Optimising Performance of Competing Search Engines in Heterogeneous Web Environments*

Rinat Khoussainov and Nicholas Kushmerick

Department of Computer Science, University College Dublin
Belfield, Dublin 4, Ireland
{rinat,nick}@ucd.ie

Abstract. Distributed heterogeneous search environments are an emerging phenomenon in Web search, in which topic-specific search engines provide search services, and metasearchers distribute user's queries to only the most suitable search engines. Previous research has explored the performance of such environments from the user's perspective (e.g., improved quality of search results). We focus instead on performance from the search service provider's point of view (e.g, income from queries processed vs. resources used to answer them). We analyse a scenario in which individual search engines compete for queries by choosing which documents to index. We propose the COUGAR algorithm that specialised search engines can use to decide which documents to index on each particular topic. COUGAR is based on a game-theoretic analysis of heterogeneous search environments, and uses reinforcement learning techniques to exploit the sub-optimal behaviour of its competitors.

1 Introduction

Heterogeneous search environments are a recent phenomenon in Web search. They can be viewed as a federation of *independently controlled* metasearchers and many specialised search engines. Specialised search engines provide focused search services in a specific domain (e.g. a particular topic). Metasearchers help to process user queries effectively and efficiently by distributing them only to the most suitable search engines for each query. Compared to the traditional search engines like Google or AltaVista, specialised search engines (together) provide access to arguably much larger volumes of high-quality information resources, frequently called the "deep" or "invisible" Web.

Previous work has mainly explored the performance of such heterogeneous search environments from the user's perspective (e.g., improved quality of search results). Examples include algorithms for search engine selection and result merging [1]. On the other hand, a provider of search services is more interested in the income from queries processed vs. resources used to answer them. To the best of our knowledge, little attention has been paid to performance optimisation of search engines from the service provider's point of view.

An important factor that affects performance of a specialised search engine in a heterogeneous search environment is *competition* with other independently controlled

* This research was supported by grant SFI/01/F.1/C015 from Science Foundation Ireland, and grant N00014-03-1-0274 from the US Office of Naval Research.

N. Lavrač et al. (Eds.): ECML 2003, LNAI 2837, pp. 217–228, 2003.

search engines. When there are many search engines available, users want to send their queries to the engine(s) that would provide the best possible results. Multiple search providers in a heterogeneous search environment can be viewed as participants in a search services market competing for user queries.

We examine the problem of performance-maximising behaviour for non-cooperative specialised search engines in heterogeneous search environments. We analyse a scenario in which individual search engines compete for queries by choosing to index documents for which they think users are likely to query. Our goal is to propose a method that specialised search engines can use to select on which topic(s) to specialise and how many documents to index on that topic to maximise their performance.

While the search engines in a heterogeneous search environment are independent in terms of selecting their content, they are not independent in terms of the performance achieved. Changes to parameters of one search engine affect the queries received by its competitors and, vice versa, actions of the competing engines influence the queries received by the given search engine. Thus, the utility of any local content change depends on the state and actions of other search engines in the system. The uncertainty about actions of competitors as well as the potentially large number of competing engines make our optimisation problem difficult. We show that naive strategies (e.g, blindly indexing lots of popular documents) are ineffective, because a rational search engine's indexing decisions should depend on the (unknown) decisions of its opponents.

Our main contributions are as follows:

- We formalise the issues related to optimal behaviour in competitive heterogeneous search environments and propose a model for performance of a specialised search engine in such environments.
- We provide game-theoretic analysis of a simplified version of the problem and motivate the use of the concept of *"bounded rationality"* [2]. Bounded rationality assumes that decision makers act sub-optimally in the game-theoretic sense due to incomplete information about the environment and/or limited resources.
- We propose a reinforcement learning procedure for topic selection, called COUGAR, which allows a specialised search engine to exploit sub-optimal behaviour of its competitors to improve own performance.

An evaluation of COUGAR in a simulation, driven by real user queries submitted to over 47 existing search engines, demonstrates the feasibility of our approach.

2 Problem Formulation

2.1 Search Engine Performance

We adopt an economic view on search engine performance from the service provider's point of view. Performance is a difference between the value of the search service provided (income) and the cost of the resources used to provide the service. The value of a search service is a function of the user queries processed. The cost structure in an actual search engine may be quite complicated involving many categories, such as storage, crawling, indexing, and searching. In our simplified version of the problem, we only

take into account the cost of resources involved in processing search queries. (Note that we also obtained similar results for a more elaborated cost model that takes into account the cost of document crawling, storage, and maintenance [3].) Under these assumptions, we can use the following formula for search engine performance: $P = \alpha Q - \beta QD$, where Q is the number of queries processed in a given time interval, D is the number of documents in the search engine index, α and β are constants.

αQ represents the service value: if the price of processing one search request for a user is α, then αQ would be the total income from service provisioning. βQD represents the cost of processing search requests. If x amount of resources is sufficient to process Q queries, then we would need $2x$ to process twice as many queries in the same time. Similarly, if x resources is enough to search in D documents for each query, then we would need $2x$ to search twice as many documents in the same time. Thus, the amount of resources (and, hence, the cost) is proportional to both Q and D, and so can be expressed as βQD, where β is a constant reflecting the resource costs. An examination of the architecture of the FAST search engine (`www.alltheweb.com`) shows that our cost function is not that far from reality [4].

We assume that all search engines in our system use the same α and β constants when calculating their performance. Having the same β reasonably assumes that the cost of resources (per "unit") is the same for all search engines. Having the same α assumes, perhaps unrealistically, that the search engines choose to charge users the same amount per query. We leave to future work, however, optimisation of search engine performance in environments where engines may have different service pricing. With no service price differentiation, selection of search engines by the metasearcher depends on what documents the engines index. Therefore, the goal of each search engine would be to select the index content in a way that maximises its performance.

2.2 Metasearch Model

We assume a very generic model of how any reasonable metasearch system should behave. This will allow us to abstract from implementation details of particular metasearch algorithms (presuming that they approximate our generic model). It is reasonable to assume that users would like to send queries to the search engine(s) that contain the most relevant documents to the query, and the more of them, the better.

The ultimate goal of the metasearcher is to select for each user query to which search engines it should be forwarded to maximise the results relevance, while minimising the number of engines involved. The existing research in metasearch (e.g. [1]), however, does not go much further than simply ranking search engines. Since it is unclear how many top ranked search engines should be queried (and how many results requested), we assume that the query is always forwarded to the *highest ranked* search engine. In case several search engines have the same top rank, one is selected at random.

The ranking of search engines is performed based on the expected number of relevant documents that are indexed by each engine. Engine i that indexes the largest expected number of documents NR_i^q relevant to query q will have the highest rank.

We apply a probabilistic information retrieval approach to assessing relevance of documents [5]. For each document d, there is a probability $\Pr(rel|q,d)$ that this document will be considered by the user as relevant to query q. In this case, $NR_i^q =$

$\sum_{d \in i} \Pr(rel|q,d)$, where by $d \in i$ we mean the set of documents indexed by engine i. Obviously, the metasearcher does not know the exact content of search engines, so it tries to estimate NR_i^q from the corresponding content summaries.

If $\Pr(rel|q_1,d) = \Pr(rel|q_2,d), \forall d$ then queries q_1 and q_2 will look the same from both metasearcher's and search engine's points of view, even though the queries may differ lexically. All engines will have the same rankings for q_1 and q_2, and the queries will get forwarded to the same search engine. Therefore, all queries can be partitioned into equivalence classes with identical $\Pr(rel|q,d)$ functions. We call such classes *topics*. We assume in this paper that there is a fixed finite set of topics and queries can be assigned to topics. Of course, this it not feasible in reality. One way to approximate topics in practice would be to cluster user queries received in the past and then assign new queries to the nearest clusters.

2.3 Engine Selection for "Ideal" Crawlers

Let us assume that users only issue queries on a single topic. We will see later how this can be extended to multiple topics. It follows from Section 2.2, that to receive queries, a search engine needs to be the highest ranked one for this topic. It means that given an index size D, a search engine would like to have a document index with the largest possible NR_i. This can be achieved, if the engine indexes the D most relevant documents on the topic.

Population of search engines is performed by topic-specific (focused) Web crawlers [6]. Since it is very difficult to model a Web crawler, we assume that all search engines have "ideal" Web crawlers which for a given D can find the D most relevant documents on a given topic. Under this assumption, two search engines indexing the same number of documents $D_1 = D_2$ will have $NR_1 = NR_2$. Similarly, if $D_1 < D_2$, then $NR_1 < NR_2$ (assuming that all documents have $\Pr(rel|d) > 0$). Therefore, the metasearcher will forward user queries to the engine(s) containing the largest number of documents.

This model can be extended to multiple topics, if we assume that each document can only be relevant to a single topic. In this case, the state of a search engine can be represented by the number of documents D_i^t that engine i indexes for each topic t. A query on topic t will be forwarded to the engine i with the largest D_i^t.

2.4 Decision Making Process

The decision making process proceeds in series of fixed-length time intervals. For each time interval, search engines simultaneously and independently decide on how many documents to index on each topic. They also allocate the appropriate resources according to their expectations for the number of queries that users will submit during the interval. Since search engines cannot have unlimited crawling resources, we presume that they can only do incremental adjustments to their index contents that require the same time for all engines. The user queries submitted during the time interval are allocated to the search engines based on their index parameters (D_i^t) as described above. The whole process repeats in the next time interval.

Let $\hat{Q}_i^t$ be the number of queries on topic t that, according to expectations of search engine i, the users will submit. Then the total number of queries expected by engine

i can be calculated as $\hat{Q}_i = \sum_{t:D_i^t>0} \hat{Q}_i^t$. We assume that engines always allocate resources for the full amount of queries expected. Then the cost of resources allocated by engine i can be expressed as $\beta \hat{Q}_i D_i$, where $D_i = \sum_t D_i^t$ is the total number of documents indexed by engine i. The number of queries on topic t *actually forwarded* to engine i can be represented as

$$Q_i^t = \begin{cases} 0 & : \quad \exists j, D_i^t < D_j^t \\ \frac{Q^t}{|B|} & : \quad i \in B, B = \left\{ b : D_b^t = \max_j D_j^t \right\} \end{cases}$$

where B is the set of the highest-ranked search engines for topic t, and Q^t is the number of queries on topic t *actually submitted* by the users. That is, the search engine does not receive any queries, if it is ranked lower than competitors, and receives its appropriate share when it is the top ranked engine (see Sections 2.2 and 2.3). The total number of queries forwarded to search engine i can be calculated as $Q_i = \sum_{t:D_i^t>0} Q_i^t$.

We assume that if the search engine receives more queries than it expected (i.e. more queries than it can process), the excess queries are simply rejected. Therefore, the total number of queries processed by search engine i equals to $\min(Q_i, \hat{Q}_i)$. Finally, the performance of engine i over a given time interval can be represented as follows: $P_i = \alpha \min(Q_i, \hat{Q}_i) - \beta \hat{Q}_i D_i$.

3 The COUGAR Approach

The decision-making process for individual search engines can be modelled as a multi-stage game [7]. At each stage, a matrix game is played, where players are search engines, actions are values of (D_i^t), and player i receives payoff P_i.

If player i knew the actions of its opponents and user queries at a future stage k, it could calculate the optimal response as the one maximising $P_i(k)$. For example, in case of a single topic it should play $D_i(k) = \max_{j \neq i} D_j(k) + 1$, if $\max_{j \neq i} D_j(k) + 1 < \alpha/\beta$, and $D_i(k) = 0$ otherwise (simply put, outperform opponents by 1 document if profitable, and do not incur any costs otherwise).

In reality, players do not know the future. Uncertainty about future queries can be largely resolved by reasonably assuming that user interests usually do not change quickly. That is, queries in the next interval are likely to be approximately the same as queries in the previous one. A more difficult problem is not knowing future actions of the opponents (competing search engines). One possible way around this would be to agree on (supposedly, mutually beneficial) future actions in advance. To avoid deception, players would have to agree on playing a Nash equilibrium [7] of the game, since then there will be no incentive for them to not follow the agreement. Agreeing to play a Nash equilibrium, however, becomes problematic when the game has multiple such equilibria. Players would be willing to agree on a Nash equilibrium yielding to them the highest (expected) payoffs, but the task of characterising all Nash equilibria of a game is NP-hard even given complete information about the game (as follows from [8]).

NP-hardness results and the possibility that players may not have complete information about the game and/or their opponents lead us to the idea of "bounded rationality" [2]. Bounded rationality assumes that players may not use the optimal strategies

in the game-theoretic sense. Our proposal is to cast the problem of optimal behaviour in the game as a learning task, where the player would have to learn a strategy that performs well against its sub-optimal opponents.

Learning in games have been studied extensively in both game theory and machine learning. Some examples include fictious play and opponent modelling. Fictious play assumes that the other players are following some Markovian (possibly mixed) strategies, which are estimated from their historical play [9]. Opponent modelling assumes that opponent strategies are representable by finite state automata. The player learns parameters of the opponent's model from experience and then calculates the the best-response automaton [10]. We apply a more recent technique from reinforcement learning called GAPS (which stands for Gradient Ascent for Policy Search) [11]. In GAPS, the learner plays a parameterised strategy represented, e.g., by a finite state automaton, where parameters are probabilities of actions and state transitions. GAPS implements stochastic gradient ascent in the space of policy parameters. After each learning trial, parameters of the policy are updated by following the payoff gradient.

GAPS has a number of advantages important for our domain. It works in partially observable games (e.g. it does not require complete knowledge of the opponents' actions). It also scales well to multiple topics by modelling decision-making as a game with factored actions (where action components correspond to topics). The action space in such games is the product of factor spaces for each action component. GAPS, however, allows us to reduce the learning complexity: rather than learning in the product action space, separate GAPS learners can be used for each action component. It has been shown that such distributed learning is equivalent to learning in the product action space. As with all gradient-based methods, the disadvantage of GAPS is that it is only guaranteed to find a local optimum. We call a search engine that uses the proposed approach COUGAR, which stands for **CO**mpetitor **U**sing **G**APS **A**gainst **R**ivals.

3.1 Engine Controller Design

The task of the search engine controller is to change the state of the document index to maximise the engine performance. When making decisions, the engine controller can receive information about current characteristics of its own search engine as well the external environment in the form of observations.

The COUGAR controllers are modelled by non-deterministic Moore automata. A controller consists of a set of Moore automata (M^t), one for each topic, functioning synchronously. Each automaton is responsible for controlling the state of the search index for the corresponding topic. The following actions are available to each automaton M^t in the controller: *Grow* – increase the number of documents indexed on topic t by one; *Same* – do not change the number of documents on topic t; *Shrink* – decrease the number of documents on topic t by one. The resulting action of the controller is the product of actions (one for each topic) produced by each of the individual automata.

A controller's observations consist of two parts: observations of the state of its own search engine and observations of the opponents' state. The observations of its own state reflect the number of documents in the search engine's index for each topic. The observations of the opponents' state reflect the relative position of the opponents in the metasearcher rankings, which indirectly gives the controller partial information about

the state of the opponents' index. The following three observations of the opponents' state are available for each topic t: *Winning* – there are opponents ranked higher for topic t than our search engine; *Tying* – opponents have either the same or a smaller rank for topic t than our search engine; *Losing* – the rank of our search engine for topic t is higher than opponents.

For T topics, the controller's inputs consist of T observations of the state of its own search engine (one for each topic) and T observations of the relative positions of the opponents (one per topic). Note, that the state of *all* opponents is summarised as a vector of T observations. Each of the Moore automata M^t in the COUGAR controller receives observations only for the corresponding topic t.

One may ask how the controller can obtain information about rankings of its opponents for a given topic. This can be done by sending a query on the topic of interest to the metasearcher and requesting a ranked list of search engines for the query. We also assume that the controller can obtain from the metasearcher information (statistics) on the queries previously submitted by user. This data are used in calculation of the expected number of queries for each topic $\hat{Q}_i^t$. In particular, for our experiments the number of queries on topic t expected by engine i in a given time interval k equals to the number of queries on topic t submitted by users in the previous interval (i.e. $\hat{Q}_i^t(k) = Q^t(k-1)$).

3.2 Learning Procedure

Training of the COUGAR controller to compete against various opponents is performed in series of simulation trials. Each simulation *trial* consists of 100 days, where each day corresponds to one state of the multi-stage game played. The search engines start with empty indices and then, driven by their controllers, adjust their index contents. In the beginning of each day, search engine controllers receive observations and simultaneously produce control actions (change their document indices). A query generator issues a stream of search queries for one day. The metasearcher distributes these queries between the search engines according to their index parameters on the day. The resulting reward in a simulation trial is calculated in the traditional for reinforcement learning way as a sum of discounted rewards from each day. After each trial, a learning step is performed. The COUGAR controller updates its strategy using the GAPS algorithm. That is, the action and state transition probabilities of the controller's Moore automata are modified using the payoff gradient (due to the lack of space see [11] for details of the update mechanism).

In our experiments, we simulated two competing search engines for a single and multiple topics. One search engine was using a fixed strategy, the other one was using the COUGAR controller. To simulate user search queries, we used HTTP logs obtained from a Web proxy of a large ISP. Since each search engine uses a different URL syntax for submission of requests, we developed URL extraction rules individually for 47 well-known search engines The total number of queries extracted was 657,861 collected over a period of 190 days. We associated topics with search terms in the logs. To simulate queries for n topics, we extracted the n most popular terms from the logs. The number of queries generated on topic t during a given time interval was equal to the number of queries with term t in the logs belonging to this time interval.

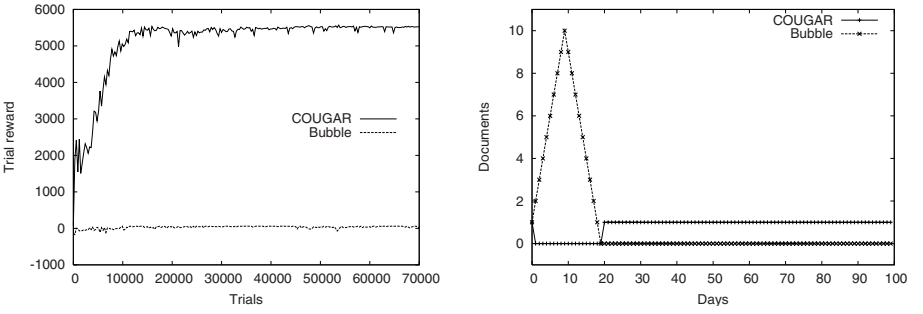

Fig. 1. "Bubble" vs COUGAR, single topic. Left: learning curve. Right: sample trial.

4 Results

4.1 "Bubble" Strategy

The "Bubble" strategy tries to index as many documents as possible without any regard to what competitors are doing. As follows from our performance formula (see Section 2.4), such unconstrained growing leads eventually to negative performance. Once the total reward falls below a certain threshold, the "Bubble" search engine goes bankrupt (i.e. it shrinks its index to 0 documents and retires until the end of the trial). This process imitates the situation in which a search service provider expands its business without paying attention to costs, eventually runs out of money, and quits. An intuitively sensible response to the "Bubble" strategy would be to wait until the bubble "bursts" and then come into the game alone. That is, a competitor should not index anything while the "Bubble" grows and should start indexing a minimal number of documents once the "Bubble" search engine goes bankrupt.

Figure 1 (left) shows how the performance of the COUGAR engine improved during learning for a single topic case. Once COUGAR reached a steady performance level, its resulting strategy was evaluated in a series of testing trials. Figure 1 (right) visualises a sample trial between the "Bubble" and the COUGAR engines by showing the number of documents indexed by the engines on each day of the trial.

In case of multiple topics, the "Bubble" was increasing (and decreasing) the number of documents indexed for each topic simultaneously. The COUGAR controller was using separate GAPS learners to manage the index size for each topic (as discussed in Section 3). Figure 2 shows the engines' behaviour (left) and performance (right) in a test trial with two different topics. Note that COUGAR has learned to wait until "Bubble" goes bankrupt, and then to win all queries for both topics.

4.2 "Wimp" Strategy

The "Wimp" controller used a more intelligent strategy. Consider it first for the case of a single topic. The set of all possible document index sizes is divided by "Wimp" into three non-overlapping sequential regions: "Confident", "Unsure", and "Panic". The

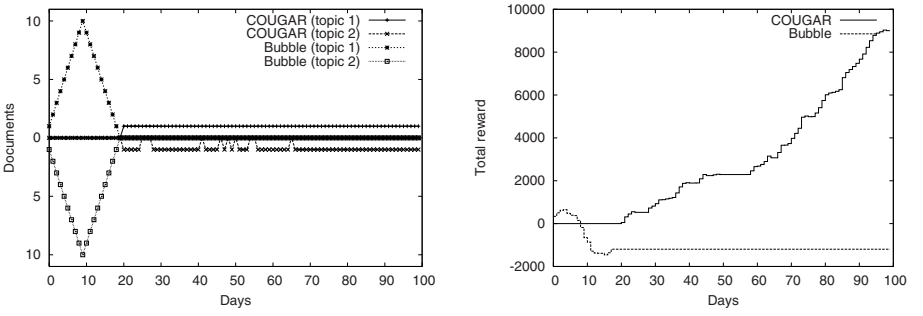

Fig. 2. "Bubble" vs COUGAR, multiple topics. Left: sample trial; the top half of Y axis shows the number of documents for topic 1, while the bottom half shows the number of documents for topic 2. Right: performance in a sample trial.

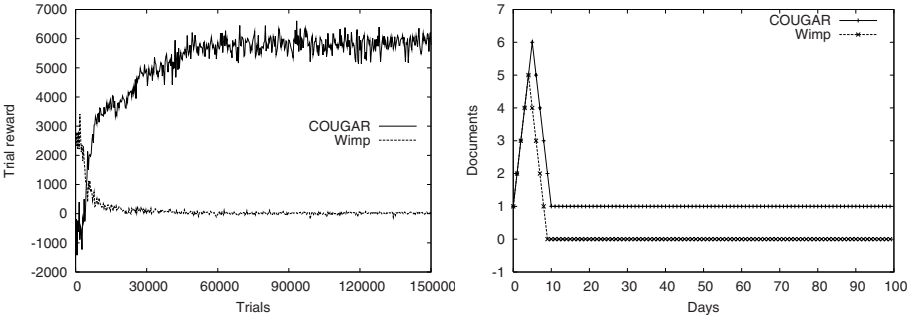

Fig. 3. "Wimp" vs COUGAR, single topic. Left: learning curve. Right: sample trial.

"Wimp's" behaviour in each region is as follows: *Confident* – the strategy in this region is to increase the document index size until it ranks higher than the opponent. Once this goal is achieved, the "Wimp" stops growing and keeps the index unchanged; *Unsure* – in this region, the "Wimp" keeps the index unchanged, if it is ranked higher or the same as the opponent. Otherwise, it retires (i.e. reduces the index size to 0); *Panic* – the "Wimp" retires straight away.

The overall idea is that the "Wimp" tries to outperform its opponent while in the "Confident" region by growing the index. When the index grows into the "Unsure" region, the "Wimp" prefers retirement to competition, unless it is already winning over or tying with the opponent. This reflects the fact that the potential losses in the "Unsure" region (if the opponent wins) become substantial, so the "Wimp" does not dare to risk.

Common sense tells us that one should behave aggressively against the "Wimp" in the beginning, to knock him out of competition, and then enjoy the benefits of monopoly. This is exactly what the COUGAR controller has learned to do as can be seen from Figure 3.

To generalise the "Wimp" strategy to multiple topics, it was modified in the following way. The "Wimp" opponent did not differentiate between topics of both queries

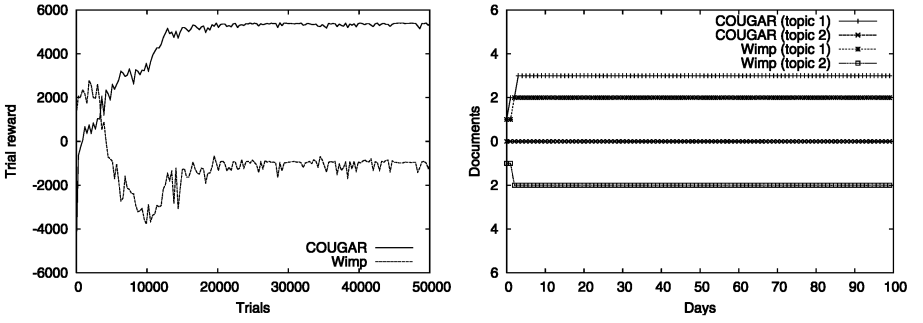

Fig. 4. "Wimp" vs COUGAR, multiple topics. Left: learning curve. Right: sample trial; the top half of Y axis shows the number of documents for topic 1, while the bottom half shows the number of documents for topic 2.

and documents. When assessing its own index size, the "Wimp" was simply adding the documents for different topics together. Similarly, when observing relative positions of the opponent, it was adding together ranking scores for different topics. Finally, like the multi-topic "Bubble", the "Wimp" was changing its index size synchronously for each topic. Figure 4 presents the learning curve (left) and a sample trial (right) respectively. COUGAR decided to specialise on the more popular topic, where it outperformed the opponent. The "Wimp" mistakenly assumed that it was winning in the competition, since its rank for both topics together was higher. In reality, it was receiving only queries for one topic, which did not cover its expenses for indexing documents on both topics.

4.3 Self-play

In the final set of experiments, we analysed behaviour of the COUGAR controller competing against itself. It is not guaranteed from the theoretical point of view that the gradient-based learning will always converge in self play. In practice, however, we observed that both learners converged to relatively stable strategies. We used the same setup with two different topics in the system. Figure 5 (left) shows that the players decided to split the query market: each of the search engines specialised on a different topic. Figure 5 (right) also shows the learning curves.

5 Related Work

The issues of performance (or profit) maximising behaviour in environments with multiple, possibly competing, decision makers have been addressed in a number of contexts, including multi-agent e-commerce systems and distributed databases.

In Mariposa [12], the distributed system consists of a federation of databases and query brokers. A user submits a query to a broker for execution together with the amount of money she is willing to pay for it. The broker partitions the query into sub-queries and finds a set of databases that can execute the sub-queries with the total cost not exceeding

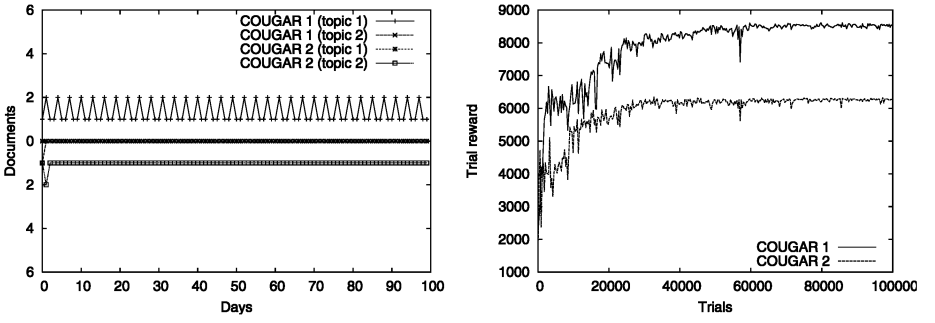

Fig. 5. COUGAR in self play, multiple topics. Left: sample trial; the top half of Y axis shows the number of documents for topic 1, while the bottom half shows the number of documents for topic 2. Right: learning curve.

what the user paid and the minimal processing delay. A database can execute a subquery only if it has all necessary data (data fragments) that are involved. The databases can trade data fragments (i.e. purchase or sell them) to maximise their revenues.

Trading data fragments may seem similar to the topic selection problem for specialised search engines. There are, however, significant differences between them. Acquiring a data fragment is an act of mutual agreement between the seller and the buyer, while search engines may change their index contents independently from others. Also, a proprietorship considerations are not taken into account.

Greenwald *et al* have studied behaviour dynamics of *pricebots*, automated agents that act on behalf of service suppliers and employ price-setting algorithms to maximise profits [13]. In the proposed model, the sellers offer a homogeneous good in an economy with multiple sellers and buyers. The buyers may have different strategies for selecting the seller, ranging from random to the selection of the cheapest seller on the market (bargain hunters), while the sellers use the same pricing strategy. A similar model but with populations of sellers using different strategies has been studied in [14]. The pricing problem can be viewed as a very simple instance of our topic selection task (namely, as a single topic case with some modifications to the performance model).

6 Conclusions and Future Work

The successful deployment of heterogeneous Web search environments will require that participating search service providers have effective means for managing the performance of their search engines. We analysed how specialised search engines can select on which topic(s) to specialise and how many documents to index on that topic to maximise their performance. We provided both an in-depth theoretical analysis of the problem and a practical method for automatically managing the search engine content in a simplified version of the problem. Our adaptive search engine, COUGAR, has managed to compete with some non-trivial opponents as shown by the experimental results. Most importantly, the same learning mechanism worked successfully against opponents using different strategies. Even when competing against other adaptive search engines (in

our case itself), COUGAR has demonstrated a fairly sensible behaviour from the economic point of view. Namely, the engines have learned to segment the search services market with each engine occupying its niche, instead of a head-on competition.

We do not claim to provide a complete solution for the problem here, but we believe it is the promising first step. Clearly, we have made many strong assumptions in our models. One future direction will be to relax these assumptions to make our simulations more realistic. In particular, we intend to perform experiments with real documents and using some existing metasearch algorithms. This should allow us to avoid the assumption of the "single-topic" documents and also to assess how closely our metasearch model reflects real-life engine selection algorithms. We also plan to use clustering of user queries to derive topics in our simulations.

While we are motivated by the optimal behaviour for search services over document collections, our approach is applicable in more general scenarios involving services that must weigh the cost of their inventory of objects against the expected inventories of their competitors and the anticipated needs of their customers. For example, it would be interesting to apply our ideas to an environment in which large retail e-commerce sites must decide which products to stock. Another important direction would be to further investigate performance and convergence properties of the learning algorithm when opponents also evolve over time (e.g. against other learners). One possible approach here would be to use a variable learning rate as suggested in [15].

References

1. Gravano, L., Garcia-Molina, H.: GlOSS: Text-source discovery over the Internet. ACM Trans. on Database Systems **24** (1999) 229–264
2. Rubinstein, A.: Modelling Bounded Rationality. The MIT Press (1997)
3. Khoussainov, R., Kushmerick, N.: Performance management in competitive distributed Web search. In: Proc. of the IEEE/WIC Intl. Conf. on Web Intelligence. (2003) To appear.
4. Risvik, K., Michelsen, R.: Search engines and web dynamics. Computer Networks **39** (2002)
5. van Rijsbergen, C.J.: Information Retrieval. 2nd edn. Butterworths (1979)
6. Chakrabarti, S., van den Berg, M., Dom, B.: Focused crawling: A new approach to topic-specific Web resource discovery. In: Proc. of the 8th WWW Conf. (1999)
7. Osborne, M., Rubinstein, A.: A Course in Game Theory. The MIT Press (1999)
8. Conitzer, V., Sandholm, T.: Complexity results about Nash equilibria. Technical Report CMU-CS-02-135, Carnegie Mellon University (2002)
9. Robinson, J.: An iterative method of solving a game. Annals of Mathematics **54** (1951)
10. Carmel, D., Markovitch, S.: Learning models of intelligent agents. In: Proc. of the 13th National Conf. on AI. (1996)
11. Peshkin, L.: Reinforcement Learning by Policy Search. PhD thesis, MIT (2002)
12. Stonebraker, M., Devine, R., Kornacker, M., Litwin, W., Pfeffer, A., Sah, A., Staelin, C.: An economic paradigm for query processing and data migration in Mariposa. In: Proc. of the 3rd Intl. Conf. on Parallel and Distributed Information Systems. (1994) 58–67
13. Greenwald, A., Kephart, J., Tesauro, G.: Strategic pricebot dynamics. In: Proc. of the 1st ACM Conf. on Electronic Commerce. (1999) 58–67
14. Greenwald, A., Kephart, J.: Shopbots and pricebots. In: Proc. of the 16th Intl. Joint Conf. on AI. (1999) 506–511
15. Bowling, M., Veloso, M.: Rational and convergent learning in stochastic games. In: Proc. of the 17th Intl. Joint Conf. on AI. (2001)