRO-I results more precisely.

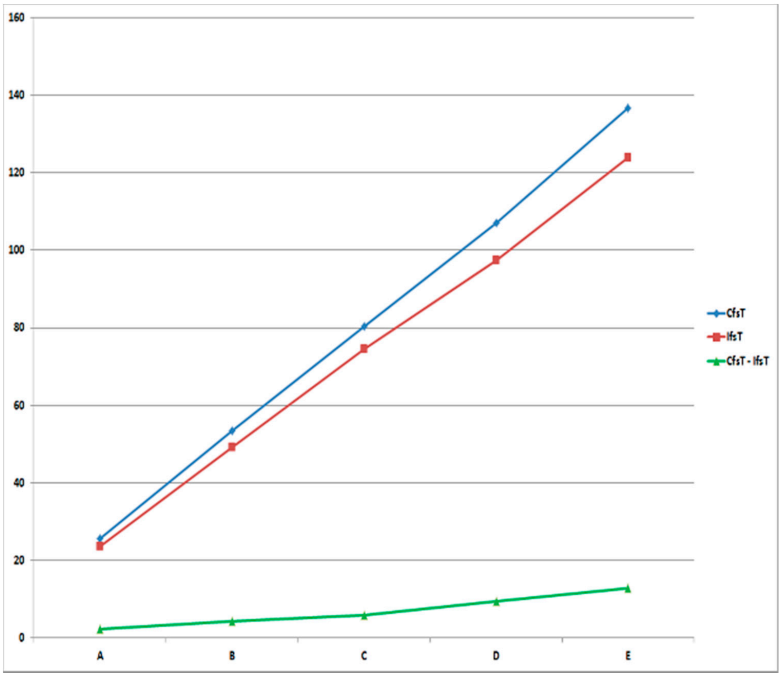


Figure 16. Execution times of the ICACRO-C and ICACRO-I for solving problems A–E.

In accordance with definitions 4 and 5, Figure 17 shows that the ICACRO-C achieved smaller TU values for all 5 problems than those achieved by the ICACRO-I. Hence, it can be concluded that with respect to the required execution time, the ICACRO-C outperforms the ICACRO-I and can achieve more proper solutions when solving the same *STCOCCSCs*. The TUO values of the five problems are also shown to be almost equal. Therefore, the same level of optimality can be achieved when solving problems of various sizes using the ICACRO-C instead of the ICACRO-I.

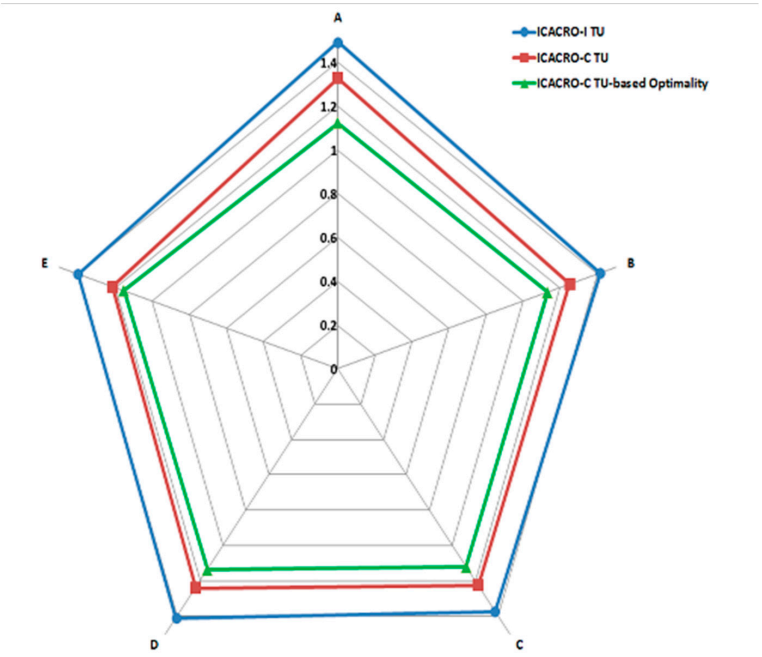


Figure 17. TU values of ICACRO-C and ICACRO-I and TU-based optimality of ICACRO-C for problems A–E.

5.2. Fps or Fs?

As mentioned in definitions 1 and 2, cloud suppliers can prepare a solution based on the APLT or OSAT policies. Table 2 shows the execution times and MVs of the ICACRO-I and ICACRO-C for problems A-E for both policies.

In accordance with definitions 6 and 7, Figure 18 demonstrates that although the MVG is very close to zero for simple problem A, it experiences an increasing trend for both investigated algorithms. Also, the ICACRO-I is shown to perform slightly better than the ICACRO-C based on the MVG investigation.

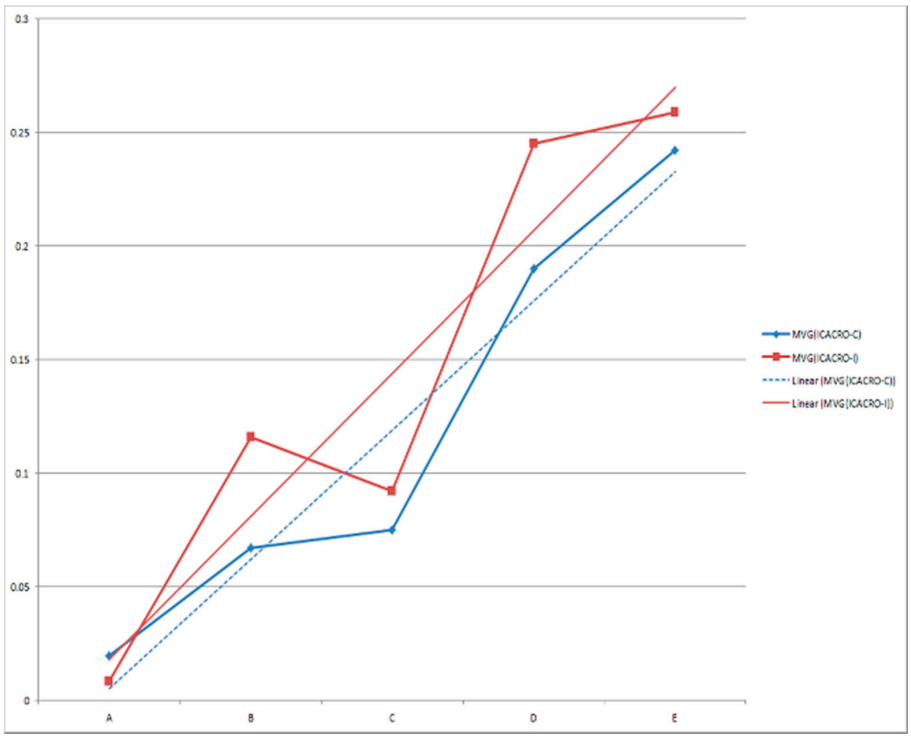


Figure 18. Merit value gradient (MVG) and trend of MVG of ICACRO-C and ICACRO-I for problems A–E.

Therefore, it can be concluded that for problems with a small number of required simple services, obtaining the fps seems to be sufficient and that execution of the algorithms to reach the fs will not lead to a significant improvement in the best obtained solution to the problem. Nevertheless, as the number of required simple services increases, the difference becomes more significant.

5.3. Performance Statistical Test

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

A repeated measures analysis of variance, with the Greenhouse-Geisser correction [60], is conducted to consider the difference in mean total service time-cost obtained using the four algorithms. The result of the repeated measures analysis of variance indicates that the mean *STs* of the four investigated algorithms are statistically significantly different for all investigated problems (see Table 3). Pairwise comparisons with the Bonferroni correction [61–63] also reveal that the two different types of ICACRO obtained a significantly lower mean total service time-cost than that of the ICA and niching PSO for all four investigated problems.

Table 3. Results of repeated measures ANOVA.

		df	Mean Square	F	Sig.
Problem A	Between Groups	1.234	1654453.748	1064352.466	<0.001
	Error	7400.915	1.554		
Problem B	Between Groups	1.079	9596518.233	258544.375	<0.001
	Error	6473.570	37.117		
Problem C	Between Groups	1.082	31014381.517	441298.360	<0.001
	Error	6493.645	70.280		
Problem D	Between Groups	1.052	46227282.898	199913.106	<0.001
	Error	6309.354	231.237		
Problem E	Between Groups	1.040	72555514.778	195415.062	<0.001
	Error	6239.068	371.289		

Furthermore, the ICA results are also significantly better than the results of niching PSO according to the results shown in Table 4. To statistically analyze the behavior of the ICACRO-C and ICACRO-I, Table 4 is provided. The table shows that there is a significant difference between the results obtained using the two algorithms. Further inspection of the results suggests that the mean difference between these two algorithms is significant and increases as the problem size increases. Hence, based on the trends of the algorithm results, it can be concluded that the larger the problem size, the more efficient the performance of the ICACRO-C compared to that of the ICACRO-I.

Table 4. Results of Bonferroni pairwise comparisons.

	(I) Algorithm	(J) Algorithm	Mean Difference (J—I)	Std. Error	Sig.
Problem A	ICACRO-C	Niching PSO	−22.836	0.018	<0.001
	ICACRO-I	Niching PSO	−22.037	0.019	<0.001
	ICA	Niching PSO	−12.575	0.009	<0.001
	ICACRO-C	ICA	−10.261	0.015	<0.001
	ICACRO-I	ICA	−9.462	0.016	<0.001
	ICACRO-C	ICACRO-I	−0.798	0.002	<0.001
Problem B	ICACRO-C	Niching PSO	−50.116	0.089	<0.001
	ICACRO-I	Niching PSO	−47.144	0.088	<0.001
	ICA	Niching PSO	−18.518	0.026	<0.001
	ICACRO-C	ICA	−31.598	0.072	<0.001
	ICACRO-I	ICA	−28.626	0.071	<0.001
	ICACRO-C	ICACRO-I	−2.972	0.01	<0.001
Problem C	ICACRO-C	Niching PSO	−92.806	0.127	<0.001
	ICACRO-I	Niching PSO	−89.269	0.121	<0.001
	ICA	Niching PSO	−51.471	0.044	<0.001
	ICACRO-C	ICA	−41.335	0.098	<0.001
	ICACRO-I	ICA	−37.798	0.091	<0.001
	ICACRO-C	ICACRO-I	−3.537	0.009	<0.001
Problem D	ICACRO-C	Niching PSO	−111.665	0.221	<0.001
	ICACRO-I	Niching PSO	−104.956	0.218	<0.001
	ICA	Niching PSO	−53.594	0.061	<0.001
	ICACRO-C	ICA	−58.071	0.178	<0.001
	ICACRO-I	ICA	−51.362	0.174	<0.001
	ICACRO-C	ICACRO-I	−6.709	0.019	<0.001
Problem E	ICACRO-C	Niching PSO	−139.396	0.293	<0.001
	ICACRO-I	Niching PSO	−132.027	0.27	<0.001
	ICA	Niching PSO	−71.467	0.086	<0.001
	ICACRO-C	ICA	−67.928	0.225	<0.001
	ICACRO-I	ICA	−60.56	0.2	<0.001
	ICACRO-C	ICACRO-I	−7.369	0.026	<0.001

6. Conclusions and Directions for Future Research

Designing effective operators for EAs can enhance the searching capabilities of the algorithms provided that they are utilized appropriately. Furthermore, the designed operators should be developed based on the same origins used as inspiration in the design of the algorithms. Hence, in this paper, a new operator termed the CRO is designed and applied to the ICA based on the color revolutions, a sociopolitical movement that has occurred in some countries. Application of the CRO significantly increases the ability of the ICA to achieve closer-to-optimal solutions when considering five problems of different size. Two types of the proposed algorithm (ICACRO-C and ICACRO-I) considerably outperformed niching PSO and the ICA in terms of the APLT and OSAT policies. The ICACRO-C is experimentally and statistically evaluated to generally be more efficient than the ICACRO-I in terms of execution time and the obtained results.

The CRO can be improved by providing a dynamic probability rate and optimum number of target countries in future research. This can be done by evaluating the trends of changes in the MV and convergence speed of solutions. The CRO can also be made more effective provided that service providers are categorized precisely based on all possible QoS parameters.

Author Contributions: methodology, A.J.; software, A.J. and N.K.N.; validation, E.A.S. and Z.O.; formal analysis, A.J.; investigation, A.J. and N.K.N.; resources, A.J., N.K.N. and E.A.S.; Writing—Original draft preparation, A.J.; Writing—review and editing, A.J., N.K.N., E.A.S. and Z.O.; visualization, A.J. and N.K.N.; supervision, E.A.S. and Z.O. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Ministry of Higher Education Malaysia Grant: FRGS/1/2014/ICT07/UKM/02/1, the Research University Grant: DIP-2018-041 and Research Incentive Grant: GPP-2020-032.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The results of the experimental tests can be found at http://www.scidb.cn/en/s/pj6b2uq.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hamdaqa, M.; Tahvildari, L. Cloud Computing Uncovered: A Research Landscape. *Adv. Comput.* **2012**, *86*, 41–85. [CrossRef]
2. Mell, P.; Grance, T. *The NIST Definition of Cloud Computing*; National Institute of Standards and Technology, U.S. Department of Commerce: Gaithersburg, MA, USA, 2011; pp. 1–7.
3. Vaquero, L.M.; Rodero-Merino, L.; Caceres, J.; Lindner, M. A break in the clouds: Towards a cloud definition. *Comput. Commun. Rev.* **2008**, *39*, 50–55. [CrossRef]
4. Jula, A.; Sundararajan, E.; Othman, Z. Cloud computing service composition: A systematic literature review. *Expert Syst. Appl.* **2014**, *41*, 3809–3824. [CrossRef]
5. Ding, S.; Yang, S.; Zhang, Y.; Liang, C.; Xia, C. Combining QoS prediction and customer satisfaction estimation to solve cloud service trustworthiness evaluation problems. *Knowl. Based Syst.* **2014**, *56*, 216–225. [CrossRef]
6. Yousefipour, A.; Rahmani Amir, M.; Jahanshahi, M. Energy and cost-aware virtual machine consolidation in cloud computing. *Softw. Pract. Exp.* **2018**, *48*, 1758–1774. [CrossRef]
7. Yuan, Y.; Zhang, W.; Zhang, X.; Zhai, H. Dynamic Service Selection Based on Adaptive Global QoS Constraints Decomposition. *Symmetry* **2019**, *11*, 403. [CrossRef]
8. Lu, P.; Zhang, L.; Liu, X.; Yao, J.; Zhu, Z. Highly efficient data migration and backup for big data applications in elastic optical inter-data-center networks. *IEEE Netw.* **2015**, *29*, 36–42. [CrossRef]
9. Fei, T.; Yuanjun, L.; Lida, X.; Lin, Z. FC-PACO-RM: A Parallel Method for Service Composition Optimal-Selection in Cloud Manufacturing System. *IEEE Trans. Ind. Inform.* **2013**, *9*, 2023–2033. [CrossRef]
10. Yu, T.; Lin, K.-J. Service selection algorithms for composing complex services with multiple qos constraints. In Proceedings of the Third International Conference on Service-Oriented Computing, Amsterdam, The Netherlands, 12–15 December 2005; pp. 130–143.

11. Anselmi, J.; Ardagna, D.; Cremonesi, P. A QoS-based selection approach of autonomic grid services. In Proceedings of the 2007 Workshop on Service-Oriented Computing Performance: Aspects, Issues, and Approaches, Monterey, CA, USA, 25 June 2007; pp. 1–8.
12. Li, J.; Zheng, X.-L.; Chen, S.-T.; Song, W.-W.; Chen, D. An efficient and reliable approach for quality-of-service-aware service composition. *Inf. Sci.* **2014**, *269*, 238–254. [CrossRef]
13. Mijumbi, R.; Serrat, J.; Gorricho, J.; Bouten, N.; Turck, F.D.; Boutaba, R. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 236–262. [CrossRef]
14. Ocampo, A.F.; Gil-Herrera, J.; Isolani, P.H.; Neves, M.C.; Botero, J.F.; Latré, S.; Zambenedetti, L.; Barcellos, M.P.; Gaspary, L.P. Optimal Service Function Chain Composition in Network Functions Virtualization. In Proceedings of the 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, Zurich, Switzerland, 10–13 July 2017; pp. 62–76.
15. Wang, M.; Cheng, B.; Li, B.; Chen, J. Service Function Chain Composition and Mapping in NFV-Enabled Networks. In Proceedings of the 2019 IEEE World Congress on Services (SERVICES), Milan, Italy, 8–13 July 2019; pp. 331–334.
16. Wada, H.; Suzuki, J.; Yamano, Y.; Oba, K. A Multiobjective Optimization Framework for SLA-Aware Service Composition. *IEEE Trans. Serv. Comput.* **2012**, *5*, 358–372. [CrossRef]
17. Hayyolalam, V.; Pourhaji Kazem, A.A. A systematic literature review on QoS-aware service composition and selection in cloud environment. *J. Netw. Comput. Appl.* **2018**, *110*, 52–74. [CrossRef]
18. Jula, A.; Othman, Z.; Sundararajan, E. A Hybrid Imperialist Competitive-Gravitational Attraction Search Algorithm to Optimize Cloud Service Composition. In Proceedings of the 2013 IEEE Workshop on Memetic Computing (MC), Singapore, 15–19 April 2013; pp. 37–43.
19. Wang, Z.-S.; Lee, J.; Song, C.G.; Kim, S.-J. Efficient Chaotic Imperialist Competitive Algorithm with Dropout Strategy for Global Optimization. *Symmetry* **2020**, *12*, 635. [CrossRef]
20. Atashpaz-Gargari, E.; Lucas, C. Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, Singapore, 25–28 September 2007; pp. 4661–4667.
21. Zibin, Z.; Yilei, Z.; Lyu, M.R. Distributed QoS Evaluation for Real-World Web Services. In Proceedings of the 8th IEEE International Conference on Web Services (ICWS 2010), Miami, FL, USA, 5–10 July 2010; pp. 83–90.
22. Kofler, K.; ul Haq, I.; Schikuta, E. A Parallel Branch and Bound Algorithm for Workflow QoS Optimization. In Proceedings of the ICPP 2009, International Conference on Parallel Processing, Vienna, Austria, 22–25 September 2009; pp. 478–485.
23. Moura, L.D.; Bjørner, N. Satisfiability modulo theories: Introduction and applications. *Commun. ACM* **2011**, *54*, 69–77. [CrossRef]
24. Worm, D.; Zivkovic, M.; van den Berg, H.; van der Mei, R. Revenue maximization with quality assurance for composite web services. In Proceedings of the 2012 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2012), Taipei, Taiwan, 17–19 December 2012; pp. 1–9.
25. Shangguang, W.; Qibo, S.; Fangchun, Y. Towards Web Service selection based on QoS estimation. *Int. J. Web Grid Serv.* **2010**, *6*, 424–443. [CrossRef]
26. Zhu, Y.; Li, W.; Luo, J.; Zheng, X. A novel two-phase approach for QoS-aware service composition based on history records. In Proceedings of the 2012 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2012), Taipei, Taiwan, 17–19 December 2012; pp. 1–8.
27. Qi, Y.; Bouguettaya, A. Efficient Service Skyline Computation for Composite Service Selection. *IEEE Trans. Knowl. Data Eng.* **2013**, *25*, 776–789. [CrossRef]
28. Hossain, M.S.; Hassan, M.M.; Al Qurishi, M.; Alghamdi, A. *Resource Allocation for Service Composition in Cloud-Based Video Surveillance Platform*; IEEE: New York, NY, USA, 2012; pp. 408–412. [CrossRef]
29. Zeng, C.; Guo, X.A.; Ou, W.J.; Han, D. Cloud Computing Service Composition and Search Based on Semantic. In *Lecture Notes in Computer Science*; Jaatun, M.G., Zhao, G., Rong, C., Eds.; Springer: Berlin, Germany, 2009; Volume 5931, pp. 290–300.
30. Huang, J.; Liu, Y.; Yu, R.; Duan, Q.; Tanaka, Y. Modeling and Algorithms for QoS-Aware Service Composition in Virtualization-Based Cloud Computing. *IEICE Trans. Commun.* **2013**, *96*, 10–19. [CrossRef]
31. Zhou, X.; Mao, F. A Semantics Web Service Composition Approach Based on Cloud Computing. In Proceedings of the 2012 Fourth International Conference on Computational and Information Sciences (ICCIS 2012), Chongqing, China, 17–19 August 2012; pp. 807–810.
32. Karim, R.; Chen, D.; Miri, A. An End-to-End QoS Mapping Approach for Cloud Service Selection. In Proceedings of the 2013 IEEE Ninth World Congress on Services (SERVICES), Santa Clara, CA, USA, 28 June–3 July 2013; pp. 341–348.
33. Barzegar, S.; Davoudpour, M.; Meybodi, M.R.; Sadeghian, A.; Tirandazian, M. Formalized learning automata with adaptive fuzzy coloured Petri net; an application specific to managing traffic signals. *Sci. Iran.* **2011**, *18*, 554–565. [CrossRef]
34. Zhao, H.; Gao, W.; Deng, W.; Sun, M. Study on an Adaptive Co-Evolutionary ACO Algorithm for Complex Optimization Problems. *Symmetry* **2018**, *10*, 104. [CrossRef]
35. Jaddi, N.S.; Alvankarian, J.; Abdullah, S. Kidney-inspired algorithm for optimization problems. *Commun. Nonlinear Sci. Numer. Simul.* **2017**, *42*, 358–369. [CrossRef]
36. He, J.; Lin, G.M. Average Convergence Rate of Evolutionary Algorithms. *IEEE Trans. Evol. Comput.* **2016**, *20*, 316–321. [CrossRef]

37. Vesterstrom, J.; Thomsen, R. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2004, Portland, OR, USA, 19–23 June 2004; pp. 1980–1987.
38. Bäck, T.; Fogel, D.B.; Michalewicz, Z. *Handbook of Evolutionary Computation*; IOP Publishing Ltd.: Bristol, UK, 1997; p. 988.
39. Naseri, N.K.; Sundararajan, E.A.; Ayob, M.; Jula, A. Smart Root Search (SRS): A Novel Nature-Inspired Search Algorithm. *Symmetry* **2020**, *12*, 2025. [CrossRef]
40. Knuth, D.E. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*; Pearson Education: Tamil Nadu, India, 2011.
41. Abonyi, J.; Akerkar, R.; Alavi, A.H.; Arango, C.; Aydogdu, I.; Brest, J.; Cai, X.; Cordeiro, J.; Cortés, P.; Costa, K.A.P.; et al. List of Contributors. In *Swarm Intelligence and Bio-Inspired Computation*; Yang, X.-S., Cui, Z., Xiao, R., Gandomi, A.H., Karamanoglu, M., Eds.; Elsevier: Oxford, UK, 2013; pp. xv–xviii. [CrossRef]
42. Zhang, X.; Dou, W. Preference-Aware QoS Evaluation for Cloud Web Service Composition Based on Artificial Neural Networks. In *Web Information Systems and Mining*; Wang, F., Gong, Z., Luo, X., Lei, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6318, pp. 410–417.
43. Wu, Q.; Zhang, M.; Zheng, R.; Lou, Y.; Wei, W. A QoS-Satisfied Prediction Model for Cloud-Service Composition Based on a Hidden Markov Model. *Math. Probl. Eng.* **2013**, *2013*, 7. [CrossRef]
44. Lie, Q.; Yan, W.; Orgun, M.A. Cloud Service Selection Based on the Aggregation of User Feedback and Quantitative Performance Assessment. In Proceedings of the 2013 IEEE International Conference on Services Computing (SCC), Santa Clara, CA, USA, 28 June–3 July 2013; pp. 152–159.
45. Ye, Z.; Zhou, X.; Bouguettaya, A. Genetic Algorithm Based QoS-Aware Service Compositions in Cloud Computing. In *Database Systems for Advanced Applications*; Yu, J., Kim, M., Unland, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6588, pp. 321–334.
46. Klein, A.; Ishikawa, F.; Honiden, S. Towards network-aware service composition in the cloud. In Proceedings of the 21st International Conference on World Wide Web, Lyon, France, 16–20 April 2012; pp. 959–968.
47. Ludwig, S.A. Clonal selection based genetic algorithm for workflow service selection. In Proceedings of the 2012 IEEE Congress on Evolutionary Computation (CEC), Brisbane, QLD, Australia, 10–15 June 2012; pp. 1–7.
48. Wang, S.G.; Sun, Q.B.; Zou, H.; Yang, F.C. Particle Swarm Optimization with Skyline Operator for Fast Cloud-based Web Service Composition. *Mob. Netw. Appl.* **2013**, *18*, 116–121. [CrossRef]
49. Liao, J.X.; Liu, Y.; Wang, J.Y.; Zhu, X.M. Service Composition Based on Niching Particle Swarm Optimization in Service Overlay Networks. *KSII Trans. Internet Inf. Syst.* **2012**, *6*, 1106–1127. [CrossRef]
50. Wang, Y.W. *Application of Chaos Ant Colony Algorithm in Web Service Composition Based on QoS*; IEEE Computer Soc: Los Alamitos, CA, USA, 2009; pp. 225–227. [CrossRef]
51. Yang, Y.; Mi, Z.; Sun, J. Game theory based iaas services composition in cloud computing environment. *Adv. Inf. Sci. Serv. Sci.* **2012**, *4*, 238–246.
52. Jula, A.; Othman, Z.; Sundararajan, E. Imperialist competitive algorithm with PROCLUS classifier for service time optimization in cloud computing service composition. *Expert Syst. Appl.* **2015**, *42*, 135–145. [CrossRef]
53. Han, J.; Kamber, M.; Pei, J. *Data Mining: Concepts and Techniques*; Morgan Kaufmann Publishers Inc.: Burlington, MA, USA, 2011; p. 696.
54. Bahrami, H.; Faez, K.; Abdechiri, M. Imperialist Competitive Algorithm Using Chaos Theory for Optimization (CICA). In Proceedings of the 2010 12th International Conference on Computer Modelling and Simulation (UKSim), Brisbane, Australia, 24–26 March 2010; pp. 98–103.
55. Zarandi, M.H.F.; Zarinbal, M.; Ghanbari, N.; Turksen, I.B. A new fuzzy functions model tuned by hybridizing imperialist competitive algorithm and simulated annealing. Application: Stock price prediction. *Inf. Sci.* **2013**, *222*, 213–228. [CrossRef]
56. Zherebkin, M. In search of a theoretical approach to the analysis of the 'Colour revolutions': Transition studies and discourse theory. *Communist Post-Communist Stud.* **2009**, *42*, 199–216. [CrossRef]
57. Marples, D.R. Color revolutions: The Belarus case. *Communist Post-Communist Stud.* **2006**, *39*, 351–364. [CrossRef]
58. Jula, A.; Nilsaz, H.; Sundararajan, E.; Othman, Z. A new dataset and benchmark for cloud computing service composition. In Proceedings of the 2014 5th International Conference on Intelligent Systems, Modelling and Simulation, Langkawi, Malaysia, 27–29 January 2014; pp. 83–86.
59. Liao, J.X.; Liu, Y.; Zhu, X.M.; Xu, T.; Wang, J.Y. Niching Particle Swarm Optimization Algorithm for Service Composition. In *2011 IEEE Global Telecommunications Conference*; IEEE: Houston, TX, USA, 2011.
60. Abdi, H. *Greenhouse-Geisser Correction. Encyclopedia of Research Design*; SAGE Publications, Inc.: Thousand Oaks, CA, USA, 2010; pp. 545–549. [CrossRef]
61. Nakagawa, S. A farewell to Bonferroni: The problems of low statistical power and publication bias. *Behav. Ecol.* **2004**, *15*, 1044–1045. [CrossRef]
62. Cabin, R.; Mitchell, R. To Bonferroni or not to Bonferroni: When and how are the questions. *Bull. Ecol. Soc. Am.* **2000**, *81*, 246–248. [CrossRef]
63. Holm, S. A Simple Sequentially Rejective Multiple Test Procedure. *Scand. J. Stat.* **1979**, *6*, 65–70. [CrossRef]