International Journal of Signal Processing, Image Processing and Pattern Recognition
Vol.9, No.4 (2016), pp.443-454
http://dx.doi.org/10.14257/ijsip.2016.9.4.39

Selection and Configuration Optimization for Customizable Cloud Services

Hongxia Zhang, Fei Wang, Jiuyun Xu and Jiashu Guo

College of computer and communication engineering, China University of Petroleum
Qingdao 266580, China

Abstract

Cloud services is a typical trait of cloud computing. It allows tenants rent customizable services transparently, and guarantees substantial cost savings for the providers. With the significant number of cloud services, a key challenge when providing customizable cloud services is how to manage those services and find an optimized configuration service rapidly that maximizes tenants' preferences, subject to resource constraints. To help address this challenge, this paper introduces a novel approach to support the modeling and configuration of customizable cloud services. In particular, we propose a customizable service selecting and configuring optimization approach based on a heuristic approach, which can quickly derive an optimized valid service selection by evaluation different configurations that optimize tenants' preferences under resource limitations. The scalability and performance of the algorithms is investigated in detail. And the obtained experimental results demonstrate the feasibility of the proposed method.

Keywords*: Cloud service; customizable service configuration; cloud computing; Binary Particle Swarm Optimization*

1. Introduction

With the fast development of cloud computing, among other benefits, cloud services provide computational & data resources dynamically, which allow applications to react faster to sudden changes in demands, while saving cost and decreasing power consumption [1]. Recently, a number of companies are deciding to utilize cloud services directly for building and deploying new systems [2].

Nowadays, many existing cloud applications only deliver a limited amount of customizability, often using a one-size-fits-all approach or limiting customizations to fit for changes. However, applications in some areas such as document processing, medical information management, and medical communication systems, applications must be high-variability, as different tenants often have similar needs, especially with different non-functional requirements. High-variability applications require different service configurations for specific requirement, which is the process of selecting proper cloud services with different quality [3]. In order to offer cost-effective solutions to multiple tenants in the cloud environment, customizable cloud services are necessary, *i.e.,* the ability to satisfy multiple tenants simultaneously based on one customizable cloud services [4]. Currently such customizable cloud services are often developed on an ad-hoc basis. This however poses difficulties concerning the management of these customizations and ensures the correctness of these customizations [5].

Using Software Product Line Engineering (SPLE) [6], this issue can be addressed. Feature model in SPLE is widely used to build customizable applications. In feature model, features are the basic units for building applications, and by selecting and deselecting features, different application variants can be created. Most approaches in

ISSN: 2005-4254 IJSIP

SPLE have focused on configuring products using features with static configuration of variation points. D. Benavides *et al.* [7] summarize the configuration method based on uses' functional requirements and guarantee the soundness of results. On the other hand, Sayyad *et al.* [8] present search-based software engineering to solve the SPL configuration problem on the value of user preferences, which only focus on quality preferences, but ignore the relationship among features. J. White *et al.* [9] present Filtered Cartesian Flattening method, which is Polynomial complexity. However, in cloud environment, facing a larger number of cloud services, configuration based on tenants preferences is much complex, and the tenants pay much attention on the efficiency of providing services. Therefore, a great challenge is how to provide services rapidly, that is, to find an optimized configuration rapidly that maximizes tenants' preferences satisfaction, subject to resource constraints.

In this paper, we focus on the management of customizable cloud services and the design of algorithms for configuring high-variability applications on cloud infrastructure. The applications are built by composing customizable services from a set of cloud services. We first build the model of customizable cloud services based on feature model, then service configuration problem and optimization objective are designed. We design a customizable service selecting and configuring optimization approach (CSSCOA) to solve the problem, CSSCOA can quickly derive an optimized service selection by evaluating different configuration both satisfy tenants' preferences under resource limitations. Besides, a feature amend function is been introduced to convert an optional service set to satisfy the constraints of services, which guarantee the correctness of the solution. The scalability and performance of the algorithm is investigated in detail.

The rest of this paper is organized as follows: Section II illustrates the motivating example and problem analysis. Section III introduces our extended feature model, configuration model and objective. Section IV proposes the customizable service optimization configuration approach. Section V presents the experimental results. Section VI concludes the paper.

2. Motivathing Example and Problem Analysis

Fig.1 shows an example of an application migrated into a cloud, which provides travel booking service. It has three tenants: 1) *Webjet*, a travel booking company offering airline ticket booking, car rental, hotel booking, and payment based in Australia; 2) *Rail Plus*, the leading dedicated rail trip specialist general sales agent through Australia & New Zealand, which provide the train ticket booking, hotel booking and payment service; and 3) *P&O Cruise*, Australia and New Zealand's leading cruise line, offering cruise ticket booking, insurance quote and payment [10].The cloud application serves the three travel agents by processing their customers' requests. A customer enters their travel requirements, *e.g.*, city of departure, destination, departure date, return date, preferred type of rental car, *etc.* In response to the request, the application returns a list of candidate travel plans for the customer to book and pay. These three tenants configure the customizable travel booking service from the candidate services and rent them. But, sometimes there exists the dependencies between the candidate services, for example, if the booking service is rented, then the pay service must be rented.

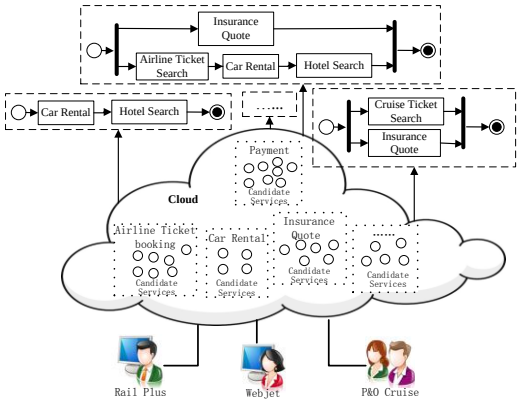


Figure 1. A Travel Booking Application in a Cloud

On the other hand, these travel agents usually have different multi-dimensional requirements for the quality of the application. For instance, P&O Cruise pays much attention to the security of the application despite a high price, Webjet requires a very fast response time of the application, and Rail Plus is more concerned about minimizing the cost of renting the application. Therefore, a set of services must be selected from the candidate services in the cloud to perform the application that serves the travel agents with satisfactory quality and achieves the application provider's optimization goal. Similarly to other researches [11-12], we assume that alternative functionally-equivalent services are available and can be categorized based on their functionalities.

3. Cloud Services Configuration Problem

In this section, we first describe the customizable service model, and then present the optimization model adopted in this research.

3.1. Extended Feature Model

In this research, we adopt the feature model to represent the customizable cloud services, which is used to model the variability of an application. The customizability of the application is represented by a collection of features. A feature can (1) capture high-level variability, such as variations in end-user functionality, or (2) document low-level variability, such as software variability (*e.g.*, variations in software implementation) [13]. Each complete architectural variant of the SPL is described as a set of selected features.

In the basic form, a feature model is a tree of features which are divided in grouped features and solitary features.

Types of solitary feature contain mandatory and optional, where

• *Mandatory (a, b)*: If a feature *a* is included, the feature *b* must be included as well.
• *Optional (a, b)*: If a feature *a* is included, the feature *b* may be included. Conversely, the feature *b* must not be included if a is not included.

The relationship of a grouped feature includes and-groups, or-groups and xor-groups, where

• *And (a,S)*: If a feature *a* is included, all features contained in the set *S* must be included. If *a* is not included, none of the features in *S* may be included.

• *Or (a, S)*: If a feature *a* is included, at least one of the features contained in the set *S* must be included. If *a* is not included, none of the features in *S* may be included.

• *Xor(a, S)*: If a feature *a* is included exactly one of the features contained in the set *S* must be included. If *a* is not included, none of the features in *S* may be included.

The relationship between features includes imply and exclude relationships:

• *Excludes (a, b)*: If a feature *a* is included, the feature *b* must not be included and vice versa.

• *Requires (a, b)*: A feature *a* may only be included if the feature *b* is included as well.

In cloud environment, we use features to express cloud services, which not only have functional property, but also have non-functional properties, such as price, performance, availability, *etc.* Therefore, extended feature model (EFM) is presented to express the customizable cloud services in cloud computing. Because we only focus on the functional, non-functional and configured variants described by the feature model, skipping the details, we informally define the extended feature model in Definition 1.

Definition 1 Extended feature model is a 5-tuple $EFM = (FD, F_L, QP, QR, R)$, where *FD* is a feature model, F_L is the set of atom services, $QP \subseteq F_L \times QR$ is the set of nonfunctional values of atom services, *QR* is the range of values for nonfunctional properties, *R* is the resource costs of atom services.

Fig. 2 shows an illustrative example of a simple extended feature model, in this model, atom services *PhonePay*, *CreditCard*, *DebitCard* and *FraudDetection* have quality properties and corresponding values, such as the availability of *PhonePay* is [300,500], the rental cost is [50,70].

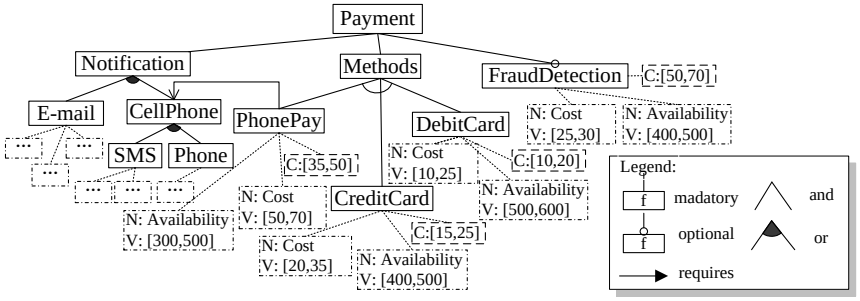


Figure 2. An Example of an Extended Feature Model

3.2 Configuration Model and Objective

In cloud environments, tenants rent configurable services from a cloud provider. Based on the tenants' requirements, the cloud provider configures the configurable service and selects services from an extended feature model. The key challenge when providing customizable cloud services is determining how to find an optimized configuration service rapidly that maximizes tenants' requirements, subject to resource constraints.

Suppose a cloud environment provide *n* customizable services $S = \{s_i\}, 1 \le i \le n$, which modeled by an EFM, C denote all the dependency constraints and cross-tree constraints defined in the EFM. Besides, each service $s_i \in S$ has an associated resource consumption $r(s_i) \in Z$, and a QoS $v_t(s_i) \in Z$. $R(S)$ indicates the set of resources consumed by all the services and $V(S)$ indicates the set of qualities provided by all the services. R_C includes the resource constraints, *e.g.*, $CPU \le 50$, $Memory \le 1G$. The service configuration problem for customizable services aims at selecting and configuring a set of services from customizable services, which achieving the tenant's optimization goal on the premise of resource constraints R_C.Therefore, cloud service configuration based on tenants' requirements is defined in Definition 2.

Definition 2 Given an extend feature model with n cloud services $S = \{s_i\}, 1 \le i \le n$ and a set of constraints C, the goal of the service configuration problem is to find a service subset $S' \subseteq 2^S$ from all valid service combinations such that

$$\sum V_i(S') + \sum (1 - V_j(S')) \text{ is maximized} \quad (1)$$

subject to

$$S' \xrightarrow{\text{conform to}} C$$

$$R(S')(\text{i.e.}\sum_{s_i \in S'} r(s_i)) \le R_C \tag{2}$$

where $V_i(S')$ is the quality which should be maximized, and $V_j(S')$ is the quality which should be minimized.

The aim of this problem is to find a set of services, which, when executed according to the tenant's customized requirements, can fulfill corresponding resource constraints. As we know, this problem is a NP-hard problem, which should be solved by optimization algorithm.

4. Customizable Service Optimization Configuration Algorithms

In this section, we present the customizable service selecting and configuring optimization approach (CSSCOA) based on Binary Particles Swarm Optimization (BPSO) Algorithm, which can quickly derive an optimized service selection by evaluation different configurations based on the tenant's requirements and resource constraints.

4.1. Binary Particles Swarm Optimization Algorithm

BPSO is an evolutionary computation technique proposed by Kennedy and Eberhart [14], which is used to solve 0-1 integer programming problem based on the binary coding scheme specifically. In BPSO, each individual of particle swarm is called particle, which express as a feasible solution. M particles cooperate to search for the global optimum in the n-dimensional search space. Each particle maintains a position x_{id} and velocity v_{id}. A position is a feasible solution to the optimization problem, and the corresponding objective function value is the fitness of particles, which used to measure the pros and cons. A velocity indicates the position of the particle velocity change of direction to guide the search process of particle motion to the optimal solution. In each iteration, each particle uses its own search experience(self-cognitive) and the whole swarm' s search experience (social-influence) to update the velocity and flies to a new position. The updating rules are as follows:

$$v_{id}^{t+1} = wv_{id}^{t} + c_1 r_1 (p_{best}(t) - x_{id}^{t}) + c_2 r_2 (p_{gbest}(t) - x_{id}^{t}) \tag{3}$$

$$x_{id}^{t+1} = \begin{cases} 1, & r < S(v_{id}) \\ 0, & r \ge S(v_{id}) \end{cases}, \text{ and } S(v_{id}^{t+1}) = \frac{1}{1+e^{-v_{id}^{t+1}}} \tag{4}$$

where P_{best} the best solution is yielded by a particle and P_{gbest} is the best-so-far solution obtained by the whole swarm. c_1 and c_2 are two parameters to weigh the importance of self-cognitive and social-influence, respectively. ω is an inertia weight. r_1 and r_2 are random numbers. x_{id}^{t} is the position of the current particle, and x_{id}^{t+1} is the position of the updated particle, v_{id}^{t} and v_{id}^{t+1} express the velocity of the current and updated particle, respectively, v_{id} is random numbers uniformly distributed in $U(0,1)$.

4.2 Customizable Service Selecting and Configuring Optimization Approaches

Although BPSO runs faster than other algorithms because of its relative simple structure, if p_{gbest} fall into a local optimum, the search will be limited in the same area, which will prevent the real optimal solution, besides, the correctness of the solution should be guaranteed for configuring the customizable services. Therefore, we put forward the customizable service selecting and configuring optimization approach based on the basic particle swarm optimization algorithm, which redefine the position of the particle velocity updating formula to enlarge the search scope, introduce the

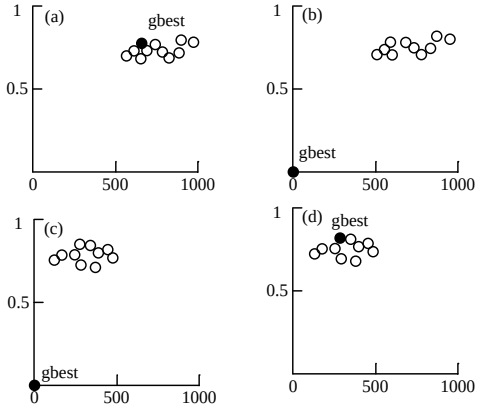
weight adjustment mechanism to improve the convergent speed of particle swarm and define a *dyAdjust* function to satisfy the correctness of the solution.

4.2.1. The Velocity and Position Updating Strategy: Avoiding the solution falls into a local optimum, we limit that the particle velocity must be reach the maximum speed $V_{\max}$ as far as possible, and should be restricted in $V_{\max}$, so the velocity updating formula is as shown in Eq. (5)

$$v_{id}^{t+1} = \begin{cases} wv_{id}^{t} + c_1 r_1 (p_{best}(t) - x_{id}^{t}) + c_2 r_2 (p_{gbest}(t) - x_{id}^{t}) & v_{id}^{t} \in (V_{\max}, V_{\min}) \\ \max(\min(V_{\max}, v_{id}^{t}), V_{\min}) & v_{id}^{t} \notin (V_{\max}, V_{\min}) \end{cases} \quad (5)$$

where $V_{\max}$ and $V_{\min}$ are specified by experience。

As we know, if p_{gbest} trap into a local optimum, the search will be limited in the same area, which will prevent to obtain the real optimal solution. Therefore, the strategy of dynamic reset is introduced in CSSCOA. If p_{gbest} does not change after the algorithm perform three times iteratively, p_{gbest} will be reset into the initial value and continue searching. If this case, the particle will leave the local optimal solution to the global area, and find in new areas, the specific process is shown in Fig.3.



(a) p_{gbest} trapped into local optima (b) reset p_{gbest}

(c) after resetting p_{gbest}, the moving process of particles (d) particles gather to the updated p_{gbest}

Figure 3. The Searching Process of the Particles under the Strategy of Dynamic Reset

4.2.2 The Dynamic Inertia Weight Adjustment Studies have shown that the local search ability is stronger when w is smaller, which facilitates the convergence. And the global search ability is stronger when w is bigger, which is conductive to jump out of local optimal point. Although decreasing inertia weight linearly can improve the performance of algorithm, but it can't reflect the complicated behavior of the search process, which results the effective of the convergence is not ideal. Therefore, a nonlinear function (5) is defined to change the inertia weight dynamically in the process of iteration, which specifies the inertia weight in iteration. This mechanism adjusts global coefficient adaptively, meanwhile gives consideration to the precision and efficiency.

$$w(Iter) = w_{initial} e^{-(\frac{Iter}{Iter_{\max}})^n} \quad (5)$$

where *n* is the nonlinear change rule control power exponents, specially, formula (5) is called probability curve function when *n=2*. Change curves in the Fig. 4 represent the change rule of w in different n, which shows that, the bigger the *n* is, the more it is

inclined to search in global, and vice-versa.

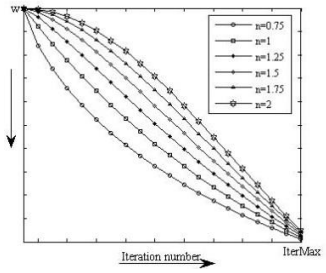


Figure 4. *w* Changing Curves with the Iteration Number under Different n

4.2.3. Feature Adaptively Adjusting Function: The important problem of applying BPSO in solving customizable service configuration problem is how to guarantee the optimal solution is correct. That means a solution must satisfy the all the dependency constraints (*e.g.* and, or, alternative, *etc.*) and cross-tree constraints (as requires and excludes) in EFM. Therefore, we should adaptively adjust an arbitrary service selection into a correct service combination. A function, called dyAdjust is defined to achieve this task, which is shown is Algorithm I.

The input of the *dyAdjust* is an arbitrary service configuration set S_R, and the known conditions are an extended feature model and a rule set C which contains the characteristics and conditions of the EFM. The output is a service set S_V which satisfies C adjusted through S_R, and the services excluded by S_V are in S_E. The process of the algorithm *dyAdjust* is shown in Algorithm I.

Algorithm I Description:

function *dyAdjust* ( )

Data: an arbitrary service configuration set S_R
Data: an extended feature model EFM
Data: a rule set SC contains the characteristics and conditions of the EFM
Data: The correct service set S_V
Data: The exclusive service set S_E

For every service $s \in S_R$ *do*
  if $s \in S_R$ *and* s is not the root *then*
    (the parenet of s) $\in S_V$
  if children of $s \in$ *and*-group then
    foreach service $s' \in$ the group of s's children
      if s' is mandatory *then* $s' \in S_V$
  if $\exists s' \in$ the group of s's children *and* s' is mandatory and $s' \in S_E$ *then*
    all children of s which is mandatory $\in S_E$
  if children of $s \in$ *alternative*-group then
    choose a service ($s' \in$ the group of s's children) $\in S_V$
  foreach $s'' \in$ the group of s's children and $s' \neq s''$
    $s'' \in S_E$
  if s *requires* $s' \in C$
    case $s \in S_V$: $s' \in S_V$
    case $s \notin S_V$: $s' \in S_E$
  if s *excludes* $s' \in C$
    case $s \in S_V$: $s' \in S_E$
    case $s \notin S_V$: $s' \in S_V$

The execution process of the customizable service selection and configuration

optimization approaches is shown as below.

Step 1: Initialization: Initialize the scale of particle swarm *N* based on the given EFM, the maximum iteration number is *K*. Generate the initial velocity and position parameters X_i and V_i randomly, and *dyAdjust* is called to guarantee the correctness of X_i. Calculate the fitness of each particle $f(X_i)$, if $f(X_i)<0$, the particle is discarded. The initial global optimum position X_{gbest} and the initial local optimum of each individual particle are obtained based on $f(X_i)$

Step 2: check the iteration number *K*, if *k* equals to the maximum iteration number *K*, then go to step 4; otherwise step 3.

Step3: iterative algorithm

For each particle in the swarm, calculate the inertia weight w according to equation (5)

Update the velocity and position of the particle according to equation (3) and (4)

Calculate the fitness function value $f(X_i)$ according to equation (1), if $(f(X_i)>f(X_{pbest})$, then $X_{pbest}=X_i$, and adjust X_{pbest} by *dyAdjust*; if $(f(X_{pbest})>f(X_{gbest})$, then $X_{gbest}=X_{pbest}$.

k ++; execute Step 2;

Step 4: output the obtained optimum and its fitness value.

5. Experimental Evaluation

This section presents the experimental evaluation of our approach. We conduct a series of in-lab experiments to verify the effectiveness (measured by success rate and objective value) and efficiency (measured by computational time). We develop a prototype that implements the proposed approaches in Eclipse IDE for Java. The experiment data is generated by BeTTY feature model generator [15]. The service number in the extended feature model is between 100 and 1000, and the quality attributes of each service are generated randomly. [16] pointed out that the characteristics of the above model can reflect the characteristics of the industrial practical application. The experimental environment: Pentium Dual 2.27GHz, 2.0GB RAM, Windows Vista, Eclipse 3.6, FeatureIDE plug-ins and Java Virtual Machine 1.6. The results are the average data for running the algorithm 100 times.

In this paper, we improve the update formula and introduce the adaptive weight adjustment mechanism based on the binary particle swarm optimization algorithm. Therefore, we compare CSSCOA with the standard BPSO with the same setting of parameters, as shown in Table (1), where *itmax* is the number of iterations, *Np* is the number of particle population. The service constraint parameters in EFM are shown in Table (2). 100 feature models for each size are generated randomly with the same parameters, and we only take the mean value of their results.

Table 1. Parameters of CSSCOA and BPSO

Np	***itmax***	***n***	$\omega_{initial}$	c_1	c_2	$V_{\min}$	$V_{\max}$
15	30	2	0.8	2.0	2.0	6	-6

Table 2. The Service Constraint Parameters in EFM

sub-service number of each service	*The relationship between father and son*			*Cross-tree constraints*
	and	*or*	**alternative**	
10(at most)	50 %	25%	25%	10%

Table 3 the time requirement for CSSCOA and BPSO. Here, the experiments are performed on each feature model whose size *FN* varies from 100 to 1000. The rows T-CSSCOA and T-BPSO represent the time requirement for the CSSCOA and BPSO solutions respectively. It is observed that solution times are increased dramatically with the increasing of features and the increasing amplitude are consistent. But, the performance of CSSCOA is slightly better than BPSO under different feature number. During the iterative process, adaptive weight adjustment mechanism enables the *w* balance the global and local search ability, reduce the number of iterations, therefore spend less time, especially in the case of large numbers of features.

Table 3. The Time Requirement for CSSCOA and BPSO under Different Number of Features

time / FN	***100***	***200***	***300***	***400***	***500***
T-CSSCOA(ms)	186.15	669.34	1654.23	2784.71	4200.70
T-BPSO(ms)	188.99	684.55	1718.26	2809.16	4261.44
time / FN	***600***	***700***	***800***	***900***	***1000***
T-CSSCOA(ms)	5583.45	9318.06	10153.22	12903.99	15621.20
T-BPSO(ms)	5604.27	9401.26	10354.91	13019.38	15681.04

The comparison of the approximate optimal solution between CSSCOA and BPSO is shown in Fig.5. As we seen, the fitness function value increased gradually with features increasing, and the fitness value of CSSCOA is better than BPSO. Through changing the global optimal particle's position during searching, CSSCOA expands the search scope of the solution space to find a better solution.

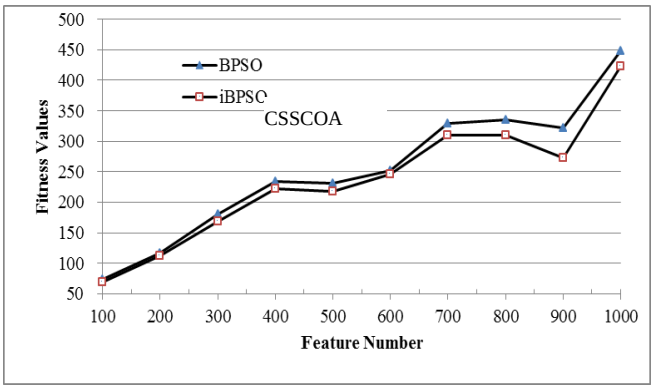


Figure 5. The Comparison of Fitness Values between CSSCOA and BSPO

Next we will analyze the influence of different service constraint parameters in EFM for algorithm. Literature [9] pointed out that in the feature model, the main factors influencing the product configuration is the dependency constraints among features.

To this end, the experiments are performed on 30 feature models for 1000 size, which are generated randomly with the same parameters, and we take the mean value of their results. Under different ratios of dependency and exclusive relationships, the efficiency and the approximation degree of the algorithm are compared, which are shown in Table 4. The columns T-CSSCOA, R-CSSCOA and SD-CSSCOA represent the solution time, the approximation degree and instability of CSSCOA. With the growth of the constraint ratio, the solution time grows linearly, the approximate solution declines gradually, and the instability of the algorithm increases gradually.

Table 4. The Influence of Different Dependency and Mutually Exclusive Ratio on the Algorithm

Constraint Ratio	*T-CSSCOA*	*R-CSSCOA*	*SD-CSSCOA*
0	446.027	0.778	0.074
10%	5031.789	0.839	0.084
20%	10429.334	0.710	0.092
30%	15909.783	0.545	0.088
40%	20927.820	0.448	0.111

6. Conclusion

With the mature of cloud computing technology, more and more applications are moved to the cloud. How to manage those services and provide customizable cloud services to tenants has become a more active research. In this paper, we introduce a novel approach to support the modeling and configuring of customizable cloud services. First, we present an extended feature model, which manage and model customizable cloud services effectively. Then, we propose a customizable service selecting and configuring optimization approach (CSSCOA), which can quickly derive an optimized service selection by evaluation different configurations that both optimize tenant preferences and honor resource limitations. The main contributions are 1. redefine the position of the velocity updating formula, which conducive to improve the solving accuracy; 2 introduce the dynamic weighting adjustment mechanism, which improve convergence speed; 3 puts forward feature adaptively adjusting function, which ensure the rationality of solving. At last, the scalability and performance of the algorithms is investigated.

References

[1] I. Kumara, "Runtime Evolution of Service-Based Multi-tenant SaaS Applications", Service-Oriented Computing. Springer Berlin Heidelberg, **(2013)**, pp. 192-206.
[2] J. G. Galán, "Migrating to the Cloud - A Software Product Line based Analysis", In Proceedings of the 3rd International Conference on Cloud Computing and Services Science, **(2013)**, pp. 416-426.
[3] Q. He, "QoS-Aware Service Selection for Customisable Multi-Tenant Service-Based Systems: Maturity and Approaches", In Proceedings of the 8th International Conference on IEEE Cloud Computing (CLOUD), **(2015)**, pp. 237-244.
[4] B. M. Ognjanovic, D. Gasevic, E. Bagheri and M. Boskovic, "A Metaheuristic Approach for the Configuration of Business Process Families", In the Proceedings of the Ninth International Conference on IEEE Services Computing, Hawaii, USA , **(2012)**, pp: 25-32.
[5] H. Moens, B. Dhoedt and F. D. Turck, "Allocating resources for customizable multi-tenant applications in clouds using dynamic feature placement", Future Generation Computer Systems , no. 53, **(2015)**, pp. 63-76.
[6] K. C. Kang, J. Lee and P. Donohoe, "Feature-oriented product line engineering", IEEE Software, vol. 19, no. 4, **(2002)**, pp. 58-65.
[7] D. Benavides, S. Segura and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later-A literature review", Information Systems, vol. 35, no. 6, **(2010)**, pp. 615-636.
[8] A. S. Sayyad, T. Menzies and H. Ammar, "On the value of user preferences in search-based software engineering: A case study in software product lines", In the Proceedings of the 35th International Conference on IEEE Software Engineering (ICSE), **(2013)**, pp.492-501.

[9] J. White, B. Dougherty and D. C. Schmidt, "Selecting highly optimal architectural feature sets with Filtered Cartesian Flattening", Journal of Systems & Software, vol. 82, no. 8, **(2009)**, pp. 1268-1284.
[10] Q.He, "QoS-Aware Service Selection for Customisable Multi-Tenant Service-Based Systems: Maturity and Approaches", In the Proceedings of the 8th International Conference on IEEE Cloud Computing (CLOUD), **(2015)**, pp. 237-244.
[11] M. Alrifai, D. Skoutas and T. Risse, "Selecting Skyline Services for QoS-based Web Service Composition", In the Proceedings of the 19th International Conference on World Wide Web (WWW2010), Raleigh, North Carolina, USA, **(2010)**, pp. 11-20.
[12] Q. He, J. Han, Y. Yang, J. Grundy and H. Jin, "QoS-Driven Service Selection for Multi-tenant SaaS", In the Proceedings of the Fifth International Conference on IEEE Cloud Computing, Honolulu, HI, USA, **(2012)**, pp. 566-573.
[13] A. Metzger, "Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis", In Proceedings of the IEEE International Requirements Engineering Conference, **(2007)**, pp. 243-253.
[14] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm", In Proceedings of the International Conference on Computational Cybernetics and Simulation, no. 5, **(1997)**, pp. 4104-4108.
[15] Homepage for BeTTy: http://www.isa.us.es/betty/betty-online.
[16] T. Thum, D. Batory and C. Kastner, "Reasoning about edits to feature models", In Proceedings of the 31st International Conference on Software Engineering, **(2009)**, pp. 254-264.

