QoS-aware Web Services Composition using GRASP with Path Relinking

José Antonio Parejo[a], Sergio Segura[a], Pablo Fernandez[a], Antonio Ruiz-Cortés[a]

[a]*Department of Computing Languages and Systems, University of Sevilla, Spain.*

Abstract

In service oriented scenarios, applications are created by composing atomic services and exposing the resulting added value logic as a service. When several alternative service providers are available for composition, quality of service (QoS) properties such as execution time, cost, or availability are taken into account to make the choice, leading to the creation of QoS-aware composite web services. Finding the set of service providers that result in the best QoS is a NP-hard optimization problem. This paper presents QoS-Gasp, a metaheuristic algorithm for performing QoS-aware web service composition at runtime. QoS-Gasp is an hybrid approach that combines GRASP with Path Relinking. For the evaluation of our approach we compared it with related metaheuristic algorithms found in the literature. Experiments show that when results must be available in seconds, QoS-Gasp improves the results of previous proposals up to 40%. Beside this, QoS-Gasp found better solutions than any of the compared techniques in a 92% of the runs when results must be available in 100ms; i.e. it provides compositions with a better QoS, implying cost savings, increased availability and reduced execution times for the end-user.

Keywords: QoS, Composite Web Service, SOA, GRASP, Path Relinking

1. Introduction

Service Oriented Computing (SOC) is a software development paradigm based on assembling web services to implement dynamic business processes and agile applications that spread across multiple organizations (Papazoglou et al., 2007). The potential of SOC lies in three of the key benefits of web services: loose coupling between consumer and provider, composability of services, and dynamic binding. Loose coupling means that web services are consumed through a contract hiding implementation details to users. Composability means that web services can be composed to create more complex and valuable services, so-called *Composite Web Services* (CWS). Finally, dynamic binding provides flexibility to the applications by enabling the selection of the specific web services to be invoked at runtime.

Web services may include information about the non functional properties that affect to their quality, so-called *Quality of Service* (QoS) attributes, e.g. cost, availability, etc. When several providers expose web services that are functionally equivalent through compatible interfaces, QoS properties can be used to drive the selection of the candidate service to invoke. For instance, one may choose the most reliable and expensive service, the cheapest one, or a third service that provides a balance.

The QoS-aware binding of CWS enables the creation of context-aware and auto-configurable applications, that can adapt itself depending on available services and user preferences (Ardagna and Pernici, 2007). For instance, consumers could specify constraints like "The total cost per invocation must be lower than 1$" and QoS criteria such as "choose the faster providers".

Given a CWS, a relevant problem is how to determine the optimal binding; *i.e.* the set of service providers to invoke that meet user constraints and optimizes the QoS according to some QoS criteria. This problem, named QoS-aware Service Composition (QoSWSC) (Strunk, 2010), is NP-hard (Bonatti and Festa, 2005; Ardagna and Pernici, 2005), and has been identified as a main research problem in the SOC field (Papazoglou et al., 2007).

The QoSWSC problem can be solved when the composition is created (i.e. at design time), just before starting the execution of the composition (i.e. at invocation time) or while the composite web service is running (i.e. at runtime). When this problem is solved at run time taking into account the current state of invocations, it

Email address: japarejo@us.es (José Antonio Parejo)

is named a *rebinding* (Zeng et al., 2004; Ardagna and Pernici, 2007). Solving rebinding problems is crucial in dynamic services markets where providers become unavailable, new providers emerge and QoS levels change frequently (Canfora et al., 2008). In this scenarios, the time spent to solve the QoSWSC problem is a critical issue that influences the overall service response time and it should be kept as low as possible (Canfora et al., 2005b).

Metaheuristic search techniques are algorithmic frameworks which use heuristics to find approximate solutions to hard problems at an affordable computational cost. Typical metaheuristic techniques are *Genetic Algorithms* (GA), *Hill Climbing* (HC), *Tabu Search* (TS), *Simulated Annealing* (SA), *GRASP* and *Path Relinking* (PR) (Gendreau and Potvin, 2010). Several heuristic (Berbner et al., 2006) and metaheuristic techniques has been proposed in the literature to solve the QoSWSC problem, such as GA (Canfora et al., 2005b) and SA(Wang et al., 2007).

This article proposes QoS-Gasp, a novel metaheuristic algorithm for solving the QoSWSC problem. This algorithm is a hybrid approach that combines GRASP and PR. QoS-Gasp is especially suitable for rebinding problems where short solving times are a must. In order to evaluate our algorithm we compared it with several metaheuristic algorithms proposed in the literature (GA (Canfora et al., 2005a) and hybrid TS with SA(Koa et al., 2008)) in rebinding scenarios. The comparison was made using several experiments with two different optimization criteria and 22 service compositions. The results show that QoS-Gasp find solutions with up to 40% higher quality than those found by related algorithms in rebinding problems that must be solved in less than one minute. Moreover, QoS-Gasp found better solutions than any of the runs of the techniques compared in a 92% of the runs when results must be available in 100ms. As a part of our evaluation we performed a rigorous statistical analysis of the data that supports our conclusions.

The remainder of this article is organized as follows: Section 2 presents a formal description of the QoSWSC problem and the metaheuristics used in our proposal (GRASP and PR). Section 3 describes QoS-Gasp in depth. The empirical evaluation of our approach is presented in section 4, along with a brief description of the previous proposals used for comparison. Section 5 presents the threats to validity of our work. The related works are presented in section 6. Finally, Section 7 describes our conclusions and future work. An extended version of the article is available as a technical report (Parejo et al., 2013).

1.1. A motivating example

In order to illustrate the QoSWSC problem, a goods ordering service inspired in the example provided in (Zheng et al., 2012) is depicted in Fig. 1 using BPMN. The diagram specifies a business process exposed as a composite web service that uses 7 *services with alternative providers* (henceforth named tasks, $t_1, \ldots, t_7$). Table 1 shows the available service providers for each task and their corresponding QoS attributes. As illustrated, two candidate services are available for each task.

The composition starts when a client sends an order. First the order is registered. Next if the payment type of the order is "Credit Card", the card is checked (t_1) and the payment (t_2) is performed. As depicted in Table 1, two banks providers are available, *A* and *B*, and each of them provide candidate services for the tasks t_1 and t_2, denoted as $s_{1,A}$, $s_{2,A}$, $s_{1,B}$ and $s_{2,B}$. Different providers could be chosen in the binding of the CWS for each task; *e.g.* *A* for t_1, and *B* for t_2.

Next the stock is checked (t_3) and the products are reserved for pick-up (t_4). If any product in the order is not in stock, the user is informed of the delay and the CWS waits for some time until activities t_3 and t_4 are repeated (creating a loop). It is worth noting that the same provider must be chosen for the tasks t_3 and t_4, since the reservation in t_4 refers to the stock of the specific provider queried in t_3. Once the order is ready for delivery two branches are performed in parallel. The pick-up and delivery (t_5) to the client is requested, and an e-mail is sent to the client with an enclosed digitally signed invoice (t_6). Once the activities on both branches are performed, the completion of an user satisfaction survey (t_7) is requested.

Additionally, Fig. 1 shows a QoS constraint that must be fulfilled. Specifically, the constraint specifies that "*The total execution time of the remainder activities after having the order ready for delivery must be lower than 0.5 seconds*".

The QoSWSC problem can be stated as finding the bindings that meet all the QoS constraints and maximize or minimize certain user-defined optimization criteria, e.g. minimize cost. Note that this may become extremely complex as the number of candidate services increases. In this example two providers are available for each task, thus 128 (2^7) different bindings are possible. This problem becomes especially convoluted in rebinding scenarios where providers can become unavailable and QoS levels may change unexpectedly.

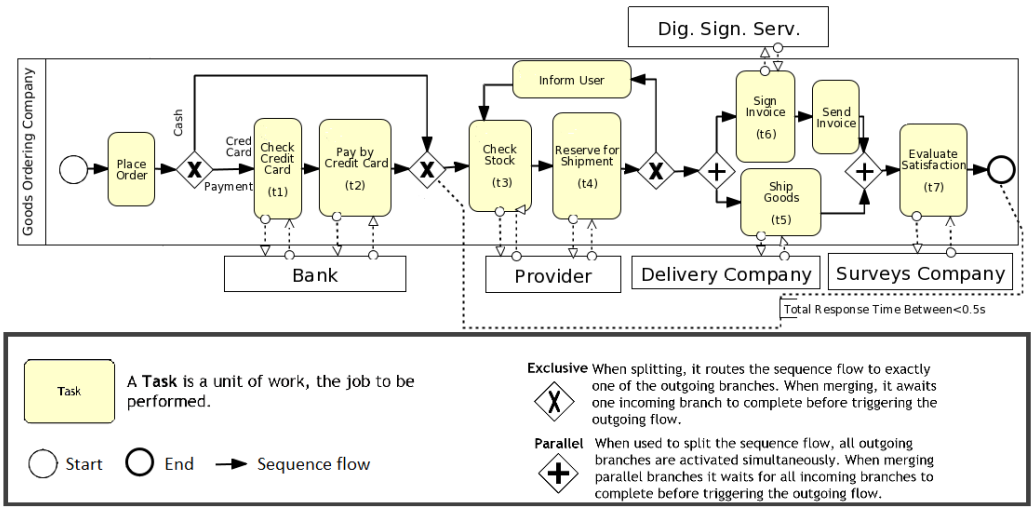


Figure 1: Goods Ordering Composite Service

Table 1: Service providers per Role and their corresponding QoS Guarantees

Actor	**BANK**				**PROVIDER**				**DELIVERY**		**DIG. SIGN.**		**SURVEYING**	
Provider	*A*		*B*		*C*		*D*		*E*	*F*	*G*	*H*	*I*	*J*
Task	t_1	t_2	t_1	t_2	t_3	t_4	t_3	t_4	t_5	t_5	t_6	t_6	t_7	t_7
Candidate Service	$s_{1,A}$	$s_{2,A}$	$s_{1,B}$	$s_{2,B}$	$s_{3,C}$	$s_{4,C}$	$s_{3,D}$	$s_{4,D}$	$s_{5,E}$	$s_{5,F}$	$s_{6,G}$	$s_{6,H}$	$s_{7,I}$	$s_{7,J}$
Cost (in cents)	1	2	1.5	5	1	2	1	5	1	2	1	2	1.5	5
Execution Time	0.2	0.2	0.1	0.15	0.2	0.2	0.4	0.25	0.2	0.2	0.2	0.2	0.1	0.15

2. Preliminaries

2.1. QoS-aware Binding of Composite Web Services

The QoS-aware binding of a CWS is performed as follows: When the CWS is invoked or a rebinding is needed (Canfora et al., 2008), the set of tasks is identified. For each task t_i, the set of service providers available $S_i = \{s_{i,1}, \ldots, s_{i,m}\}$ (named candidate services) is determined by performing a search on a service registry. For each candidate service $s_{i,j}$, the QoS information is retrieved; e.g. according to Table 1 the cost of invoking the payment service of provider A is 0.02$. Given that some registry technologies do not support QoS information, a QoS-enriched registry or alternative QoS information source (such as a Service Level Agreements Repository or a Service Trading Framework (Fernandez et al., 2006)) is needed. The set of QoS properties taken into account is denoted as $\mathbb{Q}$.

Taking into account this information the expected QoS provided by the application can be optimized. The goal of this optimization is to find the binding that maximizes the utility of the global QoS provided according to the consumers' preferences. Such preferences determine which binding is more valuable based on the global QoS levels (Q_q) provided for each QoS property q. For instance, a total execution time of 2 seconds could be fair for some users but too much for others. User preferences are expressed as weights w_q and utility functions U_q for each QoS property q. The weights define the relative importance of each property. For instance, $w_{Cost} = 0.2$ and $w_{ExTime} = 0.1$ means cost is twice as important as execution time for the user. Utility functions U_q define which values of the specific property are more useful for the user. For instance, for availability the utility function would be linear, since the higher the availability the better.

Thus, our goal translates in to finding the binding χ^* that maximizes the global user utility computed as:

$$GlobUtil(\chi) = \sum_{q \in \mathbb{Q}} U_q(Q_q(\chi)) * w_q \quad (1)$$

having $\sum_{q \in \mathbb{Q}} w_q = 1$. Similar schemes for expressing user preferences and global utility functions have been used extensively in the literature (Zeng et al., 2004; Ardagna and Pernici, 2007; Canfora et al., 2005b; Strunk, 2010).

2.2. QoS Model

2.2.1. QoS properties

The set of quality properties $\mathbb{Q} = \{C, T, A, R, S\}$ considered in this article has been used extensively in related work (Zeng et al., 2004; Ardagna and Pernici, 2007; Canfora et al., 2005b). It comprises of:

Cost (C). Fee that users must pay for invoking a service.

Execution Time (T). Expected delay between service invocation and the instant when result is obtained.

Availability (A). Probability of accessing the service per invocation, where its domain is $[0, 1]$.

Reliability (R). It measures the trustworthiness of the service. It represents the ability to meet the quality guarantees for the rest of the properties. Its value is usually computed based on a ranking performed by end users. For example, in *www.amazon.com*, the range is $[0, 5]$ where 0 means that QoS guarantees are systematically violated, and 5 means that guarantees are always respected. In this article we assume its domain is $[0, 1]$.

Security (S). It represents the quality aspect of a service to provide mechanisms to assure confidentiality, authentication and non-repudiation of the parties involved. Consequently, this property usually implies the use of encryption algorithms with different strength, different key sizes on underlying messages, and some kind of access control. In this article we use a categorization of the security, where the use of an encryption algorithm and key size in a service implies a numerical value associated to this property for the service. Its domain is $[0, 1]$, where value 0 means no security at all and value 1 means maximum security.

QoS properties are usually 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. A quality property is negative if the higher the value, the lower the utility. For instance, cost is a negative property. We apply definitions of the utility functions widely used in the literature (Zeng et al., 2004; Ardagna and Pernici, 2007; Canfora et al., 2005b). For instance, for positive QoS properties the utility of the value x for a QoS property q is defined as:

$$U_q(x) = \begin{cases} 1 & \text{if} \quad q^{max} - q^{min} = 0 \\ \frac{x - q^{min}}{q^{max} - q^{min}} & \text{if} \quad q \text{ is positive} \\ \frac{q^{max} - x}{q^{max} - q^{min}} & \text{if} \quad q \text{ is negative} \end{cases} \tag{2}$$

where q^{max} and q^{min} are the maximum and minimum values of the QoS property q for all candidate services.

2.2.2. Computing the Global QoS

Apart from the specific providers chosen for each task, the global QoS values for the CWS depend on:

The workflow of the composition and the type of QoS property. Global QoS is computed by recursively applying a QoS aggregation function according to the building blocks that define the structure of the composition. Table 2 summarizes the aggregation functions applied for each QoS property q and type of building block[1]. These functions are widely applied in literature (Zeng et al., 2004; Ardagna and Pernici, 2007; Canfora et al., 2005b; Wang et al., 2007; Strunk, 2010). For instance, the total execution time of the parallel branches is computed as the maximum execution time of any branch, but the execution time of a sequence of tasks is computed as the sum. In a very similar way, the aggregation function depends on the specific QoS property to be aggregated. For instance, given a specific workflow such as the parallel branches of our motivating example (tasks t_6 and t_5), the total cost is computed as the sum of the costs of the tasks in each branch, but the total availability is computed as the product of the availability of the tasks in each branch.

The specific branches chosen for execution and the number of iterations performed in loops. Since in general the specific run-time behaviour of loops and alternative branches is unknown in advance, an estimate of this behaviour is needed to perform QoS-aware binding (Canfora et al., 2008). For instance, given that probability of using credit card is 0.8, and 2 iterations of stock reservation are performed, the estimated global cost for the binding $\chi = (A, B, D, D, F, H, J)$ is: $Q_{Cost}(\chi)$ = Cost of switch(χ) + Cost of Loop(χ) + Cost of fork(χ)+$Cost_7(\chi)$ = 0.8∗0.025+2∗0.06+0.09 = 0.23$

Since those values are estimations, the actual global QoS values provided can differ significantly from the estimations in some invocations. In the worst case this deviation can lead to the violation of the global QoS constraints. To avoid this problem, the re-binding triggering approach proposed in (Canfora et al., 2008) could be used.

2.3. Constraints of the QoSWSC problem

The QoSWSC problem has three types of constraints (Zeng et al., 2004; Ardagna and Pernici, 2007):

Global QoS constraints. They affect the QoS of the CWS as a whole; e.g. *the total cost of the composition must be lower than five* ≡ $Q_{cost}(\chi) < 5$.

[1] In this table k means the average number of iterations performed in loops and P_i means the probability of executing branch i

Table 2: QoS Aggregation functions

	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)	$f_S(F(a_i))_{i\in 1\ldots m}$	$f_L(s^L, k))$	$f_B(\lfloor F_S(s_i^b) \rfloor, [p_i])$	$f_F(F(s_i^f))_{i\in 1\ldots p}$

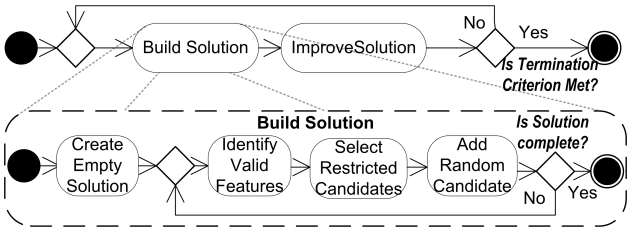


Figure 2: GRASP working scheme

Local QoS Constraints. They affect the QoS values provided by the service chosen for a specific tasks; e.g. *the cost of payment (t_2) must be lower than 1*.

Service dependence constraints. A CWS may use several services that must be bound to the same provider. This situation creates a dependence, *i.e.* if the provider is selected for one of the tasks, then it must be selected for the rest of tasks it implements. In our motivating example there exists a dependence constraint between tasks t_3 and t_4 (stock management and reservation).

2.4. GRASP

The *Greedy Randomized Adaptive Search Procedure* (GRASP) (Resende, 2009) is an iterative optimization algorithm. GRASP has been successfully applied in a plethora of real life applications and research problems (Festa et al., 2002). Its working scheme is shown in Fig. 2. Each GRASP iteration consists of two main steps: (i) building a solution and (ii) improving such solution using a local search algorithm.

In the building phase, GRASP begins by creating an empty solution. Elements are added iteratively to it until a complete and feasible solution is found. For instance, in case of the QoS-aware web service composition problem, the empty solution contains no bindings to any candidate services; *i.e.* in our motivating example the empty solution would be (?, ?, ?, ?, ?, ?, ?), meaning that no tasks are bound to an specific candidate service. The elements added are specific bindings to candidate services for each task, for instance, for task t_1 two candidate services are available, leading to partial solutions (A, ?, ?, ?, ?, ?, ?) and (B, ?, ?, ?, ?, ?, ?).

In order to add an element to the partial solution the algorithm performs three steps. First, the set of *valid elements* that could be added to the partial solution is determined. For instance, in our motivating example the element D is a candidate provider for task t_4, but given the partial solution (B, A, C, ?, ?, ?, ?), D is not a valid element, since a constraint states that tasks t_3 and t_4 should have the same provider. Thus the single valid element for task t_4 in that case is C.

Next, a subset of promising candidates is chosen from the set of valid elements. This subset is referred to as the *Restricted Candidate List* (*RCL*). The selection of the elements in the *RCL* should be greedy and adaptive.

By greedy we mean that criterion should promote the inclusion of the most promising elements in the *RCL*. For instance, a greedy criterion in our problem would be to include the best candidate services according to any of the QoS properties, the cheapest, the faster, the most secure, etc. In our example, for task t_7 service $s_{7,I}$ from provider I is faster and cheaper than service $s_{7,J}$ form provider J, thus the *RCL* according to this criterion would be $\{I\}$. On the contrary, for task t_1 service $s_{1,A}$ from provider A is the cheapest but service $s_{1,B}$ from provider B is the fastest, thus the *RCL* according to this criterion would be $\{A, B\}$.

By adaptive we mean that the selection criterion should take into account the current partial solution. As an example, a greedy and adaptive criterion in our problem would be the inclusion of the services whose QoS values are better that the average value for the elements in the current partial solution for any QoS property, and all the possible elements if such element does not exist. In our motivating example, given the partial solution (A, , ?, ?, ?, ?, ?, ?), the *RCL* for task t_2 would be $\{B\}$, since the execution time of corresponding service $s_{2,B}$ is 0.15, better than the average execution time in the composition (0.2). However, if the current partial solution is

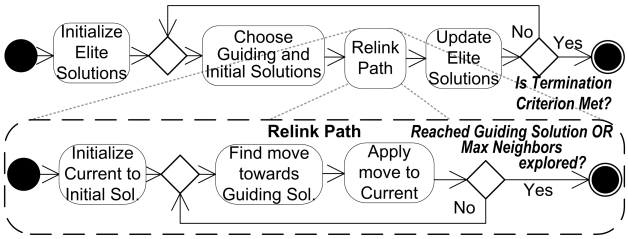


Figure 3: Path Relinking working scheme

$(A, , ?, ?, ?, ?, ?, I)$, the *RCL* for task t_2 would be $\{A, B\}$, since none of the valid candidates improves the average value for any of the QoS properties.

At the end of the iteration a randomly chosen element of the *RCL* is added to the current solution.

In the improvement phase, a local search algorithm is executed using as starting solution the result of the construction phase.

2.5. Path Relinking

Path Relinking (PR) is an metaheuristic optimization technique that generates new solutions by exploring trajectories connecting promising solutions. The basic hypothesis is that by exploring the region of the search space between promising solutions we will find more promising solutions. The working scheme of PR is shown in Fig. 3. PR manages a set of promising solutions named the "elite set". In each iteration, until the meeting of a termination criterion, PR randomly chooses two solutions from the elite set, named the initiating and guiding solution. Then, PR generates a sequence of successive solutions from the initiating to the guiding solution (Laguna and Martí, 1999). Each step is generated by replacing elements of the initial solution with the corresponding elements of the guiding solution. For instance in our motivating example, having the bindings (A, B, D, D, F, H, J) and (B, B, D, D, F, H, I) as initiating an guiding solutions respectively, the elements to be incorporated are B as provider for task t_1, and I as provider for task t_7. The order of element replacement is significant, since different orderings define different paths in the solution space. For instance, in our example we could choose to incorporate B or I first, leading to solutions (B, B, D, D, F, H, J) and (A, B, D, D, F, H, I) respectively.

After reaching the guiding solution the elite set is optionally updated. For instance, the best solution found could be added, the initiating and/or guiding solutions could be removed, etc. The key parameters of PR are the number of paths explored N_{paths} between each pair of initiating and guiding solutions, and the number of steps explored per path N_{steps}.

3. QoS-Gasp

In this section we present QoS-Gasp a novel proposal for solving the QoSWSC problem. It stands for "**QoS-**aware **G**RASP+PR **a**lgorithm for **s**ervice-based a**p**plications binding". It is an hybrid algorithm, where GRASP is used for initializing the elite set used in Path Relinking.

Next we describe how GRASP and PR have been adapted for solving the QoSWSC problem.

3.1. Solution encoding

In order to apply metaheuristic optimization algorithms to solve or problem, a suitable encoding of solutions is needed. An encoding is the mechanism used for expressing the characteristics of solutions in a form that facilitates its manipulation by the algorithm. In QoS-Gasp a vector-based encoding structure is used. This encoding has been used extensively in literature (Canfora et al., 2005a; Gao et al., 2007). Specifically, solutions are encoded as a vector of integer values, with a size equal to the number of tasks. Thus, value j at position i of this vector encodes the choice of service j as provider for task i.

For instance, in our motivating example, the vector that encodes the binding (A, B, D, D, F, H, J) would be [0|1|1|1|1|1|0|0]. The index of each provider is determined by order of appearance in table 1; *e.g.* for Banks $A \equiv 0$ and $B \equiv 1$. Note that the values in each position of the vector would be either 0 or 1, since we have only two providers per task in our motivating example, the encoding is not binary.

3.2. Constraints support

GRASP and PR do not directly support the optimization of constrained optimization problems. In order to overcome this drawback, a variant of Eq. 1 is used as objective function. This variant takes into account the penalization term defined in (Canfora et al., 2005b) using a weight w_{unf}, and a function D_f that measures the distance of a binding χ from a full constraint satisfaction. Thus our final function to be maximized is:

$$ObjFunc(\chi) = GlobUtil(\chi) - (w_{unf} * D_f(\chi)) \quad (3)$$

having $0 \leq w_{unf} \leq 1$.

The distance to full constraint satisfaction D_f is defined as:

$$D_f(\chi, C) = \frac{\sum_{c \in C} Meet(c, \chi)}{|C|} \quad (4)$$

being C the set of global and interdependence constraints of the problem [2]. $Meet(c, \chi)$ is a function that measures the distance to the fullfillment of a single constraint c by the binding χ

$$Meet(c, \chi) = \begin{cases} 0 & \text{if } c \text{ is met} \\ abs(Q_q(\chi) - T_q) \\ \text{(Dist. to threshold)} & \text{if } c \text{ is global and unmet} \\ \frac{\text{\#services missing}}{\text{\#dependant services}} & \text{if } c \text{ is an unmet dep. const.} \end{cases} \quad (5)$$

In this function, we denote the threshold of each global constraint on QoS property q as T_q. For instance, given the global constraint "*the total cost of the composition must be lower than five*" $\equiv$ $Q_{cost}(\chi) < 5$, then $T_{cost} = 5$. If the actual cost of execution the composition given a binding χ is 5.6, the value of $Meet(c, \chi)$, would be 0.6. Conversely, if the actual cost of executing the composition is 3.5, the value of $Meet(c, \chi)$ is 0, since the constraint is met. In a very similar way, when a constraint defines a dependency between taks for instance t_3 and t_4 in our motivating example, if the provider chosen for each task is different, the value of $Meet(c, \chi)$ would be $1/2 = 0.5$, since we have 1 missing service from the chosen provider, and the total number of dependent services in the constraint is 2.

3.3. GRASP building phase

In QoS-Gasp, GRASP elements represent a particular choice of a candidate service for a given task. Consequently, the solution χ is built by choosing a service for a task at each iteration of the loop until the solution is a complete binding. The partial solution at iteration k is denoted as χ^k. The specific task to bind at iteration k is randomly chosen.

The set of valid elements for the task t_i is determined by the service dependence constraints. For instance, in our motivating example there exists a dependence constraint between t_3 (stock querying) and t_4 (reservation for pickup). Thus, if a provider has been chosen for task t_3 in our partial solution χ^k, then the same provider should be chosen for t_4. If conflicting dependency constraints are found the construction phase restarts, since it is not possible to create a feasible solution from χ^{k-1}.

QoS-Gasp uses a *RCL* selection scheme that has been applied extensively in the literature of GRASP. Specifically, this selection is driven by an evaluation function g -that must be defined for the specific optimization problem to solve- and a greediness parameter α (between 0 and 1). Function g provides a value in $\mathbb{R}$ for each candidate service, where g^{min} is the minimum and g^{max} is the maximum of those values. A service $s_{i,j}$ will be in the *RCL* if $g(s_{i,j})$ is greater or equal than $g^{min} + \alpha \cdot (g^{max} - g^{min})$; *i.e.* α defines the proportion of the range $[g^{min}, g^{max}]$ in which candidates are discarded from *RCL*. Thus, for $\alpha = 0$ all the candidates are in the *RCL* (none is discarded), and the construction phase becomes random. If $\alpha = 1$ only the candidates with a value in g of g^{max} would be in the *RCL*.

The function g and value of α are crucial for the performance of GRASP. We defined up to seven novel greedy functions for the QoSWSC problem. Since the optimal values of those parameters depends on the problem to be solved, we performed a preliminary experiment testing each of function g with several values of α. All the details about the g functions and their evaluation are reported in (Parejo et al., 2013) due to space limitations. The best average results were obtained $\alpha = 0.25$, and the best performing greedy functions were $G1$, $G2$ and $G6$ showed below:

$$G_1(s_{i,j}, \chi^k) = \sum_{q \in Q} w_q \cdot U_q(q_{i,j}) \quad (6)$$

G_1 is "miopic" and unadaptive, meaning that it only considers the QoS value of each service, ignoring the current solution under construction χ^k, but its evaluation is extremely fast.

$$G_2(s_{i,j}, \chi^k) = D_f(\chi^k) - D_f(\chi^k \cup s_{i,j}) \quad (7)$$

G_2 uses the difference of distance to constraint satisfication of the current partial solution χ^k and the new partial solution, denoted as $\chi^k \cup s_{i,j}$, but it ignores the QoS weights

$$G_6(s_{i,j}, \chi^k) = ObjFunc(\chi^k \cup s_{i,j}) - GlobUtil(\chi^k) \quad (8)$$

G_6 is based directly on the gradient of the global QoS, but ignoring the distance to constraint satisfaction of the current solution. This subtle variant penalizes the selection of elements that generate constraint violations.

[2] Local constraints are not taken into account, since they can be met by preprocessing the set of candidate services (Ardagna and Pernici, 2007).

In order to evaluate D_f, $GlobUtil$, and $ObjFunc$, a random solution is generated at the beginning of the construction phase, and their elements are used to complete the choices for unassigned tasks in χ^k.

3.4. GRASP improvement phase

The GRASP improvement phase in QoS-Gasp is a local search procedure based on a neighbourhood definition. The neighbourhood of a binding χ comprises of all possible bindings that have exactly $n-1$ assignments in common with χ; i.e. have the same candidate services selected for each task except for one. QoS-Gasp uses Hill Climbing, where only a percentage of the neighbourhood is explored.

3.5. Path Relinking

QoS-Gasp uses the adaptation of GRASP described above to initialize the elite set used by PR. The length of the path between initiating and guiding solutions in QoS-Gasp is determined by the number of different service candidates. Each step of any relinking path, incorporates one service candidate from the guiding solution. It is worth noting that the order in which service candidates are incorporated defines different paths. Consequently, for each pair of initiating and guiding solutions a high number of different paths could be explored. In order to reduce the computational cost of such exploration, QoS-Gasp restricts the number of paths generated between each pair of solutions to N_{paths}. It introduces the service candidates from the guiding solution in a random order, and it limits the number of neighbours explored in each path to N_{steps}. These parameters control the balance between the diversification of the areas of the search space explored and the exhaustiveness of the search in those areas, which is crucial in rebinding scenarios where execution time is scarce.

4. Experimentation

The aim of the experimentation is to compare the performance of QoS-Gasp with previous metaheuristic proposals described in the literature for solving the QoSWSC Problem.

4.1. Previous Proposals

4.1.1. Genetic Algorithms

The proposal described in (Canfora et al., 2005a) has been implemented for comparison since it is the most cited GA-based approach for this problem. In particular, the initial population is randomly generated. A standard one-point crossover operator (Dreo et al., 2003) is used.

The mutation operator modifies the candidate to a single task, both randomly chosen. Parameter values are chosen according to (Canfora et al., 2005a) (as shown in table 3).

4.1.2. Hybrid TS with SA

A hybrid of TS with Simulated Annealing (SA) was proposed in (Koa et al., 2008) for solving the QoSWSC problem. This proposal was aimed at finding feasible solutions of constrained instances; thus, the search was driven by the constraint meeting distance and the execution terminates when a feasible solution is found. In order to enable the comparison with our proposals, and to continue optimizing according to user preferences (even when all constraints are met), a modification has been carried out. When all the constraints are met, the difference between the QoS value of current solution Q_q and the average QoS for this property Avg^q is used for guiding the search. Specifically, the QoS property selected to guide the improvement in the algorithm is the one minimizing $s * (Q_q(\chi) - Avg^q) * w_q$, where s is 1 if q is positive and -1 if it is negative; *i.e.* our modification tries to generate neighbors improving the solution in the QoS property with the bigger improvement room and importance for users. The pseudo-code of the resulting algorithm, and a detailed explanation of its working scheme is available in the additional material (Parejo et al., 2013).

4.2. Experimental Setting

In order to evaluate our proposal QoS-Gasp was implemented using FOM (Parejo et al., 2003). FOM is an object oriented framework written in JAVA that reduces the implementation burden of optimization algorithms. It also provides some experimentation capabilities (Parejo et al., 2012). Experiments were performed on a computer equipped with an 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.

4.3. Experiment #1

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 by 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 (GRASP+PR) that

use $G2$ and $G6$. The parameters used for each technique are described in table 3. These values were chosen based on the experiments reported in literature for previous proposals and on another preliminary experiment performed for GRASP and GRASP+PR (described in detail in (Parejo et al., 2013)). Positive scaling utility function were used for Availability, Reliability and Security (we denote this set of properties as $\mathbb{Q}^+ = \{A, R, S\}$), *cf.* section 2.2. Negative scaling utility functions were used for the remaining properties ($\mathbb{Q}^- = \{C, T\}$). The weights used for each QoS property were: $w_{unf} = 0.5$, $w_C = 0.3$, $w_T = 0.3$, $w_A = 0.1$, $w_S = 0.2$, $w_R = 0.1$. Since FOM solves minimization problems, an objective function that subtracts the value of *ObjFunc* (as described in equation 3) to 1.0 was used.

Table 3: Parameters of the techniques used in the experiment

Technique	**Paramenter**	**Value**
GRASP (G1)	α	0.25
	Greedy Function	G1
	LocalSearch	HC (20% neig. exploration)
GRASP+PR (G6/G2)	α	0.25
	Greedy Function	G6 / G2
	LocalSearch	HC (20% neig. exploration)
	# Elite Solutions	5
	N_{Paths}	2
	N_{steps}	50
	Intial GRASP Iter.	50
Canfora's GA (Canfora et al., 2005b)	Population Size	100
	Crossover	0.70
	Mutation Prob.	0.01
	Survival Policy	The two better individuals
	Selector	Roulette Wheel
	Initial Population	Randomly generated
Hybrid TS+SA (Koa et al., 2008)	Initial Solution	Local optimization (Koa et al., 2008)
	Services exchanged	2
	Tabu Memory	Recency based memory
	Tabu Mem. size	100 movements
	Accept. Criterion	Based on current iteration

4.3.1. Experiment design

Since our aim is to compare the performance of techniques, the dependent variable of this experiment was the evaluation of $1 - ObjFunc$ for the best solutions found in each runt. The independent variable of exp. #1 was the tehnique used for optimization. The termination criteria was maximum execution time. Specifically, the experiments were replicated using 100ms, 200ms, 500ms, 1000ms, 10000ms and 50000ms as maximum execution times. These values cover most of rebinding and binding scenarios at invocation time.

Eleven problem instances were generated by the algorithm described in appendix C of (Parejo et al., 2013), using the parameters shown in table 4. Those parameters are common in the literature on the QoSWSC problem (*cf.* table 9 of (Parejo et al., 2013)). The specific characteristics of each problem instance generated are shown in table 5 .

Table 4: Problem instances generation parameters

Composition Structure	Activities	Uniform distribution between 10 and 100
	% control flow	Uniform distribution between 20% and 50%
Parameters	% Loops	45%
	% Branches	45%
	% Flows	10%
	Max nesting	Uniform $\in [5, 10]$
Runtime inf. params	Iter. per Loop	Gaussian($\mu = 18$, $\sigma = 6$)
	Prob. of Branches	Random
Candidate Services Parameters	Candidates per Task	Uniform $\in [1, 10]$
	Cost	Uniform $\in [0.2, 0.95]$
	Exec. Time	Gaussian($\mu = 0.5$, $\sigma = 0.4$)
	Reliability	Uniform $\in [0.3, 0.9]$
	Availability	Uniform $\in [0.9, 0.99]$
	Security	Uniform $\in [0.6, 0.99]$
Constraints Parameters	Number of Const.	Uniform $\in [0, \lvert Q \rvert]$
	% of optimality	Uniform $\in [25, 75]$
Objective Function Parameters	$w_{unf} = 0.5$, $w_{Cost} = 0.3$, $w_{ExecTime} = 0.3$, $w_{Avail} = 0.1$, $w_{Sec} = 0.2$, $w_{Rel} = 0.1$	

For each combination of technique, problem instance and maximum execution time, thirty runs were performed in order to ensure the significance of results

4.3.2. Results

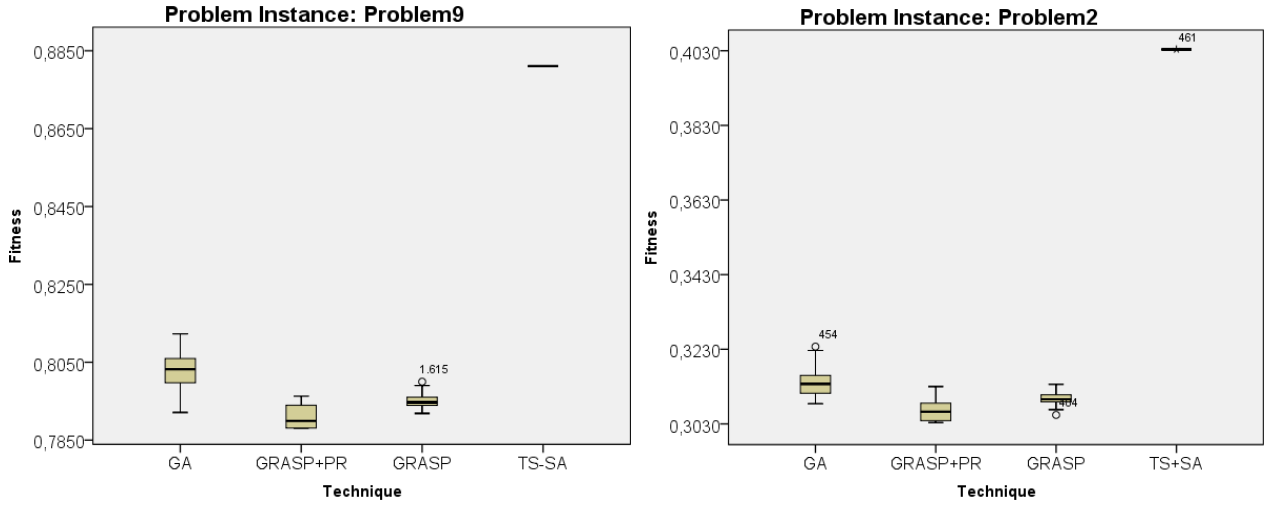
Table 6 shows the mean results per problem instance and execution time. Specifically, table 6 is divided into four sub-tables by execution time. In each sub-table, rows depict the results obtained for each problem instance, and columns depict the results obtained by each optimization technique. The best means per problem

Table 6: Means of obj. func. values for each algorithm and execution time in Experiment 1

Exec. Time	100 ms					1000 ms				
Technique	**GA**	**GRASP+PR (G6)**	**GRASP+PR (G2)**	**GRASP(G1)**	**TS/SA**	**GA**	**GRASP+PR (G6)**	**GRASP+PR (G2)**	**GRASP (G1)**	**TS/SA**
Problem P0	0,317053066	**0,31467194**	0,31559766	0,31585178	0,37823648	0,31702924	**0,31464257**	0,31514186	0,31521718	0,37819911
Problem P1	0,832070664	**0,82958546**	0,83114955	0,82996641	0,90526782	0,83257414	**0,82958531**	0,82991653	0,82992594	0,90526782
Problem P2	0,314220241	**0,30428238**	0,30821952	0,30961194	0,40335166	0,31406676	**0,30334250**	0,30495659	0,30548735	0,40335070
Problem P3	0,786458899	**0,77332510**	0,77774798	0,77939848	0,87387101	0,78422132	**0,77294846**	0,77398728	0,77478427	0,87377022
Problem P4	0,810939436	**0,81066853**	0,81082234	0,81086532	0,81292188	0,81094311	**0,81066853**	0,81072816	0,81073885	0,81291786
Problem P5	0,345341638	**0,33979296**	0,34254369	0,34201717	0,39219569	0,34514013	**0,33978323**	0,34021847	0,34087602	0,39217004
Problem P6	0,814693606	**0,79437721**	0,80665351	0,79564660	0,89469352	0,81558663	**0,79244253**	0,79865127	0,79564660	0,89464300
Problem P7	0,755621698	**0,74596884**	0,74604446	0,74597200	0,82326859	0,75901268	**0,74549613**	0,74544938	0,74549475	0,82099777
Problem P8	0,859142490	**0,85159186**	0,85524606	0,85185832	0,91504732	0,85937275	**0,85159186**	0,85210501	0,85185832	0,91504127
Problem P9	0,802587993	**0,78813945**	0,79375533	0,79500608	0,88106812	0,80275109	**0,78810276**	0,79004525	0,79100871	0,88106812
Problem P10	0,333850406	**0,33271258**	0,33290040	0,33326753	0,34420161	0,33372791	**0,33268621**	0,33266052	0,33268679	0,34393207
Exec. Time	**500 ms**					**50000 ms**				
Technique	**GA**	**GRASP+PR (G6)**	**GRASP+PR (G2)**	**GRASP(G1)**	**TS/SA**	**GA**	**GRASP+PR (G6)**	**GRASP+PR (G2)**	**GRASP (G1)**	**TS/SA**
Problem P0	0,316847884	**0,31465716**	0,31541106	0,31559366	0,37819911	0,31708794	**0,31464248**	0,31503530	0,31511347	0,37819911
Problem P1	0,832185790	**0,82958531**	0,83047811	0,82996641	0,90526782	0,83234986	**0,82958531**	0,82979438	0,82981234	0,90526782
Problem P2	0,314327155	**0,30334881**	0,30667738	0,30770218	0,40335070	0,31456350	**0,30334250**	0,30436232	0,30472537	0,40335070
Problem P3	0,785376182	**0,77303310**	0,77626168	0,77730067	0,87377022	0,78437278	**0,77284804**	0,77318822	0,77396728	0,87377022
Problem P4	0,810937976	**0,81066853**	0,81077270	0,81081414	0,81291786	0,81090575	**0,81066853**	0,81070306	0,81072227	0,81291786
Problem P5	0,345156788	**0,33979296**	0,34138343	0,34179962	0,39217004	0,34478886	**0,33977041**	0,33982145	0,34035985	0,39217004
Problem P6	0,815804062	**0,79307634**	0,80427884	0,79564660	0,89464300	0,81608455	**0,79238052**	0,79697338	0,79564660	0,89464300
Problem P7	0,756262913	**0,74577745**	0,74566567	0,74580965	0,82099777	0,75804813	**0,74542966**	0,74541133	0,74542556	0,82099777
Problem P8	0,859046630	**0,85159186**	0,85385226	0,85185832	0,91504127	0,85909064	**0,85159186**	0,85185755	0,85183389	0,91504127
Problem P9	0,803349334	**0,78810276**	0,79208361	0,79353237	0,88106812	0,80183890	**0,78810276**	0,78949346	0,78994663	0,88106812
Problem P10	0,334067627	**0,33271171**	0,33276010	0,33286655	0,34393207	0,33395456	**0,33265462**	0,33264226	0,33265207	0,34393207

Table 7: Mean percentage of solutions improving any obtained by other technique (Exp. #1)

Exec. Time	100 ms					1000 ms					
	GA	**GRASP+PR1(G6)**	**GRASP+PR(G2)**	**GRASP(G1)**	**TS+SA**		**GA**	**GRASP+PR1(G6)**	**GRASP+PR(G2)**	**GRASP(G1)**	**TS+SA**
GA		0,00%	0,30%	0,00%	100,00%	**GA**		0,00%	0,00%	0,00%	90,91%
GRASP+PR(G6)	92,42%		80,30%	3,94%	100,00%	**GRASP+PR(G6)**	86,67%		68,18%	1,52%	90,91%
GRASP+PR(G2)	35,45%	0,30%		0,30%	100,00%	**GRASP+PR(G2)**	74,85%	0,91%		4,24%	90,91%
GRASP(G1)	84,55%	0,91%	70,00%		100,00%	**GRASP(G1)**	83,03%	0,00%	60,30%		90,91%
TS+SA	0,00%	0,00%	0,00%	0,00%		**TS+SA**	0,30%	0,30%	0,30%	0,30%	
Exec. Time	**500 ms**					**50000 ms**					
	GA	**GRASP+PR1(G6)**	**GRASP+PR(G2)**	**GRASP(G1)**	**TS+SA**		**GA**	**GRASP+PR1(G6)**	**GRASP+PR(G2)**	**GRASP(G1)**	**TS+SA**
GA		0,00%	0,30%	0,00%	90,91%	**GA**		0,91%	0,91%	0,91%	36,97%
GRASP+PR(G6)	87,27%		63,33%	1,21%	90,91%	**GRASP+PR(G6)**	71,52%		36,06%	5,15%	89,39%
GRASP+PR(G2)	58,48%	1,21%		0,91%	90,91%	**GRASP+PR(G2)**	72,73%	0,91%		9,09%	76,06%
GRASP(G1)	83,94%	0,61%	60,30%		90,91%	**GRASP(G1)**	72,73%	0,00%	31,82%		90,61%
TS+SA	0,30%	0,30%	0,30%	0,30%		**TS+SA**	0,91%	0,30%	0,30%	0,61%	



(a) 1.0 – $ObjFunc$ (eq. 3) and problem inst. 9

(b) 1.0 – $ObjFunc$ (eq. 3) and problem inst. 2

Figure 4: Box plots showing the results of each technique in Experiment #1.

Table 5: Problem Instances information

Problem Name	Activities	Abstract Serv.	Candid. Serv.	Global Const.
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

instance and execution time is highlighted in boldface. In this context, it is important to note that the problem was modelled as a minimization problem for compatibility with the experimental framework FOM which implies that the lower the value the better. It is noticeable that GRASP+PR(G6) obtained the best mean results in all cases. GA provides intermediate results, better than TS+SA, but not as good as GRASP+PR and GRASP. The performance of TS+SA was bad except for tightly constrained problem instances. Our statistical analysis revealed that the differences among GRAPS+PR(G6) and the other techniques are statistically significant (with $\alpha = 0.05$) except for one problem instance and technique. Specifically, the differences between GRASP(G1) and GRAPS+PR(G6) are not significant for Problem $P7$ when execution times are longer than $500ms$. It is worth noting that $P7$ is significantly smaller that the others. It contains only 7 tasks and 63 candidate services. Thus, authors infer that for small instances of the problem, GRASP(G1) can behave nearly as well as GRASP+PR(G6). The causes of this behaviour could be: (i) the inefficiency of the intensification strategy of PR, since the probability of overlapping of paths is bigger for small problem instances; and (ii) the capability of GRASP for exploring the promising area of the search space for small problem instances.

In order to evaluate the extent to which some techniques outperform others, we computed the percentage of runs where the result obtained by one technique are better than any result (out of the 30 runs) obtained by other technique (for the same problem instance and execution time). Table 7 summarizes these results. It is divided into four sub-tables by execution time, were each sub-table contains a square matrix with the optimization techniques in rows and columns. Specifically, the value of a cell is the mean of the percentage described above for the problem instances. For instance, the value in the second row and first column of the top-left sub-table specifies that, for execution times of 100ms, on average for all the problem instances, a 92.42% of the solutions obtained by GRASP+PR(G6) are better than any solution obtained by GA. This means that the results obtained by GRASP+PR(G6) outperform those obtained by GA. Since the percentages are averaged for all the problem instances and refer to different pairs of techniques, the sum by rows and columns is not 100%. Table 7 confirms the conclusions drawn above, since the row of GRASP+PR(G6) has the higher percentage in almost any execution time and column. However, it is noticeable the small percentage of such row for the column of GRASP(G1), while the transposed cell (row GRASP(G1) and column GRASP+PR(G6)) has also a small percentage. This means that, although on average the results of GRASP+PR(G6) are better and have less dispersion than those of GRASP(G1), the latter can find occasionally better solutions than those usually found by the former. Another noticeable finding is the progressive decrease of the percentages of GRASP+PR(G6) and GRASP(G1) when execution time increases.

Fig. 4 shows box plots for two problem instances with a termination criterion of 100ms: each figure depicts four populations, defined as the values of $ObjFunc$ for the best solution obtained in the runs of an optimization technique. Thus each population has 30 samples. Results of GRASP+PR(G6) are labelled as GRASP+PR, and those of GRASP(G1) as GRASP. Specifically, for each population the boxplot shows: the minimum sample represented as the lower horizontal line segment, lower quartile (Q1) represented as the lower limit of the box, median (Q2) segment dividing the box, upper quartile (Q3) represented as the top of the box, and largest sample represented as the upper horizontal line segment. Samples considered outliers are represented as circles or stars. The distribution of the results obtained by GRASP+PR is the best in both figures. The small variability of the results provided by TS+SA is analysed in depth in (Parejo et al., 2013).

The improvements provided by our proposals are significant not only in a statistical sense, but also in terms of the actual QoS provided. 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 noteworthy when translated into costs savings and execution time decreases.

4.4. Experiment #2

In order to ensure that the differences between our proposals and the previous approaches do not depend on the specific fitness function and problem instances used, we repeated the experiment using 11 additional problem instances (described in (Parejo et al., 2013)), and the objective function defined in (Canfora et al., 2005b):

$$f^{min}_{Canf}(\chi) = \frac{\sum_{q \in Q^-} (w_q \cdot U_q(Q_q(\chi))}{\sum_{q \in Q^+} (w_q \cdot U_q(Q_q(\chi))} + w_{unf} \cdot D_f(\chi) \quad (9)$$

The information of these additional problem instances are shown in table 8.

Table 8: Additional problem instances information

Problem Name	Activities	Abstract Serv.	Candid. Serv.	Global Const.
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

The results obtained for this experiment are shown in table 9 using the same structure and notation as in table 6. GRASP+PR(G6) generates the best mean results for most problem instances. Specifically, for execution times of 500ms GRASP+PR(G6) provides the best average results for 8 out of 11 problem instances. TS+SA provided the best mean results for problem C2. This fact confirms that for tightly constrained problem instances it can perform better than GA and the GRASP-based proposal. This result is coherent, since it prioritizes constraint satisfaction in the search process (Koa et al., 2008). GRASP provided the best mean results for two problem instances (C5 and C6).

Table 10 shows the mean percentages of improvements in a similar way as table 7. Again, GRASP+PR(G6) provided the highest percentages in general. The capability of GRASP(G1) for finding sporadically the best results is confirmed by the results in table 10. Moreover, the decreasing trend of the percentages of GRASP+PR(G6) when execution time increases is also significant. A noticeable difference regarding table 7 are the percentages of TS+SA. The performance of this technique is much better in this experiment. Thus, the performance of TS+SA is highly influenced by the specific objective function used for modelling the global utility.

Statistical tests confirmed that the differences in the group of techniques were statistically significant in almost all cases. The only exception were the differences between GRASP+PR(G6) and TS+SA for problem (C2) and execution times of 50000ms.

Figure 4.4 shows two box plots depicting the results of each technique for two different problem instances with eq. 9 as objective function, and a termination criterion of 100ms. Again, the distribution of GRASP+PR is the best in both figures.

5. Threats to validity

In order to clearly outline the limitations of the experimental study, next we discuss internal and external validity threats.

Internal validity. This refers to whether there is sufficient evidence to support the conclusions and the sources of bias that could compromise those conclusions. In order to minimize the impact of external factors in our results, QoS-Gasp was executed 30 times per problem instance to compute averages. Moreover, statistical tests were performed to ensure significance of the differences identified between the results obtained by the compared proposals. Finally, the experiments were executed in a dedicated computer which provided us with a stable experimental platform.

External validity. This 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 and generalizability of the conclusions. Regarding the limitations, experiments showed no significant improvements when comparing QoS-Gasp with a simple GRASP for small problem instances and short execution times. As stated in section 4.3.2, this limitation is due to: (i) the capability of GRASP to explore a significant amount of the search space, and (ii) the overlapping of the paths explored by PR for such small problem instances.

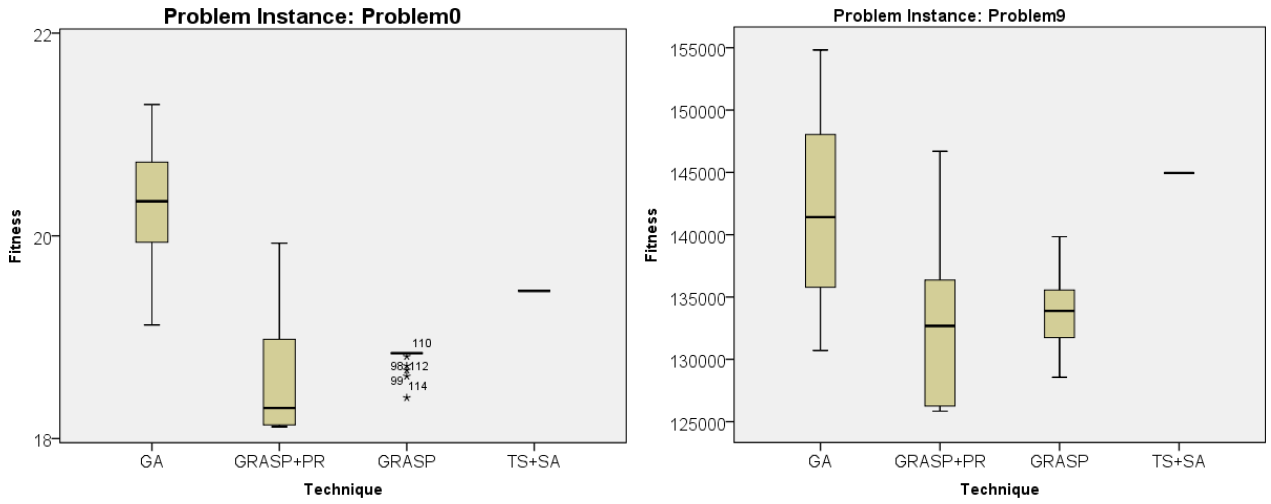
Regarding the generalizability of conclusions, two different objective functions, and two different sets of problem instances were used. Additionally the parameters and size were chosen from a survey of the most

Table 9: Means of obj. func. values for each algorithm and execution time in Experiment 2

Exec. Time	**100 ms**					**1000 ms**				
Technique	**GA**	**GRASP+PR (G6)**	**GRASP+PR (G2)**	**GRASP(G1)**	**TS/SA**	**GA**	**GRASP+PR (G6)**	**GRASP+PR (G2)**	**GRASP (G1)**	**TS/SA**
Problem C0	20,3294	**18,1494**	19,0278	18,8089	19,4567	20,2537	**18,1294**	18,7945	18,3892	19,4567
Problem C1	17546,9250	**16798,8826**	16892,6761	16883,4022	18028,5566	17344,5197	**16795,3229**	16836,7681	16799,8261	18028,5566
Problem C2	77,4635	53,3737	69,3314	50,6206	**47,2274**	77,8725	49,1226	62,7276	50,6205	**47,2274**
Problem C3	365838,7379	**354607,1630**	357263,5720	357324,9570	381218,1130	366237,9430	**353935,1220**	355399,0430	353959,6700	381218,1130
Problem C4	4660,0503	**2688,8626**	4032,7248	2817,3613	2758,6429	4729,2815	**2379,8780**	3529,8844	2817,3613	2758,6429
Problem C5	43077,7130	40087,5854	41927,6160	**39712,0757**	39804,2874	43157,9618	40039,7574	40893,9150	**39712,0757**	39804,2874
Problem C6	504,0981	353,8804	367,8332	**347,8916**	348,1642	499,4118	354,9664	368,2569	**344,4944**	348,1642
Problem C7	29445,2042	32899,8809	25257,0317	**19163,0773**	20070,3702	28586,1747	**15381,4457**	19556,9182	18875,8913	20070,3702
Problem C8	623,4833	**590,7976**	604,8967	605,9958	653,1621	622,9089	**556,9008**	581,6285	574,9983	653,1621
Problem C9	141414,7664	**129780,8750**	135381,2160	133877,3010	144955,3870	143082,5650	**125785,1540**	129145,8090	128143,6330	144955,3870
Problem C10	21682,8585	**20345,8959**	20448,2644	20392,9211	26421,6496	21699,4216	**20183,3250**	20295,5048	20228,4809	26421,6496
Exec. Time	**500 ms**					**50000 ms**				
Technique	**GA**	**GRASP+PR (G6)**	**GRASP+PR (G2)**	**GRASP(G1)**	**TS/SA**	**GA**	**GRASP+PR (G6)**	**GRASP+PR (G2)**	**GRASP (G1)**	**TS/SA**
Problem C0	20,2267	**18,1300**	18,7870	18,7038	19,4567	20,5018	**18,1271**	18,7918	18,2799	19,4567
Problem C1	17538,6257	**16793,7861**	16853,8274	16814,3450	18028,5566	17444,8214	**16789,8868**	16841,7142	16789,4079	18028,5566
Problem C2	77,2953	50,7654	64,8281	50,6206	**47,2274**	79,1106	47,3028	63,4201	50,6206	**47,227387**
Problem C3	371234,6973	**354238,9660**	354854,3030	354498,9940	381218,1130	368806,7140	**353530,0840**	354685,5790	353517,1300	381218,1130
Problem C4	4717,7005	**2522,7860**	3846,3891	2817,3613	2758,6429	4591,2618	**2474,0530**	3550,5512	2817,3613	2758,6429
Problem C5	43167,4373	40087,5854	40992,9427	**39712,0757**	39804,2874	43269,8455	40228,1459	40794,3524	**39712,0757**	39804,2874
Problem C6	512,1532	352,3514	368,4614	**347,2431**	348,1642	513,9603	360,8437	377,2588	**340,9694**	348,1642
Problem C7	27976,6404	**16228,1583**	22018,1279	19156,5452	20070,3702	28453,8342	**14439,0012**	19615,3699	18326,1133	20070,3702
Problem C8	621,2869	**565,9943**	593,6641	591,7945	653,1621	623,6269	**557,2021**	584,9948	568,9675	653,1621
Problem C9	143803,8980	**126129,2150**	131166,6470	130685,0550	144955,3870	142111,7900	**125284,279**	129281,0100	127287,3410	144955,3870
Prolem C10	21803,1250	**20189,7876**	20295,9563	20295,1878	26421,6496	21672,8291	**20204,3569**	20282,9220	20213,8838	26421,6496

Table 10: Mean percentage of solutions improving any obtained by other technique (Exp. #2)

Exec. Time	**200 ms**					**1000 ms**					
	GA	**GRASP+PR1(G6)**	**GRASP+PR(G2)**	**GRASP(G1)**	**TS+SA**		**GA**	**GRASP+PR1(G6)**	**GRASP+PR(G2)**	**GRASP(G1)**	**TS+SA**
GA		0,00%	0,00%	0,00%	41,21%	**GA**		0,61%	0,61%	0,61%	35,15%
GRASP+PR(G6)	94,85%		59,09%	13,94%	76,97%	**GRASP+PR(G6)**	75,45%		59,39%	2,42%	72,12%
GRASP+PR(G2)	62,73%	0,00%		1,21%	62,12%	**GRASP+PR(G2)**	40,91%	0,30%		2,73%	54,55%
GRASP(G1)	100,00%	7,88%	62,42%		90,91%	**GRASP(G1)**	75,45%	0,61%	57,88%		70,00%
TS+SA	45,15%	9,09%	26,97%	9,09%		**TS+SA**	36,06%	18,18%	27,27%	27,27%	
Exec. Time	**500 ms**					**50000 ms**					
	GA	**GRASP+PR1(G6)**	**GRASP+PR(G2)**	**GRASP(G1)**	**TS+SA**		**GA**	**GRASP+PR1(G6)**	**GRASP+PR(G2)**	**GRASP(G1)**	**TS+SA**
GA		0,30%	0,30%	0,30%	42,73%	**GA**		0,91%	0,91%	0,91%	35,45%
GRASP+PR(G6)	88,79%		74,24%	8,18%	73,33%	**GRASP+PR(G6)**	71,52%		56,97%	8,48%	66,97%
GRASP+PR(G2)	61,21%	0,00%		1,21%	56,06%	**GRASP+PR(G2)**	54,24%	0,00%		1,82%	48,79%
GRASP(G1)	90,61%	3,33%	73,94%		74,55%	**GRASP(G1)**	72,42%	0,00%	56,06%		65,15%
TS+SA	36,06%	18,18%	36,36%	18,18%		**TS+SA**	26,97%	18,18%	18,48%	18,18%	



(a) f^{min}_{Canf} (eq. 9) and problem instance C0

(b) f^{min}_{Canf} (eq. 9) and problem instance C9

Figure 5: Box plots showing the results of each technique in Experiment #2.

common values used in the literature (*cf.* tables of problem instance parameters in (Parejo et al., 2013) and (Strunk, 2010)). The use of bigger problem instances could introduce bias in the results, since it fosters the performance of techniques that restrict the area of the search space explored (such as GRASP). Finally, conclusions regarding the performance of QoS-Gasp are not generalizable to scenarios with longer exeuctions times, pointing out a direction of future work.

6. Related Work

QoS-aware service composition brings the dynamic and loosely coupled service selection paradigm of service orientation to its maximum expression. Apart from its implementation in working service oriented architectures (Paik et al., 2012), this problem provides an excellent application scenario for different methods and techniques, ranging from pure optimization techniques to artificial intelligence systems. Two kinds of algorithms have been proposed to solve this problem in literature (Zeng et al., 2004; Ardagna and Pernici, 2005): global and local selection algorithms. Local selection algorithms choose the best candidate for each isolated task, without taking into account the aggregated QoS of the composition. Local selection algorithms have two main drawbacks: (i) solutions obtained are sub-optimal, regarding to the overall quality of the CWS; and (ii) they do not support global or interdependence constraints. Global approaches try to optimize the whole set of services used in the composition taking into account the structure of the composition, overcoming those drawbacks. QoS-Gasp is a global selection algorithm.

Hybrid algorithms that combines local and global selection algorithms has also been proposed (Alrifai and Risse, 2009) and (Alrifai et al., 2012). The types of global selection algorithms for solving the QoS-aware web service composition problem are:

Mathematical programming techniques, such as Integer (Zeng et al., 2004) (Aggarwal et al., 2004), Linear (Cardellini et al., 2007) or Mixed (I/L) Programming techniques (Ardagna and Pernici, 2007) (Qu et al., 2006). 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 instances, genetic algorithms outperform these techniques for problem instances with an average number of candidates per service bigger than 17(Canfora et al., 2005a). Moreover, those mathematical programming techniques require the linearity of constraints and optimization criterion. For instance, such techniques could not optimize fuzzy utility functions (Wang, 2009).

Heuristic and Metaheuristic techniques. In (Jaeger et al., 2005) and (Comes et al., 2010) some specific heuristics are developed to solve the QoSWSC problem. Many to solve this problem are based on evolutionary algorithms, using genetic algorithms (Canfora et al., 2005b) and more recently adaptive genetic programming (Yu et al., 2013). Most those approaches incorporate variants to the work presented in (Canfora et al., 2005a), modifying the encoding scheme, the objective function or QoS model (Gao et al., 2007) (Su et al., 2007) (Wang et al., 2007), or using population diversity handling techniques (Zhang et al., 2006) (Zhang et al., 2007). In (Claro et al., 2005) and (Wada et al., 2012) 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 (Penta and Troiano, 2005) fuzzy logic is used to relax the QoS constraints that are not met and find alternative solutions. Using SA was proposed in (Wang et al., 2007), but no experimental results were provided. In (Zhao et al., 2013) a negative selection algorithm, *i.e.* a variant of artificial immune system, is applied to solve this problem. In (do Prado et al., 2013) the efficiency of several variants of genetic algorithms and exhaustive search are compared.

Classical strategies & other approaches. Classical problem solving strategies such as branch & bound (Liu et al., 2012), and divide & conquer (Qi et al., 2013) have been adapted to solve this problem recently. In (Zou et al., 2012) numeric temporal planning is applied to generate QoS aware web service compositions (including both the QoS-aware binding of the tasks and the composition structure).

Regarding problem variants, in (Leitner et al., 2011) a related problem that uses cost as the QoS property but takes into account service compositions with penalty clauses is solved using HC, GA, memetic algorithms and GRASP. This same problem is solved in (Leitner et al., 2013), adding a branch and bound algorithm to the comparative.

Our results are in accordance with (Leitner et al., 2011) and (Leitner et al., 2013), where GRASP provides the best execution time in general, not only for the cost-based optimization with penalties. We show that GRASP outperforms simple genetic algorithms and hybrid tabu search with simulated annealing for the general QoS-aware composition problem. Furthermore, we show that the hybridization of GRASP with PR provides significant QoS improvements.

Other variants of the problem modify the set of QoS

properties or the definition of the objective function presented in this paper comprise: the inclusion of risk analysis (Ma and Yeh, 2012), and robustness (Wagner et al., 2012) in the objective function, the use of goal oriented requirements (Oster et al., 2012), or the inclusion of network-specific QoS attributes (Klein et al., 2012) (Klein et al., 2013). Moreover, in (Ramacher and Mönch, 2012) the uncertainty of the values of the QoS attributes is taken into account, and in (Ma et al., 2013) their dependency on the parameters of the service invocation is addressed.

Finally, regarding the application contexts of the QoS-aware web service composition, it has recently applied to: optimize network latency in Cloud environments (Klein et al., 2012); improve the robustness and flexibility of systems using data from dynamic sensor networks (Geyik et al., 2013) (Efstathiou et al., 2013); and to optimize the allocation of resources in situational computing applications (Sandionigi et al., 2013).

7. Conclusion

In this paper, a novel algorithm named QoS-Gasp for solving the QoSWSC Problem has been proposed. Experiments show that QoS-Gasp outperforms previous metaheuristic proposals in rebinding scenarios. Our proposal improves the QoS of bindings found, implying cost savings, increased availability and reductions of execution times. As future work we plan to compare QoS-Gasp with IP/MP proposals (Zeng et al., 2004; Ardagna and Pernici, 2007) for instances with linear aggregation functions, and to use WS-Agreement for expressing the QoS guarantees and constraints. Additionally, we plan to compare the efficiency QoS-Gasp and previous proposals when using datasets based on real web services and QoS measurements, such as the QWS dataset (Al-Masri and Mahmoud, 2008).

Acknowledgment and Materials

This work was partially supported by the EU Commission (FEDER), the Spanish and the Andalusian R&D&I programmes grants SETI (TIN2009-07366), TAPAS (TIN2012-32273), COPAS (P12-TIC-1867) and THEOS (TIC-5906). All the source code, raw data, and statistical analysis are available at `http://wp.me/P2WIFP-v`.

References

Aggarwal, R., Verma, K., Miller, J., Milnor, W., 2004. Constraint driven web service composition in meteor-s, in: SCC '04: Proc. of the 2004 IEEE Int. Conf. on Services Computing, IEEE Comp. Society. pp. 23–30.

Al-Masri, E., Mahmoud, Q.H., 2008. Investigating web services on the world wide web, in: Proceedings of the 17th International Conference on World Wide Web, ACM, New York, NY, USA. pp. 795–804. URL: `http://doi.acm.org/10.1145/1367497.1367605`, doi:`10.1145/1367497.1367605`.

Alrifai, M., Risse, T., 2009. Combining global optimization with local selection for efficient qos-aware service composition, in: In International World Wide Web Conference, ACM. pp. 881–890.

Alrifai, M., Risse, T., Nejdl, W., 2012. A hybrid approach for efficient web service composition with end-to-end qos constraints. ACM Trans. Web 6, 7:1–7:31. URL: `http://doi.acm.org/10.1145/2180861.2180864`, doi:`10.1145/2180861.2180864`.

Ardagna, D., Pernici, B., 2005. Global and local qos guarantee in web service selection, in: BPM Workshops, pp. 32–46.

Ardagna, D., Pernici, B., 2007. Adaptive service composition in flexible processes. Software Engineering, IEEE Transactions on 33, 369–384.

Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R., 2006. Heuristics for qos-aware web service composition. ICWS '06 , 72–82.

Bonatti, P.A., Festa, P., 2005. On optimal service selection, in: WWW '05: 14th international conference on World Wide Web, pp. 530–538.

Canfora, G., Penta, M.D., Esposito, R., Villani, M., 2005a. Qos-aware replanning of composite web services. Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on 1, 121–129. doi:`10.1109/ICWS.2005.96`.

Canfora, G., Penta, M.D., Esposito, R., Villani, M.L., 2005b. An approach for qos-aware service composition based on genetic algorithms, in: GECCO '05, pp. 1069–1075.

Canfora, G., Penta, M.D., Esposito, R., Villani, M.L., 2008. A framework for qos-aware binding and re-binding of composite web services. Journal of Systems and Software 81, 1754–1769.

Cardellini, V., Casalicchio, E., Grassi, V., Presti, F.L., 2007. Efficient provisioning of service level agreements for service oriented applications, in: IW-SOSWE07, pp. 29–35.

Claro, D., Albers, P., Hao, J., 2005. Selecting web services for optimal composition, in: ICWS05.

Comes, D., Baraki, H., Reichle, R., Zapf, M., Geihs, K., 2010. Heuristic approaches for qos-based service selection, in: ICSOC'10, Springer.

Dreo, J., Petrowski, A., Taillard, E., 2003. Metaheuristics for Hard Optimization. Springer.

Efstathiou, D., Mcburney, P., Zschaler, S., Bourcier, J., 2013. Flexible QoS-Aware Service Composition in Highly Heterogeneous and Dynamic Service-Based Systems, in: WiMob - The 9th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications - 2013, Lyon, France. URL: `http://hal.inria.fr/hal-00859891`.

Fernandez, P., Resinas, M., Corchuelo, R., 2006. Towards an automatic service trading. Upgrade 7, 26–29.

Festa, P., Mauricio, Resende, G.C., 2002. Grasp: An annotated bibliography, in: Essays and Surveys in Metaheuristics, Kluwer Academic Publishers. pp. 325–367.

Gao, C., Cai, M., Chen, H., 2007. Qos-driven global optimization of services selection supporting services flow re-planning, in: Adv. in Web, Network Technologies, and Information Management. Springer. LNCS, pp. 516–521.

Gendreau, M., Potvin, J., 2010. Handbook of metaheuristics. volume 146. Springer.

Geyik, S.C., Szymanski, B.K., Zerfos, P., 2013. Robust dynamic service composition in sensor networks. Services Computing, IEEE

Transactions on 6, 560–572. doi:10.1109/TSC.2012.26.

Jaeger, M.C., Mühl, G., Golze, S., 2005. Qos-aware composition of web services: An evaluation of selection algorithms, in: On the Move to Meaningful Internet Systems. Springer. LNCS, pp. 646–661.

Klein, A., Fuyuki, I., Honiden, S., 2013. Sanga: A self-adaptive network-aware approach to service composition. Services Computing, IEEE Transactions on PP, 1–1. doi:10.1109/TSC.2013.2.

Klein, A., Ishikawa, F., Honiden, S., 2012. Towards network-aware service composition in the cloud, in: Proceedings of the 21st International Conference on World Wide Web, ACM, New York, NY, USA. pp. 959–968. URL: http://doi.acm.org/10.1145/2187836.2187965, doi:10.1145/2187836.2187965.

Koa, J.M., Kima, C.O., Kwonb, I.H., 2008. Quality-of-service oriented web service composition algorithm and planning architecture. Journal of Systems and Software 81, 2079–2090.

Laguna, M., Martí, R., 1999. Grasp and path relinking for 2-layer straight line crossing minimization. INFORMS Journal on Computing 11, 44–52.

Leitner, P., Hummer, W., Dustdar, S., 2011. Cost-based optimization of service compositions. IEEE Tran. on Serv. Comp. 99. doi:http://doi.ieeecomputersociety.org/10.1109/TSC.2011.53.

Leitner, P., Hummer, W., Dustdar, S., 2013. Cost-based optimization of service compositions. Services Computing, IEEE Transactions on 6, 239–251. doi:10.1109/TSC.2011.53.

Liu, M., Wang, M., Shen, W., Luo, N., Yan, J., 2012. A quality of service (qos)-aware execution plan selection approach for a service composition process. Future Gener. Comput. Syst. 28, 1080–1089. URL: http://dx.doi.org/10.1016/j.future.2011.08.017, doi:10.1016/j.future.2011.08.017.

Ma, H., Bastani, F., Yen, I.L., Mei, H., 2013. Qos-driven service composition with reconfigurable services. Services Computing, IEEE Transactions on 6, 20–34. doi:10.1109/TSC.2011.21.

Ma, S.P., Yeh, C.L., 2012. Service composition management using risk analysis and tracking, in: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (Eds.), Service-Oriented Computing. Springer Berlin Heidelberg. volume 7636 of *Lecture Notes in Computer Science*, pp. 533–540. URL: http://dx.doi.org/10.1007/978-3-642-34321-6_37, doi:10.1007/978-3-642-34321-6_37.

Oster, Z., Ali, S., Santhanam, G., Basu, S., Roop, P., 2012. A service composition framework based on goal-oriented requirements engineering, model checking, and qualitative preference analysis, in: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (Eds.), Service-Oriented Computing. Springer Berlin Heidelberg. volume 7636 of *Lecture Notes in Computer Science*, pp. 283–297. URL: http://dx.doi.org/10.1007/978-3-642-34321-6_19, doi:10.1007/978-3-642-34321-6_19.

Paik, I., Chen, W., Huhns, M., 2012. A scalable architecture for automatic service composition. Services Computing, IEEE Transactions on PP, 1–1. doi:10.1109/TSC.2012.33.

Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., 2007. Service-oriented computing state of the art and research challenges. IEEE Computer 40, 38–45.

Parejo, J.A., Fernández, P., Ruiz-Cortés, A., 2013. On parameter selection and problem instances generation for QoS-aware binding using GRASP and Path-Relinking. Research Report 2011-4. ETSII. Av. Reina Mercedes s/n. 41012. Sevilla. Spain.

Parejo, J.A., Racero, J., Guerrero, F., Kwok, T., Smith, K., 2003. Fom: A framework for metaheuristic optimization, in: ICCS'03, pp. 886–895.

Parejo, J.A., Ruiz-Cortés, A., Lozano, S., Fernandez, P., 2012. Metaheuristic optimization frameworks: a survey and benchmarking. Soft Computing 16, 527–561.

Penta, M.D., Troiano, L., 2005. Using fuzzy logic to relax constraints in ga-based service composition, in: GECCO.

do Prado, P.F., Nakamura, L.H.V., Estrella, J.C., Santana, M.J., Santana, R.H.C., 2013. A performance evaluation study for qos-aware web services composition using heuristic algorithms, in: ICDS 2013, The Seventh International Conference on Digital Society, pp. 53–58.

Qi, L., Ni, J., Ma, C., Luo, Y., 2013. A decomposition-based method for qos-aware web service composition with large-scale composition structure, in: SERVICE COMPUTATION 2013, Proceeding of the Fifth International Conferences on Advanced Service Computing, pp. 81–86.

Qu, Y., Lin, C., Wang, Y., Shan, Z., 2006. Qos-aware composite service selection in grids. GCC 2006 , 458–465doi:10.1109/GCC.2006.77.

Ramacher, R., Mönch, L., 2012. Dynamic service selection with end-to-end constrained uncertain qos attributes, in: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (Eds.), Service-Oriented Computing. Springer Berlin Heidelberg. volume 7636 of *Lecture Notes in Computer Science*, pp. 237–251. URL: http://dx.doi.org/10.1007/978-3-642-34321-6_16, doi:10.1007/978-3-642-34321-6_16.

Resende, M.G.C., 2009. Greedy randomized adaptive search procedures, in: Encyclopedia of Optimization, pp. 1460–1469.

Sandionigi, C., Ardagna, D., Cugola, G., Ghezzi, C., 2013. Optimizing service selection and allocation in situational computing applications. Services Computing, IEEE Transactions on 6, 414–428. doi:10.1109/TSC.2012.18.

Strunk, A., 2010. Qos-aware service composition: A survey, in: Web Services (ECOWS), 2010 IEEE 8th European Conference on, pp. 67 –74.

Su, S., Zhang, C., Chen, J., 2007. An improved genetic algorithm for web services selection, in: Distributed Applications and Interoperable Systems. Springer. volume 4531 of *LNCS*, pp. 284–295.

Wada, H., Suzuki, J., Yamano, Y., Oba, K., 2012. E3: A multiobjective optimization framework for sla-aware service composition. Services Computing, IEEE Transactions on 5, 358–372. doi:10.1109/TSC.2011.6.

Wagner, F., Kloepper, B., Ishikawa, F., Honiden, S., 2012. Towards robust service compositions in the context of functionally diverse services, in: Proceedings of the 21st International Conference on World Wide Web, ACM, New York, NY, USA. pp. 969–978. URL: http://doi.acm.org/10.1145/2187836.2187966, doi:10.1145/2187836.2187966.

Wang, H., Tong, P., Thompson, P., Li, Y., 2007. Qos-based web services selection. icebe 0, 631–637.

Wang, P., 2009. Qos-aware web services selection with intuitionistic fuzzy set under consumer's vague perception. Expert Systems with Applications 36, 4460 – 4466.

Yu, Y., Ma, H., Zhang, M., 2013. An adaptive genetic programming approach to qos-aware web services composition, in: Evolutionary Computation (CEC), 2013 IEEE Congress on, pp. 1740–1747. doi:10.1109/CEC.2013.6557771.

Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnam, J., Chang, H., 2004. Qos-aware middleware for web services composition. IEEE Tran. Sof. Eng 30, 311–327.

Zhang, C., Su, S., Chen, J., 2006. Efficient population diversity handling genetic algorithm for qos-aware web services selection, in: Computational Science. Springer. volume 3994 of *LNCS*, pp. 104–111.

Zhang, C., Su, S., Chen, J., 2007. Diga: Population diversity handling genetic algorithm for qos-aware web services selection. Comput. Commun. 30, 1082–1090.

Zhao, X., Wen, Z., Li, X., 2013. Qos-aware web service selection with negative selection algorithm. Knowledge and Information Systems , 1–25URL: http://dx.doi.org/10.1007/s10115-013-0642-x, doi:10.1007/s10115-013-0642-x.

Zheng, H., Zhao, W., Yang, J., Bouguettaya, A., 2012. Qos analysis for web service compositions with complex structures. Services Computing, IEEE Transactions on PP, 1. doi:10.1109/TSC.2012.7.

Zou, G., Lu, Q., Chen, Y., Huang, R., Xu, Y., Xiang, Y., 2012. Qos-aware dynamic composition of web services using numerical temporal planning. Services Computing, IEEE Transactions on PP, 1–1. doi:10.1109/TSC.2012.27.