Online Pricing for Web Service Providers

Shahram Esmaeilsabzali
David R. Cheriton School of Computer Science
University of Waterloo
sesmaeil@cs.uwaterloo.ca

Nancy A. Day
David R. Cheriton School of Computer Science
University of Waterloo
nday@cs.uwaterloo.ca

ABSTRACT

We consider Web service providers, which have a finite capacity, and requests for their services, which arrive sequentially over time, and propose an online algorithm for selecting from such requests and charging for such requests. We show that different variations of this problem, both as online and offline (when we know all requests a priori), are hard problems. We initially start with two naive variations of the problem and show these variations are too hard to be solved. Then, we propose an online algorithm for a variation of the problem where we make some statistical assumptions about the requests that Web service providers receive over time.

Categories and Subject Descriptors

F.1.2 [**Theory of Computation**]: Modes of Computation—*Online Computation*; K.6.0 [**Computing Milieux**]: Management Of Computing and Information Systems—*Economics*

General Terms

Algorithms, Economics, Management

Keywords

Online Algorithms, Pricing, Economics, Web services

1. INTRODUCTION

The purpose of this paper is, first, to state and formalize the problem of pricing for Web services; and, second, to provide an online algorithm for selecting from service requests and charging for these requests, which arrive over time, in a way that Web service providers achieve optimal profit.

Web services can be considered as pay-per-view functionalities that are invoked over the internet. Web service technologies are based upon a set of standards, and as such, accommodate a paradigm of software development where service requesters invoke their desired functionalities by inspecting their interfaces and calling the services over the internet. This paradigm of computation requires the Web service providers to sell their services wisely. By accepting all requests for their services early, they may exhaust their capacities early, and miss the chance to accept more profitable requests later. Conversely, they cannot be too conservative and reject all requests for their services. In this paper, we examine this problem to try to formulate it in a reasonable way, and propose a solution for it.

We are interested in studying how service providers[1] can select from their incoming requests and how they can charge for such requests, which have certain budgets. We call this problem, the *online pricing* problem. We are interested in considering the pricing problem for *public Web services*. Public Web services advertise their functionalities, and allow different service requesters to use them. *Proprietary Web services* are designed for use by specific service requesters. Public service providers have a limited capacity and there is a cost for developing and maintaining their services.

Binding between a service requester and a service provider can happen *statically*, at design time, or *dynamically*, at run time [4]. Dynamic binding relies on having *agents* for discovering the desired functionality at run time. Static binding, on the other hand, relies on the functionality of a predetermined Web service at run time. In this paper, we consider the problem of pricing in the static binding setting. We believe that static binding is a more realistic scenario than dynamic binding, because technologically the vision of dynamic binding is not mature and reliable yet. In the static binding setting, we characterize a service request as a request to consume a certain portion of the capacity of the service provider indefinitely. As an example, assume we are designing a system that uses a currency exchange functionality, provided by the service *Exchange*. Also, assume that our system may have concurrent invocations of *Exchange*, up to a maximum of three concurrent invocations. A request for the *Exchange* Web service would then ask for three units of *Exchange*, within a certain budget. *Exchange*, on other hand, commits to provide us with its service, up to three concurrent invocations, at all times. Such a paradigm of computation provides the opportunity for service providers and requesters to engage in financial relations.

We provide a model that formalizes the attributes of service providers and the requests they receive, which arrive sequentially. The goal is to provide a pricing mechanism that, in the absence of knowledge about future requests, behaves reasonably; we call such a setting an *online setting*,

[1]We use the terms "Web service", "Web service provider", and "service provider" interchangeably.

EDSER'06, May 27, 2006, Shanghai, China.

and such a problem an *online problem*. Online problems are conventionally considered in the *online algorithms* context [2]. Online algorithms process a sequence of incoming requests that arrive sequentially over time. An online algorithm is responsible for processing the requests in such a way that a goal function, *e.g.,* profit maximization, is achieved optimally. In this paper, we use online algorithms as the tool for solving our online pricing problem.

Alternatively, auctions have been used to solve online problems. Two relevant classes of auctions, in this respect, are *competitive auctions* [11], and *online auctions* [14]. Both auctions are *truthful*, *i.e.,* participants in the auction have the incentive to express their preferences truthfully. Both auctions have applications for the trade of digital goods. Also, both auctions promise to never gain profits worse than a *constant* factor of the *optimal outcome*.[2] However, auctions do not quite fit our problem. First, auctions require the buyers and the sellers to behave according to some rules, in order to achieve some elegant properties, *e.g.,* truthfulness. Web services, on the other hand, are built upon the decentralized vision of the internet. It is difficult to advocate an auction mechanism for trading among Web services and their requests, without restating the problem in a superficial way. Second, none of the aforementioned classes of auctions are of immediate use for solving our problem. Furthermore, our problem is inherently a decision problem rather than a trading problem, where we can consider strict regulations for it.

In the next section, we provide a quick overview of online algorithms, and a formalization of our problem. Our formalization does not capture all aspects of the pricing problem; however, despite its simplicity, it captures enough aspects of the pricing problem to allow us to look at some interesting variations. Next, in Section 3, we consider two naive variations of our problem, and show that they are too hard to be solved; for the first variation, we show that it cannot be solved by any online algorithm, and for the second variation, we show that the online problem is related to the well-studied problem of *online knapsack problem* [16], which in its general form has not been solved. We then, in Section 4, consider a third variation of our problem where we make some statistical assumptions about the incoming requests; we propose a threat-based, randomized, online algorithm for this variation. Finally, in Section 5, we present conclusions of our paper, and discuss our plans for future work.

2. PRELIMINARIES

In this section, we first quickly overview online algorithms, in Subsection 2.1, and then, in Subsection 2.2, formalize our pricing problem.

2.1 Online Algorithms Primer

An online algorithm $\mathcal{A}$ is meant to deal with a sequence of incoming requests, $R = \langle R(1), R(2), \cdots, R(m)\rangle$, and process them. At each point of time t, $1 \leq t < m$, $\mathcal{A}$ does not have any knowledge about any request $R(t')$, $t' > t$. Algorithm $\mathcal{A}$ should be able to process any arbitrary request, while optimizing a goal function; *e.g.,* maximizing the profit.

[2]An optimal outcome is defined differently in different auctions. In general, an optimal outcome is an outcome that we can achieve, if we have some (or complete) information about the future.

Assume that the profit of serving an *arbitrary* sequence of requests, $R = \langle R(1), R(2), \cdots, R(m)\rangle$, by $\mathcal{A}$ to be $P_{\mathcal{A}}(R)$. Also, assume that algorithm OPT knows the whole sequence of requests in advance; such an algorithm is called an *offline* algorithm for that sequence. By knowing the sequence in advance, OPT is capable of serving R and gaining maximum possible profit; let that optimal profit be $P_{OPT}(R)$. $\mathcal{A}$ is c-competitive if constants c, which is less than or equal to 1, and a exist such that:[3]

$$P_{\mathcal{A}}(R) \geq c \times P_{OPT}(R) + a$$

Assuming that $\mathcal{A}$ is deterministic and R is any arbitrary sequence of requests, c is the *competitive ratio* of $\mathcal{A}$. In other words, online algorithm $\mathcal{A}$ never gains profit worse than a constant factor of the optimal profit. We are interested in algorithms with bigger c values. In this paper, we first consider the offline versions of our online problems, as indications of how hard their online versions might be.

For more information on financial problems and online algorithms, the interested readers can refer to [5, 2, 10].

2.2 Model

Our formulation of the service selection and pricing problem is mainly based on the formulation introduced in our previous work [8]. In [8], we propose a strategy for determining quality of service for service providers, in an auction setting, but do not provide any strategy for determining the price of service providers. In this paper, we define our model based on the model in [8], with some necessary modifications for the online algorithm setting, as opposed to the auction setting.

2.2.1 Web Service Provider

Each Web service provider can potentially provide service for multiple service requesters. A Web service provider is represented by the pair (N, C). N is the capacity of the Web service; a Web service provider can serve N concurrent instances of service, which can belong to different service requesters, at each point of time. C is the cost of providing one instance of the service continuously. Both C and N are positive integer numbers.

2.2.2 Service Requests

A service request i is defined by $R(i) = (n_i, p_i)$. The number of concurrent services requested by i is represented by n_i. The *maximum* price that the service requester is willing to pay for *each* instance of service is p_i. By buying n_i services from the service provider, R_i can have at most n_i concurrent instances of service active at each point of time. A sequence of service requests is shown by $R = \langle R(1), R(2), \cdots, R(m)\rangle$; we assume that requests are indexed according to the time they arrive.

The service provider would like to employ an online algorithm that can select from the incoming requests and price its service such that its profit is maximized. In the absence of knowledge about future requests, we are interested in an online algorithm that has the largest competitive ratio with respect to the offline algorithm.

[3]Our presentation of competitiveness is based on profit maximization. Conventionally, competitiveness is described with respect to a cost function. In such presentations, c is desired to be as small as possible.

3. TWO VARIATIONS FOR PRICING

In this section, we consider two variations of the pricing problem, namely, *uniform* and *dynamic* pricing. We show that uniform pricing is not solvable by any online algorithm with a constant competitive ratio, and show that dynamic pricing is related to the well-studied *online knapsack problem* [16], which in its general form has not been solved.

3.1 Uniform Pricing

In *uniform* pricing, we are interested in predetermining a price, denoted as P, that the service provider offers to all requests in $R = \langle R(1), R(2), \cdots, R(m)\rangle$. The problem is how to choose P, such that for any arbitrary sequence of requests, P provides a decent competitive ratio. Formally, we would like to choose P and a subset of requests S ($S \subseteq R$) within the capacity N of the service provider that are willing to pay the price P, such that the profit of the service provider is maximized:

$$\max \sum_{R_i \in S} (P - C) \times n_i$$

and the following three conditions hold.

- $\sum_{R_i \in S} (n_i) \leq N$ (The service provider has the capacity to provide the service to the selected requests.)
- $P > C$ (The price is greater than the cost of providing the service.)
- $\forall i \cdot (n_i, p_i) \in S \Rightarrow (P \leq p_i)$ (P is less than or equal to the price the requesters are willing to pay for the service.)

Because it is always possible to create a bad sequence of requests, where none of the requests are willing to pay price P, there does not exist any online algorithm with a constant competitive ratio for this problem. Not surprisingly even the offline version of this problem, when we know the whole sequence of R in advance, is a difficult problem.

PROPOSITION 1. *The offline version of the uniform pricing problem for Web services is NP-hard.*

PROOF. To show that this problem is NP-hard, we reduce instances of the Subset-Sum problem, as defined in [15], to this problem. The Subset-Sum problem is a particular case of the knapsack problem where the price of items, π_j's, are equal to their weights, w_j's. For an instance of the Subset-Sum problem $\langle (w_1, w_2, \cdots, w_n), T\rangle$, we are interested in finding a subset of integer w_i's, that belong to $(w_1, w_2, \cdots, w_n)$, such that their sum is equal to T. Any instance of the Subset-Sum problem can be reduced, in polynomial time, to an instance of the uniform pricing problem, as shown in the following.

- $m = n$, $N = T$, $C = 0$ (Create an instance of the uniform pricing problem where the number of service requests is equal to the number of items in the Subset-Sum problem, the capacity of server is equal to the integer value T, and the cost of providing service is 0.)
- $\forall i \cdot 1 \leq i \leq m \Rightarrow (p_i = 1) \wedge (n_i = w_i)$ (Each service request R_i has the maximum budget of 1, and the number of concurrent services w_i.)

□

3.2 Dynamic Pricing

In this section, we relax the uniform pricing variation to an easier case, *i.e., dynamic* pricing, where a service provider dynamically announces a price for each request. Again, we consider an arbitrary sequence of requests $R = \langle (n_1, p_1), (n_2, p_2), \cdots, (n_m, p_m)\rangle$. The server, (N, C), could offer a different price, P_i, for each request $R_i = (n_i, p_i) \in R$. The problem is which requests should be accepted, and how to offer a price for each request such that it would maximize the service provider's profit. If $p_i > C$, trivially, the optimal price P_i is always p_i, and otherwise the request is rejected. In fact, in dynamic pricing, we are basically solving an online version of the classic knapsack problem [15]. Formally, we would like to select a subset of requests, $S \subseteq R$, within the capacity N of the service provider, and propose price P_i, such that the profit of the service provider is maximized:

$$\max \sum_{R_i \in S} (P_i - C) \times n_i$$

and following conditions hold.

- $\sum_{R_i \in S} (n_i) \leq N$ (The service provider has the capacity to provide the service to the selected requests.)
- $\forall i \cdot (n_i, p_i) \in S \Rightarrow (P_i > C)$ (The dynamic price that the service provider announces is greater than the cost of providing the service.)
- $\forall i \cdot (n_i, p_i) \in S \Rightarrow (P_i = p_i)$ (For each accepted request, the optimal price to charge, for the service provider, is equal to the maximum budget of that request.)

Unfortunately, this simpler problem is also NP-hard.

PROPOSITION 2. *The offline version of the dynamic pricing problem for Web services is NP-hard.*

PROOF. To show that this problem is NP-hard, we reduce the knapsack problem to dynamic pricing problem. Consider an instance of knapsack problem $\langle (w_1, w_2, \cdots, w_n), (\pi_1, \pi_2, \cdots, \pi_n), T\rangle$, it is easy to reduce this problem to an instance of dynamic pricing problem in polynomial time.

- $m = n$, $N = T$, $C = 0$ (Create an instance of the dynamic pricing problem where the number of service requests is equal to the number of items in the knapsack problem, the capacity of server is equal to integer value T, and the cost of providing service is 0.)
- $\forall i \cdot 1 \leq i \leq m \Rightarrow (p_i = \pi_i) \wedge (n_i = w_i)$ (Each service request R_i has the maximum budget of π_i, and the number of concurrent services w_i.)

□

The online version of dynamic pricing can be compared with the *dynamic (online) knapsack problem* [16, 13], where items arrive according to a Poisson process in time. Online knapsack problem can be solved, only under some "consistency conditions." The general case of online knapsack problem, however, does not have any known solution with a constant competitive ratio. The authors in [16, 13] evaluate their proposed method for solving online knapsack problem

through numerical analysis. While we find their work related to the pricing problem, the ad hoc cases of the knapsack problem that they solve are not suitable for our pricing problem. Furthermore, their method does not offer any concrete competitive ratio. Interested readers can refer to a PhD thesis on the topic [12].

Next, we consider a variation of our problem that makes some statistical assumptions about the requests.

4. DYNAMIC PRICING AGAINST A STATISTICAL ADVERSARY

In this section, we consider our problem against a *statistical adversary*. Statistical adversaries for online algorithms were first introduced by Raghavan for analyzing an online asset allocation algorithm [17]. Online algorithms always play against an adversary. An adversary tries to request the least desired requests, and the online algorithm should be able to deal with such requests. Statistical adversaries do not behave entirely randomly; inputs under "statistical adversaries" satisfy some statistical properties.

The variation of dynamic pricing using a statistical adversary is motivated by the fact that dynamic pricing, as described in the previous section, does not have a concrete solution, unless we constrain the problem. Our proposed statistical adversary, on the other hand, does not constrain the problem. Instead, in this variation of the problem, our algorithm plays against a weaker adversary than the adversary of the dynamic pricing problem.

In the following, we first introduce the statistical adversary that our algorithm plays against, and then consider the offline and online algorithms for that statistical adversary. We believe that some of the statistical information of our statistical adversary could be extracted from history data about previous requests.

Again, we consider an arbitrary sequence of requests $R = \langle (n_1, p_1), (n_2, p_2), \cdots, (n_m, p_m) \rangle$. The server, (N, C), should decide whether to accept or reject a request, and also offer a price, P_i, for each request. Again, pricing is trivial; if $p_i > C$, then $P_i = p_i$. We would like to select a subset of requests, $S \subseteq R$, within the capacity N of the service provider, and propose price P_i, such that the profit of the service provider is maximized:

$$\max \sum_{R_i \in S} (P_i - C) \times n_i$$

and the following conditions hold.

- $\sum_{R_i \in S} (n_i) \leq N$ (The service provider has the capacity to provide the service to the selected requests.)
- $\forall i \cdot (n_i, p_i) \in S \Rightarrow (P_i > C)$ (The dynamic price that the service provider announces is greater than the cost of providing the service.)
- $\forall i \cdot (n_i, p_i) \in S \Rightarrow (P_i = p_i)$ (For each accepted request, the optimal price to charge, for the service provider, is equal to the maximum budget of that request.)

Our statistical adversary is $(Money, Number, MinPrice, MaxReq, F)$. In the following we describe the role of each element. We expect that service providers can obtain the values of the statistical adversary parameters from the history data of similar service providers and service requests.

- $Money$: The optimal profit that the offline algorithm gains from sequence R.
- $Number$: The number of requests that should be accepted by the offline algorithm to achieve the profit $Money$.[4]
- $MinPrice$: The smallest price that any request would be willing to pay. We require that $MinPrice > C$, and ignore all requests with prices smaller than or equal to C.
- $MaxReq$: The maximum number of concurrent services that is requested by any request R_i. For any such R_i, we have $n_i = MaxReq$.
- Function F : For all $1 \leq x \leq MaxReq$, $F(x)$ represents the number of requests R_i's, such that for all such requests $n_i = x$.

The offline version of our problem, against the above-mentioned adversary, is still a hard problem; it is NP-hard.

PROPOSITION 3. *Pricing against the statistical adversary is NP-hard.*

PROOF. Denote the optimal offline algorithm that solves the dynamic pricing algorithm against the statistical adversary as ALG. To show that this problem is NP-hard, we reduce the *change-making* [15] problem to an instance of ALG. Again, we consider an arbitrary sequence of requests $R = \langle (n_1, p_1), (n_2, p_2), \cdots, (n_m, p_m) \rangle$. Change making is the problem where a cashier wants to provide change for a value T using some existing coins. The goal is to maximize the number of coins that are used in changing. Consider an instance of the change-making problem $\langle (w_1, w_2, \cdots, w_n), T \rangle$, where w_i's are coins with certain denominations and T is the amount for which we would like to solve the changing problem; we reduce this problem to an instance of ALG in polynomial time as follows.

- $m = n$, $N = T$, $C = 0$ (Create an instance of the dynamic pricing against a statistical adversary problem where the number of service requests is equal to the number of items in the change-making problem, the capacity of the server is equal to integer value T, and the cost of providing service is 0.)
- $\forall i \cdot 1 \leq i \leq m \Rightarrow (p_i = w_i) \wedge (n_i = 1)$ (Each service request R_i has the maximum budget of w_i, and the number of requested concurrent services 1.)

In the statistical adversary, $(Money, Number, MinPrice, MaxReq, F)$, except for the values of $Money$ and $Number$, the values for the other parameters can be extracted from the instance of change-making problem. For $Money$, we set it to T, *i.e.,* the value for which change-making should happen. For $Number$, we iterate through all values: $1, 2, \cdots, n$. In this way, to solve a change-making problem, we require ALG to be executed n times with n different statistical adversaries with different $Number$ values. After running ALG n times, we pick the statistical adversary with the largest $Number$ values, and that adversary

[4]We simplify the problem and assume that $Number$ is unique. In future work, we would like to extend our model such that the online algorithm would work based on a set of $Number$ values.

and the chosen items are the solution to the change-making problem. This transformation happens in polynomial time and thus offline dynamic pricing against a statistical adversary is NP-hard. □

4.1 Online Algorithm

In this section, we propose an online algorithm that uses randomization for deciding to accept or reject a particular request. The idea of our online algorithm is based on *threat-based online algorithms*. Threat-based algorithms are introduced in [6], with a more elaborate presentation appearing in [7]. The idea is to consider a competitive ratio r, and at each step, assume that the adversary would offer the *worst possible* requests henceforth. The online algorithm should then make decisions that *guarantee* the competitive ratio of r if the worst case scenario happens at any step. We define the worst case scenario as the situation where the adversary (1) decreases its proposed price to $MinPrice$, and (2) a big chunk of server capacity remains unsued. Since our proposed online algorithm is careful to sell its capacity as it gets closer to the end, the biggest chunk of capacity that could remain unused is of size $MaxReq$.

An important question is then what is the best r that can be achieved? The best attainable competitive ratio r can be computed from the algorithm itself. In other words, we define the algorithm based on an arbitrary r, and then try to find the best r that can be possibly attained; *i.e.,* we try to maximize r analytically in a formula that represents the competitive ratio of our algorithm in a closed form. This approach for computing the best competitive ratio is in accordance with that of [6, 7]. We defer the computation of r as a part of our future work.

Before presenting our algorithm we need to define some notation. At each step i, we define $Profit_i$ as the profit that the online algorithm has up to step i, excluding step i itself. Similarly, we define $Accept_i$ as the number of requests that the online algorithm has accepted up to step i, and $Capacity_i$ as the capacity of the Web service provider that has been sold up to step i. We have $Accept_1 = Profit_1 = Capacity_1 = 0$. We also define $F_i(x)$ as an updated version of $F(x)$, and $F_1(x) = F(x)$. We update $F_i(x) = F_{i-1}(x) - 1$, if R_{i-1} is accepted and $n_{i-1} = x$, and otherwise $F_i(x) = F_{i-1}(x)$; this process is carried out for all $1 \leq x \leq MaxReq$, and all steps $2 \leq i \leq m$.

Figure 1 presents our algorithm, which consists of three steps. For each service request, the three steps are considered in a sequential order; if a condition is satisfied in a step, then the other steps are ignored for that request. The first step of the algorithm checks whether an incoming request, R_i, contributes in maintaining the competitive ratio of r. In other words, in a threat-based environment where the adversary may drop the price to a minimum or a big chunk of the service provider may remain unused, by accepting R_i, we would like to remain r-competitive. Also, notice that if we follow the choices of the offline optimal algorithm, we should not accept more requests than the optimal offline algorithm does; *i.e.,* we should not accept more than $Number$ requests. The second step does the wise thing: when it realizes that there are not many requests left to arrive, it accepts all requests. In fact, once it realizes that the sum of the remaining requests is less than or equal to the remaining capacity, it accepts all requests. The third step is the randomized step of the algorithm;[5] if the conservative criteria for maintaining r-competitive ratio is not satisfied, we risk and probabilistically accept the request. This can be justified by the combinatorial nature of the problem. There is always the chance that the existing request is a good choice for using up the capacity of server optimally.

- At each step i, accept $R_i = (n_i, p_i)$, if both of the following conditions hold:
  - $(Profit_i + (n_i \times (p_i - C))) + \begin{pmatrix} (N - (Capacity_i + n_i + MaxReq)) \\ \times \quad (MinPrice - C) \end{pmatrix} \geq r \times Money$
    (By accepting R_i, and in the worst case scenario, where a big chunk of capacity of the service provider remains unused and the rest of capacity is sold at $MinPrice$, we remain r-competitive.)
  - $Accept_i + 1 \leq Number$
    (We are not accepting more requests than the offline algorithm.)
- At each step i, accept $R_i = (n_i, p_i)$, if:
  $$\sum_{1 \leq x \leq MaxReq} x \times F_i(x) \leq Capacity_i$$
  (If we realize that statistically, there are not enough requests left to arrive that will use the service provider's capacity, then we accept the request.)
- Otherwise, with probability $prob$ accept $R_i = (n_i, p_i)$
  (Randomly accept a request that statistically does not seem to contribute to maintaining the competitive ratio r.)

Figure 1: Threat-based online algorithm for pricing against a statistical adversary.

The competitive ratio of this algorithm is r. Notice that the best value for r is the one that for the worst case sequence of requests, would still make the algorithm r-competitive. In this paper, we do not consider analyzing the competitive ratio of this algorithm. The analysis is non-trivial and should consider the *expected* profit of the algorithm and, furthermore, should maximize r based on the expected profit and probability parameter $prob$.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we considered the problem of request selection and pricing for Web service providers, when their requests arrive over time. We first briefly looked at two naive variations of the pricing problem, namely, uniform and dynamic pricing. For uniform pricing, there does not exist any online algorithm with a constant competitive ratio; and for dynamic pricing, in its general form, there does not exist any known online algorithm with a constant competitive ratio. Our first two variations lead us to a third variation where the requests, which arrive over time, comply with some statistical properties. We proposed a randomized, threat-based, online algorithm for this third variation. Our algorithm con-

[5]For an introduction on randomized online algorithms, readers can refer to [1, 9, 3, 18].

servatively tries to maintain a certain level of competitiveness in comparison to the maximum possible profit.

In our future work, we are interested in investigating if it is possible to weaken some of the statistical assumptions that we make. We are then interested in calculating the closed formula of the competitive ratio of our proposed algorithm. We are also interested in the optimal probability value, *i.e.,* the value of *prob* variable, with which our algorithm gains the best profit. Alternatively, in our algorithm, we may be able to have dynamic values for *prob*; it can be imagined that *prob* changes as requests are viewed/accepted. In particular, it can be conceived that as the capacity of service provider decreases, *prob* should become less lenient in accepting new requests; an analogous approach has been proposed for the online knapsack problem [16].

We are also interested in enhancing our model and algorithm with the notion of time. When rejecting service requests, our algorithm does not consider any penalty for not using its unused remaining resources. In a more realistic model, the service provider will be penalized when its resources remain unused. Also, we are interested in modeling service requests as requests that ask for a certain capacity of the service provider for a *finite* amount of time; as opposed to our current unrealistic approach, where a service requester asks for a certain capacity of the service provider indefinitely.

Acknowledgments

We would like to thank Alex Lopez-Ortiz for helpful comments, and Mehdi Mirzazadeh for helpful discussions.

6. REFERENCES

[1] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 379–386. ACM Press, 1990.

[2] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[3] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *Journal of ACM*, 39(4):745–763, 1992.

[4] A. de Mes and E. Rongen. Technical note - web service credentials. *IBM Systems Journal*, 42(3):532–537, 2003.

[5] R. El-Yaniv. Competitive solutions for online financial problems. *ACM Computing Survey*, 30(1):28–69, 1998.

[6] R. El-Yaniv, A. Fiat, R. Karp, and G. Turpin. Competitive analysis of financial games. In *33rd Annual Symposium on Foundations of Computer Science*, pages 327–333. IEEE Computer Society Press, 1992.

[7] R. El-Yaniv, A. Fiat, R. M. Karp, and G. Turpin. Optimal search and one-way trading online algorithms. *Algorithmica*, 30(1):101–139, 2001.

[8] S. Esmaeilsabzali and K. Larson. Service allocation for composite web services based on quality attributes. In *The First IEEE International Workshop on Service oriented Solutions for Cooperative Organizations*, pages 71–79. IEEE Computer Society, 2005.

[9] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[10] A. Fiat and G. J. Woeginger, editors. *Online Algorithms: the State of the Art. Proceedings of the Workshop on the Competitive Analysis of On-line Algorithms at Schloß Dagstuhl*, volume 1442 of *Lecture Notes in Computer Science*. Springer, May 1998.

[11] A. V. Goldberg, J. D. Hartline, and A. Wright. Competitive auctions and digital goods. In *Symposium on Discrete Algorithms*, pages 735–744, 2001.

[12] A. J. Kleywegt. *Dynamic and Stochastic Models with Freight Distribution Applications*. PhD dissertation, School of Industrial Engineering, Purdue University, 1996.

[13] A. J. Kleywegt and J. D. Papastavrou. The dynamic and stochastic knapsack problem. *Operational Research Society*, 46(1):17–35, 1998.

[14] R. Lavi and N. Nisan. Competitive analysis of incentive compatible on-line auctions. In *ACM Conference on Electronic Commerce*, pages 233–241, 2000.

[15] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[16] J. D. Papastavrou, S. Rajagopalan, and A. J. Kleywegt. The dynamic and stochastic knapsack problem with deadlines. *Management Science*, 42(12):1706–1718, 1996.

[17] P. Raghavan. A statistical adversary for on-line algorithms. In L. A. McGeoch and D. D. Sleator, editors, *On-line Algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 79–84. AMS/ACM, February 1991.

[18] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. *IBM J. Res. Dev.*, 38(6):683–707, 1994.