Noname manuscript No.
(will be inserted by the editor)

Optimizing Queries to Remote Resources

Albert Weichselbraun

Received: date / Accepted: date

Abstract One key property of the Semantic Web is its support for interoperability. Recent research in this area focuses on the integration of multiple data sources to facilitate tasks such as ontology learning, user query expansion and context recognition. The growing popularity of such machups and the rising number of Web APIs supporting links between heterogeneous data providers asks for intelligent methods to spare remote resources and minimize delays imposed by queries to external data sources.

This paper suggests a cost and utility model for optimizing such queries by leveraging optimal stopping theory from business economics: applications are modeled as decision makers that look for optimal answer sets. Queries to remote resources cause additional cost but retrieve valuable information which improves the estimation of the answer set's utility. Optimal stopping optimizes the trade-off between query cost and answer utility yielding optimal query strategies for remote resources. These strategies are compared to conventional approaches in an extensive evaluation based on real world response times taken from seven popular Web services.

Keywords information integration · adaptive decision-making · optimal stopping · opportunity cost model · Semantic Web · heterogeneous data sources

1 Introduction

Semantic Web applications provide, integrate and process data from heterogeneous sources, including third party services. Combining information from different locations and services is one of the key benefits of the Semantic Web.

Current approaches usually limit their queries to a number of particularly useful and popular services, but research on automated Web service discovery and matching (Gupta et al, 2007) and the Web of Linked Data (Bizer, 2009) focus on enhancing applications to

Albert Weichselbraun
Vienna University of Economics and Business
Augasse 2-6, 1090 Vienna, Austria
Tel: +43-1-31336-5229
Fax: +43-1-31336-787
E-mail: albert.weichselbraun@wu.ac.at

locate, interface, and use relevant resources in real time. Such implementations can issue queries that spawn vast collections of different data sources, providing even more enhanced information. Obviously, brute-force query strategies do not scale well and impose a considerable load on the affected services, even if only small pieces of information are requested (Weichselbraun, 2009). In fact, it is also in the application developer's own interest to reduce the number of queries to external services, since they add additional processing time.

The points made above demonstrate that applications which heavily depend on external resources need to ensure that they do not overuse the remote service, and that they need to carefully decide on whether the additional delay imposed by an external query is worth its benefit. Large-scale Semantic Web projects such as the IDIOM Media Watch on Climate Change (Hubmann-Haidvogel et al, 2009; Scharl et al, 2007), which process hundreds of thousands of pages a week, demonstrate the importance of these guidelines. Querying GeoNames to geo-tag *all* the documents mirrored by IDIOM's architecture would add *days* of processing time.

Therefore, the development of accurate algorithms, which consider high level goals such as user satisfaction and help to decide which resources to query for a particular document, plays a crucial role in providing scalable solutions. This paper illustrates the use of optimal stopping to improve the performance of Semantic Web applications by optimizing queries to third party resources. The applied search test stop algorithm (MacQueen, 1964) models applications as decision makers who pay the search cost (c_s) to retrieve answer sets and an indicator (x^0) estimating the answer's expected utility. Based on x^0 they might either (i) reject the answer set and search for a better option (paying c_s again), (ii) accept the answer set and gain the appropriate utility, or (iii) retrieve additional information from external services (x^1) to get a more accurate estimation of the answer's utility. The proposed method yields optimal query strategies by optimizing the trade-off between total query cost and the answer's utility.

Such approaches gain in importance as more and more Web 2.0 and Semantic Web applications depend on and integrate external resources. In the present paper I therefore identify and address a fundamental obstacle to the further growth of the Semantic Web and its usefulness. Without appropriate methods for optimizing queries to external resources, clients will only be able to query a small subset of all potential data sources within the time constraints set by the user, undermining the Semantic Web's fundamental principle of using distributed knowledge and resources.

Despite the obvious importance of optimal stopping for the Semantic Web community, this paper shows the application of a generic approach which might also be applied to related problems in other fields such as information retrieval (Grass and Zilberstein, 2000; Weichselbraun, 2009), autonomous computing (Kephart and Das, 2007; Tesauro et al, 2007), and Web Search (Ipeirotis et al, 2007; Kukulenz and Ntoulas, 2007).

1.1 Related Literature

Action- or rule-based policies often suffer when applied to complex and distributed systems, because the number of rules required to address all possible system states grows significantly with the number of systems interacting with each other. Applying utility models to decision problems reduces complexity (Zhang et al, 2008), allows a high-level specification of the expected outcome (Kephart and Das, 2007; Tesauro et al, 2007; Zhang et al, 2008), and enables the application of methods developed in economic theory and operations research. Utility

models are particularly useful for making complex decisions, since they address issues such as goal conflicts and the problem of considering trade-offs (Kephart and Das, 2007).

The main difficulty of applying these models is predicting the future utility (and costs) for a given allocation (Vengerov, 2007). Research in the fields of autonomous computing (Verma et al, 2008) suggests the use of workload reward functions to apply a utility-based model to disk scheduling. Verma et al (2008) show that this model performs significantly better than commonly used algorithms. Changing the reward model allows adapting their model to other objectives. Strunk et al (2008) investigate how the concept of utility can be applied to the provisioning of storage systems. They combine all relevant system metrics such as disk capacity, disk latency and availability into a utility score and provide a plug-in model with specific plug-in utility functions for conversion of these metrics into a utility value. Yeo and Buyya (2007) introduce a pricing function for cluster resources considering two components: a static base price for the resource's usage and a dynamic component depending on the current utilization of the resource. They evaluate their approach using user-centric performance metrics, which indicate the cluster's profitability and the utility achieved from satisfying job requests. Zhang et al (2008) use an economic model to allocate multiple system resources to workloads in a database management system. The authors trade these resources on an electronic market where consumer wealth is determined by the workload's importance. Tesauro et al (2007) elaborate on the use of reinforcement learning for automatic resource allocation. They address the issue of poor initial performance during online training by applying a fixed policy to the decision process until the newly learned strategy supersedes the hardwired decision logic.

Ipeirotis et al (2007) and Kukulenz and Ntoulas (2007) apply utility models to search queries. The resulting query strategies might lead to less accurate results than a brute force approach but optimize the balance between accuracy and cost.

The research reviewed above successfully applies utility models to optimization problems in the information sciences. These models are highly relevant especially in regard to the transformation of model parameters to cost and utility metrics, but they do not present generic approaches for addressing decision problems under imperfect information. In contrast, the work introduced in this paper, focuses on providing practitioners with a generic framework for applying optimal stopping to decision problems. Optimal stopping has its roots in economic theory and optimizes decisions based on the utility yielded by these decisions. Freeman (Freeman, 1983) provides a comprehensive overview of the optimal stopping problem and its extensions. Recent work in this area includes research done on the approximation of optimal stopping problems (Marcozzi, 2008) and optimal stopping in connection with multiple attributes (Lim et al, 2006). For an excellent introduction to optimal stopping and its applications please refer to Ferguson (2009).

Grass and Zilberstein (2000) present a generic model for value-driven information gathering (VDIG), which considers the cost of information in query planning. VDIG focuses on the query selection problem in terms of the trade-off between response time and the value of the retrieved information. In contrast, approaches that address only the coverage problem put their emphasis solely on maximizing precision and recall. The main disadvantage of their model is the lack of support for a test step which allows the retrieval of additional information on an option's value.

1.2 Contributions and Paper outline

This paper applies MacQueen's search test stop (STS) model (Section 2) to query optimization problems (MacQueen, 1964; Weichselbraun, 2009). The work introduces a cost and utility model for the STS algorithm proposed in Weichselbraun (2009), provides practitioners with the extensions necessary to use STS in real world settings, and performs an extensive statistical evaluation of the approach.

The main contributions of this paper are (i) developing an opportunity cost-based model for the STS algorithm which extends its usefulness to utility driven information gathering, (ii) introducing the concept of slot costs which is a prerequisite for the useful application of STS to information retrieval tasks, (iii) demonstrating the application of this approach to generate contextualized information spaces, and (iv) conducting a comprehensive experiment describing how different slot cost, search and test times, utility functions, and predictors influence the method's performance to provide practitioners with background on the algorithms' behavior.

The results of this paper can be situated within the field of Artificial Intelligence (AI) research, integrating techniques from decision theory to address problems of agent decision making (Horvitz et al, 1988).

The article is organized as follows: Section 2 introduces the STS model, a classical method of optimal stopping, as an approach for coping with the challenges outlined above. Section 3 presents an abstract model for applying cost and utility metrics to decision processes which optimize the use of remote resources, discusses approaches for estimating model parameters, and presents solutions which perform well with the STS model. Section 4 provides a comprehensive evaluation of the STS approach's performance, presents comparisons to other decision models, and discusses the influence of model parameters. The paper closes with an outlook and draws conclusions in Section 5.

2 The Search Test Stop Model

MacQueen (1964) describes the idea of the STS model as follows: a decision maker searches through a population of possible actions, sequentially discovering sets of actions (S_{A_i}), and paying the search cost (c_{s_i}) each time a new set of actions (e.g. "accept $document_i$") is revealed.

For each possible action set (S_{A_i}) exists a corresponding sequence of triples (x_i^0, x_i^1, u_i) with a known joint distribution $h(x^0, x^1, u)$ indicating its value. The decision maker cannot directly measure the utility (u_i) of an action set, but only its indicators x_i^0 and x_i^1.

After each search step that yields an action set (S_{A_i}) the decision maker obtains the variable (x_i^0), which provides an estimation of the action set's utility (u_i). Based on this information, the decision maker may (i) drop the current action set and continue looking for another set of possible actions (paying search cost $c_{s_{i+1}}$), (iii) accept the current set of answers (and gain the utility u_i), or (iii) test the retrieved set of actions to obtain x_i^1 - a better estimation of the action set's value - by paying the test cost (c_{t_i}) and based on this extended information continue with option i, or finish the process with option ii. The challenge is to combine these three options in a way that maximizes total utility ($u_i - \sum_{i=1}^{m} c_{s_i} - \sum_{i=1}^{n} c_{t_i}$).

Hartmann (1985) defines the following preconditions for the application of the STS model: (i) a common probability mass function $h(x^0, x^1, u)$ exists. (ii) The expected value of u ($z = E(U|x^0, x^1)$) exists and is finite. (iii) $F(z|x^0)$ is stochastically ordered over x^0 ($\frac{\partial}{\partial x^0} F(z|x^0) < 0$).

This paper deals with discrete service response time distributions and, therefore, uses an adopted STS methodology for handling discrete data, introduced by Hartmann (1985), as base for its python stslib module. The library has been validated using unit tests which leverage known data sets and their solutions such as the ones published in Hartmann's thesis (Hartmann, 1985).

Figure 1 visualizes the computation and decision process[1]. We start with a discrete common probability function $h(x^0,x^1,u)$. From h we derive (i) the expected utility $r = E(u|x^0)$ and the probability functions $f(r)$ and $F(r)$, (ii) the expected utility $z = E(u|x^0,x^1)$ and based on r the probability functions $f(z|r)$ and $F(z|r)$. Using these values the STS algo-

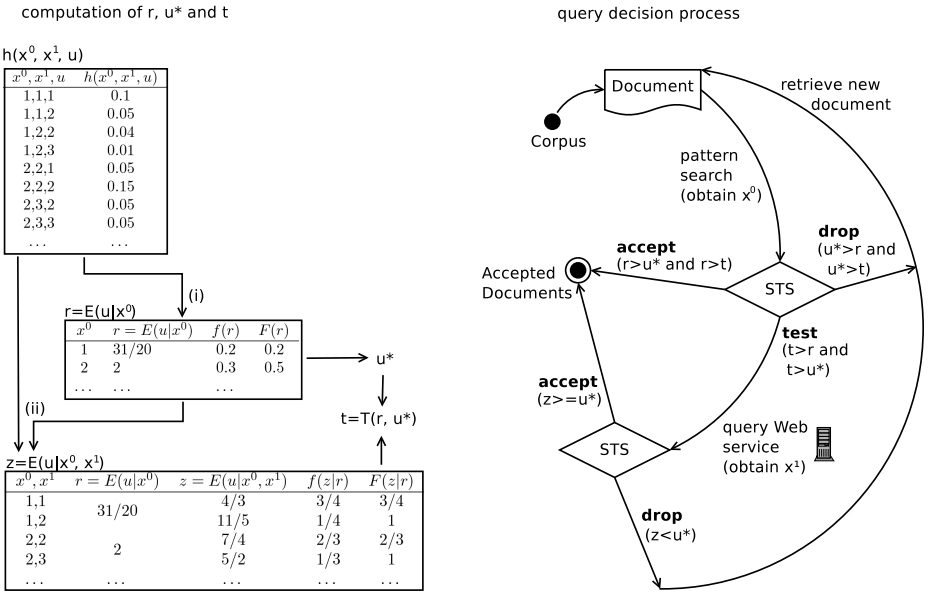

Fig. 1 The search test stop decision process.

rithm determines the discrete utilities for accepting the action set (r; Equation 1), dropping the action set and continuing the search (u^*; Equation 2), and testing the current action set (t; Equation 3) and proposes the action with the highest expected utility value.

$$r = E(u|x^0) \tag{1}$$

$$u^* = u^* F(r = u^*) + \sum_{r>u^*} r f(r) - c_s \tag{2}$$

$$t = T(r,u^*) = u^* F(z = u^*|r) + \sum_{z>u^*} z f(z|r) - c_t \tag{3}$$

MacQueen (1964) shows that this policy maximizes the decision maker's utility for the given problem class.

The right side of Figure 1 illustrates the decision making process. A portal side decides on the documents to include in their index based on the number of references to concepts relevant to the portal. Therefore, a pattern search algorithm extracts candidate references

[1] The tables in the figures are based on an example h function introduced by Hartmann (1985).

from the document which provide the indicator x^0. Based on this indicator the STS algorithm decides whether to accept, test or drop the document. If testing is necessary it retrieves additional information from a Web service to gain the indicator (x^1) and decides based on this additional information whether to accept or drop the document.

3 Method

Application of the STS model to Web services requires the availability of accurate cost and utility functions to transform application-specific parameters into comparable cost and utility scores. Figure 2 introduces an abstract model inspired by the method applied by Strunk et al (2008) to provisioning storage systems to address this issue. A domain model covers

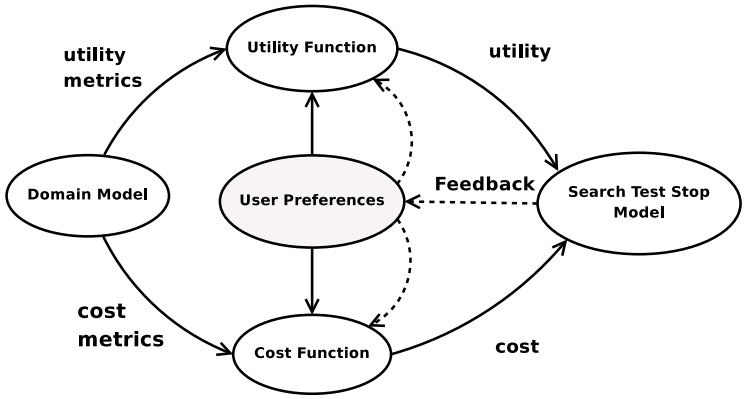

Fig. 2 Abstract model for assessing cost and utility metrics.

all relevant properties of the application and yields cost and utility metrics. Metric-specific functions that take into account the user's preferences translate these values into utility and price scores. New metrics are addressed by adding the appropriate translation functions to the model. The STS model processes tasks based on the given cost and utility, and considers feedback by adjusting the user's preference values or by modifying the translation function.

3.1 Pricing Resources

In the traditional STS model, costs refer to investment in terms of time and money spent on information gathering. When applying this idea to distributed applications, costs comprise all expenses such as CPU time, bandwidth and network resources that are necessary to search for or test certain answers.

Depending on the use case and the involved third-party services, different sets of domain specific metrics are relevant to pricing the required resources. The system architect identifies these metrics based on the domain model and specifies functions for their translation into utility scores. New metrics are included by adding new translation rules to the architecture. Common approaches toward pricing resources are the use of real-world business costs (Strunk et al, 2008), modeling search costs as exponential functions (Montgomery et al, 2004), and linear functions for cognitive costs (Shugan, 1980).

Yeo and Buyya (2007) define four essential requirements for pricing functions of cluster resources: the function should be (i) easy to configure by the owner of the resource, (ii) reflect the real resource usage, (iii) able to adjust a resource price according to its usage, and (iv) adapt to changes in supply and demand. Based on these criteria, Yeo and Buyya (2007) introduce a heuristic for pricing resources in which a resource's price consists of two components (Equation 4): a static base price for the resource's usage and a dynamic component (Equation 5) depending on the current utilization of the resource. This pricing model ensures that the framework reacts to spare resources by increasing the prices accordingly. The application and resource-specific factors α_j and β_j weight the impact of the base price and the price's dynamic component.

$$P_j = \alpha_j P_{Base,j} + \beta_j P_{Utilization,j} \tag{4}$$

$$P_{Utilization,j} = \left(\frac{Res_{used,j}}{Res_{max,j} - Res_{used,j}}\right) \cdot P_{Base,j} \tag{5}$$

This pricing strategy ensures that the system never runs out of a resources because the algorithm computes prohibitively high prices for spare resources. STS models guarded by this pricing strategy therefore never use *all* available resources, only the fraction affordable at reasonable prices, and consequently yield inferior results.

STS is best suited for situations where the test cost c_t is in the same order as the action set's utility: $O(c_t) = O(u(S_A))$. In settings with $O(c_t) \ll O(u(S_A))$, the test costs have no significant impact on the utility. If $O(c_t) \gg O(u(S_A))$, no testing will take place at all because the involved costs exceed any possible benefit of testing (Weichselbraun, 2009).

Based on these insights, we introduce an opportunity cost model that expresses search and test costs in terms of the average utility generated by using a unit of the resource $Res_{used,j}$, as outlined in Equation 6.

$$P_j = \frac{\overline{u}_j}{Res_{used,j}} \tag{6}$$

$\overline{u}_j$ indicates the average utility generated by using the amount $Res_{used,j}$ of resource j. Applying this model to query times yields Equation 7 for computing the per-request query cost c_r:

$$c_r = P_r = \frac{\overline{u}_r}{\overline{t}_r} \tag{7}$$

The common utility distribution function $h(x^0, x^1, u)$ yields the expected utility per request $\overline{u}_r$. From historical data, such as the one presented in Table 1, we derive $\overline{t}_r$. Another important issue to consider is that costs are not always known in advance. Web service query times, for instance, are influenced by many factors, which makes it very hard to predict this data. A forecaster service, therefore, provides estimates for future costs based on a weighted average of historical data points over a sliding window. More advanced versions of this service may apply time series analysis and domain-specific knowledge to yield better approximations.

Service	Protocol	response time				
		average ($\bar{t}_r$)	median ($\tilde{t}_r$)	minimum (t_r^{min})	maximum (t_r^{max})	variance ($\sigma_{t_r}^2$)
Amazon	REST	0.5	0.2	0.2	31.3	0.6
DBpedia	SPARQL	0.8	0.5	0.1	60.0	4.2
Del.icio.us	REST	0.6	0.4	0.1	24.3	0.5
GeoNames	REST	0.7	0.1	0.0	60.0	19.9
Google	Web	0.3	0.2	0.1	10.3	0.2
Swoogle	Web	4.1	1.6	0.2	60.0	98.4
Wikipedia	Web	0.5	0.2	0.1	60.0	1.3

Table 1 Response times of popular Web services in seconds (Weichselbraun, 2009).

3.2 Utility Function

Applying the STS model to economic problems yields cash deposits and payments. Transferring this idea to information science is a bit more subtle, because the utility is highly dependent on the application and its user's preferences. Even within one domain, the notion of an answer set's (S_A) value might not be clear. For instance, in a geo-spatial context the "correct" answer for a certain problem might be a particular mountain in Austria, but the geo-tagger might instead identify the surrounding region or at least the state in which it is located. Assigning concrete utility values to these alternatives is not always possible without detailed information regarding the application and user preferences. Approaches for evaluating the set's value might therefore vary from binary methods (full score for correct answers; no points for incomplete or incorrect answers) to complex ontology-based approaches that compute utility based on the grade of correctness and severity of deviations.

In general, a utility function that assumes linearly independent utility values might look like Equation 8.

$$u = u(S_A) = \sum_{a_i \in S_A} \lambda(a_i) f_{eval}(a_i) \tag{8}$$

The utility equals the sum of the utility gained by each answer a_i of the answer set S_A, which is evaluated using an evaluation function f_{eval}, and weighted with a factor $\lambda(a_i)$. To simplify the computation of the utility, we consider only correct answers as useful ($f_{eval}(a_i) = 1$ if a_i is correct and 0 otherwise) and apply the same weight ($\lambda(a_i) = const = 1$) to all answers (Vengerov, 2007).

The experiments performed on data provided by the *IDIOM Media Watch on Climate Change* (Section 4) will use a linear utility function, which increases with the number of geographic references ($|a_{geo}|$) and topics ($|a_{topic}|$) identified in a document. The application focuses on high precision rather than high recall, which led to the following quality criteria: every document included should be relevant to the domain (climate change), and contain at least one geographic reference for geo-spatial queries. Documents with a total of more than four annotations are more desirable than documents with a smaller number of annotations. Equation 9, which is used as a utility function for the evaluation presented in the next chapter, reflects these requirements.

$$u = \lambda \cdot (|a_{topic}| + |a_{geo}|) \quad \text{with} \tag{9}$$

$$\lambda = \begin{cases} 1 & if |a_{topic}| > 1 \text{ and } |a_{geo}| > 1 \text{ and } |a_{topic}| + |a_{geo}| > 4 \\ 0.5 & if |a_{topic}| > 1 \text{ and } |a_{geo}| > 1 \text{ and } |a_{topic}| + |a_{geo}| \leq 4 \\ 0 & otherwise. \end{cases} \tag{10}$$

Section 4 also includes experiments with other utility functions to assess their impact on the decision logic's performance.

3.3 Common utility mass function

The STS approach assumes that the decision maker does not benefit from accepting every possible action. As in the real world, where every potential employee could yield a positive utility to the company, every potential answer will contribute to the total utility of the answer set. Developing an optimal decision strategy, therefore, needs to consider the limited number of available positions and the costs created by accepting an answer, because otherwise an accept-everything approach would be the most successful one. In an information retrieval context every answer improves recall at the cost of precision, increasing the effort required to search for a particular document within the answer set. Considering that the answer with the highest recall would be the whole Web, the requirement to impose a penalty for every document in the corpus becomes evident. This idea is also supported by findings from Johnson and Payne (1985), which show that decision makers are prepared to trade recall for less effort required to check additional results. Therefore, we introduce the slot costs c_{sl} denoting the cost required for checking additional results. Subtracting this cost from the utility derived from Equation 9, ensures that only answers with a utility higher than the slot costs produce positive results. The evaluation section will analyze how slot costs corresponding to the top 30%, 50%, 70%, 90%, and 95% of the documents influence the method's performance.

A tagger usually identifies entities by scanning documents for string sequences containing gazetteer and topic entries. The number of identified entries ($|S_a| = x^0$) provides a first estimation of the value of the answer's utility. Applying entity identification techniques, advanced disambiguation, and leveraging external resources yields a refined indicator (x^1) of the entity set's utility. This assessment might still be flawed as the answer's *true* utility is only revealed after its acceptance. Based on the probability of a particular answer $a_i \in S_a$, yielding a certain utility u given the indicators x^0 and x^1, the joint probability h of a tuple x^0, x^1, u is determined.

If the statistical properties of the components computing x^0 and x^1 are unknown, classification accuracies found in the literature (Wang et al, 2008) or heuristics are helpful means to estimate h. In such environments the incorporation of user feedback and machine learning techniques helps refining h and therefore yields more accurate query strategies. If the joint probability function and the cost estimates are accurate, STS will return an optimal query strategy (MacQueen, 1964).

4 Evaluation

From the practitioners point of view there is a number of interesting questions regarding the application of STS: (i) will applications using STS perform better than simple brute-force approaches (always test; AT), or applications just ignoring the additional information available (search only; SO); (ii) will STS outperform the other methods regardless of the involved search and test cost; (iii) what's the influence of the request time predictions; and finally (iv) how does the chosen utility function change the algorithm's performance? The simulations below will address these questions and demonstrate how STS excels other decision logics in most real world settings.

The simulations use two weekly snapshots from the *IDIOM Media Watch on Climate Change* (Hubmann-Haidvogel et al, 2009; Scharl et al, 2007) database comprising approximately one million documents. The *IDIOM Media Watch on Climate Change's* media corpus draws upon a list of 156 news media sites from five English-speaking countries (Liu et al, 2005). The evaluation task comprises the creation of a contextualized information space (Hubmann-Haidvogel et al, 2009) for visualization in the climate change portal (www.ecoresearch.net/climate). Users can search and navigate documents within this repository along multiple dimensions and access context information through information landscapes, geographic maps, domain ontologies, and tag clouds (Hubmann-Haidvogel et al, 2009). Processing documents for the contextualized information space is costly and, therefore, only a subset of all available documents shall be selected within a given time limit of 14,400 seconds (4 hours).

Figure 3 outlines the evaluation setting. A Web crawler mirrors one million potential candidate documents for inclusion into the contextualized information space.

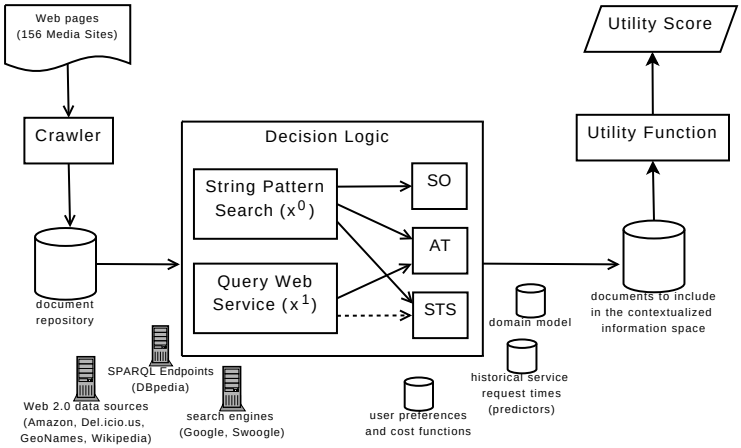

Fig. 3 Evaluation setting.

Three different decision logics (AT, SO and STS) decide on the subset of documents to include into the contextualized information space. The SO decision strategy makes decisions based on the first indicator of the document's utility (x^0), which is computed by applying a string pattern search. AT always requests the second indicator (x^1) prior to its decision. The test cost for x^1 depends on the used Web service (see Table 1). STS leverages the abstract cost model (Figure 1) introduced in Section 3 to combine: (i) user preferences and cost functions, (ii) the domain model, and (iii) historical service request time statistics to decide on whether to resort to testing or not. Applying a utility function (Equation 9) to the number of unique geographic and topic references in the document yields the document's utility. The evaluation summarizes the utility of all action sets (documents) collected by the decision logics and uses this metric to describe the decision logic's performance.

Applying the Kolmogorov-Smirnov test to this data shows that the results are not normally distributed. Therefore, the evaluation uses the Wilcoxon signed-rank test to determine whether differences between the utility values are statistically significant or not.

Two tables summarize the evaluation results: (i) Table 2, presenting the influence of different slot cost (Section 4.1) and search time levels (Section 4.2) on the decision logic's performance; and (ii) Table 3, which shows the impact of different utility functions (Section 4.3) and request time predictors (Section 4.4). The letter 'd' indicates the utility function introduced in Equation 9, 'l' a strictly linear utility function ($u = |a_{topic}| + |a_{geo}|$), 's' a squared utility function ($u = (|a_{topic}| + |a_{geo}|)^2$), and 'r' the use of the square root ($u = \sqrt{|a_{topic}| + |a_{geo}|}$).

The table's cells contain the sum of the utility of all action sets garnered by the particular decision logic. Grey cells indicate that the decision logic performed statistically significantly worse than STS with the given simulation parameters.

Figure 4 visualizes how the decision logics perform over time. The top graphs show the total utility of all action sets collected, the graph below the number of action sets gathered. The simulation on the left side uses the data set with the lowest test time average and variance (Google), the simulation on the right side draws on the Swoogle data set (highest test time average and variance).

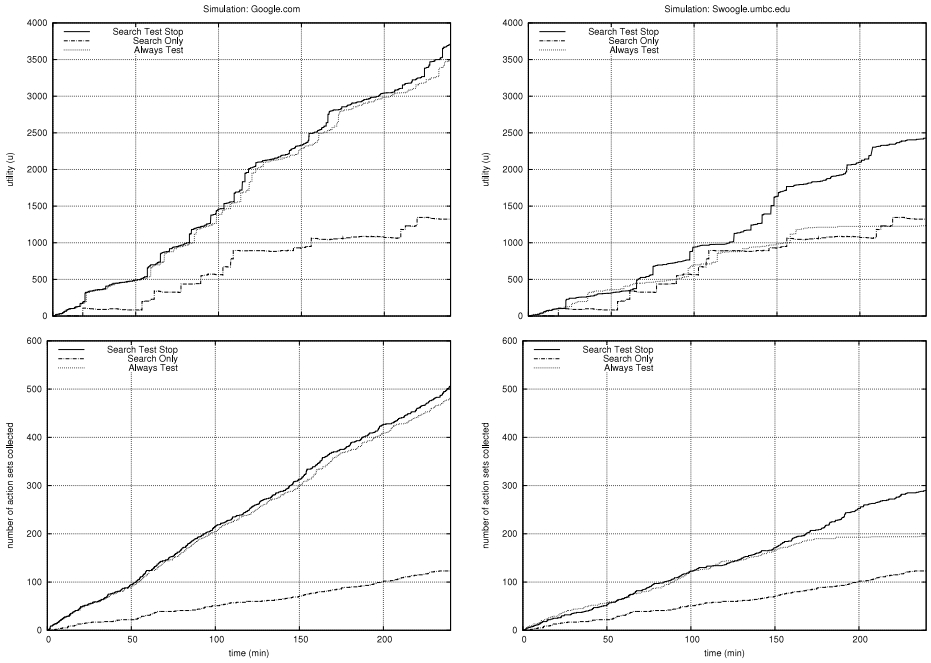

Fig. 4 Utility and number of collected action sets.

SO accepts only action sets with $E(u|x^0) > 0$ and therefore ignores sets where x^1 would be required to ensure a positive utility. In contrast, AT always determines x^1, which leads to a lower performance due to time lost for unnecessary testing. In settings with high and volatile test cost (right graphs) testing becomes much more expensive so that STS uses testing more cautiously. Due to the high test cost, the SO strategy performs comparably better. Wrong request time predictions (see Section 4.4) might lead to situations where STS performs even worse than AT or SO (see top right graph; time window approximately from minute 30 to 70).

		AT			SO			STS		
		$1/4 \cdot c_s$	$1 \cdot c_s$	$4 \cdot c_s$	$1/4 \cdot c_s$	$1 \cdot c_s$	$4 \cdot c_s$	$1/4 \cdot c_s$	$1 \cdot c_s$	$4 \cdot c_s$
Amazon	0	49963	27558	10396	201464	50891	12311	201472	50890	12310
($\bar{t}_r = 0.5$,	30	50224	29093	11357	204203	51389	13243	204203	51479	13315
$\sigma_{t_r} = 0.6$)	50	38320	21606	8905	149964	38903	10513	149963	38899	10512
	70	32132	18396	7379	116006	29634	7706	116004	29662	7899
	90	14091	8326	3178	19462	5105	1322	25707	10151	3498
	95	9766	5656	1837	20304	4387	755	22233	7784	1936
DBpedia	0	35764	22588	9754	201461	50892	12308	201472	50890	12310
($\bar{t}_r = 0.8$,	30	37289	24198	10071	204201	51474	13302	204203	51479	13315
$\sigma_{t_r} = 4.2$)	50	28129	18340	8364	149973	38890	10501	149963	38899	10512
	70	23657	15133	6950	116006	29652	7688	116004	29662	7721
	90	10249	7239	3037	19462	5105	1325	20695	8932	3244
	95	7617	4818	1703	20304	4387	755	21283	7162	1847
Del.icio.us	0	43865	25976	10280	201473	50893	12301	201472	50890	12310
($\bar{t}_r = 0.6$,	30	44958	27245	11060	204203	51459	13237	204203	51479	13315
$\sigma_{t_r} = 0.5$)	50	33633	20297	8834	149963	38916	10511	149963	38899	10512
	70	28773	16956	7295	116011	29655	7706	116004	29662	7868
	90	12461	8013	3153	19462	5096	1324	23867	9743	3425
	95	9067	5371	1748	20304	4387	755	22137	7640	1925
GeoNames	0	38088	24213	9995	201472	50770	12308	201472	50890	12310
($\bar{t}_r = 0.7$,	30	39528	25700	10514	204202	51473	13307	204203	51479	13315
$\sigma_{t_r} = 19.9$)	50	29806	19258	8599	149954	38897	10511	149963	38899	10512
	70	24929	15973	7150	116010	29768	7695	116004	29662	7782
	90	10857	7553	3060	19462	5105	1322	20962	9071	3266
	95	8212	5130	1713	20304	4387	755	21681	7010	1867
Google	0	73687	34531	11043	201470	50893	12330	201472	50890	12310
($\bar{t}_r = 0.3$,	30	73817	36033	12022	204197	51485	13316	204203	51479	13315
$\sigma_{t_r} = 0.2$)	50	57041	27385	9422	149971	38915	10511	149963	38899	10424
	70	46961	23138	7765	116011	29754	7705	116004	29651	8106
	90	21459	10040	3498	19462	5105	1324	32086	11301	3710
	95	16012	7236	1936	20304	4387	755	25309	8369	2022
Swoogle	0	8234	7173	4788	201457	50898	12340	201472	50890	12310
($\bar{t}_r = 4.1$,	30	8279	7183	4660	204203	51483	13310	204203	51479	13315
$\sigma_{t_r} = 98.4$)	50	7342	6100	3800	149963	38917	10513	149963	38899	10512
	70	6065	5394	3605	116006	29781	7716	116004	29662	7690
	90	2753	2283	1242	19462	5109	1324	19462	5105	2428
	95	1467	1358	897	20304	4387	755	20304	4984	1311
Wikipedia	0	55685	29937	10606	201461	50894	12319	201472	50890	12310
($\bar{t}_r = 0.5$,	30	56574	31514	11574	204202	51469	13315	204203	51479	13315
$\sigma_{t_r} = 1.3$)	50	43105	23209	9039	149972	38903	10514	149963	38899	10512
	70	35989	19719	7506	116007	29732	7703	116004	29662	7991
	90	16184	9111	3244	19462	5105	1328	27750	10538	3500
	95	11404	6226	1845	20304	4387	755	22820	7838	1944

Table 2 Query utility for different Web services, slot cost (0, 30, 50, 70, 90, 95), and search time levels ($1/4$, 1, 4).

4.1 Slot Cost

This evaluation verifies the hypothesis that *STS performs document evaluation tasks equally well or better than the other decision logics introduced*, regardless of the fraction of action sets yielding a positive utility.

Table 2 summarizes the evaluation results for all seven Web services using five different slot cost levels (c_{sl}), which are specified as the percentage of action sets yielding a negative utility at the given level (see Section 3.1).

The results of the evaluation emphasize the importance of the slot cost for obtaining meaningful results. Low slot costs lead to situations where STS and the SO decision logic deliver the same performance (Table 2; white cells in the SO column). This result was to be expected, considering that in such cases most (for $c_{sl} = 0$ all) choices yield a positive utility so that there is no point in testing action sets. Consequently, the AT decision logic performs badly in such situations.

		AT	SO	STS	STS					
		5	5	5	1	10	50	100	1000	c
Amazon	d	8326	5105	10134	9608	10286	10299	10009	10067	10151
($\bar{t}_r = 0.5$,	l	8156	4526	10701	9935	10802	10886	10740	10700	10568
$\sigma_{t_r} = 0.6$)	s	8017	5290	10776	10113	10954	11077	10769	10696	10762
	r	7893	5342	10150	9775	10262	10286	10241	10122	10075
DBpedia	d	7239	5105	9199	8936	9224	9267	9199	9200	8932
($\bar{t}_r = 0.8$,	l	6792	4526	9683	9321	9601	9482	9415	9295	9132
$\sigma_{t_r} = 4.2$)	s	6431	5290	9598	9306	9568	9497	9411	9292	9571
	r	6398	5342	9341	9265	9383	9471	9324	9301	9219
Del.icio.us	d	8013	5105	9708	9641	9801	9820	9829	9717	9743
($\bar{t}_r = 0.6$,	l	7757	4526	10218	10096	10287	10183	10234	10097	10239
$\sigma_{t_r} = 0.5$)	s	7565	5290	10460	10252	10558	10245	10326	10091	10097
	r	7261	5342	9758	9667	9756	9814	9785	9721	9668
GeoNames	d	7553	5105	10189	10076	9835	9474	9350	9044	9071
($\bar{t}_r = 0.7$,	l	7150	4526	10855	10851	10663	9759	9604	9409	9442
$\sigma_{t_r} = 19.9$)	s	6870	5290	10775	10698	10549	9853	9401	9630	9603
	r	6952	5342	10493	10326	10074	9673	9320	9361	9332
Google	d	10040	5105	11292	11205	11293	11254	11321	11130	11301
($\bar{t}_r = 0.3$,	l	10038	4526	11919	11663	11891	11879	11881	11895	11901
$\sigma_{t_r} = 0.2$)	s	9926	5290	12434	12196	12447	12427	12414	12398	12407
	r	9508	5342	11463	11166	11463	11412	11462	11355	11346
Swoogle	d	2283	5105	6082	5925	6091	6136	6068	5415	5105
($\bar{t}_r = 4.1$,	l	1402	4526	6515	6466	6689	6525	6366	5271	4526
$\sigma_{t_r} = 98.4$)	s	1779	5290	7260	6250	7321	7088	6946	6305	5290
	r	1623	5342	6281	5795	6214	6039	6297	5471	5342
Wikipedia	d	9111	5105	10500	10055	10484	10470	10357	10476	10538
($\bar{t}_r = 0.5$,	l	8662	4526	11039	10461	10999	11034	11088	11092	11155
$\sigma_{t_r} = 1.3$)	s	8566	5290	11206	10737	11252	11179	11174	11232	11245
	r	8494	5342	10509	10227	10421	10529	10533	10463	10431

Table 3 Comparing the per decision logic query utility for different utility functions (d, l, s, r) and predictors (moving average of 5, 1, 10, 50, 100, 1000 and constant (c) based on the average request times).

As the cost of making suboptimal decisions rises (due to higher slot cost), the STS algorithm performs significantly better than the other decision logics (Table 2; grey cells). The AT strategy outperforms the SO decision logic as testing becomes more and more beneficial. In conclusion, SO performs quite well in environments with low slot cost, while STS delivers optimal results in all tested settings.

4.2 Search and Test Times

The evaluation tasks carried out verify the hypothesis that *STS performs equally or better than the other decision logics, regardless of the search and test time levels*. Table 2 demonstrates how different search and test times influence the algorithms performance. In the table's heading, search time multipliers of a quarter, one, and four indicate the search time level, on the table's left side the Web services used are listed with the respective slot costs and average test times (see also Table 1).

All three decision logics yield higher utility for lower search times, because low search times allow them to obtain more answer sets in the same time period. AT and STS benefit from lower test times as well, as they make testing less expensive.

Lower search times (1/4) lead to more simulations where SO and STS perform equally well (Table 2; white cells in the SO column), because testing becomes less favorable as the test-cost to search-cost ratio rises. A simulation run with search costs of a quarter based on the Swoogle dataset means that testing is approximately 24 times more expensive than searching. Therefore STS never resorted to testing in this setting, which led to a behavior (and utility score) similar to SO.

4.3 Utility function

Table 3 compares results that where obtained using four different utility functions. Despite the fact that the absolute utility values differ statistically significantly, all utility functions yielded the same overall results: STS outperforms the AT decision logic in all experiments and yields the same or a higher utility in comparison with the SO decision logic. This result is also suggested in the literature, which reports the successful application of various approaches for determining a decision's utility such as electronic markets (Zhang et al, 2008), marginal utility models (Vengerov, 2007), and linear utility models (Das et al, 2006).

4.4 Predictors

Predicting request times accurately is important because otherwise decisions are based on wrong assumptions. The following evaluation is carried out to investigate the impact of the predictor choice (predictions based on a moving average of *1, 5, 10, 50, 100, 1000* and constant predictions based on the mean of the request time) on the STS algorithm's performance[2] for the presented use case (see Table 3).

A statistical analysis of Table 3 yields the following insights: results obtained from highly volatile Web services ($\sigma_{t_r} \geq 10 \cdot \bar{t}_r$: GeoNames, Swoogle) and Web services with a moderate volatility ($\sigma_{t_r} < 10 \cdot \bar{t}_r$: Amazon, DBpedia, Del.icio.us, Google, Wikipedia) differ considerably. For GeoNames and Swoogle, the performance differences due to different predictors are in 300 of 336 cases (89%) statistically significant according to the Wilcoxon signed-rank test with a maximum average performance difference of approximately 12.3%. In contrast, the predictor choice only leads to a performance difference of 2.8% for Web services with moderate volatility and only 354 of 840 computations (42%) show significant deviations. The results also indicate that the moving average of the last ten results performed best for the given use case. In highly volatile environments, very large moving averages and constant request time estimates perform badly, because they are too coarse to accurately estimate the real cost involved in the decision.

Despite the relatively small deviations in this use case, the predictor's influence on the simulation cannot be underestimated, because inaccurate predictions lead to decisions based on wrong input data. The visualization of the simulation using the Swoogle dataset (Figure 4) demonstrates the impact of such a wrong decision: the AT decision logic outperforms STS over a timespan of approximately 40 minutes (from minute 30 to 70) during the 240 min simulation run. In contrast, STS performs very well over the whole simulation period when applied to the Google dataset, where request time predictions are much more accurate due to the smaller variance of search times.

4.5 Final Remarks

Table 4 contrasts the number of Web service calls, the Web service query time and the number of accepted documents for the document selection task from a corpus of 850 documents. In these simulations the AT and SO decision logic always accept the same number of documents, because their decision strategy is independent from the Web service's query cost.

[2] SO and AT do not consider costs.

Service	DL	Web service queries		Web service query time		Documents accepted	
		no.	%	(sec)	%	no.	%
Amazon	AT	850	100	611	100	80	100
	STS	381	45	219	36	70	88
	SO	0	0	0	0	18	22
DBpedia	AT	850	100	498	100	80	100
	STS	338	40	179	36	72	90
	SO	0	0	0	0	18	22
Del.icio.us	AT	850	100	468	100	80	100
	STS	352	41	181	39	69	86
	SO	0	0	0	0	18	22
GeoNames	AT	850	100	552	100	80	100
	STS	462	54	225	41	76	95
	SO	0	0	0	0	18	22
Google	AT	850	100	241	100	80	100
	STS	452	53	125	52	75	94
	SO	0	0	0	0	18	22
Swoogle	AT	850	100	1972	100	80	100
	STS	90	11	161	8	25	31
	SO	0	0	0	0	18	22
Wikipedia	AT	850	100	291	100	80	100
	STS	446	52	152	52	76	95
	SO	0	0	0	0	18	22

Table 4 Web service queries, query time and number of documents accepted.

AT yields the highest recall (100%) at the cost of long query times. SO does not query external resources but yields the lowest recall. Table 4 identifies the following major benefits of applying STS to queries:

1. *handling of trade-offs:* the simulations using STS retrieve around 90% of the documents identified by the AT algorithm at only 36-52% of AT's query time. Swoogle is the only notable exception due to its prohibitively high query cost ($O(c_t) \gg O(u(S_A))$) which yields in less queries and therefore a much lower document recall. The user's preferences control how STS handles the trade-off between recall and query time.
2. *intelligent querying:* the percentage of Web service queries is always higher or at least equal to the percentage of the query time because STS only queries remote resources *if* the expected gain outweighs the costs in terms of additional response time. Therefore, queries are more likely to occur during periods of short request times and the algorithm avoids querying busy services.

The results from the simulations performed in this section suggest that STS adjusts well to Web services with reasonable query cost ($O(c_t) = O(u(S_A))$) and provides significantly shorter query times at moderately lower recall.

5 Outlook and Conclusions

This paper presents an approach for optimizing access to remote resources. Optimizing the client's resource access strategy yields higher query performance and saves remote resources by preventing unnecessary queries.

STS performed equally well or better than the other decision logics (Section 4). It maximizes answer quality and quantity based on the *current* search and test cost adjusting queries

to the responsiveness of the service and the user's preferences. These preferences formalize the trade-off between quality and quantity by specifying a transformation function between search cost and search times. STS therefore optimizes the agent's behavior in terms of user utility. This does *not* necessarily mean that STS minimizes resource usage. Instead, it dynamically adjusts the resource utilization based on the cost of searching (c_s) and testing (c_t), providing the user with optimal results in terms of accuracy *and* response times. Applying this approach to the IDIOM Media Watch on Climate Change yielded a far better performance than the previously used brute-force (AT) approach. Due to the intelligent testing performed by STS, statistically significantly more documents where processed than with the AT approach, yielding a higher total utility than the other decision logics. The overhead from applying the python-based libsts prototype is currently about one second per 23 decisions on a 3 GHz Intel Pentium D CPU. Moving time-critical computations such as matrix operations to an external C library will considerably improve the library's throughput.

The presented approach also addresses the need for a generic method with support for an own testing step to optimize access to remote resources, as outlined in the literature section. To practitioners this paper provides important information such as usable utility functions and a cost model for implementing STS, and identifies common pitfalls (e.g., low slot costs, high test-cost to search-cost ratios). Finally, based on the evaluation performed in Section 4 it identifies the following four major factors influencing the performance of STS implementations: (i) The *percentage of entries yielding a negative utility*: if all choices do yield a positive utility, an accept-everything strategy would be the optimal choice and there would be no point in applying a decision logic for selecting answers. Therefore, this paper introduced the notion of slot costs (Section 3.1) and Section 4.1 elaborated on their influence on the algorithm's performance. (ii) The *test-cost to search-cost ratio*: Section 4.2 describes how high test cost to search cost ratios make testing more expensive and therefore decrease STS's effectiveness when compared to SO. (iii) The *order of the test cost compared to the order of the utility* of an action set: as already outlined by Weichselbraun (2009), STS performs best if costs and utility are in the same order ($O(c) = O(u(S_A))$). Section 3.1, therefore, suggested the use of opportunity costs (Equation 6) for pricing resources, which ensures that resource costs and answer set utility are of the same order. (iv) The *request time predictions*, which have been covered in Section 4.4.

Hartmann (1985) has shown that extending STS to n-levels of testing is a straight forward task. Nevertheless, determining the optimal sequence of tests is still an interesting research avenue. Future versions of the algorithm will also optimize the number and order of external services they query.

Developing utility functions considering partially correct answers based on the user's preferences will allow more fine-grained control over the process's performance yielding highly accurate querying strategies and therefore better results. We will also transfer these techniques and results to more complex use cases integrating multiple data sources such as semi-automatic ontology extension (Liu et al, 2005). Extensions considering planning and high-latency resources such as user feedback garnered from online games (Siorpaes and Hepp, 2008) provide another area of interesting research challenges.

Acknowledgment

The project results have been developed in the RAVEN (Relation Analysis and Visualization) project funded by the Austrian Ministry of Transport, Innovation and Technology and

the Austrian Research Promotion Agency. The author would like to thank Wolfgang Janko for his valuable suggestions during the preparation of this article.

References

Bizer C (2009) The emerging web of linked data. IEEE Intelligent Systems 24(5):87–92, DOI 10.1109/MIS.2009.102

Das R, Whalley I, Kephart JO (2006) Utility-based collaboration among autonomous agents for resource allocation in data centers. In: AAMAS '06: Proceedings of the fifth international joint conference on autonomous agents and multiagent systems, ACM, New York, NY, USA, pp 1572–1579, DOI 10.1145/1160633.1160935

Ferguson TS (2009) Optimal Stopping and Applications. Mathematics Department, University of California, URL http://www.math.ucla.edu/˜tom/Stopping/Contents.html, online publication, last visited: 2 June 2010

Freeman PR (1983) The secretary problem and its extensions: A review. International Statistical Review 51(2):189–206

Grass J, Zilberstein S (2000) A value-driven system for autonomous information gathering. Journal of Intelligent Information Systems 14(1):5–27, DOI 10.1023/A:1008718418982

Gupta C, Bhowmik R, Head MR, Govindaraju M, Meng W (2007) Improving performance of web services query matchmaking with automated knowledge acquisition. In: Web Intelligence, IEEE Computer Society, pp 559–563

Hartmann J (1985) Wirtschaftliche Alternativensuche mit Informationsbeschaffung unter Unsicherheit. PhD thesis, Universität Fridericiana Karlsruhe

Horvitz EJ, Breese JS, Henrion M (1988) Decision theory in expert systems and artificial intelligence. International Journal of Approximate Reasoning 2:247–302, DOI 10.1016/0888-613X(88)90120-X

Hubmann-Haidvogel A, Scharl A, Weichselbraun A (2009) Multiple coordinated views for searching and navigating web content repositories. Information Sciences 179(12):1813–1821, DOI 10.1016/j.ins.2009.01.030

Ipeirotis PG, Agichtein E, Jain P, Gravano L (2007) Towards a query optimizer for text-centric tasks. ACM Transactions on Database Systems 32(4):21, DOI 10.1145/1292609.1292611

Johnson EJ, Payne JW (1985) Effort and accuracy in choice. Management Science 31(4):395–414, DOI 10.1287/mnsc.31.4.395

Kephart JO, Das R (2007) Achieving self-management via utility functions. IEEE Internet Computing 11(1):40–48, DOI 10.1109/MIC.2007.2

Kukulenz D, Ntoulas A (2007) Answering bounded continuous search queries in the world wide web. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, ACM, New York, NY, USA, pp 551–560, DOI 10.1145/1242572.1242647

Lim C, Bearden JN, Smith JC (2006) Sequential search with multiattribute options. Decision Analysis 3(1):3–15, DOI 10.1287/deca.1050.0044

Liu W, Weichselbraun A, Scharl A, Chang E (2005) Semi-automatic ontology extension using spreading activation. Journal of Universal Knowledge Management 0(1):50–58, URL http://www.jukm.org/jukm_0_1/semi_automatic_ontology_extension

MacQueen J (1964) Optimal policies for a class of search and evaluation problems. Management Science 10(4):746–759

Marcozzi MD (2008) On the approximation of infinite dimensional optimal stopping problems with application to mathematical finance. Journal of Scientific Computing 34(3):287–307, DOI 10.1007/s10915-007-9168-2

Montgomery AL, Hosanagar K, Krishnan R, Clay KB (2004) Designing a better shopbot. Management Science 50(2):189–206, DOI 10.1287/mnsc.1030.0151

Scharl A, Weichselbraun A, Liu W (2007) Tracking and modelling information diffusion across interactive online media. International Journal of Metadata, Semantics and Ontologies 2(2):136–145, DOI 10.1504/IJMSO.2007.016807

Shugan SM (1980) The cost of thinking. The Journal of Consumer Research 7(2):99–111

Siorpaes K, Hepp M (2008) Games with a purpose for the semantic web. IEEE Intelligent Systems & their Applications 23:50–60, DOI 10.1109/MIS.2008.45

Strunk JD, Thereska E, Faloutsos C, Ganger GR (2008) Using utility to provision storage systems. In: FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies, USENIX Association, Berkeley, CA, USA, pp 1–16

Tesauro G, Jong NK, Das R, Bennani MN (2007) On the use of hybrid reinforcement learning for autonomic resource allocation. Cluster Computing 10(3):287–299, DOI 10.1007/s10586-007-0035-6

Vengerov D (2007) A reinforcement learning approach to dynamic resource allocation. Engineering Applications of Artificial Intelligence 20(3):383–390, DOI 10.1016/j.engappai.2006.06.019

Verma A, Jain R, Ghosal S (2008) A utility-based unified disk scheduling framework for shared mixed-media services. ACM Transactions on Storage 3(4):1–30, DOI 10.1145/1326542.1326546

Wang YJ, Sanderson R, Coenen F, Leng P (2008) Document-base extraction for single-label text classification. In: Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery (DaWaK-2008), Springer-Verlag, Berlin, Heidelberg, pp 357–367, DOI 10.1007/978-3-540-85836-2_34

Weichselbraun A (2009) Applying optimal stopping for optimizing queries to external semantic web resources. In: Cordeiro J, Shishkov B, Ranchordas A, Helfert M (eds) Software and Data Technologies, Communications in Computer and Information Science, vol 47, Springer, Berlin-Heidelberg, pp 105–118, DOI 10.1007/978-3-642-05201-9

Yeo CS, Buyya R (2007) Pricing for utility-driven resource management and allocation in clusters. International Journal of High Performance Computing Applications 21(4):405–418, DOI 10.1177/1094342007083776

Zhang M, Martin P, Powley W, Bird P (2008) Using economic models to allocate resources in database management systems. In: CASCON '08: Proceedings of the 2008 conference of the center for advanced studies on collaborative research, ACM, New York, NY, USA, pp 248–259, DOI 10.1145/1463788.1463814