Optimal Service Selection Heuristics in Service Oriented Architectures

Emiliano Casalicchio[1], Daniel A. Menascé[2], Vinod Dubey[3], and Luca Silvestri[1]

[1] Dipartimento di Informatica Sistemi e Prod.
Università di Roma "Tor Vergata", Roma, Italy
{casalicchio,silvestri}@ing.uniroma2.it

[2] Department of Computer Science,
George Mason University, Fairfax, VA, USA
menasce@gmu.edu

[3] The Volgenau School of Information Technology and Engineering,
George Mason University, Fairfax, VA, USA
vdubey@gmu.edu

Abstract. Service Oriented Architectures allow service brokers to execute business processes composed of network-accessible loosely-coupled services offered by a multitude of service providers, at different Quality of Service (QoS) and cost levels. To optimize their revenue and the offered QoS level, service brokers need to solve the problem of finding the set of service providers that minimizes the total execution time of the business process subject to cost and execution time constraints. This optimization problem is clearly NP-hard. Optimized algorithms that find the optimal solution without having to explore the entire solution space have been proposed to solve problems of moderate size. A heuristic search of the sub-optimal solution scales to problems of large size and is appropriate for runtime service selection. This paper evaluates the performance of three heuristic service selection algorithms that are candidates for implementation in scalable service brokers. Our goal is to identify which algorithm provides the solution closest to the optimal and how many selections are evaluated to find the solution. The comparison is made over a wide range of parameters including the complexity of the business process topology and the the tightness of the QoS and cost constraints.

Keywords: Service Oriented Architecture, Web services, service composition, QoS, heuristics.

1 Introduction

The Service Oriented Architecture (SOA) model enables a market of services, where service providers (SPs) provide services at different QoS levels and at different cost. In this emerging market it makes sense to investigate mechanisms to properly select a set of services, characterized by different QoS and cost levels, that when composed together satisfy the QoS needs and cost constraints of the resulting business process (BP). This problem is referred in the literature as the

QoS-aware Service Selection or Optimal Service Selection problem [1,2,3,4,8,9, 10,12,13,14].

The execution of a business process is coordinated by a service broker (or broker for short). The broker needs runtime and scalable mechanisms to solve the optimal service selection problem and to exploit the dynamics of the service marketplace characterized by potential and rapidly changing conditions in workload intensity, QoS level, and cost.

In [8], the authors provided a performance model that takes into account the business process structure, including cycles, parallel activities, and conditional branches, and computes the end-to-end execution time and cost for the business process. For some performance metrics (e.g., cost, availability, reputation) the composition is a trivial linear combination of the performance measure of the composing services. On the contrary, for other metrics such as execution time, we have a non linear function of the performance level of the services being composed. In that paper, we also provided an efficient algorithm, called JOSeS algorithm, that finds the optimal solution without resorting to an exhaustive search of the of solution space. This efficient optimal algorithm can only handle problems of small to moderate size. That paper also presented and throughly evaluated a heuristic solution that is compared here with other two proposed heuristics.

Several approaches can be used to solve the service selection problem. Current proposals use exact algorithms or heuristics (e.g., [2] or genetic algorithms [3]) to solve the QoS-aware (optimal) service selection problem for each request, whose exact solution has an exponential complexity. In [12], the authors define the problem as a multi-dimension multi-choice 0-1 knapsack one as well as a multi-constraint optimal path problem. A global planning approach to select an optimal execution plan by means of integer programming is used in [13]. In [1], the authors model the service composition as a mixed integer linear problem where both local and global constraints are taken into account. A linear programming formulation and flow-based approach is proposed in [4]. There, the authors consider not only sequential composition of services but also cycles and parallel activities. Algorithms for the selection of the most suitable candidate services to optimize the overall QoS of a composition are also discussed in [7]. A different approach, based on utility functions as the QoS measure, is used in [9], where the authors propose a service selection mechanism based on a predictive analytical queuing network performance model. Other contributions to the issue of service selection and composition can be found in [6,11].

This paper presents a performance comparison of runtime heuristic algorithms to evaluate their accuracy in finding the sub-optimal solution and their scalability to large size problems. The algorithms we consider conduct a heuristic search of the solution space in order to find a sub-optimal solution that is very close to the optimal solution but is obtained by examining a drastically reduced number of selections. In fact, the experimental studies reported in this paper show that the heuristic solutions come very close to the optimal solution (less than 9.6%

worse) after having examined a very small number of possible solutions (less then 9.45 on average versus 125,794 for the efficient optimal search).

The paper is organized as follows. Section 2 introduces the problem formulation. The optimal solution approach is described in Section 3. The heuristic solutions are described in Section 4. Experimental results are discussed in Section 5. Section 6 concludes the paper.

2 Problem Formulation

We use the average execution time of the business process (BP) as its main QoS metric. As previously discussed, this metric is a nonlinear function of the execution times of individual business activities and depends on the BP structure and composition constructs used. The extension to other performance metric is straightforward.

We assume that the probability density function (pdf) and cumulative distribution function (CDF) of the execution times of each SP are known. We also assume that the execution cost of each business activity provided by the SPs is given.

Let,

– A business process B be composed of N business activities $a_i, i = 1, \cdots, N$.
– $R_{\max}$ be the maximum average execution time for B.
– $C_{\max}$ be the maximum cost for the execution of B.
– $R_{i,j}$ be the execution time for business activity a_i when implemented by service provider $s_{i_j} \in S_i$. $R_{i,j}$ is a random variable with a probability density function $p_{i,j}$ and a cumulative distribution function $P_{i,j}$.
– $C_{i,j}$ be the execution cost of business activity a_i when it is implemented by service provider $s_{i_j} \in S_i$.
– $\mathcal{Z}$ be the set of all possible service provider selections of the business activities of B.
– $z \in \mathcal{Z}$ be a service selection of N service providers that support the execution of business process B.
– $z(k)$: service provider allocated to activity a_k in service selection z.
– $R(z)$ and $C(z)$ be the average execution time and the cost for associated with service selection z, respectively.

The Optimal Service Selection problem is formulated as a nonlinear programming optimization problem where the objective is to find a service selection z that minimizes the average execution time subject to cost constraints:

$$\begin{aligned} &\min \quad R(z) \\ \text{subject to} \quad & \\ &R(z) \leq R_{max} \\ &C(z) \leq C_{max} \\ &z \in \mathcal{Z} \end{aligned}$$

$R(z)$ is, in general, a complex nonlinear function that can be obtained from well known results from order statistics.

The Optimal Service Composition problem formulated above can be solved using two different approaches. The first is an optimal solution approach (Optimal Service Selection) that avoids doing an exhaustive search of the solution space $\mathcal{Z}$ (e.g., the JOSeS algorithm proposed by the authors in [8]).

The second approach (Heuristic Service Selection) adopts a heuristic solution that reduces the problem complexity. In the following we compare the performances of three heuristics that scale to large size problems.

The first required step for both the optimal reduced search and the heuristic is to be able to extract from the BPEL code that describes the business process, an expression for the global average execution time and another for the total execution cost. This expression needs to take into account the structure of the business process as well as the execution times and cost of the individual business activities.

3 Optimal Service Selection

BPEL offers different constructs to combine business activities into a business process. The business logic is a structured activity obtained by putting together elementary business activities (in the following, the term business process and business logic are used alternatively). Each business activity is essentially a synchronous or asynchronous invocation of a Web service operation. The main construct a structured BPEL activity includes are: sequential control (`<sequence>`, `<switch>`, and `<while>`), non-deterministic choice (`<pick>`), and concurrency and synchronization of elementary activities (`<flow>`).

In [8], the authors showed how one can obtain an expression for the execution time R of a business process and its execution cost C directly from its structure described in BPEL or an equivalent tree-like representation. While the execution cost of a business process is the sum of the execution costs of the activities of the business process (plus eventually some additional overhead), the execution time depends on how the business activities are structured. For example, if we have a sequence of business activities $a_1, \ldots, a_n$, and a service selection z, the execution time of the business process is $R(z) = \sum_{i=1,\ldots,n} R_{i,j}$ where s_{i_j} is the service provider assigned to activity a_i in z. The execution time of an activity a_i that is repeated n times and that is supported by service provider $s_{i,j}$ is simply $R(z) = n \times R_{i,j}$. In the case of deterministic or non deterministic choices, the computation of the total execution is easily computed as $R(z) = \sum_{i=1,\ldots,n} q_i \times R_{i,j}$ where q_i is the probability that activity a_i is invoked. Finally, the execution time of the parallel execution of n business activities is given by $R(z) = \max_{i=1,\ldots,n}\{R_{i_j}\}$.

The computation of the average execution time for a business process that has `<flow>)` constructs is quite involved, especially in the case where execution times are random variables, and is described in [8].

Optimal service selection can be done in a naive way by enumerating all possible service selections and computing their execution time and cost. A more efficient approach avoids generating selections such that their subselections already violate the execution time and cost constraint. Such an algorithm, called JOSeS algorithm, was presented in [8], and is used here for comparing the heuristic algorithms presented in the next section.

4 Heuristic Service Selection

We present two new heuristics—Fastest First (**FF**) and Cheapest First (**CF**)—and compare them with the high reduction in execution cost, low increase in execution time ratio (**hrCliR**) heuristic presented by the authors in [8]. The goal of the proposed heuristic solutions is to reduce the cost of finding the optimal solution, providing a sub-optimal selection as close as possible to the optimal.

Figure 1 shows the solution space and the feasible solution space of our problem. The solution space is the area delimited by the dashed lines, which indicate the lower and upper bounds for cost and execution time of the business process. The lower bound C_C for the execution cost is the cost obtained by selecting the cheapest service providers for each activity. Similarly, the upper bound R_S for the execution time can be obtained by selecting the slowest service provider for each business activity. On the contrary, the upper bound C_E for the execution cost is obtained by selecting the most expensive service provider for each activity, and the lower bound R_F for the execution time is obtained by selecting the fastest service provider for each activity. The feasible solution space is represented by the dotted area and is the the portion of the solution space delimited by the lower bound for execution time and execution cost and by the time and cost constraints (bold lines). We assume that the execution cost of a service provider is inversely proportional to its execution time.

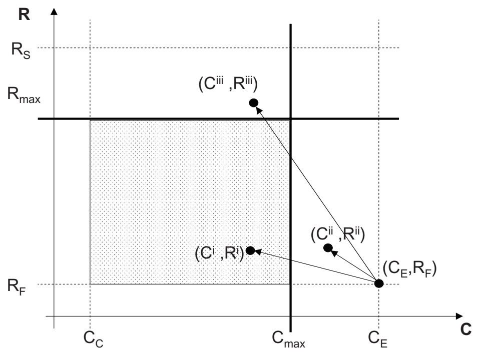

Fig. 1. A conceptual representation of the solution space and of the feasible solution space

4.1 Fastest First

The proposed heuristic is based on the following idea (see Fig. 1). We start from the service selection z_0 characterized by the lowest execution time, i.e., the point (C_E, R_F). Assume that this point is outside the feasible solution space and that the cost constraint is violated. To find a feasible solution as close as possible to the optimum, we have to choose a selection z' that moves the problem solution inside the dotted area, say the point (C', R'). To choose the solution z', we determine the activity a_k such that the service provider allocated to a_k in z_0, i.e., $z_0(k)$, provides the lowest average execution time among all allocated service providers in z_0. We then replace $z_0(k)$ with the second fastest service provider for a_k.

We then evaluate the execution cost and execution time for the new service selection z'. If the constraints are satisfied we have a suboptimal solution (see point (C', R')). Otherwise, there are two possibilities: if the cost constraint is still violated and the time constraint is not yet violated, we are in a point such as (C'', R'') and we have to repeat the above mentioned process, i.e., the replacement of the fastest service provider for the new allocation z'.

If the execution time constraint is violated but the cost constraint is satisfied, we are at point (C''', R''') and we cannot accept such solution as we would continue to violate the execution time constraint at any further attempt of cost reduction. Then, we go back to the previous allocation (z_0 in this case) and we replace the service provider that has the second lowest execution time by its next fastest.

The details of this heuristic are shown in Algorithm 1. The function `GetFastest` (z, h) returns the service provider in allocation z that has the h-th smallest average execution time when $h \leq N$ and returns `NULL` when $h > N$. The function `next (k)` returns the next, not yet evaluated, service provider in the list of service providers for activity a_k. This list is assumed to be sorted in increasing order of average execution time. This function returns NULL if all the providers for activity a_k have been already evaluated. We use the following notation in the algorithm. Let $z \ominus_k s$ stand for the operation of removing from solution z provider s for activity k. Similarly, let $z \oplus_k s$ denote the addition to solution z of provider s to activity k.

4.2 Cheapest First

The CF heuristic is based on the same criteria used in the Fastest First. The main difference is that the search for a sub-optimal solution starts from the point (C_C, R_S) in Fig. 1, which is the cheapest and slowest selection of service providers. Assume that this point is outside the feasible solution space and that the execution time constraint is violated. To find a feasible solution we have to choose a selection z' that moves the problem solution inside the dotted area. To choose the solution z', we determine the activity a_k such that the service provider allocated to a_k in z_0, i.e., $z_0(k)$, provides the lowest cost among all allocated service providers in z_0. We then replace $z_0(k)$ with the service provider with the second lowest for a_k.

Algorithm 1. Fastest First Algorithm Solution

```
1: function FFHeuristic()
2: Find $z$ such that $E[R(z)] = R_F$;
3: if $(E[R(z)] \leq R_{\max})$ and $(C(z) \leq C_{\max})$ then
4:    return $z$
5: end if
6: while $C(z) > C_{\max}$ do
7:    $h \leftarrow 1$;
8:    while $(s_{k_i} \leftarrow \text{GetFastestSP}(z, h)) \neq \text{NULL}$ do
9:       if $(s_{k_j} \leftarrow \text{next}(k)) \neq \text{NULL}$ then
10:         $z \leftarrow z \ominus_k s_{k_i}$;
11:         $z \leftarrow z \oplus_k s_{k_j}$;
12:         if $C(z) \leq C_{\max}$ then
13:            if $E[R(z)] \leq R_{\max}$ then
14:               return $z$;
15:            else
16:               $z \leftarrow z \ominus_k s_{k_j}$;
17:               $z \leftarrow z \oplus_k s_{k_i}$;
18:               $h \leftarrow h + 1$;
19:            end if
20:         end if
21:      else
22:         $h \leftarrow h + 1$;
23:      end if
24:   end while
25: end while
26: return *infeasible solution*
27: end function
```

We then evaluate the execution cost and execution time for the new service selection z'. If the constraints are satisfied we have a suboptimal solution. Otherwise, there are two possibilities: if the execution time constraint is still violated and the cost constraint is not violated, we have to repeat the above mentioned process, i.e., the replacement of the cheapest service provider for the new allocation z'.

If the cost constraint is violated but the execution time is satisfied we cannot accept such solution as we would continue to violate the cost constraint at any further attempt of cost increase. Then, we go back to the previous allocation and replace the service provider that has the second lowest cost to be replaced by its next cheapest.

4.3 hrCliR Algorithm

This heuristic, proposed in [8], starts evaluating the service selection z_0 characterized by the lowest execution time, i.e., the point (C_E, R_F). Assume that this point is outside the feasible solution space and that the cost constraint is violated. To find a feasible solution as close as possible to the optimum, we have to choose a selection z' that moves the problem solution inside the dotted area, say the point (C', R'). To choose the solution z' we replace the service provider that provides the highest reduction in the execution cost C with the lowest increase in the execution time R. To determine such provider, we need to compute the ratio

$$\Delta_{i,j,j'} = p_i \times \frac{C_{i,j} - C_{i,j'}}{R_{i,j'} - R_{i,j}} \quad j' > j \tag{1}$$

for each activity a_i $(i = 1, \cdots, N)$. In Eq. (1), j represents the service provider allocated to activity a_i, j' represents an alternate service provider for a_i, and p_i is the probability that activity a_i is executed in the business process. This probability is a function of the structure of the business process and its branching probabilities. We then select the activity for which there is an alternate provider that maximizes the value of the ratio for all such ratios. More precisely,

$$(k, m) = \text{argmax}_{i=1,\cdots,N; j' \neq j} \{\Delta_{i,j,j'}\} . \tag{2}$$

According to Eq. (2), the service provider m when replacing service provider j in activity k yields the maximum value for the ratios Δ.

We then evaluate the execution cost and execution time for the new service selection z'. If the constraints are satisfied we have a suboptimal solution. Otherwise, there are two possibilities: if the cost constraint is still violated and the time constraint is not yet violated, we are in point such as (C'', R'') and we have to repeat the above mentioned process, i.e., the selection of a new service provider that maximizes the ratio Δ among all activities. If the execution time constraint is violated but the cost constraint is satisfied, we are at point (C''', R''') and we cannot accept such solution as we would continue to violate the execution time constraint at any further attempt of cost reduction. Then, we select the service provider that has the second best ratio Δ. The process is repeated until a feasible solution is found.

5 Experiments

We implemented the heuristics and the optimal JOSeS algorithm [8] to conduct experiments aimed at evaluating the efficiency of the former. In particular, we wanted to: 1) determine how close the heuristics solution are to the optimal, 2) compare the number of points in the solution space examined by each algorithm, 3) compare the three heuristics, CF, FF and hrCliR, over a wide range of parameters including the complexity of the business process topology, the tightness of the response time and cost constraints, and the number of SPs per activity.

5.1 Description of the Experiments

The experimental methodology and metrics computed mirrors pretty closely what the authors did in [8]. Fifty business processes were randomly generated and the expression for the average response time and execution costs were computed according to section 3. The process for the generation of business processes determined randomly when to generate sequences, flows, switches (and their switching probabilities) as well as the number of branches of flows and switches.

The number of activities for the randomly generated business processes varied from 6 to 10. The number of flows and switches in these business processes varied in the range zero to three and zero to two, respectively.

The experiments assumed that the execution time of each service provider s is exponentially distributed. The cost of obtaining an average execution time $E[R_s]$ from service provider s was assumed to be equal to $1/E[R_s]$. In other words, the cost decreases with the inverse of the average service time offered by a service provider.

For each experiment, the number of SPs per activity *nspa* was the same for all activities and that number was varied as follows: 2, 3, 4, 5, 6, and 7.

The complexity $\mathcal{C}(B)$ of a business process B is defined as

$$\mathcal{C}(B) = \#\text{activities} + \#\text{flows} + \sum_{\forall \text{ switch } i} \text{fanout}_i \tag{3}$$

using an adapted version of the control flow complexity and other metrics discussed in [5]. After all business processes are generated, we compute for each a normalized complexity $\mathcal{C}^{'}(B)$ as follows

$$\mathcal{C}^{'}(B) = \frac{\mathcal{C}(B) - \min_{\forall s} \mathcal{C}(s)}{\max_{\forall s} \mathcal{C}(s) - \min_{\forall s} \mathcal{C}(s)}. \tag{4}$$

It can be easily seen that $0 \leq \mathcal{C}^{'}(B) \leq 1$ for any business process B.

We then apply the k-means, with $k = 3$, clustering algorithm on all business processes using $\mid \mathcal{C}^{'}(B) - \mathcal{C}^{'}(q) \mid$ as the distance between business processes B and q. The business processes in the cluster with the smallest centroid are called *simple* business processes, the ones in the cluster with the largest centroid are called *complex*, and the remaining ones *medium* business processes. The performance of the heuristics are also compared along this dimension.

For a given business process p and for a given number of SPs per activity, we compute the coordinates of the feasibility region (C_C, R_F), (C_E, R_F), (C_C, R_S), and (C_E, R_S). We then compute three sets of values for the constraints $R_{\max}$ and $C_{\max}$ according to how tight they are. We call them *strict*, *medium*, and *relaxed* constraints, and their values are:

$$\begin{aligned}
C_{\max} &= C_C + (C_E - C_C)/3 && \text{for tight,}\\
R_{\max} &= R_F + (R_S - R_F)/3 && \text{for tight,}\\
C_{\max} &= C_C + (C_E - C_C)/2 && \text{for medium,}\\
R_{\max} &= R_F + (R_S - R_F)/2 && \text{for medium,}\\
C_{\max} &= C_E - (C_E - C_C)/6 && \text{for relaxed, and}\\
R_{\max} &= R_S - (R_S - R_F)/6 && \text{for relaxed.}
\end{aligned}$$

We also carried out a set of experiments to evaluate the scalability of the algorithms. The number of activities was set to 10 and *nspa* ranged from 5 to 40.

For the complexity of the experiments the heuristics were evaluated only in the *medium* constraints scenario.

5.2 Results of the Experiments

The following metrics are used to evaluate the heuristic algorithms discussed here.

- ε_R: absolute relative percentage average execution time difference defined as

$$\varepsilon_R = 100 \times \frac{\mid R_h - R_o \mid}{R_o} \tag{5}$$

  where R_h and R_o are the average execution times obtained using the heuristic and JOSeS algorithm, respectively.
- ε_C: absolute relative percentage average execution cost difference defined as

$$\varepsilon_C = 100 \times \frac{\mid C_h - C_o \mid}{C_o} \tag{6}$$

  where C_h and C_o are the average execution costs obtained using the heuristic and JOSeS algorithm, respectively.
- δ: the percentage of not feasible solutions found, defined as

$$\delta = 1 - \frac{N_h}{N_o} \tag{7}$$

  where N_o is the number of optimal solutions found by the JOSeS algorithm (equal to the number of experiments) and N_h is the number of sub-optimal solutions found by the heuristic algorithm. Note that while N_o is equal to the number of exeriments, N_h could be less than N_o because heuristics are not able to find a solution for all the constraints combination.

In [8], after conducting an exhaustive ANOVA analysis, we showed that both ε_R and ε_C depend on *nspa* and on the business process complexity ($\mathcal{C}^{'}(p)$). Therefore, in the experiments, we compare the performance of the proposed heuristics for different value of the number of SPs per activity and for the three different classes of the normalized business complexity.

Our results can be summarized as follows:

1. The Fastest First and hrCilR heuristic algorithms have a comparable behaviour with an absolute relative percentage average execution time difference ε_R less then 7.3%$\pm$3.3% at a 95% confidence interval on average and less then 9.6% $\pm$ 4.2% at worst. The value of ε_C is always less then 4.8% $\pm$ 1.7% at a 95% confidence interval.
2. The Fastest First and hrCilR heuristic algorithms need a similar number of iterations to find the sub-optimal solution (9.15 $\pm$ 1.35 for hrCilR and 9.45$\pm$0.98 for FF in the worst case, that is for *tight* constaints). This number is five orders of magnitude less then the optimal JOSeS's algorithm (125,794 iterations on average).

3. The Fastest First and hrCilR heuristic algorithms find a sub-optimal (or optimal in some cases) solution in 95.6 percent of the cases and in 97.8%, respectively. On the contrary, the cheapest first has a high percentage of not found solutions (17.7%). We should mention that the heuristics are not able to find the optimal solution only when the constraints are *tight*.
4. The Cheapest First heuristic algorithm shows very high values of ε_R and ε_C, regardless of the type of constraints, the number of SPs per activity and the BP complexity. For example, in the case of $nspa = 5$ and relaxed constraints $\varepsilon_R = 123\% \pm 15.6\%$ at a 95% confidence interval.
5. FF and hrCliR scale for a wide range of $nspa$ (from 5 to 40) and the number of iterations to find a sub-optimal solution was always less than 6 ± 0.66 at a 95% confidence level.

Figures 2 and 3 show the performance of the heuristics for the three types of constraints and for $nspa = 3$ and $nspa = 5$, respectively. The results show that while Fastest First and hrCilR heuristic have the same behaviour in terms of relative percentage average execution time difference, the Cheapest First heuristic shows a very high value of ε_R, regardless of the type of constraints. This behavior can be explained as follows. The CF heuristic, starting from the cheapest solution, tries to find a sub-optimal solution that has a lower cost. Therefore, the goal of the Cheapest First is opposite to the optimization problem defined by Equation 1. Usually, the solution determined by the CF has a cost lower then the optimum, and the high value for ε_C is due to values of $\mathcal{C}_h$ less then $\mathcal{C}_o$. On the contrary, the sub-optimal execution time is significantly higher than the optimum. From the experiments, it emerges also that the performance of CF drastically improves for tight constraints. In this scenario, the number of feasible allocations is reduced and the distance between the solutions is very small; therefore the probability of finding a solution near the optimum is higher.

Figures 4 and 5 show the values of ε_R for different values of business process complexity. Also along this dimension, the trend is confirmed, i.e., FF and hrCilR achieve very similar performance and outperform the Cheapest First algorithm.

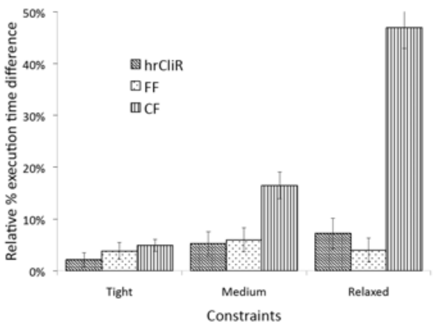

Fig. 2. ε_R as funtion of the type of constraints. In this scenario $nspa = 3$.

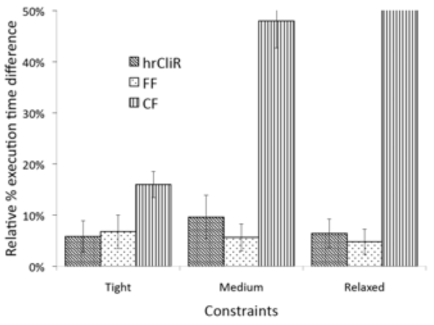

Fig. 3. ε_R as function of the type of constraints. In this scenario $nspa = 5$.

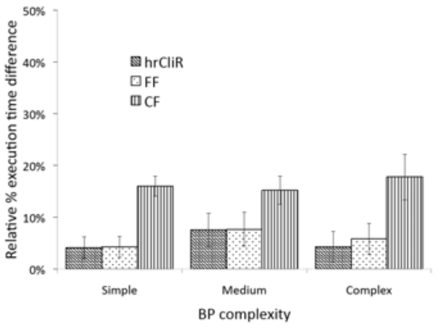

Fig. 4. ε_R as function of the business process complexity. In this scenario $nspa = 3$.

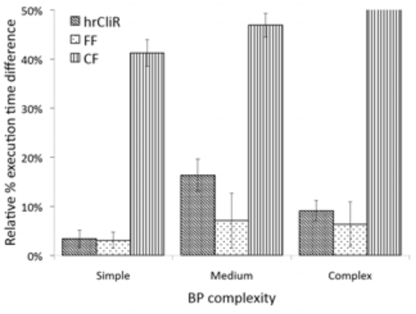

Fig. 5. ε_R as function of the business process complexity. In this scenario $nspa = 5$.

In the last set of experiments we evaluated the scalability of FF and hrCliR when the selection is done over a large set of SPs (from *nspa* from 5 to 40) and for complex business processes. Figure 6 shows that the number of iterations to find a sub-optimal solution is always less then 6 ± 0.66 at a 95% at a confidence level. The hrCliR performs better then FF (at worst it takes 5.28 ±0.54 iterations to find a solution). The weakness of this set of experiments is that it is impossible (with the systems we have access to) to compute the optimal solution and then the values of ε_R and ε_C. Therefore we are not certain of the goodness of the solutions found in the case of a large set of SPs.

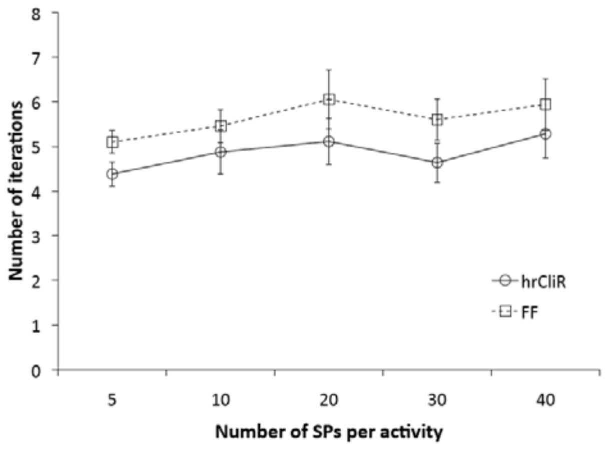

Fig. 6. Number of Iterations for medium constraints and complex BP

6 Concluding Remarks

The SOA model enables re-use and sharing of components through dynamic discovery. The benefit of service composition stimulates also the growth of a market of heterogeneous and volatile services. Service brokers are aware that: each possible service selection of services brings different levels of QoS and cost; and that the service marketplace environment is highly heterogeneous and volatile. Therefore, brokers need scalable mechanisms that can be used for runtime service selection among a set of service providers.

This paper presented two such efficient mechanisms that, in all experiments reported and all experiments carried out and not-reported due to lack of space, come very close to the optimal solution (less than 7.3% worse) after having examined a very small number of possible solutions (less than 9.45 worse).

Acknowledgment

The work of Daniel Menascé is partially supported by grant CCF-0820060 from the National Science Foundation.

The work of Emiliano Casalicchio and Luca Silvestri is partially supported by the PRIN project D-ASAP founded by the Italian Ministry of Education, University and Research.

References

1. Ardagna, D., Pernici, B.: Global and Local QoS Guarantee in Web Service Selection. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 32–46. Springer, Heidelberg (2006)
2. Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R.: Heuristics for QoS-aware Web Service Composition. In: Proc. Int'l Conf. on Web Services (September 2006)
3. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An Approach for QoS-aware Service Composition Based on Genetic Algorithms. In: Proc. Genetic and Computation Conf. (June 2005)
4. Cardellini, V., Casalicchio, E., Grassi, V., Francesco, L.P.: Flow-based service selection for web service composition supporting multiple qos classes. In: ICWS 2007. IEEE Intl. Conf. Web Services, July 9-13, pp. 743–750 (2007)
5. Cardoso, J., Mendling, J., Neumann, G., Reijers, H.A.: A Discourse on Complexity of Process Models. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 117–128. Springer, Heidelberg (2006)
6. Fung, C.K., Hung, P.C.K., Wang, G., Linger, R.C., Walton, G.H.: A Study of Service Composition with QoS Management. In: Proc. of the IEEE ICWS (2005)
7. Jaeger, M., Muhl, G., Golze, S.: Qos-aware composition ofweb services: A look at selection algorithm. In: Proc. 2005 IEEE Intl. Conf. Web Services, ICWS 2005 (2005)
8. Menascé, D.A., Casalicchio, E., Dubey, V.: On optimal service selection in Service Oriented Architectures. In: Performance Evaluation (in press)

9. Menascé, D.A., Dubey, V.: Utility-based QoS brokering in service oriented architectures. In: Proc. of the IEEE ICWS, Application Services and Industry Track, Salt Lake City, Utah, July 9-13, pp. 422–430 (2007)
10. Menascé, D.A., Ruan, H., Gomma, H.: QoS management in service oriented architectures. Performance Evaluation Journal 64(7-8), 646–663 (2007)
11. Serhani, M.A., Dssouli, R., Hafid, A., Sahraoui, H.: A QoS Broker based Architecture for Efficient Web Service Selection. In: Proc. 2005 IEEE ICWS (2005)
12. Yu, T., Lin, K.J.: Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In: Proc. of 3rd Int'l Conf. on Service Oriented Computing, December 2005, pp. 130–143 (2005)
13. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Trans. Softw. Eng. 30(5), 311–327 (2004)
14. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F., Mirandola, R.: QoS driven runtime adaptation of service-oriented architectures. In: Proc. of the 7th ACM SIGSOFT ESEC/FSE 2009, Amsterdam, The Netherlands (August 2009)