TECHNICAL REPORT ISA-2011-TR-04 (v. 1.0)

On parameter selection and problem instances generation for QoS-aware binding of composite web services using GRASP with Path Relinking

Josè Antonio Parejo Maestre, Pablo Fernandez and Antonio Ruiz Cortés
Computer Science and Engineering School
Department of Computing Languages and Systems
University of Seville
{japarejo,pablofm,aruiz}@us.es

Applied Software Engineering Research Group
http://www.isa.us.es

University of Sevilla
http://www.us.es

This work is partially supported by:

Ministerio de Ciencia e Innovación

Consejería Innovación, Ciencia y Empresa, Junta de Andalucía

November 24, 2011

Contents

1. Introduction **4**

2. The QoS-aware Web Service Composition problem **5**
2.1. An ilustrative example 5
2.2. Components of the Problem 7
2.3. Composition Structure and Run Time Information 9
2.4. Candidate Services and QoS Model 9
2.4.1. Constraints treatment 11
2.5. Additional Problem Metrics 12
2.5.1. Extreme QoS values of the Composition 12
2.5.2. Average QoS 12

3. GRASP with Path Relinking for QoS-aware Web Services Composition **13**
3.1. Solution Encoding 15
3.2. Construction Phase 15
3.2.1. Greedy function G1 16
3.2.2. Greedy function G2 16
3.2.3. Greedy functions G3,G4 and G5 16
3.2.4. Greedy function G6 16
3.2.5. Greedy function G7 16
3.3. Choosing a Construction method 17
3.4. GRASP Improvement Phase 17
3.5. Path Relinking 17

4. Experimentation **18**
4.1. Previous Proposals 19
4.1.1. Genetic Algorithms 19
4.1.2. Hybrid Tabu Search with Simulated Annealing 19
4.2. Experimental Setting 21
4.3. Experiment 21
4.3.1. Experiment design 22
4.3.2. Experiment conduction and Results 22
4.3.3. Analysis and Interpretation of Results 24
4.4. Threats to Validity 28

5. Related Work **29**

6. Conclusion **31**

A. Preliminary Experiment 1 **32**

B. Preliminary Experiment 2 **33**

C. QoSWSC Problem Instances Generation Algorithm 34
C.1. Composition structure generation algorithm 34
C.2. Global Constraints Generation Algorithm 36

D. QoS-aware Web Service Composition Formulation 37

E. Acnowledgements* 39

1. Introduction

Web Services are software systems designed to support interoperable machine-to-machine interaction over a network based on an interface described in a machine processable format [38]. The design and development of applications and whole software architectures based on web services, known as Service Oriented Computing (SOC), is attracting a great attention both from academia and industry [20] [30]. In Service Oriented environments, complex applications are developed by composing web services, specifying the sequence of invocation of composed services. These applications are expressed typically in BPEL [2], being deployed as services, in such a way that other compositions can use it recursively (as services being composed).

Moreover, these applications can be composed using *abstract services*, where the services to be invoked are chosen dynamically at runtime from a set of candidates that implement the same functionality and its corresponding interface. In this context, Quality of Service (QoS) has been identified as a key element to guide the selection of those candidates, providing a set of properties, such as execution time, invocation cost or availability for each candidate service. The development of QoS-aware composite services leads to the creation of context-aware and automatically optimized applications, depending on available services and user preferences. This problem represents an important SOC research challenge [29, 30] identified as part of the Search Based Software Engineering area [19].

QoS-aware Web Service Composition (QoSWSC) implies solving a NP-hard optimization problem [6, 4]. Moreover, QoSWSC is essentially dynamic since QoS levels provided by a service may change frequently, and even some services can become unavailable or new services emerge [10]. Consequently, dynamic composition approaches are needed; which take into account runtime changes in the QoS of component services. When this problem is solved at runtime (during the execution of the sequence of invocations of the composition or immediately before its start, i.e. at invocation time), taking into account the current execution state; It is called a *reoptimization* or *rebinding* problem [41][3]. In these circumstances, the time to obtain a good solution is a critical issue, and the use of heuristics appears as a promising approach [5]. In literature some metaheuristic techniques has been proposed to solve this problem [7, 40, 24].

This techreport is a companion to a shorter article, where the state of the art on QoS-aware binding of composite web services is improved in two aspects:

1. Providing a novel approach for solving the QoSWSC Problem based on GRASP with Path Relinking.

2. Showing that this approach outperforms previous metaheuristics-based approaches for the QoSWSC Problem in rebinding scenarios. Specifically, results of this approach improve up to 40% the results of previous proposals in scenarios where results must be available in less than a minute, generating the best average results in nearly all experiment runs for those execution times.

Specifically, this techreport provides additional information by adding a set of four appendixes to the original paper: : the first and second one discuss the preliminary experiments performed; the third appendix details the algorithms designed to create the

instances of the QoSWSC problem; the fourth provides a complete formulation of the QoSWSC problem.

The remainder of the tech report is organized as follows: Section 2 presents a formalization of the QoSWSC problem. Section 3 describes our proposal for solving this problem. Section 4 shows the experiments performed, their analysis and the interpretation of results. This section also describes previous proposals used for comparison in the experiments. Section 5 presents related work. Finally, Section 6 describes our conclusions and future work. In addition, four appendixes are presented as described above.

2. The QoS-aware Web Service Composition problem

When invoking a composite service, its specification is analyzed, obtaining the set of abstract web services (tasks in the remainder of the report) with each requiring the selection of a concrete service. The set of available services is obtained for each task, usually by performing a search on a registry using technologies such as UDDI or ebXML. In this search, functional features (typically specified as the service interface) determine the set of available candidates, where QoS properties (such as service cost or availability) drive the selection of the best service for each task. Given that some registry technologies do not support QoS information, a QoS-enriched registry or alternative QoS information data source (such as a Service Level Agreements Repository or a Service Trading Framework [**?**]) is needed.

After gathering the QoS information, the QoSWSC problem instance can be optimized. The goal of this optimization is to select the combination of services that maximizes the global QoS. As a result, an execution plan χ is developed, specifying the actual service for each task in the composition. Finally, a composition execution engine proceeds to the execution of the composition, performing service invocations that conform to χ.

2.1. An ilustrative example

In order to illustrate the problem, a "Travel Planner" service is depicted in fig. 1 as an UML activity diagram, this example was first introduced in [41]. The diagram specifies the sequence of invocation of composed services. The two circles at the ends of the diagram denote the start (left) and end (right) of the execution of service composition. Arrows ($\longrightarrow$) specify the sequence of composed service invocations. Diamonds symbols ($\diamond$) denote either a set of alternative execution flow branches ($\rightarrow \diamond\nearrow\searrow$, where only one of the branches will be executed, being this one chosen by a conditional expression), or a convergence on the branches of a bifurcation ($\nearrow\searrow\diamond \rightarrow$). Bolded-bar symbols denote the start ($\rightarrow |\nearrow\searrow$) or end ($\nearrow\searrow| \rightarrow$) of the parallel execution of several flow branches; i.e. all the branches will start executing at the same time, and the execution from the end bar will only continue until the execution of all incoming branches is ended.

In this example a composite service for travel planning is provided, based on a set

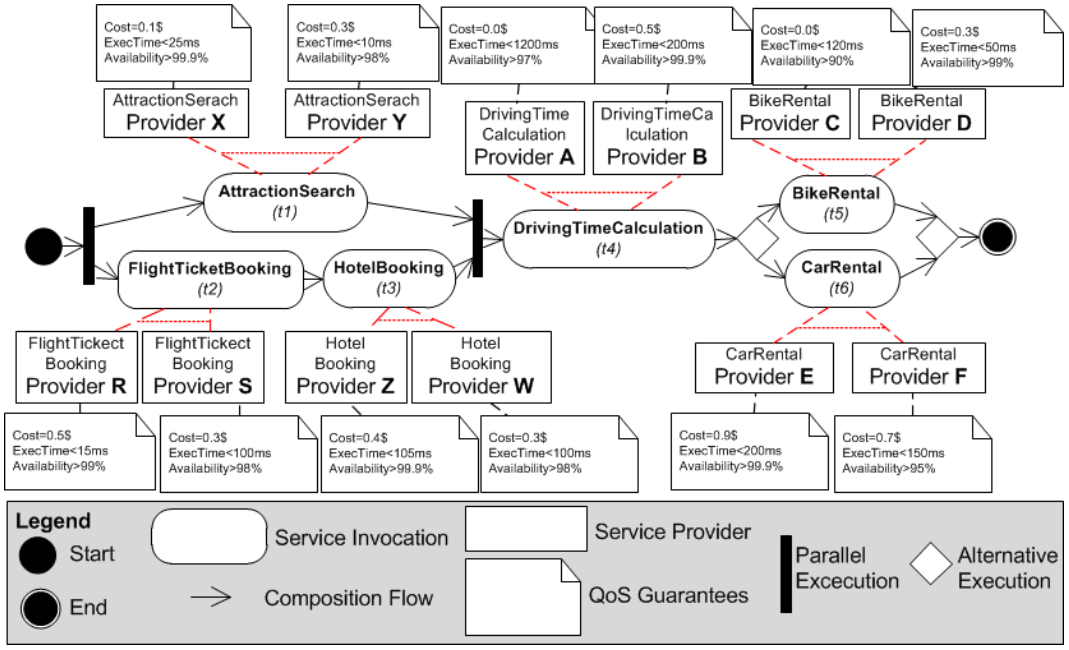

Figure 1: "Travel Planner" Composite Service

of common independent services, it provides an added value service that frees the user from the burden of travel planning activities, by automating invocation of the corresponding services from a proper input data. The composition starts with a search for attractions, that is done in parallel with a flight and an accommodation booking. After the searching and booking operations complete, the distance from the hotel to the attractions is computed, and either a car or a bike-rental service is invoked. Available providers for each service are also shown as boxes, where only one provider can be used to perform each task actually. Specifically, for each task in this example two providers are available. Each provider guarantees different QoS values for their services; in this example, the value of QoS properties *cost*, *execution time* and *availability* are shown. Noticeably, this small sized problem presents 64 (2^6) different possible execution plans, such as $\{< t1, X >, < t2, R >, < t3, Z, >, < t4, A >, < t5, C >, < t6, E >\}$, where provider X is chosen to perform task t_1, provider R is chosen to perform task t_2, etc.

The global QoS values for the composite service depend on: (i) the structure of the composition, i.e. the combination of branches, parallel executions and sequences of invocations, (ii) the provider selected for each task, that ultimately determines the QoS provided for the task, (iii) the concrete set of branches chosen for execution, and (iii) the own nature of the QoS property. For instance, the global cost of the execution plan specified above, assuming that car rental (t_6) is needed (not t_5), will be:

$$Global_{Cost} = Cost(t_1) + Cost(t_2) + Cost(t_3) + Cost(t_4) + Cost(t_6) =$$
$$= Cost(X) + Cost(R) + Cost(Z) + Cost(A) + Cost(E) = 0.1 + 0.5 + 0.4 + 0 + 0.9 = 1.9$$

However, since the total execution time of two parallel branches is equal to the maximum of their execution times, the global execution time under the same circumstances will be:

$$Global_{ExecTime} = Max((ExecTime(t_1), ExecTime(t_2)) + ExecTime(t_3)) +$$
$$+ExecTime(t_4) + ExecTime(t_6) = Max(ExecTime(X), ExecTime(R) + ExecTime(Z))$$
$$+ExecTime(A) + ExecTime(E) = Max(25, 15 + 105) + 1200 + 200 = 1520$$

2.2. Components of the Problem

One of the key reasons for the complexity of this problem is the number of components that are involved and their intricate relationships. Next we enumerate the components that are considered in our formalization of the QoSWSC problem:

Tasks (T): set $T = \{t_1, \ldots, t_n\}$ of tasks in the composition. As an example, figure 1 shows a composition involving six different tasks.

QoS Properties (Q): set $Q = \{q_1, \ldots, q_l\}$ of QoS properties handled during optimization. Figure 1 shows services that provide guarantees on three different QoS properties; Cost, Execution time and Availability.

Composition Structure (Φ): this component specifies the sequence of the invocation of tasks. Figure 1 shows a composition structure specifying the parallel execution of activities $t1$ and the sequence of $t2$ and $t3$, next a sequential invocation of $t4$, with a switch s with two independent branches invoking $t5$ and $t6$ respectively. Note that the executions of the composition will perform five tasks, since only one of the branches of the switch will executed.

Run time information (R): Once the composition is being executed, the underlying platform -usually an Enterprise Service Bus (ESB) or BPEL execution engine- can collect data about the composition execution. This information allows for the creation of a predictive model of the tasks that will be invoked by the composition, and optimize according to its predictions. The amount and type of information needed depends on the accuracy of the model. In this report we assume that this information, denoted as $R = \{R_L, R_S\}$ contains information about loops R_L and switches R_S. For each loop l, R_L contains its average number of iterations k. For each branch b of a switch s, R_S contains an average probability of execution of each branch p_b.

Candidate Services (S): In order to implement the composition for each task t_i a set of functionally equivalent candidate services is available $S_i = \{s_{i,1}, \ldots, s_{i,m}\}$, having $S = \{S_1 = \{s_{1,1}, \ldots, s_{1,m}\}, \ldots, S_n = \{s_{n,1}, \ldots, s_{n,l}\}\}$ and being $S^* = \bigcup_{i=1}^{n} S_i$ the whole set of services available. Each candidate service $s_{i,j}$ offers a QoS level $F_q(s_{i,j})$ for each property $q \in Q$. (e.g. given a property *Cost*, $F_{cost}(s_{i,j}) = 0.3\$$ is the price of each invocation of $s_{i,j}$). Once one candidate service $s_{i,j}$ is selected for each task t_i in the composition, we have an execution plan of the composition $\chi = \{\langle t_1, s_{1,j}\rangle, \ldots, \langle t_n, s_{n,l}\rangle\}$. For each task, the QoS level provided by the execution of the task t_i is the QoS level provided by the service $s_{i,j}$ selected where $\langle t_i, s_{i,j}\rangle \in \chi$.

Global QoS Model (Ψ): In addition to the QoS levels provided by each service in the composition, it is important to express the global QoS of the whole. The global QoS model Ψ describes the way we compute the global QoS value for each property based on three elements: the set of selected candidate services, expressed as an execution plan χ, the composition structure and the run time information. It is formalized as $\psi_q(\chi, \Phi, R)$ having $q \in Q$, being $\Psi = \bigcup_{q \in Q} \{\psi_q\}$. As an example, the property *Execution Time* has a value for each single service and a global value for the whole composition, that depend on the number of iterations of loops, the probability of execution of each branch in the switches and the own structure of the composition.

User preferences (W): This component determines which execution plan is more valuable based on the QoS levels provided. Some proposals take a multi-objective approach to QoSWSC problem solving [13][39], using each QoS property as a different

objective to optimize, obtaining a set of execution plans as a result. In this report we use Simple Additive Weighting (SAW), where user preferences are expressed as weights w_q for each QoS property q according to their relative importance, having $\sum_{q \in Q} w_q = 1$. A complementary way to express user preferences are through Global QoS constraints as shown below.

Constraints (C): This component limits the execution plans that can be selected by ensuring some properties. Three different types of constraints $C = \{C_g, C_l, C_d\}$ have been identified for this problem [41, 3]:

- Global QoS constraints (C_g) express a restriction on the value of some QoS properties of the whole composition as obtained by the application of the QoS Model $\psi_q(\chi, \Phi, R) < C_q$; e.g. *the total cost of the composition is lower than five monetary units* $\equiv \ \psi_{cost}(\chi, \Phi, R) < 5$. Thus, for each QoS property considered $q \in Q$, C_g can contain a threshold c_q that specify its corresponding constraint.

- Local QoS Constraints (C_l) express a restriction on the value of some QoS properties of the service selected for a particular task; e.g. *the cost of task* t_2 *must be lower than 1 monetary unit* $\equiv \ F_{cost}(t_2) < 1$).Thus, for each QoS property considered $q \in Q$, and task of the composition $t_i \in T$, C_l can contain a threshold c_q^i that specify its corresponding local QoS constraint. Local constraints have been omitted from the test bed developed in this report, since they can be satisfied by pre-processing the set of candidate services, excluding the candidates that do not meet the constraint.

- Service interdependence constraints (C_d). In some cases, composition makes use of state-full web services, where one of the candidate services implements various tasks, imposing an interdependence constraint; i.e. if the service is selected for one of the tasks, it must be selected for the rest of dependent tasks. Some examples of these interdependences can be found in [3]. Those constraints can be expressed by the set of services $c_d = \{s_{i,j}, \ldots, s_{k,l}\}$ that are inter-dependent. Thus C_d contains a set of those sets.

Based on the previous description, a QoSWSC problem P is defined as
$P = \{T, S, Q, \Phi, \Psi, R, W, C\}$.

It is important to note that, due to the use of an average number of iterations in loops and probabilities of branch execution in switches, our global QoS values are just estimates, and the real QoS values of a specific execution of the composition can differ significantly. For instance, the number of iterations performed in a loop for a concrete execution could be twice those performed on average, and consequently the execution time could become much higher, and in the worst case global QoS constraints could be violated. In order to avoid this problem, the use of the re-binding triggering algorithm described in [10] is recommended. Furthermore, given that our proposal obtains better results for the execution times used for runtime re-binding, it is convenient to combine this algorithm with our proposal instead of with the original evolutionary algorithm proposed in [10].

Table 1: Basic building blocks of the composite web service description language

Building Block	Description
Sequence s	Sequential execution of n activities $[a_1, \ldots, a_n]$
Switch b	Alternative choice of execution of sequences of activities $\left\{s_1^b = [a_{1,1}, \ldots, a_{1,p}], \ldots, s_m^b = [a_{m,1}, \ldots, a_{m,q}]\right\}$
Fork f	Parallel asynchronous execution of sequences of activities $\left\{s_1^f = [a_{1,1}, \ldots, a_{1,p}], \ldots, s_m^f = [a_{m,1}, \ldots, a_{m,q}]\right\}$
Loop l	Iterative execution of a sequence of activities $s^L = [a_1, \ldots, a_n]$

2.3. Composition Structure and Run Time Information

There are different standard languages that could be used to specify the structure Φ of composite services, from UML activity diagrams (used in figure 1) or state charts (used in [3] and [41]), to BPEL [2]: used as an operational language that can be deployed and executed in the appropriate engine.

In the remainder of the report, we assume that the language used to express the structure of the composition contains a set of basic *building blocks*, without binding our proposals to a specific language. The different kinds of building blocks supported are summarized in table 1. In this point, we can fine-tune the idea of composition to integrate building blocks, by defining a composition as a set of *activities* divided into two types: building blocks and tasks. According to this model, loops, switches, and sequences of tasks can be nested one in each other without limitations, in order to implement the concrete logic of the composite service, creating a hierarchical structure of activities.

2.4. Candidate Services and QoS Model

In this report, we use a set of quality properties, which have been used for QoS-aware selection literature ([41], [3], [9]). Although the set of quality properties used in this report is limited to domain-independent properties, it is extensible; new and possibly domain-dependent quality properties can be added without fundamentally altering our approach, just like the example shown in [8].

QoS properties are classified as *negative* or *positive*. A quality property is positive if the higher the value, the higher the user utility. For instance, availability is a positive property, since the higher the availability the better for the user. A quality property is negative if the higher the value, the lower the user utility. Thus, cost is a negative property, since the higher the cost the worst it is for the user. So, $Q = Q^+ \cup Q^-$, being Q^+ and Q^- the set of positive and negative QoS properties respectively. The following set of quality properties Q is considered in this report:

- *Cost* (C) fee that a user has to pay for invoking $s_{i,j}$.
- *Execution Time* (T) expected delay between candidate web service $s_{i,j}$ invocation and the moment when result is obtained, usually measured in seconds.

Table 2: Aggregation functions per Building Block and QoS property

	Sequence (S)	Loop (L)	Branch (B)	Fork (F)
Cost (C)	$\sum_{i=1}^{m} C(a_i)$	$k \cdot \sum_{i=1}^{n} C(a_i)$	$\sum_{i=1}^{m} P_i \cdot C(s_i^b)$	$\sum_{i=1}^{p} C(s_i^f)$
Time (T)	$\sum_{i=1}^{m} T(a_i)$	$k \cdot \sum_{i=1}^{n} T(a_i)$	$\sum_{i=1}^{m} P_i \cdot T(s_i^b)$	$max\left\{T(s_i^f)\right\}$
Reliability (R)	$\prod_{i=1}^{m} R(a_i)$	$(\prod_{i=1}^{n} R(a_i))^k$	$\sum_{i=1}^{m} P_i \cdot R(s_i^b)$	$\prod_{i=1}^{p} R(s_i^f)$
Avaliability (A)	$\prod_{i=1}^{m} A(a_i)$	$(\prod_{i=1}^{n} A(a_i))^k$	$\sum_{i=1}^{m} P_i \cdot A(s_i^b)$	$\prod_{i=1}^{p} A(s_i^f)$
Security (S)	$\min(S(a_i))_{i\in\{1\ldots m\}}$	$\min(S(a_i))$	$\sum_{i=1}^{m} P_i \cdot S(s_i^b)$	$\min_{i=1}^{p} S(s_i^f)$
Custom attribute (F)	$f_S(F(a_i))_{i\in 1\ldots m}$	$f_L(s^L, k))$	$f_B(\left[F_S(s_i^b)\right], [p_i])$	$f_F(F(s_i^f))_{i\in 1\ldots p}$

- *Availability* (A) is the probability of accessing the service per invocation, where $A(s_{i,j}) \in [0, 1]$.
- *Reliability* (R) is a measure of the service's trustworthiness $s_{i,j}$; it represents the ability to respect the quality expected for the rest of the properties. Usually, this property is based on a ranking performed by end users rank. For example, in *www.amazon.com*, the range is $[0, 5]$. In this report we will assume $R(s_{i,j}) \in [0, 1]$. This QoS property is equivalent to the *Reputation* specified in [9]
- *Security* (S) is the quality aspect of a service $s_{i,j}$ to provide mechanisms to assure confidentiality, authentication and non-repudiation of the parties involved. Usually this property implies the use of encryption algorithms with different strength, different key sizes on underlying messages, and some kind of access control. In this report we will assume a categorization of the security, where the use of an encryption algorithm and key size in a service $s_{i,h}$ implies a numerical value associated to this property for the service, where $0 \leq S(s_{i,j}) \leq 1$, where 0 means no security at all and 1 maximum security.

The approach to compute the global QoS of execution plan χ is similar to the proposal stated in [12], that has been used extensively in literature [41, 3, 9, 40]. Table 2.4 summarizes the aggregation functions applied for each QoS property q and type of building block, that conform our QoS-model $\Psi = \bigcup_{q\in Q} \Psi_q$, the aggregation functions correspond to those proposed [9]. These functions are recursively defined on activities of the composition structure Φ, allowing to compute global values by the recursive application of the corresponding function on each building block $\phi_i \in \Phi$. For instance, having run time information $R = \{R_L, R_S\} = \left\{\{\}, \left\{\left\{p_{b_{top}}, p_{b_{bottom}}\right\}\right\}\right\} = \{\{\}, \{\{0.6, 0.4\}\}\}$, i.e. *the probability of execution of top branch is 0.6 while the probability of the bottom branch is 0.4*; Global cost of composition structure Φ shown in figure 1, for the execution plan $\chi = \{< t_1, s_{1,X} >, < t_2, s_{2,R} >, < t_3, s_{3,Z} >, < t_4, s_{4,A} >, < t_5, s_{5,C} >, < t_6, s_{6,E} >\}$ will be:

$\psi_{Cost}(\chi, \Phi, R) = \sum$(Cost of activities in the main sequence) =
$\sum$(Cost of activities in the parallel Flow) $+ \psi_{Cost}(t_4)+$
$+ \sum$(Cost act. in each branch of the final switch) $\cdot$ (Prob. of exec. of the branch)) =
$\psi_{Cost}(t_1) + \psi_{Cost}(t_2) + \psi_{Cost}(t_3) + \psi_{Cost}(t_4) + 0.6 \cdot \psi_{Cost}(t_5) + 0.4 \cdot \psi_{Cost}(t_6) =$
$F_{Cost}(s_{1,X}) + F_{Cost}(s_{2,R}) + F_{Cost}(s_{3,Z}) + F_{Cost}(s_{4,A}) + 0.6 \cdot F_{Cost}(s_{5,C}) + 0.4 \cdot F_{Cost}(s_{6,E}) = 0.1 + 0.5 + 0.4 + 0 + 0.6 \cdot 0 + 0.4 \cdot 0.9 = 1.36$

The goal of solving the QoSWSC problem is to execute the composition with as much QoS as possible, i.e. finding the best execution plan χ. The meaning of "best"

depends on the class of QoS property under consideration -either positive or negative-. In order to support a clear formulation of our objective function, QoS values of services are normalized and scaled. This scaling is performed for each property $q \in Q$ and candidate service $s_{i,j}$:

$$F'_q(s_{i,j}) = \begin{cases} \frac{F_q(s_{i,j}) - q^{min}}{q^{max} - q^{min}} & \text{if} \quad \text{q is positive} \\ \\ \frac{q^{max} - F_q(s_{i,j})}{q^{max} - q^{min}} & \text{otherwise} \end{cases} \tag{1}$$

where q^{max} and q^{min} are the maximum and minimum value of $F_q(s_{i,j})$, $s_{i,j} \in S^*$. The scaling occurs prior to optimization as a pre-processing step. The global QoS value of a solution $\chi = \{< t_1, s_{1,j} >, \ldots, < t_n, s_{n,l} >\}$is computed by the evaluation of the global QoS provided by the given solution (that represents an execution plan of the composition), conforming to the QoS Model Ψ_q for each QoS property q, described in table 2.4. This process results in a set of global QoS properties values $\{Q_{cost}, Q_{time}, Q_{avail}, Q_{rel}, Q_{sec}\}$. Finally, we use simple additive weighting to compute a global QoS value:

$$GlobalQoS(\chi) = \sum_{q \in Q} w_q \cdot \Psi_q(\chi, \Phi, R) \tag{2}$$

having $\sum_{q \in Q} w_q = 1$.

A complete formulation of the QoSWSC problemas specified above is provided in appendix D.

2.4.1. Constraints treatment

$GlobalQoS$ as defined in the previous subsection does not consider constraints.In order to support the optimization of constrained problem instances, Lagrangean relaxation is used. Some results in literature [9] show that a dynamic penalization function does not improve the performance of Genetic Algorithms for this problem. Thus, for the purpose of this report, we will use a simple penalization based on a weight w_{unf}, and a value D_f that measures the distance of χ from full constraint satisfaction.

$$D_f(\chi, \Omega) = \frac{\sum_{c \in C} Meet(c, \chi)}{|C|} \tag{3}$$

having $Meet(c, \chi) = 1$ if χ meets ρ, 0 otherwise if $c \in C_d$ or $c \in C_l$. If $c \in C_g$, i. e. it is a global constraint, $Meet(c, \chi) = |t(c) - \Psi_q(\chi, \Phi, R)|$, where q is the QoS property on which the constraint is imposed and $t(c)$ is the threshold value of c. Moreover, we apply a correction factor N to $GlobalQoS$ defining $NormGlobalQoS(\chi) = GlobalQoS(\chi)/N$ where

$$N = GlobalQoS^{max} = \sum_{q \in Q} w_q \cdot \Psi_q^+ \tag{4}$$

where Ψ_q^+ is the best possible global value for property q (computed after normalization an scaling, c.f. section 2.5), having $0 \leq NormGlobalQoS(\chi) \leq 1$.

Our final objective function to be maximized is:

$$FinalQoS(\chi) = NormGlobalQoS(\chi) - (w_{unf} \cdot D_f(\chi)) = \tag{5}$$

$$= \frac{\sum_{q \in Q} w_q \cdot \Psi_q(\chi, \Phi, R)}{\sum_{q \in Q} w_q \cdot \Psi_q^+} - w_{unf} \cdot (\frac{\sum_{c \in C} Meet(c, \chi)}{|C|}) \tag{6}$$

having $0 \leq w_{unf} \leq 1$ where $-1 \leq FinalQoS(\chi) \leq 1$.

2.5. Additional Problem Metrics

The following problem metrics must be defined to properly describe the techniques used to solve this problem:

2.5.1. Extreme QoS values of the Composition

Given a QoS property q and a QoSWSC problem instance $P = \{T, S, Q, \Phi, \Psi, R, C\}$, the extreme QoS values of composition for property $q \in Q$, are defined as follows:

$$\Psi_q^+ = \Psi_q(\chi_q^+, \Phi, \Gamma) \tag{7}$$

$$\Psi_q^- = \Psi_q(\chi_q^-, \Phi, \Gamma) \tag{8}$$

where χ_q^+ and χ_q^- denote solutions where for each task $t_i \in T$, the service $s_{i,j}$ that provides the best and worst value for q is used respectively. It is important to note that Ψ_q^+ and Ψ_q^- are upper a lower bounds of the global QoS value of the composition for q.

2.5.2. Average QoS

Given a QoS property q and the set of tasks T and candidate services S of a QoSWSC problem, we define the average QoS value of q for task t_i, denoted as $Avg_{t_i}^q$ as the average value of q for the candidate services $s_{i,j}$ of t_i:

$$Avg_{t_i}^q = \frac{\sum_{s_{i,j} \in S_i} q(s_{i,j})}{|S_i|} \tag{9}$$

Global average value of q, denoted as Avg^q, is defined as the average value of q for the services in S:

$$Avg^q = \frac{\sum_{s_{i,j} \in S^*} q(s_{i,\Psi_q^{Avg}(\chi^{Avg}, \Phi, \Gamma)})}{|S^*|} \tag{10}$$

Based on the previous definitions, an average value of property q for a composition Φ and run-time information R can be defined:

$$\Psi_q^{Avg} = \Psi(\chi^{Avg}, \Phi, R) \tag{11}$$

This definition is based on the usage of a virtual solution χ^{Avg} where all services present average QoS values; i. e. $\forall(q \in Q)\forall(\langle t_i, s_{i,j}\rangle \in \chi^{Avg}) \bullet F^q(s_{i,j}) = Avg_{t_i}^q$.

3. GRASP with Path Relinking for QoS-aware Web Services Composition

The Greedy Randomized Adaptive Search Procedure (GRASP)[35] is an iterative problem solving technique where each iteration consists of constructing a trial solution and then applying an improvement procedure, typically a local search method. This technique has been successfully applied in a plethora of real life applications and research problems [16].

Algorithm 1 shows the pseudo-code of the basic GRASP. In this algorithm a subroutine named $greedyRandomizedSolution()$ is used for implementing the construction phase. This subroutine is depicted in detail in algorithm 2.

In the GRASP construction phase, solutions are constructed by iteratively adding elements to a partial solution. Those elements are each of the individual components that a valid solution to the problem contains. This paper presents an adaption of GRASP to solve the QoSWSC Problem, where features represent the concrete services chosen for implementing each task. Since some elements can be incompatible with others (its addition would lead to the unfeasibility of the partial solution) the *elements* function is used to get the valid elements for the current partial solution.

The next subsection describes the solution encoding used for all the algorithms compared in this paper. The remaining subsections describe how the adaption of grasp is performed, by specifying the solution construction scheme, and the RCL updating policy. Additionally, a search procedure that combines GRASP with Path Relinking [26] is proposed. This approach generates new solutions by exploring trajectories that connect high-quality solutions previously found with the GRASP method.

Algorithm 1 GRASP pseudocode

$currentSolution \leftarrow \oslash$
$currentEval \leftarrow +\infty$
$bestSolution \leftarrow \oslash$
$bestEval \leftarrow +\infty$
{Main loop}
repeat
  $currentSolution \leftarrow greedyRandomizedSolution()$ {Construction Phase}
  $currentSolution \leftarrow localSearch(currentSolution)$
  $currentEval \leftarrow f(currentSolution)$
  if $currentEval < bestEval$ **then**
    $bestSolution \leftarrow currentSolution$
    $bestEval \leftarrow currentEval$
  end if
until Termination Criteria is satisfied
return $bestSolution$

Algorithm 2 GRASP Construction Phase

$currentSolution \leftarrow neutralSolution()$
$validFeatures \leftarrow \{\}$
$RCL \leftarrow \{\}$
{Main Construction loop}
repeat
  $validFeatures \leftarrow features(currenSolution)$
  $RCL \leftarrow selectCandidateFeatures(validFeatures, g)$
  $chosenFeature \leftarrow selectFeature(RCL)$
  $addFeature(currentSolution, chosenFeature)$
until $isComplete(currentSolution)$
return $currentSolution$

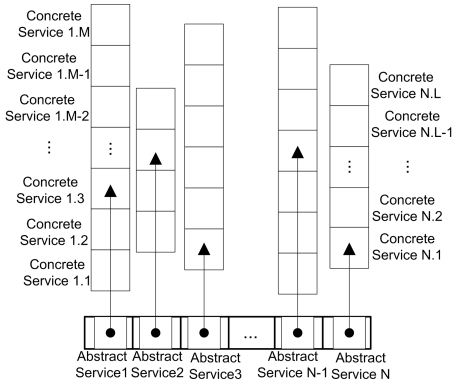

Figure 2: Solution encoding

3.1. Solution Encoding

A suitable encoding of solutions is needed in order to apply some optimization techniques. This report uses a structure similar to the vector-based encoding described in [7]. Solutions are encoded as a vector of integer values, with a size equal to the number of tasks $|T|$. Value j at position i of this vector represents the service $s_{i,j}$ selected from S_i for task t_i, encoding a pair $\langle t_i, s_{i,j} \rangle$, allowing us to encode execution plans $\chi = \{\langle t_1, s_{1,j} \rangle, \ldots, \langle t_n, s_{n,l} \rangle\}$. In figure 2 we show this structure graphically.

3.2. Construction Phase

In our adaption, a feature $f_k = \langle t_i, s_{i,j} \rangle$ represents the use of specific service $s_{i,j}$ for a task t_i. Thus, the solution χ is built by choosing a service $s_{i,j}$ for a task t_i at each iteration. The concrete task t_i is chosen randomly from the tasks that remaining unassigned in the solution.

At each iteration, a *Restricted Candidate List* (RCL) is generated, containing a subset of the candidate services $RCL \subseteq S_i = \{s_{i,1}, \ldots s_{i,n}\}$ for the task t_i at hand. The greedy function $g : S_i \longrightarrow \mathcal{R}$ provides a value for each candidate service in S_i, driving the selection of the elements contained in the RCL. Specifically, the scheme proposed in [36] is used for RCL building.
Thus, at each iteration $RCL = \{s_{i,j} \in S_i | g(s_{i,j}, \chi^i) \geq \underline{g^i} + \alpha \cdot (\overline{g_i} - \underline{g^i})\}$, where $\overline{g_i} = max\{g(s_{i,j}, \chi^i) | s_{i,j} \in S_i\}$ and $\underline{g^i} = min\{g(s_{i,j}, \chi^i) | s_{i,j} \in S_i\}$, being α a parameter of the algorithm. The feature to add to the current solution at each iteration of the construction phase is chosen randomly. At this step, if the partially built solution χ^i and chosen feature violate a service interdependence constraint, the chosen feature is discarded, and the feature that satisfies the constraint is used.

The construction phase is extremely important for GRASP success [35, 33], since it must provide a proper balance between diversification and intensification in the search. The factors that affect this balance are the value of α and the specific greedy function g used. The next subsections describe how seven different greedy functions were defined and tested in a preliminary experiment in order to ensure such a balance.

3.2.1. Greedy function G1

This greedy function is "miopic" and unadaptive, meaning that it only considers the QoS value of each service, ignoring the current solution under construction:

$$G1(s_{i,j}, \chi^i) = \sum_{q \in Q} w_q \cdot F'_q(s_{i,j}) \tag{12}$$

Where χ^i means the solution under construction at iteration i.

3.2.2. Greedy function G2

This greedy function uses the margin to reach meeting the constraints of the problem, once the service is added to the execution plan (our solution). This function ignores QoS weights:

$$G2(s_{i,j}, \chi^i) = D_f(\chi^i) - D_f(\chi^i \cup \langle t_i, s_{i,j} \rangle) \tag{13}$$

In order to evaluate D_f, a random solution is generated at the beginning of the construction phase, that is used to complete the choices for unassigned tasks.

3.2.3. Greedy functions G3,G4 and G5

Those greedy functions are linear combinations of G1 and G2 based on a parameter β:

$$GX(s_{i,j}, \chi^i) = \beta \cdot G1(s_{i,j}, \chi^i) + (1 - \beta) \cdot G2(s_{i,j}, \chi^i) \text{ where } X \in \{3, 4, 5\} \tag{14}$$

where G3 implies $\beta = 0.25$, G4 implies $\beta = 0.5$ and G5 implies $\beta = 0.75$.

3.2.4. Greedy function G6

This greedy function is based directly on the improvement of the objective function:

$$G6(s_{i,j}, \chi^i) = FinalQoS(\chi^i \cup \langle t_i, s_{i,j} \rangle) - FinalQoS(\chi^i) \tag{15}$$

In order to evaluate FinalQoS, a random solution is generated at the beginning of the construction phase, that is used to complete the choices for unassigned tasks.

3.2.5. Greedy function G7

This greedy function evaluates the improvement global QoS of the partially chosen solution, by assigning a virtual value of 0 to the QoS values of unassigned tasks:

$$G7(s_{i,j}, \chi^i) = GlobalQoS(\chi^i \cup \langle t_i, s_{i,j} \rangle) - GlobalQoS(\chi^i) \tag{16}$$

having

$$F_q(s_{i,j}) = \begin{cases} F_q(s_{i,j}) & \text{if} \quad \langle t_i, s_{i,j} \rangle \in \chi^i \\ 0 & \text{otherwise} \end{cases}$$

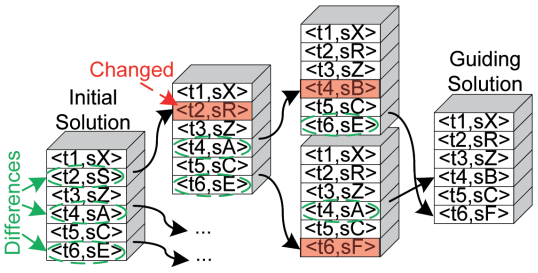

Figure 3: Path Relinking between two execution plans of the "Travel Planner" Composite Service

3.3. Choosing a Construction method

A preliminary experiment was performed in order to choose a construction method . In this experiment, both the values of α and the greedy function G were explored. Brief conclusions show that the best results were obtained for $\alpha = 0.25$, and that on average the best greedy function was $G6$. Moreover, for problem instances with more than two constraints, $G2$ performs better than $G6$, and $G1$ also performs well for unconstrained instances. Details of this experiment and its results are described in detail in appendix A.

3.4. GRASP Improvement Phase

The GRASP improvement phase typically consists of a local search procedure based on an neighborhood definition. In this paper, an execution plan χ' is a neighbor of other execution plan $\chi = \{\langle t_1, s_{1,i}\rangle \ldots, \langle t_n, s_{n,j}\rangle\}$ if $\exists(k) | \langle t_k, s_{k,l}\rangle \in \chi' \wedge \langle t_k, s_{k,m}\rangle \notin \chi \wedge \forall(p) | p \neq k) \bullet \langle t_p, s_{p,q}\rangle \in \chi \Rightarrow \langle t_p, s_{p,q}\rangle \in \chi'$. Thus, the neighborhood of execution plan χ is composed by all possible execution plans for this QoSWSC problem instance such as those that have exactly $n - 1$ pairs $\langle t_i, s_{i,j}\rangle$ in common with χ; i.e. have the same services selected for each task except for one t_i which has a different candidate service associated to it.

Our local search procedure is a simple hill climbing algorithm where only a percentage γ of the neighborhood is explored (in order to reduce the size of the neighborhood). The solutions explored until completion of this percentage are randomly chosen.

3.5. Path Relinking

Path relinking is an intensification strategy that generates new solutions by exploring trajectories that connect high quality solutions. It starts from one of these good solutions, which is called an initiating solution, and generates a path in the neighborhood space that leads toward the other solutions, called guiding solutions [26].

Figure 1 shows the application of path relinking between two execution paths of the travel planning example, which is depicted in figure 3. Three differences between task assignments are identified using our neighborhood definition; leading to three neighbors to the initial solution, that will conform paths reaching to the guiding solution.

Our implementation of path relinking is similar to that presented in [33], and it has two phases. In the first one a set of elite solutions is generated with GRASP. Instead of

retaining only the best solution overall when running GRASP, this phase stores the best solutions obtained with the method. In the second phase we apply the relinking process. In order to reduce the computational cost of the search associated to the relinking we have limited the number of paths N_{paths} relinked in each iteration, and the number of neighbors $N_{neighbors}$ explored in each pair. This allows us to perform more iterations of this technique in rebinding scenarios where execution time is limited, obtaining more control in the balance between exploration and exploitation of the search space. N_{paths} and $N_{neighbors}$ become parameters of our implementation of Path Relinking.

Algorithm 3 Path Relinking Pseudocode

$eliteSolutions \leftarrow generateEliteSolutions(nEliteSols)$
$pairs \leftarrow \oslash$
$bestSolution \leftarrow \oslash$
$bestEval \leftarrow -\infty$
for all $candidate \in eliteSolutions$ **do**
  if $f(candidate) > bestEval$ **then**
    $bestEval \leftarrow f(candidate)$
    $bestSolution \leftarrow candidate$
  end if
end for
{Main Loop}
repeat
  $pairs \leftarrow choosePairsToRelink(N_{paths}, eliteSolutions)$
  for all $pair \in pairs$ **do**
    $x \leftarrow pair.originSolution$
    $neighoursExplored \leftarrow 0$
    repeat
      $x \leftarrow RelinkMove(x, pair.guidingSolution)$
      $nextEval \leftarrow f(x)$
      if $nextEval < bestEval$ **then**
        $bestEval \leftarrow nextEval$
        $bestSolution \leftarrow x$
      end if
      $neighoursExplored \leftarrow neighoursExplored + 1$
    until $x = pair.guidingSolution$ OR $neighboursExplored \geq N_{neighbours}$
  end for
  $eliteSolutions \leftarrow updateEliteSolutions(eliteSolutions)$
until Termination Criteria is satisfied
return $bestSolution$

4. Experimentation

The aim of the experimentation is to compare the performance of the algorithms proposed in this paper with the previous proposals described in literature to solve the

QoSWSC Problem. In order to perform this comparison, we next enumerate the most relevant metaheuristic proposals in literature to solve the QoSWSC Problem.

4.1. Previous Proposals

In this section we provide a brief description of the previous metaheuristics-based proposals for solving the QoSWSC problem. These proposals are used in the experiments of this study. Section 5 provides a more general and exhaustive review of the techniques used for solving this problem in the literature.

4.1.1. Genetic Algorithms

Genetic algorithms maintain a population of individuals that represent solutions. This population evolves along generations until a termination criterion is reached, where individuals that represent better solutions survive and generate new individuals. During each generation, operators such as mutation or crossover are applied to the individuals generating changes in the individuals and the whole population. Genetic Algorithm (GA) is the metaheuristic technique most widely applied to the QoSWSC problem. In some approaches (such as [7]) a single objective function is used to compute a global measure of QoS, in other cases [13, 39] a multi-objective optimization is performed, obtaining an estimation of the Pareto front using each QoS property as a different objective function.

In this report we have implemented the proposal described in [7]. In particular, the initial population is generated randomly and a standard two-points crossover operator [15] is used, interchanging subsections of each parent solution and selecting randomly one of the generated child.

The implemented mutation operator selects the concrete service $s_{i,j}$ used to implement a task t_i in the composition solution and exchanges it for a different service $s_{i,k}$ where $k \neq j$ is chosen randomly from the candidate set of this task. Parameters of this technique are: *population size, percentage of population that generates offspring, probability of mutation for each selected service and criteria to select survival and crossover individuals*. Parameters are chosen according to [7] as shown in table 4, *roulette wheel selection* is used, where the best 2 individuals are kept alive across generations.

4.1.2. Hybrid Tabu Search with Simulated Annealing

Ideas behind Tabu Search(TS) were proposed by Glover in [18]. This technique uses procedures designed to cross boundaries of local optima by establishing an *adaptive memory to guide the search process*, avoiding searching in circles through the solution space. A hybrid of TS with Simulated Annealing (SA) was proposed in [24] for solving the QoSWSC problem. The original proposal was aimed at finding feasible solutions of hardly constrained instances of the problem -the search was guided by constraint meeting distance-. A modification has been carried out to support optimization using our objective function (eq. 6), making it comparable with the rest of the proposals. Since our implementation contains a significant modification of the original proposal, we provide the description of the implemented technique in detail:

Algorithm 4 Hybrid Tabu Search with Simulated Annealing pseudocode

$currentSolution \leftarrow createInitialSolution()$
$currentEval \leftarrow f(currentSolution)$
$bestSolution \leftarrow currentSolution$
$bestEval \leftarrow currentEval$
$\tau \leftarrow initialTemp$
{Main loop}
repeat
  $nextSolution \leftarrow generateNeighbor(currentSolution)$
  $nextEval \leftarrow f(nextEval)$
  if $nextEval < bestEval$ **then**
    $bestSolution \leftarrow nextSolution$
    $bestEval \leftarrow nextEval$
  end if
  if $acceptanceCriteria(curentEval, nextEval, \tau)$ **then**
    $makeTabu(currentSolution, nextSolution)$
    $currentSolution \leftarrow nextSolution$
    $currentEval \leftarrow nextEval$
  end if
  $\tau \leftarrow updateTemperature(\tau)$
until Termination Criteria is satisfied
return $bestSolution$

Algorithm 5 Neighbor generation algorithm

$unmetConstraints \leftarrow arrangeByMeetingDistance(\Omega_g, currentSolution)$
$\leftarrow currentSolution$
if $unmetConstraints = \oslash$ **then**
  $q \leftarrow arrangeByAvgQoSDistance(W, Q, currentSolution)$
  $nextSolution \leftarrow improve(curretSolution, q)$
else
  for all $constraint \in unmetConstraints$ **do**
    $q \leftarrow constraint.property$
    $nextSolution \leftarrow improve(curretSolution, q)$
  end for
end if
return $nextSolution$

The main body of the algorithm is based on simulated annealing, as shown in alg. 4. Given a solution, the algorithm creates a neighbor and accepts it depending on the difference in the objective function with a probabilistic rule. The modification operation is recorded in the tabu list. While generating the neighbor, the algorithm prevents the use of the operations in the tabu list so as not to return to solutions previously visited. Specifically, the neighbor generation step focuses on a concrete QoS property for impromevent. In the original proposal this step is guided by the global constraints violated, and it selects its corresponding QoS property for improvement until constraint meeting. When all global constraints are met the original proposal described in [24] ends. The modification proposed in this paper is applied when all global constraints are met in order to continue optimizing. Under those circumstances, our modification uses the difference between the QoS value of current solution $\Psi_q(\chi, \Phi, R)$ and the average QoS for this property Avg^q for choosing the QoS property to improve. Specifically, the QoS property selected to guide the improvement is the one minimizing $s * (\Psi_q(\chi, \Phi, R) - Avg^q) * w_q$, where s is 1 if q is positive and -1 if it is negative. Thus our modification tries to generate a neighbour improving the solution in the QoS property with the bigger improvement room and importance for the users. This value is computed in the pseudocode of alg. 2 by the subroutine $arrangeByAvgQoSDistance$ in line 7.

4.2. Experimental Setting

A java implementation was developed in order to evaluate our proposal. The algorithm was developed on top of the FOM optimization framework [31], an object oriented framework that reduces the burden of implementation when solving optimization problems using metaheuristics and also provides experimentation capabilities [22]. Since FOM is designed to solve minimization problems, the implementation uses an objective function that subtracts the value of *FinalQoS* (as described in equation 6) to 1.0. This objective function has the same properties that the original one, but transforms the QoSWSC problem into a minimization problem.

The experiments were performed on a computer equipped with a Intel Core I7 Q870 CPU with 8 cores working at 1.87 Ghz, running Windows 7 Professional 64bits and Java 1.6.0.22 on 8 GB of memory. An implementation of the Mersenne Twister random number generator [27], available at `http://www.cs.gmu.edu/~sean/research/mersenne/MersenneTwister.java` was used for generating the random numbers in the experiments of this paper. Both FOM and our implementation of the algorithms are public and can be downloaded (cf. materials section).

4.3. Experiment

The aim of this experiment is to compare the performance of our proposal and previous ones in terms of the QoS of solutions they provide. Previous proposals (as described in sec. 4.1) are compared to ours when solving a number of instances of the QoSWSC problem. Specifically, we compare Genetic Algorithms (GAs) and Hybrid Tabu Search with Simulated Annealing (TS/SA), with a GRASP using G1 (GRASP(G1)), and two variants of GRASP with Path Relinking. These two variants use G2 and G6, and are

labeled GRASP+PR(G2) and GRASP+PR(G6) respectively. The choice of our proposals was performed based on the preliminary results, since G2 has shown the best results for constrained instances of the problem, G1 works well for unconstrained ones and G6 provides the best results in general. The parameters used for each technique are described in table 4, those values were chosen based on the experiments reported in literature for previous proposals and on our second preliminary experiment for GRASP and GRASP+PR. The details of this experiment and its results are described in appendix B. The fact that the number of paths per iteration is 2 and the number of neighbors to be explored per path is 50 stands out. Thus, on each iteration of the Path Relinking a maximum number of 100 solutions on the paths are explored, making its computational cost similar to the iterations of the Genetic Algorithm, where a population size of 100 individuals is used.

4.3.1. Experiment design

Since we want to compare the performance of techniques, in this experiment we use the final QoS (eq. 6 of the best solutions found for each technique as the dependent variable. The termination criteria of optimization was a maximum execution time, that we increased exponentially in order to evaluate its effect on the performance of the techniques. The values of the execution time are directly related to the binding and rebinding scenarios or QoS-aware Sevice Composition we focus on. In rebinding scenarios, a solution should be provided in fraction of seconds or seconds at maximum, since the time invested on finding the optimum is a capital issue. Thus maximum execution times used as termination criteria are: 100ms, 200ms, 500ms,1000ms,10000ms and 50000ms. These values cover most of the re-binding scenarios where composition users (either human or automated systems) are not willing to wait minutes for an optimal composition.

Eleven problem instances were generated by the algorithm described in C, using the parameters shown in table 3. As table 9 shows (third column) those parameters are common in the literature on the QoSWSC Problem. The basic characteristics of problem instances generated are shown in table 5

.

4.3.2. Experiment conduction and Results

For each combination of technique, problem instance and maximum execution time, thirty runs were performed in order to ensure the significance of results. Table 6 shows the results obtained for these experiments. It specifically shows the means of each technique for each problem instance and execution time . The differences between our proposals and previous ones shown in this table are important.

In order to ensure that the differences between our proposals an the previous approaches are significant, we repeated the experiment using the objective function defined in ([9]). The definition of that objective function using our notation is:

$$FinalQoS_{Canf}(\chi) = \frac{\sum_{q \in Q^+} w_q \cdot \Psi_q(\chi, \Phi, R)}{\sum_{q \in Q^-} w_q \cdot \Psi_q(\chi, \Phi, R)} - w_{unf} \cdot D_f(\chi) \tag{17}$$

Table 3: Problem instances generation parameters

Composition Structure Parameters	Number of Activities	Uniform distribution between 10 and 100
	Percentage of Control Flow Activities	Uniform distribution between 20% and 50%
	Percentage of Loops	45%
	Percentage of Branches	45%
	Percentage of Flows	10%
	Maximum nesting level	Uniform distribution between 5 and 10
Runtime Information Parameters	Iterations per Loop	Gaussian dist. with mean=18 and StdDev=6
	Probability of Branches	Randomly Chosen
Candidate Services Parameters	Candidates per Task	Uniform distribution between 1 and 10
	Cost	Uniform distribution between 0.2 and 0.95
	Exec. Time	Gaussian distribution with Mean=0.5 and StdDev=0.4
	Reliability	Uniform distribution between 0.3 and 0.9
	Availability	Uniform distribution between 0.9 and 0.99
	Security	Uniform distribution between 0.6 and 0.99
Constraints Parameters	Number of Constraints	Uniform distribution between 0 and |Q|
	% of optimality	Unifform distribution between 25% and 75%
Objective Function Parameters	w_{unf}	0.5
	w_{Cost}	0.3
	$w_{ExecTime}$	0.3
	w_{Avail}	0.1
	w_{Sec}	0.2
	w_{Rel}	0.1

In order to implement and execute this new experiment in FOM, we need to transform the QoSWSC problem of [9] into a minimization one. Thus, the objective function used in our experiments is:

$$FinalQoS^{min}_{Canf}(\chi) = \frac{\sum_{q \in Q^-} w_q \cdot \Psi_q(\chi, \Phi, R)}{\sum_{q \in Q^+} w_q \cdot \Psi_q(\chi, \Phi, R)} + w_{unf} \cdot D_f(\chi) \tag{18}$$

We generated 11 new problem instances with the same parameters and algorithm and executed the experiment using this objective function. The information of these additional problem instances are shown in table 7.

The results obtained for this alternative execution of the experiment are shown in table 8. In both tables (6 and 8) the best mean for each problem and execution time is highlighted in boldface (remember that the objective functions were transformed to get minimization problems).

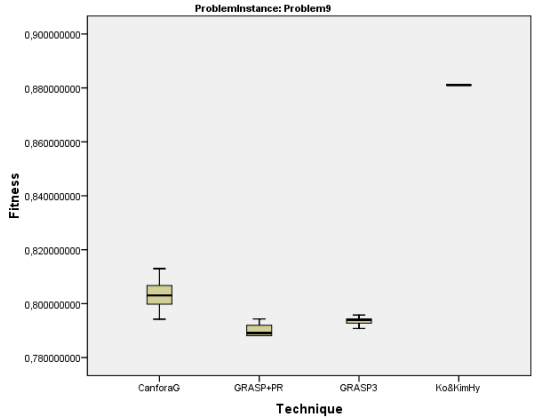

Figure 4: Box plot showing the results of each technique for Obj. function:1.0 - FinalQoS (eq. 6) and Problem instance 9.

Figures 4.3.2 to 4.3.2 depict box plots showing the results of each technique for different problem instances and experimental configurations. All the box plots correspond

Table 4: Parameters of the techniques used in the experiment

Technique	Paramenter	Value
GRASP (G1)	α	0.25
	Greedy Function	G1
	LocalSearch	SteepestDescent with 20 % of neighborhood exploration
GRASP+PR (G6/G2)	α	0.25
	Greedy Function	G6 / G2
	LocalSearch	SteepestDescent with 20 % of neighborhood exploration
	Number of Elite Solutions	5
	Number of Paths per Iter.	2
	Number of neighbors to be explored per path	50
	Intial Grasp Iterations (number of grasp iterations prior to the intesification phase)	50
Canfora´s GA [9]	Population Size	100
	Crossover	0.70
	Mutation Prob.	0.01
	Survival Policy	The two best individuals survive in the next generation
	Selector	Roulette Wheel
	Initial Population	Randomly generated
Hybrid TS+SA [24]	Initial Solution	Using local optimization as described in [24]
	Number of services exchanged for improvement	2, as describen in [24]
	Tabu Memory	Recency based memory as a tabu list (as in [24])
	Tabu list size	100 movements (authors do not provide the values used in [24])
	Acceptance Criteria	Based on the curren iteration (as describen in [24])

to an execution time of 500ms and different objective functions are used; 1.0 - Final QoS as defined in 6, is used in subfigures 4.3.2 and 4.3.2; and $FinalQoS^{min}_{Canf}$ is used in subfigures 4.3.2 and 4.3.2. In these figures GA is labeled as CanforaG, GRASP+PR(G6) as GRASP+PR, GRASP(G1) as GRASP, and TS/SH as Ko&KimHy. GRASP+PR(G6) has the best results on all figures.

4.3.3. Analysis and Interpretation of Results

As shown in table 6 GRAPS+PR(G6) has obtained the best mean results for all problem instances an execution times when using our fitness function (eq. 6). GRAPS+PR(G2) obtains good mean results but is worse than GRAPS+PR(G6) in general. The performance of TS/SA was not good except for highly constrained instances.

The following describes how, in order to ensure the statistical significance of these results, we performed statistical tests for each execution time and problem instance. First, normality tests were performed on results, showing the they do not follow a normal distribution (we used Kolmogorov-Smirnov tests). Non-parametric statistical tests were therefore used in the remainder of the study.

Kruskal-Walls and Friedman tests were then performed, having as null hypothesis that there is no difference on the QoS of solutions provided by the techniques. This

Table 5: Problem Instances information

Problem Name	Activities (tasks and building blocks)	Abstract Services (tasks)	Candidate Services	Global Constraints
Problem 0	72	55	220	0
Problem 1	89	46	92	2
Problem 2	51	34	170	4
Problem 3	25	15	75	2
Problem 4	82	51	102	1
Problem 5	47	3	68	0
Problem 6	79	42	252	1
Problem 7	12	7	63	3
Problem 8	54	37	74	4
Problem 9	24	18	144	4
Problem 10	58	41	82	4

Table 6: Means of obj. func. values (1.0 - value of eq. 6) for each algorithm and execution time

Execution Time	100					1000				
Technique	GA	GRASP+PR (G6)	GRASP+PR (G2)	GRASP(G1)	TS/SA	GA	GRASP+PR (G6)	GRASP+PR (G2)	GRASP(G1)	TS/SA
Problem 0	0,317053066	**0,31467194**	0,31559766	0,31585178	0,37823648	0,31702924	**0,31464257**	0,31514186	0,31521718	0,37819911
Prolem 10	0,333850406	**0,33271258**	0,3329004	0,33326753	0,34420161	0,33372791	**0,33268621**	0,33266052	0,33268679	0,34393207
Problem 1	0,832070664	**0,82958546**	0,83114955	0,82996641	0,90526782	0,83257414	**0,82958531**	0,82991653	0,82992594	0,90526782
Problem 2	0,314220241	**0,30428238**	0,30821952	0,30961194	0,40335166	0,31406676	**0,3033425**	0,30495659	0,30548735	0,4033507
Problem 3	0,786458899	**0,7733251**	0,77774798	0,77939848	0,87387101	0,78422132	**0,77294846**	0,77398728	0,77478427	0,87377022
Problem 4	0,810939436	**0,81066853**	0,81082234	0,81086532	0,81292188	0,81094311	**0,81066853**	0,81072816	0,81073885	0,81291786
Problem 5	0,345341638	**0,33979296**	0,34254369	0,34201717	0,39219569	0,34514013	**0,33978323**	0,34021847	0,34087602	0,39217004
Problem 6	0,814693606	**0,79437721**	0,80665351	0,7956466	0,89469352	0,81558663	**0,79244253**	0,79865127	0,7956466	0,894643
Problem 7	0,755621698	**0,74596884**	0,74604446	0,745972	0,82326859	0,75901268	**0,74549613**	0,74544938	0,74549475	0,82099777
Problem 8	0,85914249	**0,85159186**	0,85524606	0,85185832	0,91504732	0,85937275	**0,85159186**	0,85210501	0,85185832	0,91504127
Problem 9	0,802587993	**0,78813945**	0,79375533	0,79500608	0,88106812	0,80275109	**0,78810276**	0,79004525	0,79100871	0,88106812
Execution Time	500					50000				
Problem 0	0,316847884	**0,31465716**	0,31541106	0,31559366	0,37819911	0,31708794	**0,31464248**	0,3150353	0,31511347	0,37819911
Prolem 10	0,334067627	**0,33271171**	0,3327601	0,33286655	0,34393207	0,33395456	**0,33265462**	0,33264226	0,33265207	0,34393207
Problem 1	0,83218579	**0,82958531**	0,83047811	0,82996641	0,90526782	0,83234986	**0,82958531**	0,82979438	0,82981234	0,90526782
Problem 2	0,314327155	**0,30334881**	0,30667738	0,30770218	0,4033507	0,3145635	**0,3033425**	0,30436232	0,30472537	0,4033507
Problem 3	0,785376182	**0,7730331**	0,77626168	0,77730067	0,87377022	0,78437278	**0,77284804**	0,77318822	0,77396728	0,87377022
Problem 4	0,810937976	**0,81066853**	0,8107727	0,81081414	0,81291786	0,81090575	**0,81066853**	0,81070306	0,81072227	0,81291786
Problem 5	0,345156788	**0,33979296**	0,34138343	0,34179962	0,39217004	0,34478886	**0,33977041**	0,33982145	0,34035985	0,39217004
Problem 6	0,815804062	**0,79307634**	0,80427884	0,7956466	0,894643	0,81608455	**0,79238052**	0,79697338	0,7956466	0,894643
Problem 7	0,756262913	**0,74577745**	0,74566567	0,74580965	0,82099777	0,75804813	**0,74542966**	0,74541133	0,74542556	0,82099777
Problem 8	0,85904663	**0,85159186**	0,85385226	0,85185832	0,91504127	0,85909064	**0,85159186**	0,85185755	0,85183389	0,91504127
Problem 9	0,803349334	**0,78810276**	0,79208361	0,79353237	0,88106812	0,8018389	**0,78810276**	0,78949346	0,78994663	0,88106812

Table 7: Additional problem instances information

Problem Name	Activities (tasks and building blocks)	Abstract Services (tasks)	Candidate Services	Global Constraints
Problem C0	41	33	64	0
Problem C1	46	29	84	1
Problem C2	40	32	279	0
Problem C3	46	27	78	0
Problem C4	78	52	459	0
Problem C5	64	48	94	2
Problem C6	12	8	63	0
Problem C7	82	51	450	2
Problem C8	58	35	136	1
Problem C9	61	35	170	4
Problem C10	29	22	42	5

Table 8: Means of obj. func. values (using equation 18) for each algorithm and execution time

Exec. Time	100					1000				
Technique	GA	GRASP+PR (G6)	GRASP+PR (G2)	GRASP(G1)	TS/SA	GA	GRASP+PR (G6)	GRASP+PR (G2)	GRASP(G1)	TS/SA
Problem 0	20,3293962	**18,1494425**	19,0278313	18,8088962	19,456701	20,2537398	**18,1293883**	18,794495	18,3892283	19,456701
Problem 10	21682,8585	**20345,8959**	20448,2644	20392,9211	26421,6496	21699,4216	**20183,325**	20295,5048	20228,4809	26421,6496
Problem 1	17546,925	**16798,8826**	16892,6761	16883,4022	18028,5566	17344,5197	**16795,3229**	16836,7681	16799,8261	18028,5566
Problem 2	77,46346587	53,3737008	69,3313881	50,6205653	**47,227387**	77,8725994	49,1226642	62,7275717	50,6205653	**47,227387**
Problem 3	365838,7379	**354607,163**	357263,572	357324,957	381218,113	366237,943	**353935,122**	355399,043	353959,67	381218,113
Problem 4	4660,050302	**2688,86261**	4032,72479	2817,36132	2758,64294	4729,28151	**2379,87803**	3529,88439	2817,36132	2758,64294
Problem 5	43077,71302	40087,5854	41927,616	**39712,0757**	39804,2874	43157,9618	40039,7574	40893,915	**39712,0757**	39804,2874
Problem 6	504,0981333	353,88043	367,833198	**347,891644**	348,164253	499,411777	354,966396	368,256943	**344,494453**	348,164253
Problem 7	29445,20424	32899,8809	25257,0317	**19163,0773**	20070,3702	28586,1747	**15381,4457**	19556,9182	18875,8913	20070,3702
Problem 8	623,4833397	**590,797592**	604,896714	605,995777	653,162126	622,908984	**556,900895**	581,62854	574,998387	653,162126
Problem 9	141414,7664	**129780,875**	135381,216	133877,301	144955,387	143082,565	**125785,154**	129145,809	128143,633	144955,387
Exec. Time	500					50000				
Problem 0	20,22673263	**18,1300329**	18,7870556	18,7037792	19,456701	20,5017695	**18,127089**	18,7918048	18,2799535	19,456701
Prolem 10	21803,12505	**20189,7876**	20295,9563	20295,1878	26421,6496	21672,8291	**20204,3569**	20282,922	20213,8838	26421,6496
Problem 1	17538,62571	**16793,7861**	16853,8274	16814,345	18028,5566	17444,8214	**16789,8868**	16841,7142	16789,4079	18028,5566
Problem 2	77,29528988	50,7654103	64,8281533	50,6205653	**47,227387**	79,1106454	47,3028287	63,42012	50,6205653	**47,227387**
Problem 3	371234,6973	**354238,966**	354854,303	354498,994	381218,113	368806,714	**353530,084**	354685,579	353517,13	381218,113
Problem 4	4717,700463	**2522,78602**	3846,38906	2817,36132	2758,64294	4591,26167	**2474,05301**	3550,55124	2817,36132	2758,64294
Problem 5	43167,43732	40087,5854	40992,9427	**39712,0757**	39804,2874	43269,8455	40228,1459	40794,3524	**39712,0757**	39804,2874
Problem 6	512,1531946	352,35142	368,461442	**347,243117**	348,164253	513,960347	360,843707	377,25881	**340,969439**	348,164253
Problem 7	27976,64044	**16228,1583**	22018,1279	19156,5452	20070,3702	28453,8342	**14439,0012**	19615,3699	18326,1133	20070,3702
Problem 8	621,2869551	**565,994332**	593,664127	591,794515	653,162126	623,626913	**557,202175**	584,994824	568,967591	653,162126
Problem 9	143803,898	**126129,215**	131166,647	130685,055	144955,387	142111,79	**125284,279**	129281,01	127287,341	144955,387

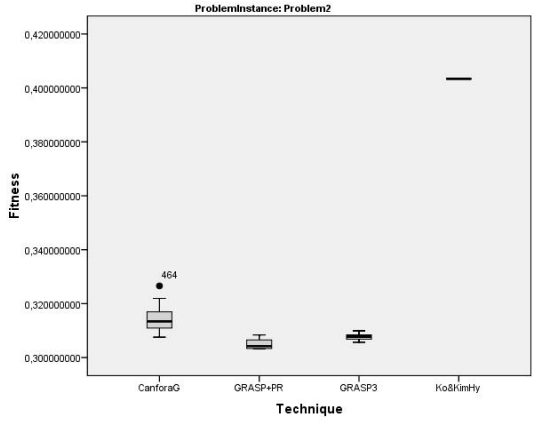

Figure 5: Box plot showing the results of each technique for Obj. function: 1.0 - FinalQoS (eq. 6) and Problem instance 2.

hypothesis was rejected on all cases for both tests, thus, there are significant differences on the performance of the techniques. Mann-Withney tests were then carried out in order to ensure the significance of the differences between GRASP+PR(G6) and GAs; results showed that differences of performance are significant in all cases (null hypotheses were all rejected). The same happens for GRASP+PR(G6) and TS/SA. Last of all, a test comparing (GRAPS+PR(G6)) and GRASP(G1) was carried out. Results show that differences of performance are significant for all problem instances except for Problem 7. For this problem the tests did not reject null hypotheses for execution times greater than 500ms. This problem is significantly smaller than the rest, since it contains only seven tasks and 63 candidate services. Thus, we infer that for those smaller instances of the problem, GRASP(G1) can behave nearly as well as GRASP+PR(G6). Authors believe that the root cause of this is that the intensification strategy of Path Relinking is not as effective as with bigger problem instances, because a bigger percentage of the promising area of the solution space can be explored directly by the GRASP.

With regards to the results obtained using the objective function defined in eq. 18,

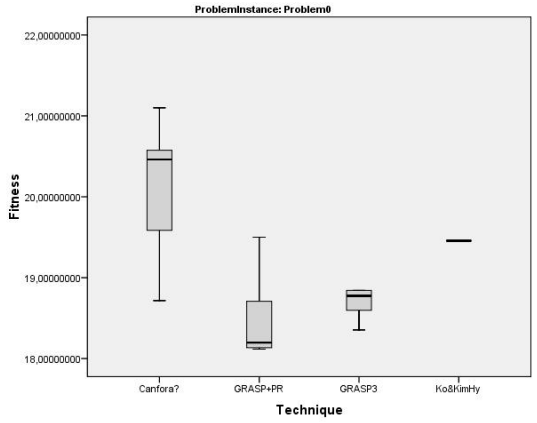

Figure 6: Box plot showing the results of each technique for Obj. function: $FinalQoS^{min}_{Canf}$ (eq. 18) and Problem instance C0.

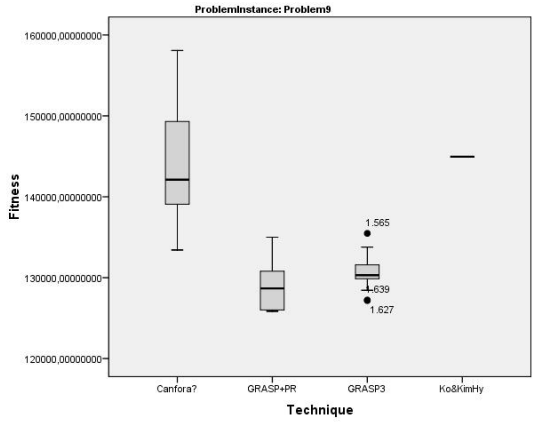

Figure 7: Box plot showing the results of each technique for Obj. function: $FinalQoS^{min}_{Canf}$ (eq. 18) and Problem instance C9.

GRASP+PR(G6) shows again the best mean results for most of the problem instances. However, there is a problem instance (C2) for which the best results were obtained by TS/SA. Additionally, GRASP(G1) found the best solutions for two problem instances (C5 and C6).

The same statistical study was performed for the results obtained using the objective function defined in eq. 18. The results did not follow a normal distribution (Kolmogorov-Smirnov tests were performed), so non-parametric statistical tests were used. Mann-Whitney tests show that for problem (C2) and short execution times, the difference of performance between GRASP+PR(G6) and TS/SA is significant. However, the p-values of the difference between TS/SA become bigger with longer execution times, where the difference of performance is not significant for execution time 50000ms.

One of the most surprising results of the study is the low dispersion of results obtained by the TS/SA algorithm between different executions, specially when using the objective function of eq. 18. Authors logged the execution of this technique in order to determine the origins of these results. One of the causes of these results was the acceptance criteria defined: according to [24], the probability of acceptance of a solution as the next current solution for search is $exp((F(s) - F(s')) \cdot iteration)$. For functions

with large scale values such as eq. 18, it is rare to choose solutions that do not improve, since if there is a large difference in the objective function and the number of iterations increase, the probability of accepting them becomes extremely small. Therefore, the algorithm only chooses improving neighbors very early in the optimization process. Furthermore, for constrained problem instances the criteria of constraint satisfaction prevails, since it is applied earlier than the optimization criteria. Under these circumstances, the neighbor generation subroutine tends to always use the same properties for improvement, thus leading the search to similar areas in the search space. Finally, the deterministic initial solution generation method (local optimization based initialization, in Table 4 of [24]), also contributes to generate the nearly constant results.

It is remarkable that results about the improvement provided by our proposals are not only significant in a statistical sense but also for the real consequences of QoSWSC problem solving. In this sense, the QoS of solutions provided by our proposals are better than those provided by previous ones. As a motivating example, the QoS of solutions provided by GRASP+PR(G6) for problem instance C4 are 49.25% and 28% better on average than those provided by GAs and TS/SA respectively. These improvements are important for users when translated into cost savings and execution time decrease.

Note that execution times 200ms and 10000ms were omitted from results in order to have tables of an affordable size. Results for these execution times were similar to those shown for 500ms and 50000ms respectively. All the raw data, basic analysis and descriptive statistics, and statistical analysis files are available for download, in csv/txt, MS Excel and SPSS v17 formats respectively (cf. materials section at the end of the paper).

4.4. Threats to Validity

In order to clearly delineate the limitations of the experimental study, we next discuss internal and external validity threats [23].

Internal validity refers to whether there is sufficient evidence to support the conclusions and the sources of bias that could compromise those conclusions. In order to ensure validity of the experimental approach, experiments were performed in a randomized order on the same isolated and exclusively dedicated computer, and runs were replicated 30 times for each experimental configuration and technique. Moreover, statistical tests were performed to ensure significance of differences between the results of our proposals and previous ones. More specifically, as detailed in Section 4.3.3, Kurskal-Walls, Friedman and Mann-Withney U tests were performed. Results of tests shown that differences on performance between our proposals and previous ones are in general significant.

External validity is concerned with how the experiments capture the objectives of the research and the extent to which the conclusions drawn can be generalized. This can be mainly divided into:

- **Limitations of the approach**. Experiments showed that no significant difference exists between GRAPS+PR(G6) and GRASP (G1) when using our algorithm with problems of low complexity, i.e. problem with less than 10 tasks and 100 candidate services, such as problem instance 2. Moreover, results shown in this study

cover only execution times in the context of rebinding or binding at run-time. Thus results don´t apply for longer execution times. Additional experimentation must be performed for those cases, but those experiment are out of the scope of this work, and are included in conclusions as future work.

- **Generalizability of the conclusions**. In our experiments, we used 22 different problem instances covering the range of parameters used in literature (cf. third column of table 9). Moreover, in order to ensure the relevance and generalizability of results, we performed the experiments both using the objective function defined by authors (eq. 6) and other previously defined in literature (eq. 18). One of the problems in terms of generalizability and replicability of results is the absence of a standard library of problem instances for the QoSWSC Problem. In order to overcome this problem, we provide both the algorithm used for generating the instances used in this paper (cf. appendix C) and the set of instances used in this study (cf. Materials section). Those instances are expressed using a a plain text description that cover the full set of components of the QoSWSC problem (cf. materials section). Thus, experiments described in this study become replicable. We think that this set of instances and the generation algorithm could be used as the seed of a wider library of instances of the problem to be used as an standard in the research on this problem, in order to help creating a cohesive body of knowledge with the previous and future research on this problem.

5. Related Work

QoS-aware service composition is one of the most promising possibilities that Service Oriented technologies brings to service developers and software architects. It has in fact been identified as a main research area in the Service Orientation field [28] [25], and therefore is being addressed as a main research line by both academy and industry. QoSWSC brings the dynamic, loosely coupled service selection paradigm of service orientation to its maximum expression. This problem provides an excellent application scenario for different methods and techniques, ranging from pure optimization techniques to artificial intelligence systems. In this context, two kinds of optimization strategies have been formulated for this problem in literature[41] [4]; global and local selection.

- Local approaches have two main drawbacks: (i)Obtained solutions are suboptimal with regards to the overall quality of the composite web service. (ii) Global constraints according the structure of the composite web service and their quality properties cannot be imposed.

- Global approaches try to optimize the whole set of services used in the composition according to their QoS properties, taking into account the structure of the composition. Therefore, global QoS constraints can be formulated. In doing so, more information about the expected properties of the composition must be provided, in order to overcome the variability that could arise in the optimization process.

Year	Authors	Approach & Contribution	Experiments
2004	Zeng et al. [41]	Problem proposal and solving based on IP. Problem is formulated using loop unfolding.	Randomly generated compositions with 10 to 80 tasks (after unfolding), and 10 to 40 candidates per task (both varying with steps of 10). Global QoS Constraints and 5 QoS Properties whose values are randomly genereated.
2005	Canfora et al. [9]	Genetic Algorithm proposal for solving the QoS-Aware Web Service Composition problem. The GA-based approach is better than IP-based apporaches for small compositions (7 tasks) with more than 17 candidates per tasks (finding 95% optimal solutions).	Randomly generated compositions with 8 tasks, and 5 to 25 candidates per task (varying with steps of 5).
2007	Ardagna & Pernici. [3]	Problem reformulation, MIP (Mixed Integer Programing) solving. Usage of loop peeling instead of loop unfolding. Global constraints allowing the execution of stateful web services are introduced.	Randomly generated compositions, with up to 100 execution paths and 1000 tasks (after peelling), 50 candidates per task and 5 global constraints. 5 QoS Properties whose values were generated based on gaussian and uniform distributions (depending on the QoS property).
2008	Koa et al. [24]	Hybrid approach of Tabu Search and Simulated Annealing. The performance of the approach is compared with Zeng´s proposal [41] and result show that the new tecnique is better for large instance problems.	Randomly generated compositions with 10 to 100 tasks (after unfolding), and 10 to 50 candidates per task (both varying with steps of 10). Global QoS constraints and 6 QoS properties whose values are randomly genereated.

Table 9: Relevant Literature

Global approaches to solve the QoSWSC problem comprise:

- The use of Integer [41] [1],Linear [11] or Mixed (I/L) Programming techniques [3] [34]. These kind of approaches model the problem using integer and/or real variables and a set of constraints. Although these approaches provide the global optimum of the problem, and their performance is better for small size instances of the problem, genetic algorithms outperform these techniques for problem instances with $avg(|S_i|) > 17$ [7]. The use metaheuristics is more flexible, because those techniques can consider non-linear composition rules and different fitness function formulations [4].

- Heuristic techniques. In [21] and [14] some specific heuristics are developed to solve the service composition problem. Applications of metaheuristics to this problem are present in the literature, mainly using different genetic algorithm based approaches. These incorporate variants to the work presented in [7]; either on the encoding scheme, the fitness function or QoS model [17] [37] [40], or using population diversity handling techniques [42] [43]. In [13] and [39] a multi objective evolutionary approach is used to identify a set of optimal solutions according to different quality properties without generating a global ranking. In [32] fuzzy logic is used to relax the QoS constraints, in order to find alternative solutions when it is not possible to find any solution to the problem. Some authors have proposed the use of simulated annealing [40], but no experimental results have been provided.

6. Conclusion

In this tech report, an algorithm based on GRASP with Path Relinking for solving the QoSWSC Problem has been presented, along with experimental evidence showing that this algorithm outperforms previous proposals based on metaheuristics in rebinding scenarios.

Acknowledgment

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366), and Andalusian Government projects ISABEL (TIC-2533) and THEOS (TIC-5906). Authors are thankful to Barbara Pernici by their review and useful comments in early versions of this work.

Materials

All the source code, raw data of results and analysis of information is available for downloading online at `http://www.isa.us.es/uploads/papers/japarejo/GRASPwPRforQoSWSC/materials.zip`. It is a big file (5 MB), because it contains all the data and sources. The source code and executable classes of the experiments shown in this report are distributed in this zip file that contains a Java Maven project. In this project five different executable files can be found: (i) a basic test file used for testing our implementations named "BasicTest.java". (ii) The source code of the preliminary experiment 1 described in appendix A is implemented in a file named "Experiment-0.java". (iii) The source code of the preliminary experiment 2 described in appendix B is implemented in a file named "Experiment-1-0.java", and its corresponding results and problem instances are in a folder with the same name. (iv) The source code of the main experiments of this study are available in files "Experiment-1-1.java" and "Experiment-1-2.java", and their results and problem instances are available on folders with the same names. Problem instances are expressed in plain text files, following the format described in the "ProblemFileFormat.txt" file.

Regarding the raw data of results, a log file containing the results of the execution of each technique for each experimental configuration is available (as a .log file). Data in these files is stored with values separated by commas. A set of files containing basic descriptive statistics for each experiment is also available. Those files have extension ".txt" and contain data separated by semicolons.

Finally, advanced descriptive statistics, some charts and statistical analysis files with tests of each experiment are provided in excel and SPSS 17 formats respectively.

A. Preliminary Experiment 1

The aim of preliminary experiment 1 was to choose the parameters of the GRASP construction phase to be used in the experimentation. These parameters are two: (i) α, that controls the level of elitism versus randomness when creating the RCL; and (ii) the greedy function G, to be used in the evaluation of features at each iteration of the construction phase. This experiment is similar to those presented in [33], which we have used for inspiration.

In this experiment we assume that the extreme values of α, zero and one, are not valid options. When α is zero the constructions phase turns into a purely elitist algorithm, similar to the local optimization methods described in Section 5. When α is one, the construction method is a purely random solutions generation procedure. Consequently, in this experiment the following range of values for α is used: 0.25, 0.5, and 0.75. The range of possible greedy functions along with their formulations are described in Section 3.2, from G1 to G7.

A good greedy function should provide not only near-optimal solutions, but generate enough diversity as to explore the multiple local optima of the search space [35, 33]. A measure of diversity of solutions is defined based on our notion of neighboring solutions (cf. Section 3.4). The diversity of two solutions χ_i and χ_j to a QoSWSC Problem instance P, is defined as the number of assignments of services that have in common divided by the total number of tasks of P; i.e.:

$$Diversity(\chi_i, \chi_j) = \frac{\sum_{k=1}^{|T|} Eq(\chi_i, \chi_j, k)}{|T|} \tag{19}$$

where $Eq(\chi_i, \chi_j, k)$ returns one if the service assigned to task t_k in χ_i is the same as the service assigned to t_k in χ_j.

Thus, we can define the diversity of a set X of n solutions $X = \{\chi_1, \ldots, \chi_n\}$ as:

$$Diversity(X) = \frac{\sum_{i=1}^{N} \sum_{j=1 \wedge i \neq j}^{N} Diversity(\chi_i, \chi_j)}{N^2 - N} \tag{20}$$

A factorial design is used for this preliminary experiment, generating all the possible combinations of parameter values. A number of 11 problem instances were generated by the algorithm described in Appendix C, using the parameters shown in table 3. For each problem instance and parameters configuration, solution were generated and the mean QoS computed (as evaluated by decreasing to 1.0 the value of eq. 6, as described in Sec. 4.2 in order to have a minimization problem).

Table 10 shows both the average QoS provided by each experimental configuration, and the diversity of the value of the set of solutions generated. Moreover, problem instances were classified into hard-constrained (those with three or more constraints) and soft-constrained (those with one or zero constraints), specific results were computed for those sets (from the sixth to the tenth column). Specifically, in table 10 the best values of each column (mean and diversity) for the same value of α are in boldface, showing the greedy function that best performs for that value of alpha (and group of problems). Moreover, the best global value per column is underlined showing the best overall configuration for each problem group.

Table 10: Mean QoS and Diversity of solutions generated by each value of α and greedy function G

		ALL		SOFT-CONSTRAINED		HARD- CONSTRAINED	
Alpha	**Technique**	Mean	Div	Mean	Div	Mean	Div
0.25	**G1**	0.62474703	0.21304265	0.51731048	0.48076142	0.71427748	0.22534343
	G2	0.61333567	**0.62468482**	0.51463859	**0.47164242**	0.69558323	0.6812282
	G3	0.62534596	0.17672353	0.51775962	0.48069143	0.71500124	0.17258037
	G4	0.62492818	0.19684101	0.5172063	0.48078729	0.71469641	0.20334984
	G5	0.625128	0.19287986	0.51761126	0.48064958	0.71472529	0.19224344
	G6	**0.58125744**	0.17292293	**0.50039478**	0.49763382	**0.64864298**	0.2311484
	G7	0.61467777	0.62331184	0.51416351	0.47056025	0.69843964	**0.67886608**
0.5	**G1**	0.61831675	0.32475299	0.50481	0.4914657	0.71290571	0.39327706
	G2	0.63848977	0.61884664	0.66803946	0.29253323	**0.61386504**	0.6046426
	G3	0.61818339	0.32608253	0.50466206	0.49132503	0.7127845	0.39345729
	G4	0.61811012	0.32168708	0.50453063	0.49151367	0.7127597	0.38383705
	G5	0.61823378	0.32312512	0.50450911	0.49016852	0.71300433	0.37929564
	G6	**0.61204608**	0.24672428	**0.50263654**	**0.49364574**	0.70322069	0.32529862
	G7	0.61270349	**0.62809594**	0.5156759	0.46835542	0.69355981	**0.68308396**
0.75	**G1**	0.62051958	0.43237198	0.50710818	0.48528711	0.71502908	0.50370889
	G2	0.61995518	**0.62828543**	0.51566876	0.468973	0.70686053	0.67974565
	G3	0.62033167	0.43713301	0.50658881	**0.48825115**	0.7151174	0.50257664
	G4	0.62095321	0.44053165	0.5076308	0.4822471	0.71538856	0.50104319
	G5	0.62064763	0.44037755	0.50721391	0.48683499	0.71517573	0.51411211
	G6	**0.59482258**	0.33137821	**0.50532487**	0.48343613	**0.66940402**	0.41949602
	G7	0.61367541	0.6277052	0.51462828	0.47161421	0.69621468	**0.6833674**

In general, the best global configuration is $\alpha = 0.25$ and $G = G6$. However, for hardly constrained instances, the configuration $\alpha = 0.5$ and $G = G2$ provides better mean results, and should be used for this kind of problem. It is not surprising that G2 generates much diversity, since for unconstrained instances of the problem its use is equivalent to a random selection of features. Note that the evaluation of this greedy function will always be zero for unconstrained problems and thus all the features are inserted in the RCL independently of the value of α. G1 and G4 have also a good performance in general but are worse than $G2$.

B. Preliminary Experiment 2

The aim of preliminary experiment 2 was to choose the parameters of the GRASP+PR approach to be used in the experimentation. The parameters values for the underlying GRASP technique are chosen based on the results of the previous section. Thus, only the values of parameters specific of Path Relinking left. These parameters are: (i) Initial Grasp Iterations (number of grasp iterations prior to the intensification phase of PR), (ii) Number of Elite Solutions, (iii) Number of Paths per Iteration; and (iv) Number of neighbors to be explored on each path.

A factorial design is used for this preliminary experiment, generating all the possible combinations of parameter values in a certain range. The ranges used for each parameter are described in table 11. These ranges were determined by authors criteria and some exploratory executions of GRASP+PR.

A number of 11 problem instances were generated by the algorithm described in Appendix C, using the parameters shown in table 3. GRASP+PR where run 30 times for each problem instance and parameters configuration, and the mean QoS computed (as

Paramenter	Range
Initial Grasp Iterations	$\{10, 50, 100\}$
Number of Elite Solutions	$\{2, 5, 10\}$
Number of paths per Iteration	$\{2, 5, 10\}$
Number of neighbors to explore per path	$\{5, 10, 50\}$

Table 11: Parameters Ranges

evaluated by decreasing to 1.0 the value of eq. 6, as described in Sec. 4.2 in order have a minimization problem). Techniques were configured for using a termination criteria based on a maximum execution time of 100ms. The execution time could have been considered as an additional parameter to be included in the factorial design. However, the size of this new design, with $NumberOfexcutionTimesConsidered \cdot 54$ different combinations of parameter values discourages its use. The execution time chosen is the shortest of those considered in this study for rebinding scenarios, assuming that with longer execution times the performance of chosen configurations could only increase, while if choosing longer execution times, performance for the shortest ones will be disadvantaged.

Due to the size of the experiment design, we simply describe the conclusions of the experiment. Both the source code of the implementation and the raw data of results are available on-line for interested readers.

The configuration with best mean results was: (i) Initial Grasp Iterations = 50; (ii) Number of Elite Solutions = 10; (iii) Number of paths per Iteration = 2; and (iv) Number of neighbors to explore per path = 50.

C. QoSWSC Problem Instances Generation Algorithm

In this appendix we describe the algorithms used to generate instances for the QoSWSC Problem. Specifically, two algorithms are described: one the creation of pseudo-random composition structures Φs and other for the addition of global constraints c_g to previously a created instance of the QoSWSC problem P. Given these two algorithms, the rest of components of the QoSWSC problem can be created easily. Thus, a general mechanism for creating

C.1. Composition structure generation algorithm

The parameters of the composition structure generation algorithm are: number of activities N (that can be tasks or building blocks containing other activities), percentage of control flow β activities, percentage of loops β^L, percentage of switches β^S, percentage of forks β^F and maximum nesting level $Nest_{max}$. Moreover we assume that each invocation is performed on a different task. As a consequence, the number of tasks of the structure is $N_{tasks} = |T| = N \cdot \frac{100-\beta}{100}$, being the number of building blocks $N_{bb} = N \cdot \frac{\beta}{100}$. In this point, we assume that each switch and fork has two branches (note that algorithm remains valid if there are more than two branches, but the probabil-

ity of creation of branches with more than one branch without task invocations becomes high). The algorithm assumes that the composition structure to be created contains a sequence S_{main} of activities, called main sequence, where all the rest of activities of the composition are nested. The algorithm initializes S_{main} to a void sequence and creates the set $|T|$ of tasks to invoke.

The algorithm uses three different sets of activities BB, CBB and CBB_{empty}. BB is initialized to $\{S_{main}\}$, representing the set of building blocks (activities that are not tasks, and can nest other activities) connected to S_{main}; i. e. the activities such that are nested on S_{main}, or are nested on an activity a_{parent} that is connected to S_{main} recursively. CBB is initialized to $\oslash$, and in the second loop (line 8) is filled with all the building blocks of this composition except for S_{main}. In this loop we have used a function *createBuildingBlock* that creates a building block selecting the kind (loop, switch or fork) based on parameters β^l, β^S and β^F. CBB_{empty} is initialized with CBB at this moment, containing all building blocks present in this composition except for S_{main}, and representing the set of building blocks that do not have a nested task invocation.

In the third loop (line 15) each element in CBB is randomly nested in an building block connected to S_{main}. In so doing, we select randomly the building block to be nested b_{nested} from CBB and the building block where we nest b_{parent} form BB. In the case of b_{parent} we take into account the nesting level in order to assure that the nesting level of b_{nested} is lower than $Nest_{max}$. After nesting, we remove b_{nested} for CBB and add it to BB; as a consequence, it will be a candidate where to nest at next iteration. Moreover we remove b_{parent} form CBB_{empty} (if present). As a result, when CBB is empty, all building blocks of the composition are present at BB, and CBB_{empty} contains the set of building blocks that have no activities nested. This implies that, if we do not nest a task invocation on the building blocks in CBB_{empty}, it will create wait loops (i.e. loops without any activity) or dummy forks and switches, that have no activities on their branches. To avoid this, in the fourth loop (line 24) we nest a task invocation on each building block present at CBB_{empty}. It is important to note that the algorithm only ensures the avoidance of wait loops and dummy switches and forks if at this point $|T| > |CBB_{empty}|$. Finally in the fifth loop (line 31) we insert the remaining task invocations randomly along the composition structure.

Algorithm 6 Create Composition Structure

$S_{main} \Leftarrow$ empty sequence
$T \Leftarrow \oslash$
for $i = 0$ to N_{tasks} **do**
  $T = T \cup \ create\ taskt_i$
end for
$BB \Leftarrow S_{main}$
$CBB \Leftarrow \oslash$
for $i = 0$ to N_{bb} **do**
  $b_i \Leftarrow createBuildingBlock(\beta^l, \beta^S, \beta^F)$
  $b_i \rightarrow NestingLevel \Leftarrow 0$
  $CBB \Leftarrow CBB \cup \{b_i\}$
  $i \Leftarrow i + 1$
end for
$CBB_{empty} \Leftarrow CBB$
while $CBB \neq \oslash$ **do**
  $b_{nested} \Leftarrow SelectRandomly(CBB)$
  $b_{parent} \Leftarrow SelectRandomly(BB, NestingLevel)$
  $NestRandomly(b_{nested}, b_{parent})$
  $b_{nested} \rightarrow NestingLevel \Leftarrow b_{nested}.NestingLevel + 1$
  $CBB \Leftarrow CBB - \{b_{nested}\}$
  $CBB_{empty} \Leftarrow CBB_{empty} - \{b_{parent}\}$
  $BB \Leftarrow BB \cup \{b_{nested}\}$
end while
while $CBB_{empty} \neq \oslash \wedge T \neq \oslash$ **do**
  $t_{nested} \Leftarrow SelectRandomly(T)$
  $b_{parent} \Leftarrow SelectRandomly(CBB_{empty})$
  $NestRandomly(t_{nested}, b_{parent})$
  $T \Leftarrow T - \{t_{nested}\}$
  $CBB_{empty} \Leftarrow CBB_{empty} - \{b_{parent}\}$
end while
while $T \neq \oslash$ **do**
  $t_{nested} \Leftarrow SelectRandomly(T)$
  $b_{parent} \Leftarrow SelectRandomly(BB)$
  $NestRandomly(t_{nested}$,$b_{parent})$
  $T \Leftarrow T - \{t_{nested}\}$
end while

C.2. Global Constraints Generation Algorithm

The parameters of our global constraints generation algorithm are: QoSWSC problem instance P, mean μ and typical deviation σ of the number of global constraints and the percentage $percent$ of global optimality of constraints to generate that determines the hardness of constraints (we will generate one global constraint per Qos property q selected, but all generated constraints use the same value of $percent$).

Algorithm 7 Global QoS Constraints generation algorithm for a given problem instance P

$NConstraints \Leftarrow Normal(\mu, \sigma)$
$Qprops \Leftarrow Q$
while $|C_g| < NConstraints \wedge Qprops \neq \oslash$ **do**
  Select randomly $q \in Qprops$
  $Max_q \Leftarrow \Psi_q^+$
  $Min_q \Leftarrow \Psi_q^-$
  if $Qprop.Positive$ **then**
    $C_g \Leftarrow C_g \cup \{q \geq Min_q + percent \cdot (Max_q - Min_q)/100\}$
  else
    $C_g \Leftarrow C_g \cup \{q \leq Min_q + (100 - percent) \cdot (Max_q - Min_q)/100\}$
  end if
  $Qprops \Leftarrow Qprops - \{q\}$
end while

In this algorithm constraints are generated conform to the type of the QoS property q that restrict (positive of negative), using inequalities consequently ($\geq$ and $\leq$ respectively).

D. QoS-aware Web Service Composition Formulation

The QoS-aware Web Service Composition problem as specified in this report is formulated as a Mixed Linear Integer Programming (MIP) problem, including the global QoS computing functions $\Psi_q(X, \Phi, R)$ in the constraints. Those functions have not a close arithmetic form without knowing the specific composition structure and the runtime information, that depend on the specific composition to be optimized, and the general QoS model.

QoSWSC Problem

$$\max \sum_{q \in Q} w_q \cdot v_q \tag{21}$$

s.t.:

$$\sum_{q \in Q} w_q = 1 \tag{22}$$

$$\Psi_q(X, \Phi, R) = v_q \qquad \forall q \in Q \tag{23}$$

$$\sum_{j=1}^{|S_i|} x_{i,j} = 1 \qquad i = 1, \ldots, |T| \tag{24}$$

$$v_q \geq c_q \qquad \forall c_q \in C_g \text{ having } q \in Q^+ \tag{25}$$

$$v_q \leq c_q \qquad \forall c_q \in C_g \text{ having } q \in Q^- \tag{26}$$

$$\sum_{j=1}^{|S_i|} x_{i,j} \cdot F_q(s_{i,j}) \geq c_q^i \qquad \forall c_q^i \in C_l \text{ having } q \in Q^+ \tag{27}$$

$$\sum_{j=1}^{|S_i|} x_{i,j} \cdot F_q(s_{i,j}) \leq c_q^i \qquad \forall c_q^i \in C_l \text{ having } q \in Q^- \tag{28}$$

$$x_{i,j} = x_{k,l} \qquad \forall s_{i,j}, s_{k,l} \in c_d, \forall c_d \in C_d \tag{29}$$

$$x_{i,j} \in 0, 1 \qquad \forall x_{i,j} \tag{30}$$

$$F_q(s) \in \Re^+ \qquad \forall s \in S \quad \forall q \in Q \tag{31}$$

$$\Psi_q(X, \Phi, R) \qquad \forall q \in Q \tag{32}$$

In this problem formulation a set X of binary decision variables $x_{i,j}$ is used. $x_{i,j} = 1$ means that service $s_{i,j}$ is chosen for execution task t_i. Thus this set of variables allows us to encode composition instantiations χ, since $x_{i,j} = 1$ means that $\langle t_i, s_{i,j} \rangle \in \chi$.

The objective function to be maximized in this problem is equivalent to GlobalQoS (eq. 2 in sec. 2.4), where global QoS values stored in variables v_q are weighted and added according to user preferences, specified as the weight w_q for each QoS property $q \in Q$. Constraint 22 guarantees that those weights sum 1.

Constraints family 23, specifies that the QoS values will depend on the current providers selection X, the composition structure Φ, and the run-time information R, according to the global QoS model. In sec. 2.1 for an example computing the global cost and execution time of a concrete composition.

Constraints family 24 guarantees that only one candidate provider is chosen for each task t_i.

Constraints families 25 and 26 guarantee that global constraints on positive and negative QoS properties are meet respectively. For instance, equation 26 could specify that the global cost of executing the composition is lower or equal than 1 monetary unit.

Constraints families 27 and 28 guarantee that local tasks constraints on positive and negative QoS properties are meet respectively. For instance, equation 28 could specify that execution of the task t_1 is lower or equal than 0.1 seconds.

Constraint family 29 specifies interdependences between service providers choices. For instance, it could specify that if service $s_{1,1}$ is chosen for executing task t_1, i.e. $x_{1,1} = 1$, then service $s_{2,2}$ must be chosen for executing task t_2, i.e. $x_{2,2} = 1$, and vice versa.

Interested readers can find alternative formulations of the problem in [9], [3] and [44], that use the concept of execution path to compute the global QoS values of the composition. In this paper, given that our proposal is able to handle the composition structure without computing the whole set of possible execution paths, the formulation described above is preferred.

E. Acnowledgements*

This work was partially supported by the European Commission (FEDER) and Spanish Government under Web-Factories (TIN2006-00472) and SETI (TIN 2009-07366) projects, and by the Andalusian Government under project ISABEL (TIC-2533) and THEOS (TIC-5906).

http://www.micinn.es/

http://www.juntadeandalucia.es/organismos/economiainnovacionyciencia.html

References

[1] Rohit Aggarwal, Kunal Verma, John Miller, and William Milnor. Constraint driven web service composition in meteor-s. In *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 23–30, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. BPEL4WS specification, 2003.

[3] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *Software Engineering, IEEE Transactions on*, 33(6):369–384, 2007.

[4] Danilo Ardagna and Barbara Pernici. Global and local qos guarantee in web service selection. In *Business Process Management Workshops*, pages 32–46, 2005.

[5] Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, and Ralf Steinmetz. Heuristics for qos-aware web service composition. *Web Services, 2006. ICWS '06. International Conference on*, pages 72–82, Sept. 2006.

[6] P. A. Bonatti and P. Festa. On optimal service selection. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 530–538, New York, NY, USA, 2005. ACM.

[7] G. Canfora, M. Di Penta, R. Esposito, and M.L. Villani. Qos-aware replanning of composite web services. *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, 1:121–129, 11-15 July 2005.

[8] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, Francesco Perfetto, and Maria Luisa Villani. Service composition (re)binding driven by application-specific qos. In *ICSOC*, Lecture Notes in Computer Science, pages 141–152. Springer, 2006.

[9] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An approach for qos-aware service composition based on genetic algorithms. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1069–1075, New York, NY, USA, 2005. ACM.

[10] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. A framework for qos-aware binding and re-binding of composite web services. *Journal of Systems and Software*, 81(10):1754–1769, 2008.

[11] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, and Francesco Lo Presti. Efficient provisioning of service level agreements for service oriented applications. In *IW-SOSWE '07: 2nd international workshop on Service oriented software engineering*, pages 29–35, New York, NY, USA, 2007. ACM.

[12] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281–308, April 2004.

[13] D.B. Claro, P. Albers, and J.K. Hao. Selecting web services for optimal composition. In *Proc. Int. Conf. Web Services (ICWS 05)*, 2005.

[14] Diana Comes, Harun Baraki, Roland Reichle, Michael Zapf, and Kurt Geihs. Heuristic approaches for qos-based service selection. In *8th International Conference on Service Oriented Computing (ICSOC)*, San Francisco, USA, Dec 2010. Springer.

[15] Johann Dreo, Alain Petrowski, and Eric Taillard. *Metaheuristics for Hard Optimization*. Springer, 2003.

[16] Paola Festa, Mauricio, and G. C. Resende. Grasp: An annotated bibliography. In *Essays and Surveys in Metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[17] Chunming Gao, Meiling Cai, and Huowang Chen. Qos-driven global optimization of services selection supporting services flow re-planning. In *Advances in Web and Network Technologies, and Information Management*, Lecture Notes in Computer Science, pages 516–521. Springer, 2007.

[18] F. Glover. Tabu search ? part i. *ORSA Journal on Computing*, 1:190?206, 1989.

[19] Mark Harman. The current state and future of search based software engineering. *Future of Software Engineering, 2007. FOSE '07*, pages 342–357, 23-25 May 2007.

[20] Randy Heffner, Larry Fulton, Mike Gilpin, and Ken Vollmer Henry Peyret, and Jacqueline Stone. Topic overview: Service-oriented architecture. by Forrester Inc., June 2007. Freely available by Forrester Inc. athttp://www.forrester.com/Research/Document/0, ,42528,00.html.

[21] Michael C. Jaeger, Gero Mühl, and Sebastian Golze. Qos-aware composition of web services: An evaluation of selection algorithms. In *Lecture Notes in Computer Science*, Lecture Notes in Computer Science, pages 646–661. Springer, 2005.

[22] Sebastián Lozano José Antonio Parejo, Antonio Ruiz-Cortés and Pablo Fernandez. Metaheuristic optimization frameworks: A survery and benchmarking. Technical report, Applied Software Engineering Research Group.. University of Seville, December 2010.

[23] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. Emam, and J Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28:721–734, 2002.

[24] Jong M. Koa, Chang O. Kima, and Ick-Hyun Kwonb. Quality-of-service oriented web service composition algorithm and planning architecture. *Journal of Systems and Software*, 81(11):2079–2090, 2008.

[25] Kostas Kontogiannis, Grace A. Lewis, Dennis B. Smith, Marin Litoiu, Hausi Muller, Stefan Schuster, and Eleni Stroulia. The landscape of service-oriented systems: A research perspective. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, Washington, DC, USA, 2007. IEEE Computer Society.

[26] Manuel Laguna and Rafael Martí. Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.

[27] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8:3–30, January 1998.

[28] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *IEEE Computer*, 40(11):38–45, November 2007.

[29] Mike P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann, and Bernd J. Krämer. Service-oriented computing: A research roadmap. In Francisco Curbera, Bernd J. Krämer, and Mike P. Papazoglou, editors, *Service Oriented Computing*, volume 05462 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[30] Mike P. Papazoglou and Willem-Jan van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *VLDB J.*, 16(3):389–415, 2007.

[31] J. A. Parejo, J. Racero, F. Guerrero, T. Kwok, and K. Smith. Fom: A framework for metaheuristic optimization. In *Proceedings of the 2033 International conference on Computational science*, volume 2660 of *ICCS'03. Lecture Notes in Computer Science*, pages 886–895, 2003.

[32] Massimiliano Di Penta and Luigi Troiano. Using fuzzy logic to relax constraints in ga-based service composition. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, June 2005.

[33] Estefanía Piñana, Isaac Plana, Vicente Campos, and Rafael Martí. Grasp and path relinking for the matrix bandwidth minimization. *European Journal of Operational Research*, 153(1):200–210, 2004.

[34] Yang Qu, Chuang Lin, YuanZhuo Wang, and Zhiguang Shan. Qos-aware composite service selection in grids. *Grid and Cooperative Computing, 2006. GCC 2006. Fifth International Conference*, pages 458–465, Oct. 2006.

[35] Mauricio G. C. Resende. Greedy randomized adaptive search procedures. In *Encyclopedia of Optimization*, pages 1460–1469. 2009.

[36] Mauricio G. C. Resende, Leonidas S. Pitsoulis, and Panos M. Pardalos. Fortran subroutines for computing approximate solutions of weighted max-sat problems using grasp. *Discrete Applied Mathematics*, 100(1-2):95–113, 2000.

[37] Sen Su, Chengwen Zhang, and Junliang Chen. An improved genetic algorithm for web services selection. In *Distributed Applications and Interoperable Systems*, volume 4531/2007 of *Lecture Notes in Computer Science*, pages 284–295. Springer, 2007.

[38] W3C. Web services glossary, 2004. Available at `http://www.w3.org/TR/ws-gloss/`.

[39] H. Wada, P. Champrasert, J. Suzuki, and K. Oba. Multiobjective optimization of sla-aware service composition. *Congress on Services - Part I, 2008. SERVICES '08. IEEE*, pages 368–375, July 2008.

[40] Hongbing Wang, Ping Tong, Phil Thompson, and Yinsheng Li. Qos-based web services selection. *icebe*, 0:631–637, 2007.

[41] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.

[42] Chengwen Zhang, Sen Su, and Junliang Chen. Efficient population diversity handling genetic algorithm for qos-aware web services selection. In *Computational Science ? ICCS 2006*, volume 3994/2006 of *Lecture Notes in Computer Science*, pages 104–111. Springer, 2006.

[43] Chengwen Zhang, Sen Su, and Junliang Chen. Diga: Population diversity handling genetic algorithm for qos-aware web services selection. *Comput. Commun.*, 30(5):1082–1090, March 2007.

[44] Li Zhang and Danilo Ardagna. Sla based profit optimization in autonomic computing systems. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 173–182, New York, NY, USA, 2004. ACM.