On Optimal Service Selection

P. A. Bonatti[*]
Università di Napoli FEDERICO II
Napoli, Italy
bonatti@na.infn.it

P. Festa
Università di Napoli FEDERICO II
Napoli, Italy
paola.festa@unina.it

ABSTRACT

While many works have been devoted to service matchmaking and modeling nonfunctional properties, the problem of matching service requests to offers in an optimal way has not yet been extensively studied. In this paper we formalize three kinds of optimal service selection problems, based on different criteria. Then we study their complexity and implement solutions. We prove that one-time costs make the optimal selection problem computationally hard; in the absence of these costs the problem can be solved in polynomial time. We designed and implemented both exact and heuristic (suboptimal) algorithms for the hard case, and carried out a preliminary experimental evaluation with interesting results.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based Services; F.2.2 [**Nonnumerical Algorithms and Problems**]: Computations on Discrete Structures

General Terms

Algorithms, Experimentation, Theory

Keywords

Service selection problem, Automatic service composition, Service matchmaking, Nonfunctional properties

1. INTRODUCTION

There exists an increasing body of work on automated service selection, based on criteria such as quality of service (QoS), trust, cost, etc. Some works focus on service matchmaking [26, 20, 2, 19, 28, 11, 3, 4, 21, 12, 13], that is, a process that given a service request returns the set of available services that can be used to fulfill that request (offers may be ranked according to their similarity to the request). Some other papers focus on modelling nonfunctional properties such as the above criteria, that induce preference orderings on the available services [16, 5, 17].

However, no paper tackles in depth the optimization problem that follows matchmaking and nonfunctional property evaluation: What is the best way of binding each service request to a matching service? The problem may be nontrivial if the optimization involves multiple service requests at once. Consider for example composite services; they can be modelled as workflows [6, 12], where each activity potentially corresponds to a different service. In this framework, the decision problem consists in finding an optimal matching (w.r.t. the adopted criteria) between the set of activities occurring in the workflow and the set of available services that can be used to carry out those activities.

In this paper we consider optimal service selection based on a given set of service requests (such as the activities occurring in a workflow), a set of service offers (the available services), the result of the matchmaking process (that associates each request to the set of offers that can satisfy it), and a numeric preference measure. Numeric measures are well-suited to a number of preference criteria of practical interest, based on costs of various sorts, as well as bandwidth, trust [1, 30, 24, 29], and other QoS criteria. Moreover, different criteria can often be merged into a single numerical value [5].

Preferences and costs may be associated to services, service invocations, or both, as illustrated by the following examples:

- Trust is often associated to services, not service invocations. User preferences driven by privacy protection and security usually refer to services, independently of the specific call.
- However, an information service may be trusted on some queries and not on others; in this case trust is associated to individual invocations.
- Some services have an activation cost or a registration cost, to be paid only the first time the service is invoked, or before the first use. Such costs are associated to the services and do not depend on the number of calls nor on their nature.
- Other services work on a pay-per-use basis (such as paper downloads from a digital library and other electronic purchases). In this case, costs and preferences may depend on the specific request and are associated to each service invocation. Some services have both a per-use cost and an activation cost (such as telephone providers).
- Also bandwidth and transmission speed may vary across different service calls. For example a service may be

[*]Partially supported by the EU FP6 Network of Excellence REWERSE, IST-2004-506779.

faster at certain times of the day. Another example is given by connection costs that depend on the duration of each particular call.

In this paper we shall contribute to the understanding of the service selection problem (SSP, for short) by formalizing and studying three classes of SSP problems where selection is based on costs and on two different QoS-like criteria, respectively. For simplicity, in this paper we assume that costs and preferences are totally ordered and static (i.e., time independent); partially ordered and dynamic nonfunctional properties will be dealt with in a forthcoming paper.

We shall prove that in general—and despite the aforementioned simplifying assumptions—the optimal service selection problem is harder than **NP** (unless the polynomial hierarchy collapses). More precisely, some SSPs are in $\mathbf{FP^{NP}}$, like many famous hard optimization problems, while checking whether the optimal cost equals a given constant K is **DP**-complete. We shall identify practical cases where the problem can be solved in polynomial time. In particular, we show that the high computational complexity of the service selection problem is caused by the one-time costs associated to service offers (e.g., initialization and registration costs). In the absence of one-time costs, the optimal selection problem can be solved in polynomial time by applying a greedy approach. Finally, we shall illustrate the results of an experimental evaluation of both exact and heuristic algorithms over different classes of problem instances.

The paper is organized as follows. In Section 2 we recall the definition of the complexity classes needed in this paper. In Section 3 the service selection problems are formalized. Section 4 contains the complexity results and the algorithms for the first class of SSPs (based on cost-like criteria), and reports the experimental results for these algorithms. Section 5 illustrates the complexity results and the algorithms for the remaining two classes of SSPs (based on QoS-like criteria). Section 6 concludes the paper with a discussion of the results and a list of interesting directions for future work.

2. PRELIMINARIES ON COMPLEXITY

We assume the reader to be familiar with the basics of computational complexity. We refer to [22] for more details.

The class **DP** is a class of decision problems containing **NP**. **DP** can be defined as the class of all languages $\mathcal{L}$ such that $\mathcal{L} = \mathcal{L}_1 \cap \mathcal{L}_2$, for some $\mathcal{L}_1$ in **NP** and some $\mathcal{L}_2$ in co-**NP**. If $\mathcal{L}_1$ and $\mathcal{L}_2$ are complete for **NP** and co-**NP**, respectively, then $\mathcal{L}$ is complete for **DP**.

The class $\mathbf{FP^{NP}}$ is the class of all function problems (i.e. problems that compute a value, not only a yes-no answer) that can be solved in polynomial time by a deterministic Turing machine with an oracle for **NP**.

Many standard optimization problems are complete for $\mathbf{FP^{NP}}$. For example, the Traveling Salesman Problem and Max-weight SAT are $\mathbf{FP^{NP}}$-complete [22, Chapter 17.1].

3. PROBLEM FORMALIZATION

The *instances* of the service selection problems (SSP) addressed in this paper are tuples $\langle R, O, M, c, k\rangle$ where:

- $R = \{1, 2, \dots, m\}$ is a nonempty set of service requests;
- $O = \{1, 2, \dots, n\}$ is a nonempty set of service offers;
- $M \subseteq R \times O$ is a matching between requests and offers such that
$$\forall r \in R,\ \exists s \in O,\ \langle r, s\rangle \in M \tag{1}$$
(intuitively, if some request cannot be satisfied then the decision phase is never reached);
- $c :\ O \to \mathbb{Q}$ is a function that assigns a cost or quality measure c_s to each offer $s \in O$;
- $k :\ M \to \mathbb{Q}$ is a function that assigns a cost or quality measure $k_{rs} \in \mathbb{Q}$ to each pair $\langle r, s\rangle \in M$ (that is, to each possible service call).

The goal is finding a binding between requests and offers, compatible with the given matching and optimal w.r.t. the preferences associated to services and invocations.

Formally, a *binding* for $\langle R, O, M, c, k\rangle$ is a total function $b :\ R \to O$ such that $b \subseteq M$.[1] Condition (1) on M ensures that a binding always exists.

As anticipated in the introduction, in this paper the optimality of bindings will be evaluated against different objective functions.

The first objective function, denoted by $\mathcal{C}_b$, is appropriate for criteria based on totally ordered costs (money, time, etc.)[2] The overall cost of a binding is obtained by summing up the costs of all the calls specified in the binding, plus the one-time costs associated to the called services (e.g., initialization and registration costs). More precisely, let
$$b[R] = \{s \in O \mid \exists r \in R.\ b(r) = s\}$$
denote the range of b (informally speaking, $b[R]$ is the set of services "used" by b); then the total cost of binding b is given by
$$\mathcal{C}_b = \sum_{r\in R} k_{r\,b(r)} + \sum_{s\in b[R]} c_s\,. \tag{2}$$

The second objective function, denoted by $\mathcal{Q}_b$, is appropriate for many QoS-like criteria. Suppose that the aim of the optimization problem, in this case, is maximizing simultaneously the quality of each requested service. Then the overall quality of a binding b can be modelled by summing up the qualities of each selected request-offer match:
$$\mathcal{Q}_b = \sum_{r\in R} f(k_{r\,b(r)}, c_{b(r)})\,. \tag{3}$$

Here $f : \mathbb{Q}^2 \to \mathbb{Q}$ computes the quality of the solution provided by the selected service $b(r)$ to request r by appropriately combining the measure $k_{r\,b(r)}$ associated to the service call and the measure $c_{b(r)}$ associated to the service. We assume only that f can be computed in polynomial time (w.r.t. the given instance), because different applications may require different functions f.

For example, suppose r is satisfied by invoking $b(r)$ via a network connection. Packet rate is influenced both by the server's speed and by the bandwidth allowed by the intermediate routers; the lowest rate determines the overall rate for the connection. Suppose the values k_{ij} measures the

[1]This inclusion means that for all requests $r \in R$, $\langle r, b(r)\rangle \in M$.

[2]Totally ordered costs are typically appropriate for uniform costs, and for multi-dimensional costs with a total preference over dimensions (e.g., money over time, and so on).

packet rate allowed by the connection between i and j, and the values c_j measure the packet rate of the servers; then it is appropriate to set $f = \min$.

For another example, suppose the values k_{ij} measure the quality of the connections between i and j, and the values c_j measure the level of trust in the information released by service j. Then a service $b(r)$ may be preferred because (i) the quality of the connection is good, and at the same time (ii) the level of trust in $b(r)$ is high. In this case $f = \min$ does not seem adequate; it is not sensitive to any increment of the maximal argument, therefore it does not forces simultaneous improvement of the two values k_{ij} and c_j. A function more sensitive to both of its parameters seems more appropriate (e.g., one may adopt $f = +$ or $f = \times$).

The third objective function, denoted by $\mathcal{Q}'_b$, is appropriate for QoS-like criteria, too. Sometimes, in a compound service, the quality of the worst component service affects the quality of the entire service. For example, the overall privacy preservation degree of a compound service issuing a set of requests R, is determined by the minimal privacy preservation degree of the service components (i.e. the individual invocations $b(r)$). In this kind of scenario, the quality estimates $f(k_{r\,b(r)}, c_{b(r)})$ are combined by taking their minimum:

$$\mathcal{Q}'_b = \min\{f(k_{r\,b(r)}, c_{b(r)}) \mid r \in R\}\,. \qquad (4)$$

The three objective functions $\mathcal{C}_b$, $\mathcal{Q}_b$, and $\mathcal{Q}'_b$ induce three classes of SSP:

SSP$_\mathcal{C}$: Given a SSP instance I, find a binding b for I that minimizes the cost function $\mathcal{C}_b$.

SSP$_\mathcal{Q}$: Given a SSP instance I, find a binding b for I that maximizes the quality function $\mathcal{Q}_b$.

SSP$'_\mathcal{Q}$: Given a SSP instance I, find a binding b for I that maximizes the quality function $\mathcal{Q}'_b$.

The last two problems, $\mathcal{Q}_b$ and $\mathcal{Q}'_b$, are not much different from each other, as stated by the following result:

THEOREM 3.1. *For each SSP instance I,*

1. *All the solutions of I under SSP$_\mathcal{Q}$ are also solutions of I under SSP$'_\mathcal{Q}$.*
2. *Conversely, at least one solution of I under SSP$'_\mathcal{Q}$ is also a solution of I under SSP$_\mathcal{Q}$.*

Intuitively, the reason is that SSP$'_\mathcal{Q}$ considers only the bottlenecks, while SSP$_\mathcal{Q}$ tries to improve all services.

4. COMPLEXITY OF AND ALGORITHMS FOR SSP$_\mathcal{C}$

We prove that SSP$_\mathcal{C}$ is **NP**-hard by reduction from the *Uncapacitated Facility Location Problem* (UFLP), which is defined as follows. We are given a bipartite graph (F, C) with set of n facilities F and m cities C. Let f_j represent the cost of opening a facility at location j in F, and c_{ij} represent the cost of serving city i from an open facility j. The goal is to find a subset I of F along with an assignment function $\Phi : C \to I$ to assign the cities such that the total cost is minimized.

There is a great variety of types of facility location problems depending on the features of the components that contribute in the model definition. Some basic classes of facility location problems are listed below.

1. If there are given upper bounds on the number of cities a facility can serve, then the corresponding problem is classified as a *Capacitated Facility Location Problem.*
2. If some data are given by a probability distribution, the problem is considered to be *stochastic*; otherwise it is referred to as a *deterministic* one.
3. If the decision process is concerned not only with the location of the facilities to be open but also with "the moment" of their opening, then the corresponding problem is called a *Dynamic Facility Location Problem*; otherwise it is called a *Static Facility Location Problem.*

The uncapacitated facility location problem we need in this paper is static and deterministic and admits the following integer programming formulation.

$$\begin{array}{llll}
(\textbf{UFLP}) & \min & \displaystyle\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\, x_{ij} + \sum_{j=1}^{n} f_j y_j & \\
 & \text{s.t.} & & \\
 & & \displaystyle\sum_{j=1}^{n} x_{ij} \quad = 1, \quad 1 \le i \le m & \text{(a)} \\
 & & y_j - x_{ij} \quad \ge 0, \quad 1 \le i \le m,\ 1 \le j \le n & \text{(b)} \\
 & & x_{ij},\ y_j \in \{0,1\} \quad 1 \le i \le m,\ 1 \le j \le n, &
\end{array}$$

where constraints (a) impose that each city is assigned to at least one facility, while constraints (b) restrict assignments to open facilities only.

Despite their simple formulation, most location problems are very difficult to solve. Except for some special cases, their decision version (*For a given K, is there a solution with cost $\le K$?*) have been shown to be **NP**-hard by reduction from the *Vertex Cover Problem* (membership in **NP** is straightforward). An extensive survey of location problems, their complexities and applications can be found in the book edited by Mirchandani and Francis [18].

By setting $R = C$, $O = F$, $c_j = f_j$, and $k_{ij} = c_{ij}$ ($1 \le i \le m$, $1 \le j \le n$), UFLP can be reduced to SSP$_\mathcal{C}$, and viceversa. Then, we can prove the following result:

PROPOSITION 4.1. *Deciding whether the optimal cost of a given instance of SSP$_\mathcal{C}$ is less than or equals a given rational K is* **NP***-complete.*

With this result, we can express the optimality check as the conjunction of an **NP**-complete test and a **co-NP**-complete test, so we get the following theorem.

THEOREM 4.2. *Deciding whether the optimal cost of a given instance of SSP$_\mathcal{C}$ equals a given rational K is* **DP***-complete.*

The optimal cost can be computed through a binary search of K, based (by the above proposition) on an oracle for **NP**. This procedure provides an upper bound to the complexity of the optimization problem.

THEOREM 4.3. *Computing the optimal cost of a given instance of SSP$_\mathcal{C}$ is in* $\mathbf{FP}^{\mathbf{NP}}$.

Note that by Theorem 4.2, the optimization problem is harder than **NP**, unless the polynomial hierarchy collapses. The source of complexity lies in the one-time costs associated to services, as shown in the next two subsections.

4.1 Exact and approximated algorithms

The algorithms described in this section accept slightly modified instances of SSP$_{\mathcal{C}}$, where the function k is extended to all of $R \times O$ by setting $k_{ij} = +\infty$ for all $\langle i, j\rangle \in (R \times O) \setminus M$.

Algorithm 1 solves exactly the problem in the obvious way, by exhaustively trying all possible bindings. The only optimization consists in aborting a tentative binding construction whenever the value of the current partial binding exceeds the best cost found so far.

Nevertheless, the intractable nature of the problem makes approximate solutions the natural choice for dealing with large instances. The first constant factor approximation algorithm for facility location problems due to Shmoys et al. appeared in the literature in 1997 [23]. In 1999 Guha and Khuller [8] proved that it is impossible to get an approximation guarantee of 1.463 unless $\mathbf{NP}\subseteq$DTIME$[n^{O(\log\log n)}]$. Since then, several scientific papers have been published along this line of research [10, 9, 14, 25].

In our exploration of the approximate solutions we have implemented the best known approximation algorithm (Algorithm 2) proposed by Mahdian et al. [15]. This algorithm ensures that the ratio between the cost of the returned solution and the optimal cost is bounded by 1.52. Algorithm 2 combines the greedy algorithm proposed by Jain et al. [9] (Algorithm 3) with the idea of cost scaling and can be implemented in quasi-linear time, as showed by the authors using a result of Thorup [27].

We have also investigated a simple heuristic approach requiring time $O(mn^2)$. Algorithm 4 consists of two phases: a greedy adaptive construction phase (line 3, calling Algorithm 5) and a local search phase (lines 4–21). These algorithms use the sets of requests R_s served by each service s, formally defined by

$$R_s = \{r \in R \mid b(r) = s\}\,.$$

Algorithm 4 and Algorithm 5 return *inverse* bindings, represented by pairs $(y, \{R_s\}_{s\in O})$ where (i) y is a boolean vector such that $y_s = 1$ iff offer s is used in the binding, and (ii) the family $\{R_s\}_{s\in O}$ defines for each offer s the requests satisfied by s.

Starting from an empty solution, the first phase (Algorithm 5) iteratively constructs a feasible solution in a greedy and adaptive fashion with a greedy function defined on both matching and service costs. At each iteration, a new matching is determined between an unmatched request and the most convenient offer. In future iterations, the cost of this offer will not be considered again while evaluating the greedy choice (a greedy adaptive schema). The running time of Algorithm 5 that performs this phase is $O(mn)$.

Starting from the feasible binding found by the construction phase, the local search phase tries (in time $O(mn)$) to find a better binding by slightly perturbing it. In particular, for each invoked offer $s \in \{1, 2, \ldots, n\}$ the algorithm looks for an alternative and more convenient offer $l \neq s$ that can serve the requests currently matched to s (l may have been already associated to other requests, but not necessarily). If such a service l is found, then all the requests served by s are redirected to l.

This strategy is expected to work especially well in the presence of multi-function services that make per-use discounts to users that register to many of the service's options. In case of heavy use of these functionalities, the algorithm is likely to find that the service is more convenient even if its one-time cost is higher than those of the competing services. An experimental evaluation of Algorithms 2 and 4 is discussed in Section 4.4.

Algorithm 1
ExhaustiveSearch $(UR, CO, PC, BC, c, k, b)$

```
1: Inputs: $UR$:unmatched requests, $CO$:called offers, $PC$:partial cost, $BC$:best cost, $c$:vector of costs associated to services, $k$:matrix of costs associated to invocations.
2: Outputs: best cost and an optimal binding $b$.
3: begin
4: if $PC \geq BC$ then
5:    return $BC$ {abort search; keep current best cost}
6: else if $UR = \emptyset$ then {we found a better complete solution, as $PC < BC$}
7:    save the current binding $b$;
8:    return $PC$ {new best cost}
9: else
10:    choose $r \in UR$;
11:    for all $s$ such that $k_{rs} < +\infty$ do
12:       $b(r) := s$; {bind $r$ to $s$}
13:       if $s \in CO$ then
14:          $PC_s := PC + k_{rs}$
15:       else
16:          $PC_s := PC + k_{rs} + c_s$
17:       $BC$ := ExhaustiveSearch$(UR \setminus \{r\}, CO \cup \{s\}, PC_s, BC, c, k)$
18:    return $BC$
19: end
```

4.2 A polynomially solvable subclass

Let us suppose that the one-time costs associated to services are null, that is:

$$\forall s \in O,\ c_s = 0\,. \tag{5}$$

(the costs k_{rs} associated to service invocations may be greater than zero). This special case of SSP is equivalent to a special transportation problem and can be polynomially solved by following a greedy approach with greedy function given by $k:\ R \times O \to \mathbb{Q}$. The optimal solution activates all services and simply matches a service request with the cheapest service. It is easy to show that the optimal cost can be computed through GreedyAdapt (Algorithm 5) with null input cost vector c:

Theorem 4.4. *The binding corresponding to the values $y, \{R_s\}_{s\in O}$ returned by Algorithm 5 is optimal if c is null.*

Note that in this case GreedyAdapt is a pure greedy algorithm running in $O(mn)$ time. The next corollary follows immediately.

Corollary 4.5. *If (5) holds, then SSP$_{\mathcal{C}}$ can be solved in time $O(mn)$.*

4.3 The source of complexity of SSP$_{\mathcal{C}}$

In the light of Section 4.2, it is interesting to investigate the complexity of SSP$_{\mathcal{C}}$ when the invocation costs are null, that is:

$$\forall\ \langle r, s\rangle \in M,\ k_{rs} = 0 \tag{6}$$

Algorithm 2
1.52APPROX (m, n, c, k, δ)

```
Outputs: for $\delta = 1.504$, 1.52-approximate binding - represented by $y$ and $\{R_s\}_{s\in O}$ - and its cost $C$.
begin
for all $s = 1$ to $n$ do
    $c(s) := c(s) * \delta$
$(y, \{R_s\}_{s\in O}, C, d) :=$ JAIN $(m, n, c, k)$
for all $s = 1$ to $n$ do
    $c(s) := \frac{c(s)}{\delta}$
bool:=true
while (bool) do
    max:= 0
    for $s = 1$ to $n$ s.t. $y_s = 0$ do
        $\hat{C} := 0$, $\bar{C} := 0$
        $Q := \emptyset$
        for $r = 1$ to $m$ s.t. $k_{rs} < +\infty$ do
            $Q := Q \cup \{r\}$
            $\hat{C} := \hat{C} + k_{rs}$
            $\bar{C} := \bar{C} + k_{rb(r)}$
        if (max$< (\bar{C} - \hat{C} - c_s)/c_s$) then
            max:= $(\bar{C} - \hat{C} - c_s)/c_s$
            $v := s$
            $\hat{C}_v := \hat{C}$
            $Q_v := Q$
    if (max$> 0$) then
        $y_v := 1$, $d_v := \hat{C}_v$, $R_v := Q_v$
        for $r \in R_v$ do
            $j := b(r)$
            $d_j := d_j - k_{rj}$
            $R_j := R_j \setminus \{r\}$
            if $(R_j = \emptyset)$ then
                $y_j := 0$
            $b(r) := v$
    else
        bool:=false
return $C$
end
```

Algorithm 3
JAIN (m, n, c, k)

```
Outputs: 1.61-approximate binding - represented by $y$ and $\{R_s\}_{s\in O}$ - and its cost $C$.
begin
$C := 0$
for all $r = 1$ to $m$ do
    $b(r) := 0$, budget$(r) := 0$
for all $s = 1$ to $n$ do
    $y_s := 0$
while (there exists $r \in \{1, 2, \ldots, m\}$ s.t. $b(r) = 0$) do
    for all $r = 1$ to $m$ s.t. $b(r) = 0$ do
        budget$(r)$ :=budget$(r) + 1$
    for all $s = 1$ to $n$ do
        if $y_s = 0$ then
            totoffer:= 0, $i := 0$
            for all $r = 1$ to $m$ do
                if $(b(r) = 0)$ then
                    if (budget$(r) - k_{rs} > 0$) then
                        totoffer:=totoffer+budget$(r) - k_{rs}$
                        $i := i + 1$, $L(i) := r$
                else
                    if $(k_{rb(r)} - k_{rs} > 0)$ then
                        totoffer:=totoffer+$k_{rb(r)} - k_{rs}$
                        $i := i + 1$, $L(i) := r$
            if (totoffer$\geq c_s$) then
                $y_s := 1$
                for $k = 1$ to $i$ do
                    $v := L(k)$
                    $j := b(v)$
                    if $(j \neq 0)$ then
                        $R_j := R_j \setminus \{v\}$
                        $d_j := d_j - k_{vj}$
                        if $(R_j = \emptyset)$ then
                            $y_j = 0$
                    $b(v) := s$, $R_s := R_s \cup \{v\}$
                    $d_s := d_s + k_{vs}$
        else
            for $r = 1$ to $m$ do
                if $(b(r) = 0)$ then
                    if (budget$(r) = k_{rs}$) then
                        $R_s := R_s \cup \{r\}$
                        $b(r) := s$, $d_s := d_s + k_{rs}$
for all $s = 1$ to $n$ do
    if $(y_s = 1)$ then
        $C := C + c_s + d_s$
return $(y, \{R_s\}_{s\in O}, C, d)$
end
```

Algorithm 4
GreedyAdaptHeur (m, n, c, k)

```
1: Outputs: suboptimal binding – represented by $y$ and $\{R_s\}_{s\in O}$ – and its cost $C$.
2: begin
3: $(y, \{R_s\}_{s\in O}, C, d) :=$ GreedyAdapt $(m, n, c, k)$
   {Local search phase}
4: for all $s = 1$ to $n$ do
5:   if $y_s = 1$ then
6:     $improved := false$
7:     $l := 1$
8:     while (not $improved$ and $l \leq n$) do
9:       if $l \neq s$ then
10:        $q := \sum_{r\in R_s} k_{rl}$
11:        gain $:= (\ c_s + d_s\ ) - [\ c_l(1 - y_l) + q\ ]$
12:        if gain$> 0$ then
13:          $R_l := R_l \cup R_s$
14:          $d_l := d_l + q$
15:          $R_s := \emptyset$
16:          $d_s := 0$
17:          $y_s := 0$
18:          $y_l := 1$
19:          $improved := true$
20:          $C = C-$gain
21:      $l := l + 1$
22:      {end while}
23: return $(y, \{R_s\}_{s\in O}, C)$
24: end
```

Algorithm 5
GreedyAdapt (m, n, c, k)

```
1: Outputs: suboptimal binding – represented by $y$ and $\{R_s\}_{s\in O}$ –, its cost $C$ and the costs $d$ (see below).
2: begin
3: for all $s = 1$ to $n$ do
4:   {Init structures}
5:   $R_s := \emptyset$
6:   $y_s := 0$ {i.e. $s$ not used}
7:   $d_s := 0$ {total cost of all calls to $s$}
8: $C := 0$
9: for all $r = 1$ to $m$ do
10:  min $:= +\infty$
11:  for all $s = 1$ to $n$ do
12:    if min $> c_s(1 - y_s) + k_{rs}$ then
13:      min $:= c_s(1 - y_s) + k_{rs}$
14:      $best := s$
15:  $C := C +$ min
16:  $R_{best} := R_{best} \cup \{r\}$
17:  $d_{best} := d_{best} + k_{r\ best}$
18:  $y_{best} := 1$
19: return $(y, \{R_s\}_{s\in O}, C, d)$
20: end
```

(the costs c_j may be nonzero.) In this case, SSP$_\mathcal{C}$ remains difficult. Its computational complexity remains high even if (6) holds *and* the costs c_j are all identical (but nonzero), that is,

$$\forall\ \{s, t\} \subseteq O,\ \ c_s = c_t \neq 0\,. \tag{7}$$

To prove this, we note that the *hitting set problem* [7] can be reduced to the decision version[3] of SSP$_\mathcal{C}$ satisfying (6) and (7). The hitting set problem can be formulated as follows:

> Given a finite set S, a collection of sets $S_i \subseteq S$ $(1 \leq i \leq z)$, and a positive K, decide whether there exists $S' \subseteq S$ such that for all $i = 1 \ldots z$, $S' \cap S_i \neq \emptyset$ and $|S'| < K$.

The hitting set problem is known to be **NP**-complete. The hitting set problem can be reduced to the decision version of SSP$_\mathcal{C}$ under restrictions (6) and (7) by defining: $R = \{1, \ldots, z\}$, $O = S$ (we may assume w.l.o.g. that S is a finite initial segment of $\mathbb{N}$), $M = \{\langle i, j\rangle \mid j \in S_i\}$, and $c = \{1\}^m$.

Then, by analogy with the cost estimates for the general case, we can prove that for the class of SSP$_\mathcal{C}$ instances satisfying (6) and (7):

- Checking whether the optimal cost equals a given rational K is **DP**-complete.
- Computing the optimal cost is in $\mathbf{FP}^{\mathbf{NP}}$.

From this result and the results of the previous section, we conclude that the costs c_j associated to services are entirely responsible for the high computational complexity of SSP$_\mathcal{C}$. This holds even if the service offers all have the same cost. Intuitively, in this case, it is hard to choose among services with the same activation cost that compete by offering different, partially overlapping sets of free functionalities.

4.4 Experimental results

We performed some preliminary experiments with Algorithms 1, 2, and 4, using a C implementation running on a Pentium 4, 2.4GHz, 512Mb.

To compare the algorithms, we applied them to a set of 300 randomly generated instances, according to the following criteria. Recall that m is the number of requests and n is the number of offers. We have considered instances with $5 \leq m \leq 100$ and $100 \leq n \leq 10000$ (assuming that the set of offers in practice will be significantly larger than the set of requests in a workflow). We fixed the range of the invocation costs k to $[0, 100]$ and the range of the one-time costs c to $[1, p \cdot 100]$ for $p = 0.1, 1, 10$, in order to check the influence of the relative weight of k and c. For each triple (m, n, p) 10 instances have been randomly generated. The runs longer than 1 hour have been killed.

Algorithm 1 (that computes an optimal solution) exhibited a satisfactory performance for all the instances with $m \leq 10$ and $n \leq 100$. The maximal elapsed time was 0.35 seconds.

The performance started to decrease for $(n, m) = (15, 150)$.

- For $(n, m) = (15, 150)$ the maximal elapsed time was $5'$:$31''$.

[3]The decision version of SSP$_\mathcal{C}$ is: Given an SSP instance and a cost K, decide whether there is a solution with cost $\leq K$.

- For $(n, m) = (20, 200)$, 20% of the runs have been killed and the maximal elapsed time of the other runs was $21'{:}01''$.
- For $(n, m) = (20, 200)$, 73% of the runs have been killed and the maximal elapsed time of non-killed runs was over 59 minutes.

Algorithms 2 and 4 are much faster, of course. The former has been killed only once ($m = 100$, $n = 10000$), the latter has never been killed. The average time of Algorithm 2 was 2.5 minutes. Algorithm 4 seems to be faster (average less than 30 seconds), but more extensive experimentations are needed to confirm and explain this observation.

Some of the average execution times of Algorithm 2 are reported in Figure 1. The figure illustrates both how execution time grows with the size of the problem instance, and the influence of one-time costs on performance. In particular, it appears that as one-time costs become negligible, Algorithm 2 becomes faster. When the upper bound for one-time costs used by the random generator is one tenth of the upper bound for per-use costs, the average time drops down to 65.19 seconds.

We measured the quality of the approximate solutions returned by Algorithm 4 by evaluating the *relative error* of each solution; the relative error is $\frac{\mathcal{A}-\mathcal{C}}{\mathcal{C}}$, where $\mathcal{A}$ is the approximate cost and $\mathcal{C}$ is the optimal cost (we computed the error only for those instances whose optimal cost was available, i.e. the exact algorithm was not killed). The average of the errors is around 70%, which is not bad for a naive heuristics. Also in this case, we need more experiments to validate this observation.

5. COMPLEXITY OF AND ALGORITHMS FOR SSP$_\mathcal{Q}$ AND SSP$'_\mathcal{Q}$

Unlike SSP$_\mathcal{C}$, SSP$_\mathcal{Q}$ and SSP$'_\mathcal{Q}$ are always easy. These two problems can be solved almost in the same way. Algorithm 6 solves the version of SSP$_\mathcal{Q}$ with objective functions $\mathcal{Q}_b$.

Algorithm 6
GreedyAdapt-$\mathcal{Q}\,(m, n, c, k)$

```
1:  Outputs: an optimal binding $b$ and its quality level $L$.
2:  begin
3:  $L := +\infty$
4:  for all $r = 1$ to $m$ do
5:     $maxlev_r := -\infty$
6:     for all $s = 1$ to $n$ do
7:        if $maxlev_r < f(k_{rs}, c_s)$ then
8:           $maxlev_r = f(k_{rs}, c_s)$
9:           $best := s$
10:    $L := L + maxlev_r$
11:    $b(r) := best$
12: return $(b, L)$
13: end
```

To solve the version based on $\mathcal{Q}'_b$, only one change to Algorithm 6 is required: replace line 10 with

$$10: \quad L := \min\{maxlev_r, L\}\,.$$

It is not hard to prove the correctness of the two versions of Algorithm 6 w.r.t. SSP$_\mathcal{Q}$ and SSP$'_\mathcal{Q}$; from this property and a straightforward analysis of Algorithm 6, we conclude that:

Theorem 5.1. *SSP$_\mathcal{Q}$ and SSP$'_\mathcal{Q}$ can be solved in time $O(mn)$.*

This approach can be easily extended to any objective function similar to $\mathcal{Q}$ and $\mathcal{Q}'$, based on polynomially computable, monotonic combination functions besides $\sum$ and min. The details will be given in an extended version of the paper.

6. CONCLUSIONS

Summarizing, we formalized three kinds of optimal service selection problems—based on cost minimization and on two different quality maximization criteria—and we proved that the cost minimization problem, SSP$_\mathcal{C}$, is generally hard, while the two quality maximization problems, SSP$_\mathcal{Q}$ and SSP$'_\mathcal{Q}$, can be solved in polynomial time. In particular, SSP$_\mathcal{C}$ is in $\mathbf{FP^{NP}}$ and harder than $\mathbf{NP}$(unless the polynomial hierarchy collapses).

We proved that the reason of the high computational complexity of SSP$_\mathcal{C}$ lies in the one-time costs associated to service offers (such as initialization and registration costs). When these costs are all null, SSP$_\mathcal{C}$ is solvable in polynomial time (on the contrary, in the absence of per-use costs the problem does not become easier).

We designed and implemented algorithms for computing exact solutions for all these versions of SSP. The exact algorithm for SSP$_\mathcal{C}$ (Algorithm 1) has been evaluated experimentally. According to the current results, instances with up to 10 requests and 100 offers can be nicely handled by this algorithm; for larger instances, the performance quickly decreases, making the algorithm inapplicable.

We have also designed and implemented suboptimal solutions and evaluated them experimentally. Currently, it seems that the algorithm with a guaranteed 0.52 bound on relative error is too slow for real-time service selection over large workflows and offer sets. The heuristic algorithm (Algorithm 4) seems to be faster, but it has no guarantees on the quality of the solution.

We are planning to carry out more experiments to validate and refine these preliminary observations. Moreover, we are trying to sharpen the complexity bounds (we do not yet know whether SSP$_\mathcal{C}$ is complete for $\mathbf{FP^{NP}}$).

Finally, we are generalizing the framework presented in this paper by considering optimization problems that involve simultaneous cost minimization and quality maximization, as well as multidimensional measures that induce partially ordered measures of nonfunctional properties (although in several cases multiple criteria can be reduced to single, totally ordered numeric measures [5]). Another direction for generalization concerns time-dependent costs and preferences.

7. ACKNOWLEDGMENTS

The authors are grateful to the anonymous referees for their insightful and constructive comments, and to Alessio Contino for his help in the implementation and experimentation phase. This work has been partially supported by the Network of Excellence REWERSE, EU Sixth Framework Program, IST-2004-506779.

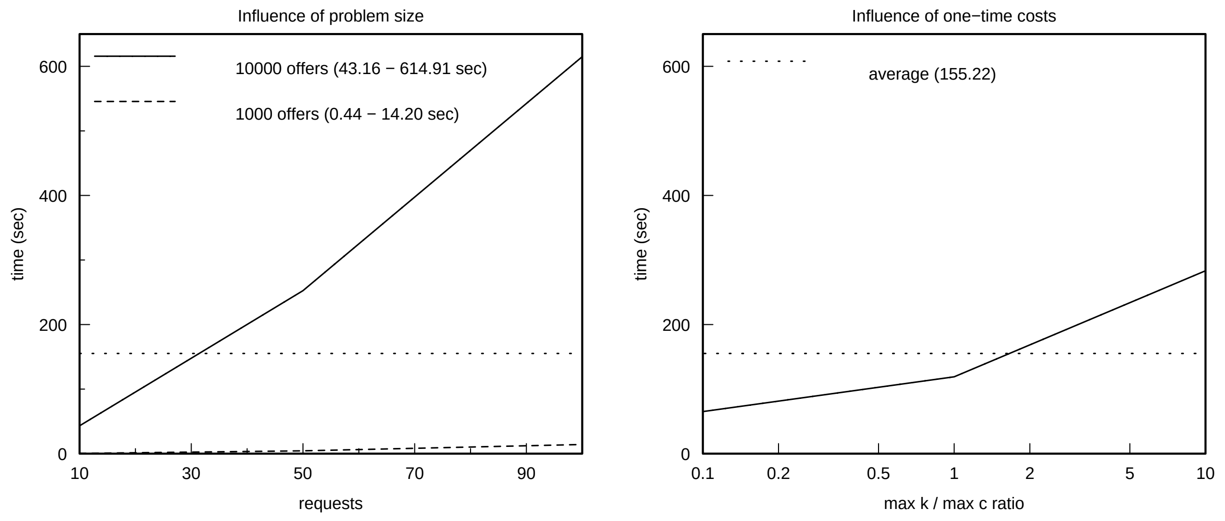

Figure 1: Performance of Algorithm 2

8. REFERENCES

[1] K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *Cooperative Information Systems, 9th International Conference, CoopIS 2001*, volume 2172 of *LNCS*, pages 179–194. Springer, 2001.

[2] P. Bonatti. Towards service description logics. In *Logics in Artificial Intelligence, European Conf., JELIA 2002*, volume 2424 of *LNCS*, pages 74–85. Springer, 2002.

[3] I. Constantinescu, W. Binder, and B. Faltings. An Extensible Directory Enabling Efficient Semantic Web Service Integration. In *3rd International Semantic Web Conference (ISWC04)*, Hiroshima, Japan, November 2004.

[4] I. Constantinescu, B. Faltings, and W. Binder. Type-based composition of information services in large scale environments. In *The 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)*, Beijing, China, September 2004.

[5] V. Deora, J. Shao, W. Gray, and N. Fiddian. A quality of service management framework based on user expectations. In *Service-Oriented Computing - ICSOC 2003*, volume 2910 of *LNCS*, pages 104–114. Springer, 2003.

[6] I. Fikouras and E. Freiter. Service discovery and orchestration for distributed service repositories. In *Service-Oriented Computing - ICSOC 2003*, volume 2910 of *LNCS*, pages 59–74. Springer, 2003.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[8] S. Guha and S. Khuller. Greedy strikes back: improved facility location algorithms. *Journal of Algorithms*, 31:228–248, 1999.

[9] K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *34th Annual ACM Symposium on Theory of Computing*, pages 731–740, 2002.

[10] K. Jain and V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of ACM*, 48:274–296, 2001.

[11] T. Kawamura, J.-A. D. Blasio, T. Hasegawa, M. Paolucci, and K. Sycara. Preliminary report of public experiment of semantic service matchmaker with uddi business registry. In *Service-Oriented Computing - ICSOC 2003*, volume 2910 of *LNCS*, pages 208–224. Springer, 2003.

[12] O. Lassila and S. Dixit. Interleaving discovery and composition for simple workflows. In *Semantic Web Services, AAAI Spring Symposium Series*. AAAI, 2004.

[13] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the Twelfth International World Wide Web Conference (WWW'2003)*, pages 331–339. ACM, 2003.

[14] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *5th International Workshop on Randomization and Approximation Techniques in Computer Science (APPROX 2002)*, pages 127–137. Springer-Verlag, 2002.

[15] M. Mahdian, Y. Ye, and J. Zhang. Approximation algorithms for metric facility location problems. *Submitted to SIAM Journal on Computing*, 2004.

[16] O. Martín-Díaz, A. Ruiz-Cortés, A. Durán, D. Benavides, and M. Toro. Automating the procurement of web services. In *Service-Oriented Computing - ICSOC 2003*, volume 2910 of *LNCS*, pages 91–103. Springer, 2003.

[17] A. Maurino, S. Modafferi, and B. Pernici. Reflective architectures for adaptive information systems. In *Service-Oriented Computing - ICSOC 2003*, volume 2910 of *LNCS*, pages 115–131. Springer, 2003.

[18] P. Mirchandani and R. Francis. *Discrete location theory*. John Wiley and Sons, 1990.

[19] P. Oaks, A. ter Hofstede, and D. Edmond. Describing what services can do. In *Service-Oriented Computing - ICSOC 2003*, volume 2910 of *LNCS*, pages 1–16. Springer, 2003.

[20] A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors. *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer, 2001.

[21] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC2002)*, 2004.

[22] C. Papadimitriou. *Computational complexity*. Addison Wesley, 1994.

[23] D. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *29th Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[24] A. Singh and L. Liu. Trustme: Anonymous management of trust relationships in decentralized P2P systems. In *3rd International Conference on Peer-to-Peer Computing (P2P 2003)*, pages 142–149. IEEE Computer Society, 2003.

[25] M. Sviridenko. An improved approximation algorithm for the metric uncapacitated facility location problem. In *9th Conference on Integer Programming and Combinatorial Optimization*, pages 240–257, 2002.

[26] K. Sycara, J. Lu, M. Klusch, and S. Widoff. Matchmaking among heterogeneous agents on the Internet. In *Proc. of the AAAI Spring Symposium on Intelligent Agents in Cyberspace*, 1999.

[27] M. Thorup. Quick and good facility location. In *14th ACM-SIAM Symposium on Discrete Algorithms*, pages 178–185, 2003.

[28] Y. Wang and E. Stroulia. Semantic structure matching for assessing web service similarity. In *Service-Oriented Computing - ICSOC 2003*, volume 2910 of *LNCS*, pages 194–207. Springer, 2003.

[29] Y. Wang and J. Vassileva. Bayesian network-based trust model. In *2003 IEEE / WIC International Conference on Web Intelligence, (WI 2003)*, pages 372–378. IEEE Computer Society, 2003.

[30] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Trans. on Knowledge and Data Engineering*, 16(7):179–194, 2004.