

# Distributed Web Search as a Stochastic Game\*

Rinat Khoussainov and Nicholas Kushmerick

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

**Abstract.** Distributed search systems are an emerging phenomenon in Web search, in which independent topic-specific search engines provide search services, and metasearchers distribute user’s queries to only the most suitable search engines. Previous research has investigated methods for engine selection and merging of search results (i.e. performance improvements from the user’s perspective). We focus instead on performance from the 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 user queries by choosing which documents (topics) to index. The challenge is that the utilities of an engine’s actions should depend on the uncertain actions of competitors. Thus, naive strategies (e.g. blindly indexing lots of popular documents) are ineffective. We model the competition between search engines as a stochastic game, and propose a reinforcement learning approach to managing search index contents. We evaluate our approach using a large log of user queries to 47 real search engines.

## 1 Introduction

Distributed heterogeneous search environments are an emerging phenomenon in Web search. Consider a federation of *independently controlled* metasearchers and many specialised search engines. The 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, specialised search engines (together) may provide access to arguably much larger volumes of high-quality information resources, frequently called “deep” or “invisible” Web.

We envisage that such heterogeneous environments will become more popular and influential. However, to unlock the benefits of distributed search for the users, there must be an incentive for search engines to participate in such a federation, i.e. an opportunity to make money. Previous research has mainly targeted performance of such environments from the user’s perspective (i.e. improved quality of search results). On the other hand, a provider of search services is more interested in the utility of the service compared to the cost of the resources used (e.g. the income from search queries processed versus the amount of resources needed for answering those queries).

---

\* 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.

An important factor that affects a particular search engine’s performance in heterogeneous search environments is *competition* with other independently controlled search engines. When there are many engines available, users will send queries to those that would provide the best possible results. Thus, the service offered by one search engine influences queries received by its competitors. Multiple search providers 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. In particular, we analyse a scenario in which independent topic-specific search engines compete for user queries by choosing which documents (topics) to index.

*EXAMPLE.* Consider a heterogeneous environment with two search engines *A* and *B* having equal resource capabilities. Assume that users are only interested in either “sport” or “cooking”, with “sport” being more popular. If *A* and *B* each decide to index both “sport” and “cooking” (i.e. everything, like Google tries to do), they will receive an equal share of all user queries. If *A* decides to spend all its resources only on “sport” while *B* stays on both topics, *A* will provide better search for “sport” than *B*. Then users will send queries on “sport” to *A*, and on “cooking” to *B*. Therefore, *A* will receive more queries (and so will have higher performance). If, however, *B* also decides to index only “sport”, both search engines will compete only for the same queries and, thus, will each receive even fewer requests than in the 2 previous cases.

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. Actions of one search engine affect the queries received by its competitors, and vice versa. The uncertainty about competitors as well as the potentially large number of competing engines make our optimisation problem difficult. For example, in one experiment (Sec. 4), several search engines that competed head-to-head for the most popular topic were less profitable than an engine that cornered the market on another topic that was less popular but ignored by the competition. This example illustrates that naive strategies such as blindly indexing popular documents can be suboptimal.

We model the competition between specialised search engines as a partially observable stochastic game, and exploit the concept of “*bounded rationality*” [1]. Bounded rationality assumes that decision makers are unable to act optimally in the game-theoretic sense due to incomplete information about the environment and/or limited computational resources. We cast our problem as a reinforcement learning task, where the goal of a specialised search engine is to learn a good behaviour strategy against given (potentially sub-optimal) competitors. The effectiveness of our approach is evaluated in a simulation environment. The simulator implements a simplified formalisation of the problem and is driven by user queries submitted to over 47 existing search engines.

## 2 Problem Formalisation

The questions of optimal behaviour in computational markets have been researched extensively from both consumers and suppliers points of view [2, 3]. However, services markets in heterogeneous Web search environments have a number of features requiring new techniques. Given the complexity of the problem domain, attempting to make

our analysis 100% realistic from the very beginning is neither feasible nor reasonable. Instead, we start with a simplified model. Our goal is to select an approach that allows us in principle to factor more realistic details into our models in future.

## 2.1 Performance metric

We adopt an economic view on search engine performance. Performance is a difference between the value (utility) of the search service provided 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. We use the following formula for the search engine performance:  $P = \alpha_1 Q - \alpha_2 QD - \alpha_3 C - \alpha_4 D$ , where  $Q$  is the number of queries received in a given time interval,  $D$  is the number of documents in the engine's index,  $C$  is the number of new documents added to the index during the given time interval, and  $\alpha_x$  are constants.

$\alpha_1 Q$  represents the service value: if the price of processing one search request for a user is  $\alpha_1$ , then  $\alpha_1 Q$  would be the total income from service provisioning.  $\alpha_2 QD$  represents the cost of processing search requests. If  $x$  amount of resources is sufficient to process  $Q$  queries, then we need  $2x$  to process twice as many queries in the same time. Similarly, twice as many resources are needed to search twice as many documents in the same time. Thus, the amount of resources can be expressed as  $\alpha_2 QD$ , where  $\alpha_2$  reflects the resource costs. The example of the FAST search engine confirms that our cost function is not that far from reality [4].  $\alpha_3 C$  is the cost of crawling and indexing new documents. While the cost of indexing new documents is indeed proportional to the number of the documents added, the cost of crawling can vary between documents. Here, we rather assume an average crawling cost. Finally,  $\alpha_4 D$  is the cost of document storage and maintenance. This includes storing a copy of the document as well as keeping the document description up-to-date.

We assume that all search engines use the same  $\alpha_x$  constants when calculating their performance. Having the same  $\alpha_2$ – $\alpha_4$  reasonably assumes that the cost of resources (CPU, memory, network) per “unit” is the same for all search engines. Having the same  $\alpha_1$  assumes, perhaps unrealistically, that the search engines choose to charge users the same amount per query. We leave to future work the optimisation problem in environments where engines may have different service pricing.

## 2.2 Engine selection model

We use a very generic model of the metasearch component, so we can abstract from implementation details or particular metasearch algorithms. Essentially, we 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 search engines that maximise the results relevance, while minimising the number of engines involved. The existing research in metasearch (e.g. [5]), however, does not go much further than simply ranking search engines. 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 based on the expected number of relevant documents that are indexed by each engine. The 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 [6]. For each document  $d$ , there is a probability  $\Pr(\text{rel}|q, d)$  that this document will be considered by the user as relevant to query  $q$ . In this case,  $NR_i^q = \sum_{d \in i} \Pr(\text{rel}|q, d)$ , where  $d \in i$  iterates over the documents indexed by engine  $i$ . If  $\Pr(\text{rel}|q_1, d) = \Pr(\text{rel}|q_2, d), \forall d$  then queries  $q_1$  and  $q_2$  will look the same from both the metasearcher's and search engine's points of view, even though  $q_1 \neq q_2$ . Therefore, all queries can be partitioned into equivalence classes with identical  $\Pr(\text{rel}|q, d)$  functions. We call such classes *topics*. We assume that there is a fixed finite set of topics and queries can be assigned to topics. 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

Assume that users only issue queries on a single topic. We will see later how this can be extended to multiple topics. To receive queries, a search engine needs to be the highest ranked one for this topic. Given an index size  $D$ , engine  $i$  would like to index a set of  $D$  documents with the largest possible  $NR_i$  (expected number of documents relevant to the topic).

Search engines use topic-specific (focused) Web crawlers to find documents to index. 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  $NR_1 < NR_2$  (if all documents have  $\Pr(\text{rel}|d) > 0$ ). 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 sequence

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 (incurring the corresponding costs). Since 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 users submit queries during the time interval, which are allocated to the search engines based on their index parameters ( $D_i^t$ ) as described above. Then the process repeats.

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  $\hat{Q}_i = \sum_{t:D_i^t > 0} \hat{Q}_i^t$ . Obviously, we only expect queries for those topics, for which we index documents (i.e. for which  $D_i^t > 0$ ). We assume that engines always allocate resources for the full amount of queries expected, so that in case they win the competition, they will be able to answer all queries received. Then the cost of resources allocated by engine  $i$  can be expressed as  $\alpha_2 \hat{Q}_i D_i - \alpha_3 C_i - \alpha_4 D_i$ , where  $D_i = \sum_t D_i^t$  is the total number of documents indexed by engine  $i$ , and  $C_i = \sum_t C_i^t$  is the total number of documents added to the engine's index in this time interval. For the given resource allocation,  $\hat{Q}_i$  will be the total number of queries that engine  $i$  can process within the time interval (its query processing capacity).

The number of queries on topic  $t$  *actually forwarded* to engine  $i$  (presuming that  $D_i^t > 0$ ) can be represented as  $Q_i^t = 0$  if engine  $i$  is ranked lower than its competitors (i.e.  $\exists j, D_i^t < D_j^t$ ), and  $Q_i^t = Q^t / |K|$  if  $i$  is in the set of the highest-ranked engines  $K = \{k : D_k^t = \max_j D_j^t\}$ .  $Q^t$  is the number of queries on topic  $t$  *actually submitted* by the users. 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, performance of engine  $i$  over a given time interval can be represented as follows:  $P_i = \alpha_1 \min(Q_i, \hat{Q}_i) - \alpha_2 \hat{Q}_i D_i - \alpha_3 C_i - \alpha_4 D_i$ . Note that even if an engine wins in the competition, its performance decreases as the index grows, and eventually becomes negative. This effect accords with the intuition that a huge index must eventually cost more to maintain than can ever be recovered by answering queries, and serves to justify our economic framework for analysing optimal search engine behaviour.

## 2.5 Decision making as a stochastic game

The decision making process can be modelled as a stochastic game. A *stochastic game* (SG) [7] is a tuple  $\langle n, S, A_{1..n}, T, R_{1..n} \rangle$ , where  $n$  is the number of decision makers (players),  $S$  is a set of game states,  $A_i$  is a set of actions for player  $i$ ,  $T : S \times A \times S \rightarrow [0, 1]$  is a transition function (where  $A = A_1 \times \dots \times A_n$ ), and  $R_i : S \times A \rightarrow \mathbb{R}$  is a reward function for player  $i$ . SGs are very similar to Markov Decision Processes (MDPs) except there are multiple decision makers and the next state and rewards depend on the joint action of the players.

In our case, players are search engines, the state of the game at stage  $k$  is defined by the state of the indices of all search engines ( $(D_1^t(k)), \dots, (D_n^t(k))$ ) and by time (which determines user queries at the given stage). A player's action  $a_i(k) \in A_i$  is a vector  $(a_i^t(k))$  of index adjustments for each topic  $t$ . The following index adjustments  $a_i^t$  are possible: *Grow* (increase the number of documents indexed on topic  $t$  by one); *Same* (do not change the number of documents on topic  $t$ ); and *Shrink* (decrease the number of documents on topic  $t$  by one). The reward to player  $i$  is calculated using the formula for  $P_i$ , where  $C_i^t = 1$  if  $a_i^t(k)$  in this time interval was "Grow", and  $C_i^t = 0$  otherwise.

## 2.6 Strategies and observations

A player’s strategy (policy) in the game is a function that maps a history of the player’s current (and possibly past) observations of the game to a probability distribution over player’s actions. In our SG, a player’s observations consist of two parts: observations of the state of its own search engine and, observations of the opponents’ state. For  $T$  topics, the player’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), i.e.  $2T$  observations in total.

The observations of its own state reflect the number of documents in the search engine’s index for each topic  $t$ . We do not assume that search engines know the contents of each other’s indices. Instead, observations of the opponents’ state reflect the relative position of the opponents in the metasearcher rankings, which indirectly gives the player information about the states of the opponents’ index. The following three observations are available for each topic  $t$ : *Winning* – there are opponents ranked higher for topic  $t$  than our search engine (i.e. they index more documents on the topic than we do); *Tying* – there are opponents having the same and smaller ranks for topic  $t$  than our search engine (opponents index the same and smaller number of documents on the topic); and *Losing* – the rank of our search engine for topic  $t$  is higher than opponents (opponents index less documents on the topic than we do).

How can a player know relative rankings of its opponents? It can send a query on the topic of interest to the metasearcher (as a search user) and request a ranked list of search engines for the query. We also assume that players 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, the number of queries on topic  $t$  expected by engine  $i$  at stage  $k$  equals to the number of queries on topic  $t$  submitted by users at the previous stage (i.e.  $\hat{Q}_i^t(k) = Q^t(k - 1)$ ).

## 3 The COUGAR Approach

Optimal behaviour in an SG is in general opponent-dependent: to select their future actions, players need to know future opponents’ strategies. Nash equilibrium from game theory [8, 7] provides a way to resolve this uncertainty about opponents: if players agree to play a Nash equilibrium, then they do not have incentive to unilaterally deviate (thus they become certain about each other’s future actions).

Agreeing to play a Nash equilibrium, however, is problematic in games with multiple equilibria (which is frequently the case). There is a large body of research on equilibrium selection in game theory. Ultimately, it requires characterising all Nash equilibria of a game, which is NP-hard even given complete information about the game [9]. These results and the possibility that players may not have complete information lead to the idea of “bounded rationality”, when players are limited in their abilities to make optimal decisions. Our goal is to learn a strategy that performs well against the given opponents rather than trying to calculate some equilibrium behaviour.

Learning in SGs have been studied extensively in game theory and machine learning. We propose to use a recent reinforcement learning algorithm called GAPS (which

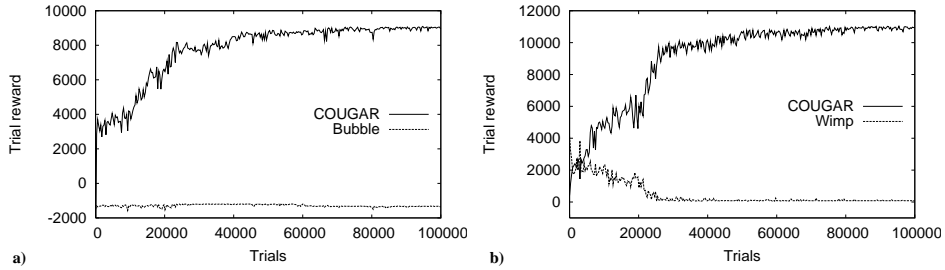
stands for **Gradient Ascent for Policy Search**) [10]. In GAPS, the learner plays a parameterised strategy represented by a non-deterministic Moore automaton, where the parameters are the probabilities of actions and state transitions. GAPS implements stochastic gradient ascent in the space of policy parameters. After each learning trial, parameters of the policy are updated by following the reward gradient.

GAPS has a number of advantages important for our problem domain. Unlike model-based reinforcement learning algorithms, GAPS does not attempt to build a model of the game or the opponents from the interaction experience, and thus can cope with partial observability and scale well with the number of opponents. The policies learned by GAPS can be both non-deterministic and state-full. The ability to play non-deterministic policies means that GAPS can potentially achieve the optimal performance in the games where only mixed (non-deterministic) strategy equilibria exist [8]. Learning state-full policies can be advantageous in partially observable settings [11]. Finally, GAPS 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 [10]. As with most 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 (COmpetitor Using GAPS Against Rivals)**. The COUGAR controllers are represented by a set of non-deterministic Moore automata ( $M^t$ ), one for each topic, functioning synchronously. Inputs of each automaton  $M^t$  are game observations for topic  $t$ . Outputs of each automaton are actions changing the number of documents  $D_i^t$  indexed for topic  $t$ . The resulting action of the controller is the product of actions (one for each topic) produced by each of the individual automata.

Training of COUGAR is performed in series of learning trials. Each learning *trial* consists of 100 days, where each day corresponds to one stage of the SG played. The search engines start with empty indices and then, driven by their controllers, adjust their index contents. In the beginning of each day, engine controllers receive observations and simultaneously produce control actions (change their document indices). Users issue a stream of search queries for one day. The metasearcher distributes queries between the engines according to their index parameters on the day, and the search engines collect the corresponding rewards. For the next day, the search engines continue from their current states, and users issue the next batch of queries. The resulting performance in the whole trial is calculated as a sum of discounted rewards from each day. After each trial, the COUGAR controller updates its strategy using the GAPS algorithm. That is, the action and state transition probabilities of the controller’s automata are modified using the payoff gradient (see [12] for details).

To simulate user search queries, we used HTTP logs obtained from a Web proxy of a large ISP and extracted queries to 47 well-known search engines. The total number of queries extracted was 657,861 collected over a period of 190 days, and we used a 100-day subset of these queries in our simulations. We associated topics with search



**Fig. 1.** Learning curves: (a) COUGAR vs “Bubble”; (b) COUGAR vs “Wimp”.

terms in the logs. To simulate queries for  $T$  topics, we extracted the  $T$  most popular terms from the logs.

## 4 Experimental Results

Our experiments consist of two parts. In the first part, we evaluate the performance of COUGAR against various fixed opponents. In the second part, we test COUGAR against evolving opponents, which in our case are also COUGAR learners. In the experiments with fixed opponents, we simulated 2 search engines competing for 2 topics.

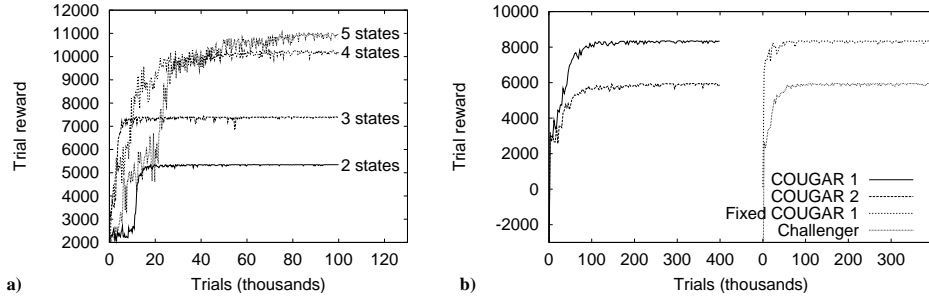
### 4.1 The “Bubble” opponent

The “Bubble” strategy tries to index as many documents as possible without any regard to what competitors are doing. From our performance formula, such unconstrained growing leads eventually to negative performance. Once the total reward falls below a certain threshold, the “Bubble” search engine goes bankrupt (it shrinks its index to 0 documents). This process imitates the situation in which a search 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 “Bubble” goes bankrupt.

Fig. 1a shows how COUGAR’s performance improved during learning. Once the COUGAR engine reached a steady performance level, its resulting strategy was evaluated in a series of testing trials. Analysis of a sample testing trial shows that COUGAR has learned to wait until “Bubble” goes bankrupt, and then to win all queries for both topics.

### 4.2 The “Wimp” opponent

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”



**Fig. 2.** (a) “Wimp” vs COUGAR with different number of states; (b) COUGAR in self play: learning (left), evaluation (right).

into three non-overlapping sequential regions: “Confident”, “Unsure”, and “Panic”. The “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, “Wimp” stops growing and keeps the index unchanged; *Unsure* – in this region, “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* – “Wimp” retires straight away.

An overall idea is that “Wimp” tries to outperform its opponent while in the “Confident” region by growing the index. When the index grows into the “Unsure” region, “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 “Wimp” does not dare to risk.

To generalise the “Wimp” strategy to multiple topics, it was modified in the following way. When assessing its own index size, “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”, “Wimp” was changing its index size synchronously for each topic. Common sense tells us that one should behave aggressively against “Wimp” in the beginning (i.e. index more than “Wimp”), to knock him out of competition, and then enjoy the benefits of monopoly (i.e. index minimum possible number of documents). This is what COUGAR has learned to do. Fig. 1b also presents the learning curve.

### 4.3 Policy complexity and performance

As pointed out earlier, the ability of COUGAR to learn non-deterministic and state-full policies can be a potential advantage under partial observability. A policy with more states can represent more complex (and potentially more optimal) behaviours. To analyse the effects of the policy complexity on the performance achieved, we evaluated COUGAR controllers with different number of states against the “Wimp” opponent. Fig. 2a demonstrates that indeed COUGARs with more states performed better, though in most cases learned more slowly.

#### 4.4 Performance bounds

While COUGAR was superior to both fixed strategy opponents, an interesting question is how its performance relates to the maximum and minimum values obtainable for the given opponents. To assess this, we evaluated “Bubble” and “Wimp” against the corresponding best-case (omniscient) and worst-case strategies. COUGAR has achieved 99.8% of the optimal performance against “Bubble” and 88.6% against “Wimp”. These facts not only show COUGAR’s near-optimality but also demonstrate the fact that such performance was not a result of dumb opponents: COUGAR could have performed much worse.

#### 4.5 COUGAR in self play

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 learners converged to relatively stable strategies. The simulation setup was the same as for the fixed opponent experiments. We trained two COUGAR learners, each using a 3-state GAPS policy, in self play in a scenario with two topics. The players decided to split the query market: each search engine specialised on a different topic. The winner happened to pick the more popular topic.

As previously mentioned, a policy learned by COUGAR in self-play is, in general, only locally optimal against the policies simultaneously learned by its opponents. A question we are interested in is how well such locally optimal policy can perform against an unbiased COUGAR learner. To test this, we fixed the strategy of the winner from self play, and trained another COUGAR against it. We can envisage a possible failure of the winner’s policy, if it did not learn to compete well for the more popular topic (because of its opponent), and learned not to compete for the less popular topic. Then an unbiased challenger might be able to capture queries for both topics. Evaluation against a 3-state COUGAR challenger showed, however, that the performance of the winner’s policy is quite stable. Fig. 2b shows the performance of the search engines during self play (left) and evaluation (right). The fixed COUGAR achieves the same performance against the challenger as it had in self play.

#### 4.6 Scaling up

An important advantage of GAPS is the ability to handle well multiple topics by modelling decision-making as a game with factored actions. To analyse the scalability of our approach with the number of topics, we simulated 10 COUGAR learners competing in a scenario with 10 different topics. Analysis showed that, similarly to the case of 2 learners and 2 topics, they decided to split the query market. An important detail is that more popular topics attracted more search engines. This confirms that the market split was really profit-driven and demonstrates COUGAR’s “economic thinking”. However, the most profitable engines actually specialised on the relatively unpopular topics, while the engines that competed for the most popular topics ended up earning substantially less money. Therefore, a naive strategy of indexing the most popular documents was suboptimal in this experiment.

## 5 Conclusions

Stochastic games provide a convenient theoretical framework for modelling competition between search engines in heterogeneous Web search systems. However, finding optimal behaviour strategies in stochastic games is a challenging task, especially in partially observable games, and also when other players may evolve over time. We demonstrated that reinforcement learning with state-full policies seems an effective approach to the problem of managing the search engine content. Our adaptive search engine, COUGAR, has shown the potential to derive behaviour strategies allowing it to win in the competition against non-trivial fixed opponents, while policies learned in self-play were robust enough against evolving opponents (other COUGARs in our case).

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 algorithm. We currently assume that users are insensitive to the quality of the search results; we intend to extend our metasearch and service pricing models to account for user satisfaction. Another direction would be to further investigate the issues of the learning convergence and performance robustness in self-play, and against other learners.

## References

1. Rubinstein, A.: *Modelling Bounded Rationality*. The MIT Press (1997)
2. Greenwald, A., Kephart, J., Tesauro, G.: Strategic pricebot dynamics. In: *Proc. of the 1st ACM Conf. on Electronic Commerce*. (1999) 58–67
3. J.Hu, Wellman, M.: Learning about other agents in a dynamic multiagent system. *Cognitive Systems Research* **2** (2001)
4. Risvik, K., Michelsen, R.: Search engines and Web dynamics. *Computer Networks* **39** (2002)
5. Gravano, L., Garcia-Molina, H.: GLOSS: Text-source discovery over the Internet. *ACM Trans. on Database Systems* **24** (1999) 229–264
6. van Rijsbergen, C.J.: *Information Retrieval*. 2nd edn. Butterworths (1979)
7. Filar, J., Vrieze, K.: *Competitive Markov Decision Processes*. Springer Verlag, New York (1997)
8. Osborne, M., Rubinstein, A.: *A Course in Game Theory*. The MIT Press (1999)
9. Conitzer, V., Sandholm, T.: Complexity results about Nash equilibria. In: *Proc. of the 18th Intl. Joint Conf. on AI*. (2003)
10. Peshkin, L., Meuleau, N., Kim, K.E., L.Kaelbling: Learning to cooperate via policy search. In: *Proc. of the 16th Conf. on Uncertainty in AI*. (2000)
11. Singh, S., Jaakkola, T., Jordan, M.: Learning without state-estimation in partially observable Markovian decision processes. In: *Proc. of the 11th Intl. Conf. on Machine Learning*. (1994)
12. Peshkin, L.: Reinforcement learning by policy search. MIT AI Lab Technical Report 2003-003 (2002)