Applying QoS-Aware Service Selection on Functionally Diverse Services

Florian Wagner[1,⋆], Fuyuki Ishikawa[2], and Shinichi Honiden[1,2]

[1] The University of Tokyo, Japan
[2] National Institute of Informatics, Tokyo, Japan
{florian,f-ishikawa,honiden}@nii.ac.jp

Abstract. The Service-Oriented Computing (SOC) paradigm envisions the composition of loosely coupled services to build complex applications. Most current selection algorithms assume that all services assigned to a certain task provide exactly the same functionality.

However, in realistic settings larger groups of services exist that share the same purpose, yet provide a slightly different interface. Incorporating these services increases the number of potential solutions, but also includes functional invalid configurations, resulting in a sparse solution space. As a consequence, applying naïve heuristic algorithms leads to poor results by reason of the increased probability of local optima.

For that purpose, we propose a functionality clustering in order to leverage background knowledge on the compatibility of the services. This enables heuristic algorithms to discover valid workflow configurations in shorter time. We integrate our approach into a genetic algorithm by performing repair operations on invalid genomes. In the evaluation we compare our approach with related heuristic algorithms that use the same guided target function but pick services in a random manner.

1 Introduction

1.1 Service Composition

Services are interoperable and reusable software components that are published on company intranets and recently more and more on public servers and cloud systems. Standardized interface description languages such as WSDL specify the functionality of these services. Furthermore, services may provide Service-Level Agreements (SLA) that declare the non-functional parameters [8].

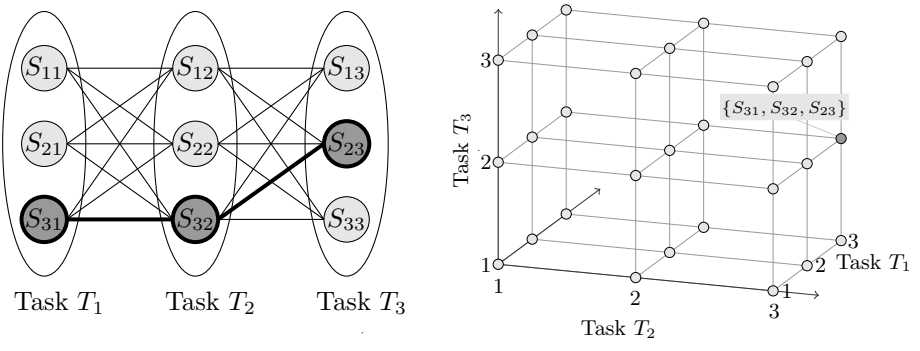
Service composition algorithms are applied in order to provide a virtually unlimited number of functionalities. In general, in the first step a workflow template containing service tasks is provided by the user or by a planning algorithm [12]. A service task is associated with a set of functionally equivalent services, each having varying QoS-attributes. In the next step, a QoS-aware service selection algorithm computes a workflow configuration that determines for each task one single service. The goal is to optimize the utility of the service composition and

⋆ The work of Florian Wagner is partially supported by the KDDI Corporation.

G. Pallis et al. (Eds.): ICSOC 2011, LNCS 7221, pp. 100–113, 2012.

meet the QoS-constraints. Service selection optimizes the consumption of resources and is therefore a major concern for companies, but is also considered as an important issue in research [9].

Since the problem has been proven to be NP-hard [10], enumerating all possible workflow configurations is not feasible. Therefore, approaches that employ near-optimal heuristic algorithms such as hill-climbing [4] and genetic algorithms [1,14] have been applied. All these approaches assume that for each task a set of functionally equivalent services exists. Fig. 1 depicts the service selection problem. In Fig. 1b the solution space is visualized where each solid point is a valid workflow configuration.



(a) All possible service combinations, highlighting one sample workflow configuration

(b) Complete solution space

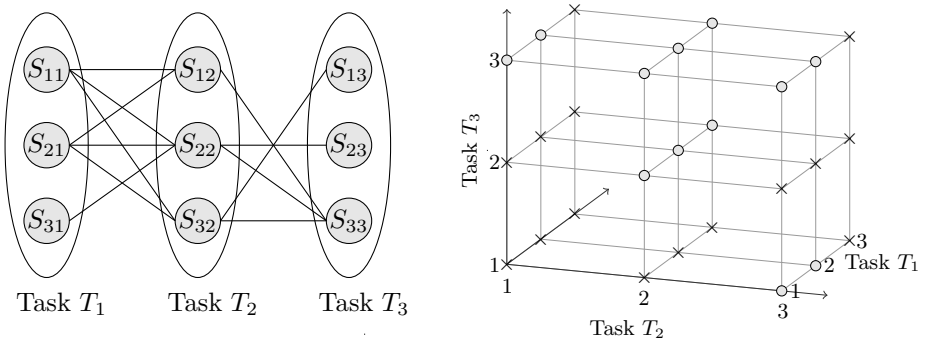
Fig. 1. Classical service selection problem

1.2 Functionally Diverse Services

However, requiring that services associated to a workflow task must have exactly the same functionality might be too restrictive. In reality, various providers offer services that share the same purpose but have slightly varying interfaces, e.g. having optional input parameters or additional result variables. Moreover, service interfaces might be modified and updated during the lifecycle of a service.

Apart from these differences in the interfaces, even if services yield a valid link on the conceptual level, the link may be invalid on the concrete data-structural level [2]. By allowing functionally diverse services in workflow tasks, more candidate services are available, offering more choices to optimize the utility of the workflow.

For these reasons, a certain amount of workflow configurations may be invalid as some services are not functionally compliant. Even though in this scenario the number of possible service combinations in fact decreases, heuristic algorithms achieve only poor results as the solution space is sparse and local optima are more likely. The consequences on the solution space are shown in Fig. 2b. Crosses indicate invalid workflow configurations, e.g. the configuration $\{S_{31}, S_{32}, S_{23}\}$ from Fig. 1a is invalid since there are no valid links between these services.



(a) Restricted service combinations (b) Sparse solution space

Fig. 2. Service selection on functionally diverse services

1.3 Our Contributions

Most related studies present selection algorithms that require functionally equivalent services for each task. Our approach does not have this restriction and is therefore applicable to a wider range of selection problems. Our contributions to the selection problem are as follows:

1. We discuss the implications of introducing functionally diverse services to the selection task. We analyze which heuristic algorithms are applicable to solve the problem and present how these algorithms can be adjusted to find valid workflow configurations and evaluate the resulting performance.
2. We present a method to find valid configurations efficiently based on a functional clustering of candidate services. This method leverages background knowledge on the compliance of the services to prune the search space.
3. We demonstrate how this method can be embedded into a genetic algorithm to repair erroneous genomes. In our detailed evaluation we show which parameters yield the best results for our modified algorithm.

The proposed clustering algorithm is related to the algorithm shown in [11] but uses a more general clustering method. Moreover, the annotations of the graph vary from the previous version and are used for a different purpose.

In the next section we discuss the preliminaries of the selection problem. In Section 3 we present our approach based on a functional clustering of services combined with a genetic algorithm and Section 6 concludes this paper.

2 Preliminaries

In this section we discuss the general service selection problem and further explain the consequences of including functionally diverse services.

2.1 Services

Services provide a certain functionality on a public or private network. In order to specify the functional interface of a service, a description document written in an interface description language such as WSDL is published. Moreover, each service has varying QoS-attributes that are determined in a Service-Level Agreement (SLA) document. In this paper we consider the price, the response time, the reliability, and the availability [13] of a service. A detailed classification on QoS-attributes can be found in [8].

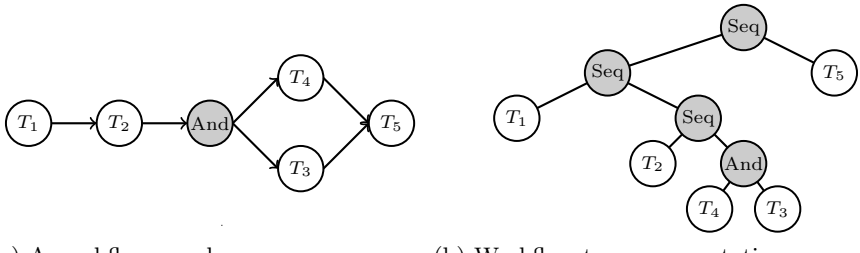
Given the functional interface of a service, service S_1 can be connected to S_2 if S_1 provides an output parameter that is required by S_2.

Comparing a service S_1^T with service S_2^T associated to the same task T, we distinguish three cases:

1. Both services can be connected to the same set of services, in other words the two services are *equivalent*.
2. Service S_1^T can be applied at least in all cases where S_2^T is applicable. In that case we say that S_1^T provides *more functionality* than S_2^T.
3. In all other cases, the two services are *unrelated*.

2.2 Workflows

A workflow is a directed graph where each node is either a task or a control node, depicted in 3a. A task is a set of functional related services that have varying QoS-attributes. A control node determines the control flow of the workflow. In this paper we consider sequences, AND/OR branches, and loops. In the case of loops we "unroll" the loop retrieving the number of expected calls from past invocations. In [3] several possible workflow patterns are discussed in detail.



(a) A workflow graph (b) Workflow tree representation

Fig. 3. Comparison of workflow trees and graph representations. $S_1 - S_5$ depict services, gray nodes are control nodes.

For aggregating the QoS attributes, workflows are often represented as workflow trees, illustrated in 3b. This way, the computations of Table 1 can be applied to the root node. Inner nodes are the control nodes, leaf nodes are the task nodes. For the sake of simplicity we use binary trees in the following.

In order to execute a workflow, a workflow configuration is computed that selects for each task T a service S, where $S \in T$.

QoS-Aggregation. In order to compute the QoS-attributes of a workflow configuration the QoS Q_i of the composite services are aggregated. First, all attributes are normalized between 0 and 1, with 1 being the best value (cf. Equation 1). Therefore, the minimal and the maximum values of the attributes are either pre-defined or determined by comparing all available services. In case the utility of an attribute grows with higher values, the attribute is categorized as a positive attribute Q_{pos}, otherwise as a negative attribute Q_{neg} [6]. For instance, reliability and availability are positive attributes, price and response time are negative attributes.

$$Q'_i = \begin{cases} \frac{Q_i - Q_{min}}{Q_{max} - Q_{min}} & \text{if } Q_i \in Q_{pos} \text{ and } Q_{min} \neq Q_{max}, \\ \frac{Q_{max} - Q_i}{Q_{max} - Q_{min}} & \text{if } Q_i \in Q_{neg} \text{ and } Q_{min} \neq Q_{max}, \\ 1 & \text{else} \end{cases} \tag{1}$$

Subsequently, the normalized attributes Q'_i of the services are aggregated to the QoS $\mathcal{Q}$ of the workflow depending on the characteristics of the attributes and the control flow. Table 1, taken from [1], depicts the aggregation for common control structures and QoS-attributes. The probability p refers to the execution probability of the branch, k is the expected loop count. Both variables are calculated based on either execution logs or estimated by humans. We apply the same computations, but instead of using a probabilistic approach for concurrent execution branches we adopt a worst-case computation.

These computations are first applied to the root node of the workflow tree, iterating subsequently to the child nodes until all nodes have been visited. In each node the QoS of the subtree are aggregated, with the root node containing the aggregated QoS of the workflow.

Table 1. QoS-aggregation of composite services, adopted from [1, Table 1]

Attribute	Sequence	AND-Branch	OR-Branch	Loop
Response time (t)	$\sum_{i=1}^{n} S_i.Q_t$	$\max_{S \in N}(S_i.Q_t)$	$\max_{S \in N}(S_i.Q_t)$	$k \cdot S_i.Q_t$
Price (c)	$\sum_{i=1}^{n} S_i.Q_c$	$\sum_{i=1}^{n} S_i.Q_c$	$\max_{S \in N}(S_i.Q_c)$	$k \cdot S_i.Q_c$
Reliability (r)	$\prod_{i=1}^{n} S_i.Q_r$	$\prod_{i=1}^{n} S_i.Q_r$	$\min_{S \in N}(S_i.Q_r)$	$S_i.Q_r^k$
Availability (a)	$\prod_{i=1}^{n} S_i.Q_a$	$\prod_{i=1}^{n} S_i.Q_a$	$\min_{S \in N}(S_i.Q_a)$	$S_i.Q_a^k$

Utility Computation. After computing the QoS-attributes of the entire workflow, the QoS vector $\mathcal{Q}$ is aggregated to a single value. For that purpose, utility functions are defined, in most studies a Simple Additative Weighting (SAW) is applied as in Equation 2.

$$\mu(\mathcal{Q}) = \sum_{i=1}^{|\mathcal{Q}|} w_i \cdot \mathcal{Q}_i \qquad \text{where } \sum_{i=1}^{|\mathcal{Q}|} w_i = 1 \tag{2}$$

The user provides for each QoS-attribute a certain weight w_i, indicating the user's preferences towards e.g. the price or the response time.

Global QoS-Constraints. Apart from the utility, the user may state global QoS-constraints, determining limits on e.g. the price of the whole workflow composition. In case the constraints are violated, the utility of a workflow becomes 0. Therefore, in this setting QoS-constraints are considered as hard constraints.

2.3 Heuristic Approaches

Since the selection problem is a NP-hard problem, near-optimal heuristic approaches are applied. In case all services provide the same functionality, the functional compliance is always satisfied. Therefore, the algorithms only have to take the QoS-attributes into consideration, by using a heuristic function that subtracts a penalty from the utility if constraints are violated (cf. Eq. 3).

$$\mu_{heu}(\mathcal{Q}) = \mu(\mathcal{Q}) - (w_{qos} \cdot d_{qos}) \tag{3}$$

The value d_{qos} equals the number of violated QoS-constraints multiplied by a certain weight w_{qos}. This way, the algorithm is guided towards a solution that meets the QoS-constraints. In the following, we will discuss two popular heuristic approaches that employ the modified utility function μ_{heu}.

Hill-Climbing. Hill-climbing is a simple greedy approach which tries to improve a selection by replacing a single service in each iteration. First, a random configuration is computed. Next, for a random task the service that yields the best heuristic value is selected.

An extension of the classical hill-climbing algorithm iterates over a set of candidate solutions and in the end chooses the selection with the best heuristic value. This way, the effect of local optima can be reduced but the computation time increases as well.

Genetic Algorithms. Genetic Algorithms maintain a set of genomes that encode a service selection. The genome length equals the workflow length and the cells contain the indices of the selected services, depicted in Fig. 4.

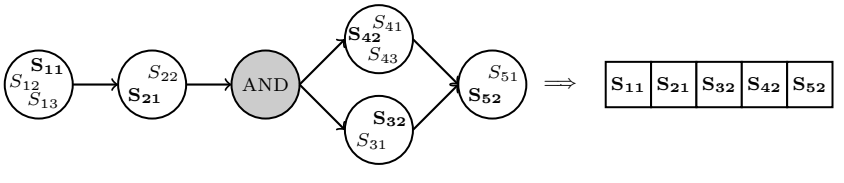


Fig. 4. Translating a workflow configuration (left) to a genome (right)

Genomes are evolved in each generation by applying the following operations:

Crossover given a probability P_{cro}, two genomes are combined by picking a service from the genomes for each task. Alternatively, both genomes are cut into halves and concatenated.

Mutation given a certain probability P_{mut}, for each task the currently selected service is replaced with a random service from the same task.

Selection. In the last step, the genomes are sorted according to their fitness (the utility function μ_{heu}) and only the best ones are kept.

These three steps are iterated either a fixed number of times or until no change occurred to the fitness of the best genome in a certain number of generations. In each iteration the top-k genomes are kept to ensure that these genomes are not discharged.

2.4 Functional Compliance of Services

In case all services associated to a workflow task provide the same functionality, theoretically every possible configuration is executable. However, in reality certain links between services may be invalid. Workflow configurations that contain at least one invalid link cannot be executed.

In most cases, these restrictions can be automatically derived by verifying the type hierarchy of the parameters or the WSDL interface. If a service provides exactly the required output parameters or parameters with additional data to another service, then these services yield a valid link. Apart from that, the user may also declare certain service combinations as invalid, e.g. if two services are hosted on servers that are connected with a low-bandwidth cable. In the following we assume that a relation $combinable : S \times S$ exists, that indicates whether two services can be combined with each other.

Similar to the QoS-constraints, these restrictions are hard constraints, therefore if a selection contains an invalid link, the utility becomes 0.

3 Approach

In this section we present our algorithm that integrates a repair operation for invalid workflow configurations into a genetic algorithm. This operation leverages a functional clustering on the candidate services to resolve invalid links efficiently.

3.1 Functional Clustering

For each workflow task the services are clustered and arranged in a direct-acyclic graph based on their functionality. The functionality of a service S depends on the services that can be connected to S.

Converting the Workflow Tree. Since the functionality of a certain service in a workflow depends on the incoming and outgoing links in a workflow, the workflow tree is first converted into a graph representation to detect these links, described in Algorithm 1. The algorithm starts with the root node, descending first to the task nodes and adding them to the graph G (line 1).

Subsequently, the inner nodes are traversed. In case a sequence is found the incoming task of the left subtree and the outgoing tasks of the right subtree are stored in the sets M_1 and M_2 (line 6 and 7). Then, in line 10 the services of both sets are connected with a directed edge and added to $G.E'$.

Algorithm 1. convert(T, n)

```
   Input: Workflow tree T = (V, E), node pointer n
   Output: Workflow graph G = (V', E')
 1 if n ∈ Tasks then G.V' := G.V' ∪ {n};
 2 else
 3 |   convert(T, n.left);
 4 |   convert(T, n.right);
 5 |   if n is Sequence then
 6 |   |   M1 := outgoingTasks(n.left);
 7 |   |   M2 := incomingTasks(n.right);
 8 |   |   foreach Task t1 ∈ M1 do
 9 |   |   |   foreach Task t2 ∈ M2 do
10 |   |   |   |   G.E' := G.E' ∪ {t1, t2}
11 |   |   |   end
12 |   |   end
13 |   end
14 end
```

In Alg. 2 the helper function `incomingTasks` is shown, computing all services that provide outgoing links in a subtree. In case the parameter n is a task node then the task itself is returned (line 2). Otherwise, if the control node is a sequence then we descend the tree to the right child node (line 4). In case of a parallel control node, the union of the left and right node is returned in line 5.

Algorithm 2. incomingTasks

```
   Input: Node n
   Output: Set M of outgoing tasks in the subtree of n
 1 if n ∈ Tasks then
 2 |   return {n.t};
 3 else
 4 |   if n is Sequence then return incomingTasks(n.left);
 5 |   else return incomingTasks(n.left) ∪ incomingTasks(n.right);
 6 end
```

The helper function `outgoingTasks` is defined in the same way, except for providing `n.right` as a parameter in line 5.

After computing the incoming and outgoing links we determine for each service the set of functional compatible predecessor IL and successor OL services, in other words services that can provide outputs and receive inputs from a service respectively. Given two tasks t and t' where $(t, t') \in G.E'$, for all services

pairs $\{S, S' | S \in t, S' \in t'\}$ that are contained in *combinable*, service S' is added to the set $S'.IL$ and S is added to the set $S'.OL$.

Computing the Functionality Graph. Based on the two sets IL and OL of each service a functionality graph is computed by comparing all services in a task with another. Two rules apply:

1. If $S.IL = S'.IL$ and $S.OL = S'.OL$ then the services are functionally equivalent. In that case, these services are clustered in the same composite node.
2. Otherwise, if $S.IL \subseteq S'.IL$ and $S.OL \subseteq S'.OL$ then S and S' are connected with a directed edge.

Each connected component forms a cluster. In Fig. 5 an example cluster with 12 services is shown, taken from the clustering of the OWLS-TC test[1].

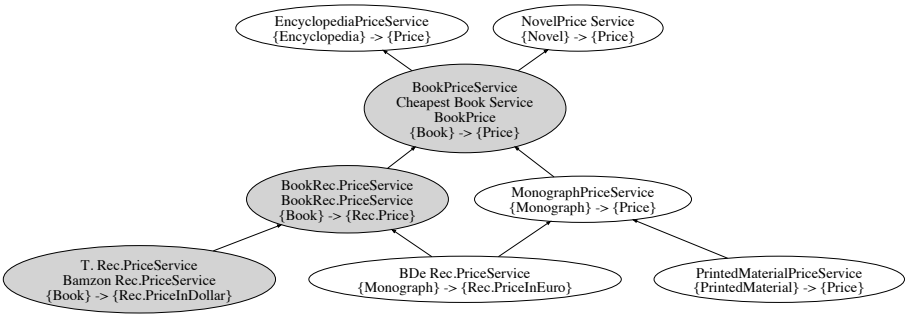
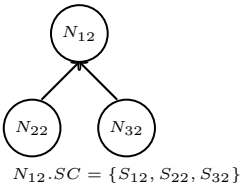


Fig. 5. An example cluster from the resulting clustering of the OWLS-TC testset. Composite services are marked by gray nodes. The relation *combinable* is computed by comparing the parameter types of the services (last line of the nodes).

In the last step, we aggregate in each node the accumulated output list $\mathcal{OL}$ of all OL sets in the subcluster and $\mathcal{SC}$ containing the set of all services in the subcluster.

Nodes	Services	N.IL	N.OL
N_{12}	S_{12}	$\{S_{11}, S_{21}\}$	$\{S_{33}\}$
N_{22}	S_{22}	$\{S_{11}, S_{21}, S_{31}\}$	$\{S_{23}, S_{33}\}$
N_{32}	S_{32}	$\{S_{11}, S_{21}\}$	$\{S_{13}, S_{33}\}$

(a) The computed N.IL and N.OL sets



(b) Resulting graph

Fig. 6. Results of applying the algorithm to Task T_2 of Fig. 2a

[1] http://projects.semwebcentral.org/projects/owls-tc/

The resulting functionality graph of the second task of Fig. 2a is shown in Fig. 6b as an example.

A specific form of the clustering technique is described in [12] where the method is applied to automatic service composition. The functional comparison is based on the parameter types and pre- and postconditions of the services. We present a more general approach to cluster services in this paper that can be applied to any kind of services, even if no type hierarchy and pre- and postconditions are given.

3.2 Combining Functional Clustering with a GA

In our approach we apply a genetic algorithm that performs an intermediate repair operation on the genomes between the mutation and the selection step. The fitness function extends the function μ_{heu} (cf. Eq. 3) in a way that functional compliance is taken into account, shown in Eq. 4.

$$\mu'_{heu}(\mathcal{Q}) = \mu(\mathcal{Q}) - (w_{qos} \cdot d_{qos} + w_{func} \cdot d_{func}) \tag{4}$$

The variable d_{func} indicates the ratio of invalid service links and w_{func} assigns a certain weight to this value. This function is used in the genetic algorithm as a fitness function of the genomes to guide the algorithm towards selections that are functional compatible and meet the QoS constraints.

In each generation an invalid genome is repaired with a certain probability P_{rep}. In general, increasing the repair probability performs repair operations more aggressively but also increases the computational complexity of each generation. By experiment we have evaluated that 33% is a good tradeoff between utility and performance.

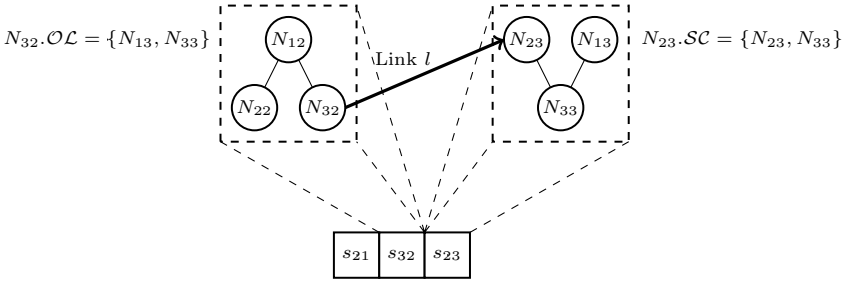


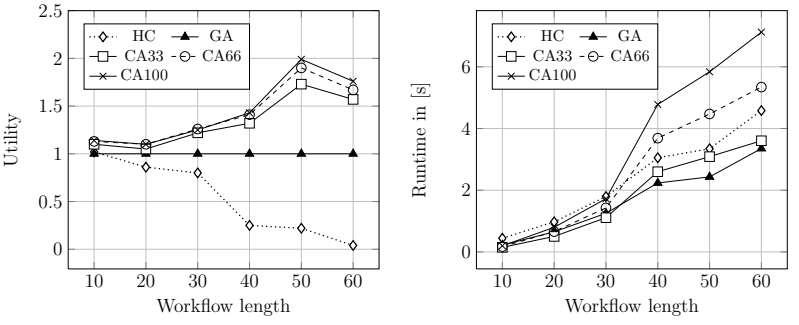
Fig. 7. Repairing an invalid link of a genome

In order to repair an invalid genome, services S_{in} and S_{out} that are part of an invalid link are detected and replaced. First, we detect whether the invalid link can be resolved by replacing one service with a service from its subtree. This is the case when the intersection of $S_{in}.\mathcal{OL}$ and $S_{out}.\mathcal{SC}$ is not empty, depicted in Fig. 7. In the example, N_{23} is replaced by N_{33}.

In case the intersection is empty, one of the parent nodes of S_{in} and S_{out} is selected randomly. If both nodes are already root nodes, then we pick a random root node from a different cluster of the same task.

4 Evaluation

In our evaluations, we use a hill-climbing algorithm (HC) that iterates 100 initial solutions 100 times, therefore 10.000 iterations in total. The genetic algorithm (GA) has 200 seeds, iterating at most 200 times. Furthermore, we evaluate our cluster algorithm (CA) that is based on the GA, including the repair operation and with 100 seeds. We provided various repair probabilities in order to evaluate the optimal parameter, e.g. CA33 is a CA with 33% repair probability. All experiments regarding the performance have been executed on an Intel Quad-core 3.00Ghz with 4GB ram.



(a) Utility comparison (GA baseline) (b) Comparing the runtime

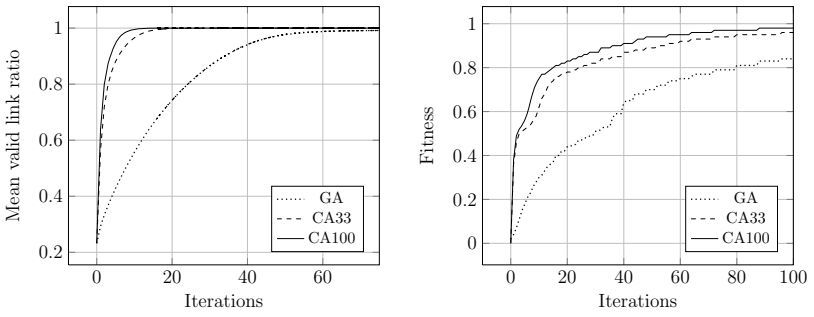
Fig. 8. Evaluating the utility and performance with varying workflow length

We have created workflows with varying lengths, consisting of sequences and AND/OR branches. Each task contains 20 concrete services. The QoS attributes of the services were chosen randomly, except for the price which was chosen partially randomly and partially depending on the other attributes.

In the first two experiments, each test case is executed 1,000 times with varying QoS constraints on price and response time. We set the utility of the GA as the baseline to 100% in order to evaluate the benefit of the additional repair operation. In Fig. 8a and 8b we vary the workflow length with a fixed average service compatibility of approx. 40%.

In relation to the baseline, the utility of the HC algorithm decreases with growing workflow length, whereas the utility of the CA grows (cf. Fig. 8a). Regarding the performance, with increasing repair probability more runtime is required as shown in Fig. 8b. Based on these results, we choose the CA33 algorithm as a reasonable tradeoff between utility and performance.

In the last experiments we evaluate the evolution of the genomes. We use a workflow of length 60 with service compliance 20%. In Fig. 9a we compare the mean ratio of valid links of all genomes. The genetic algorithm repairs all genomes in approx. 50 iterations. In contrast, the CA repairs the genomes in 10 (CA33) and 5 (CA100) iterations respectively. In Fig. 9b the fitness value of the best genomes are compared. For instance, the fitness of the GA in iteration 60 is achieved by the CA in iteration 10.



(a) Ratio of valid links of all genomes (b) Convergence of the fitness

Fig. 9. Comparing the evolution of the genomes

5 Related Work

We review some representative selection approaches that deal with the selection problem for functionally equivalent services and studies that analyze the functional compliance of services.

Zeng et al. formulate in [13] the composition problem as a Multi-Knapsack Decision Problem, proposing Integer Programing (IP) to tackle the problem. Their approach computes an optimal solution, but with growing problem size this approach cannot compute a solution in feasible time. Moreover, invalid workflow configurations are difficult to integrate in this approach.

In order to overcome the performance issues by computing an optimal solution with IP, near-optimal heuristic approaches such as hill-climbing [4] and genetic algorithms [1] have been investigated. A penalty is subtracted from the utility if constraints are violated in order to guide the algorithm to a feasible solution.

Nebil et al. [6] apply a clustering method on the service tasks before the actual selection process. In contrast to our approach they apply the clustering on the QoS-attributes of the services. For that purpose, they employ a k-nearest neighbor algorithm to identify the clusters. The algorithm computes a solution in reasonable time but the achieved utility is rather low $(60\% - 80\%)$.

Similar to our approach, Lécué and Mehandjiev apply in [5] a genetic algorithm to optimize the QoS and the matching quality of the service links. For that purpose, they combine these two values in the fitness function of the genomes. As a consequence, configurations with better QoS attributes but less functional compliance might be ranked higher. Contrary to their approach we treat the functional compliance like a QoS constraint, adding a penalty factor to the fitness of invalid genomes.

In [2] the authors investigate the gap between the conceptual and data-structural level of service composition. They claim that even if the conceptual types of the parameters of two services are compatible, the WSDL implementation may be incompatible. The focus of their work is on the automatic composition, taking no QoS-attributes into account.

The authors of the EASY project [7] arrange services in a functionality graph. The edges of this graph are based on the different Plug-in relationship as we apply it. QoS attributes are not considered in this work.

6 Conclusion and Future Work

In this paper we have discussed the problem of applying QoS-aware service selection on functionally diverse services and shown the impact on heuristic algorithms. We have presented which adjustments are necessary to apply heuristic algorithms and evaluated the algorithms on the modified problem. In the next step we have integrated a repair operation for the erroneous genomes based on a functional clustering into a genetic algorithm. This modification leverages background knowledge on the service compliance. We have shown that this modification outperforms naïve heuristic algorithms that randomly select services without taking the service compliance into account. Moreover, we have determined the optimal parameters for our algorithm by experiment.

As a next step we intend to improve the integration of the functional compliance criteria that is currently based on adding a penalty to the target function, similar to QoS-constraints. We will consider multi-objective optimization to balance the conflicting functional and non-functional requirements.

Furthermore, we plan to investigate on how to integrate the proposed solution into our automatic service composition approach [12]. The approach computes workflow templates that contain tasks with functionally diverse services.

References

1. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO (2005)
2. Ishikawa, F., Katafuchi, S., Wagner, F., Fukazawa, Y., Honiden, S.: Bridging the Gap Between Semantic Web Service Composition and Common Implementation Architectures. In: IEEE International Conference on Services Computing, SCC (2011)
3. Jaeger, M.C., Rojec-Goldmann, G., Mühl, G.: QoS Aggregation for Web Service Composition using Workflow Patterns. In: EDOC (2004)
4. Klein, A., Ishikawa, F., Honiden, S.: Efficient Heuristic Approach with Improved Time Complexity for QoS-aware Service Composition. In: IEEE International Conference on Web Services (ICWS), Washington D.C., USA (2011) (to appear)
5. Lécué, F., Mehandjiev, N.: Seeking Quality of Web Service Composition in a Semantic Dimension. IEEE Trans. on Knowl. and Data Eng. 23, 942–959 (2011)
6. Ben Mabrouk, N., Beauche, S., Kuznetsova, E., Georgantas, N., Issarny, V.: QoS-Aware Service Composition in Dynamic Service Oriented Environments. In: Bacon, J.M., Cooper, B.F. (eds.) Middleware 2009. LNCS, vol. 5896, pp. 123–142. Springer, Heidelberg (2009)
7. Mokhtar, S.B., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y.: EASY: Efficient semantic service discovery in pervasive computing environments with QoS and context support. Journal of Systems and Software 81(5), 785–808 (2008)

8. O'Sullivan, J., Edmond, D., ter Hofstede, A.H.M.: What's in a Service? Distributed and Parallel Databases 12(2/3), 117–133 (2002)
9. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer 40(11) (2007)
10. Pisinger, D.: Algorithms for Knapsack Problems. Ph.D. thesis, DIKU, University of Copenhagen, Denmark, Technical Report 95-1 (1995)
11. Wagner, F.: Efficient, Failure-Resilient Semantic Web Service Planning. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 686–689. Springer, Heidelberg (2010)
12. Wagner, F., Ishikawa, F., Honiden, S.: Qos-aware automatic service composition by applying functional clustering. IEEE International Conference on Web Services, ICWS (2011)
13. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality Driven Web Services Composition. In: Proceedings of the 12th International Conference on World Wide Web (WWW), pp. 411–421. ACM (2003)
14. Zhang, W., Schütte, J., Ingstrup, M., Hansen, K.M.: A Genetic Algorithms-Based Approach for Optimized Self-protection in a Pervasive Service Middleware. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 404–419. Springer, Heidelberg (2009)