Do What I Mean: Online Shopping with a Natural Language Search Agent

Barry G. Silverman
University of Pennsylvania, basil@seas.upenn.edu

Mintu Bachann
Equidity

Khaled Al-Akharas
Equidity

Follow this and additional works at: http://repository.upenn.edu/ese_papers

Recommended Citation
Do What I Mean: Online Shopping with a Natural Language Search Agent

Abstract
Ineffective search engines on e-catalog sites are driving away potential customers. Natural-language querying improves precision and parsing capability, and with advances in the technology, it can also meet these shopping sites' performance demands.

Comments
Publisher URL: http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=20376&puNumber=5254

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Pennsylvania's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org. By choosing to view this document, you agree to all provisions of the copyright laws protecting it.
Intelligent E-Business

Do What I Mean: Online Shopping with a Natural Language Search Agent

Barry G. Silverman, University of Pennsylvania
Mintu Bachann and Khaled Al-Akharas, Equidity

Market exchanges are Internet sites that virtually integrate multiple vendors’ catalogs. They let users browse, search for, bid on, finance, buy, or ship products. Such sites—Amazon.com and Ariba.com are two examples among many—constitute a major form of e-commerce in both the business-to-consumer and business-to-business segments. Over half of today’s 80 million Web users shop for or buy products online, and analysts expect B2B purchasing to eclipse that level rapidly.¹

However, in the rush to establish an online presence, many enterprises have built their e-market sites with little infrastructure and few of the capabilities necessary for such an e-business. For example, in March 2000, the Boston Consulting Group reported that more than 80 percent of Web shoppers have at some point left e-markets without finding what they wanted and that 23 percent of all attempted e-shopping transactions end in failure.¹ Four of the top five failure modes are search-related—long page-loading times, failures to find the product, system crashes, and calls to customer service.

Research also shows that about 15 percent of online search failures stem from spelling errors and another 40 percent result from customers using different terms from those in the Web site¹—that for example, using “patching” when the Web site uses “concrete.” Because keyword-based search engines can’t interpret the meaning of users’ queries, they typically bring back innumerable hits of everything even remotely relevant, often burying the best choice deep within the list or omitting it altogether. On top of that, such search engines seldom help the shopper narrow down the returned set of items.

Two approaches exist for improving keyword-based search: conceptual query (CQ) and natural language query (for a comparison, see the sidebar). This article describes an agent that uses restricted natural language to help users search product catalogs in large-scale market exchanges.

The challenges

Before we turn to our NLQ search, let’s examine the challenges that catalogs pose for any search method. Specifically, in shopping site searches, one challenge is handling incomplete and inconsistent product descriptions. Some descriptions include partial parameter information, such as “2”H11033, black.” Others focus on how the product might be used, and still others are highly terse and omit most details.

A second challenge is the difficulty of matching the buyer’s search terms to the wording in the descriptive fields. When exact term matches don’t exist, the search must consider issues such as word stemming, spelling errors, abbreviations, and synonyms.

A third challenge is that although product descriptions represent numeric attributes poorly and incompletely, some users will nonetheless want to search by size, weight, height, and so on. Every catalog offers many products, the product lines are continually changing, and each of the thousands of product categories has a different set of attributes. As a result,
catalogs do not store attributes as fields of a table. Rather, they store attribute names and their value settings as data items; this makes searches more difficult.

Shopping catalogs often include many dozens, even thousands, of tables to describe the products being offered and to support services and user needs. To improve runtime performance, these sites often use a denormalized field called a munge as the search engine target. This munge places into its subfields copies of each of the catalog’s searchable fields, such as item name, item ID, category name and ID, model or part number, description, price, maker, condition, and all attribute-value pairs. In effect, the munge is like a document on each product in the catalog.

Because most search algorithms can’t infer the subfields to search, they search the entire munge. For example, a keyword search processes across all subfields of the full munge for a strict match on terms. CQ traverses the same ground but can also look for synonyms, alphabetically similar terms, and related conceptualizations. NLQ is the only search strategy that infers the labels or field names of each token in the query string. Hence, it can then send the (conceptual) search to the precise subfields of the catalog in which the query’s tokens should exist, provided they are in the database.

The reason that CQ does worse than NLQ is that it must deal with term semantics and ambiguities without a parser. NLQ, on the other hand, can parse the entire query string, label its tokens, and hold an interactive conversation about the query to confirm its interpretation. NLQ researchers argue that CQ languages would benefit from the addition of a parser with a grammar restricted to the database’s domain. Although such parsers have less-than-general interpretative power, they can still improve trouble management and help the interface conform to the user’s language. Several systems in the lab purport to provide these extensions, and some offer formal slotted grammars (with semantically typed slots) for merging the natural language approach with CQ.

Proponents of pure CQ, in turn, argue that natural language extensions tend to be impractical: indeed, none of the NLQ systems just cited has been evaluated on a large scale. The CQ people argue that the “natural language problem” is too difficult and remains unsolved on any reasonable scale. As a result, most large-scale relational database management products (for example, Oracle, Sybase, DB II, or AltaVista) that are widely used by market exchanges and other large-scale e-commerce sites include CQ features for those who choose to deploy them, but exclude NLQ. Some smaller, less formally defined NLQ systems appear to work atop specific databases or environments.7,8

Certain e-commerce Web sites have deployed natural language self-help or chatterbots, but these bots handle site navigation and document retrieval issues and cannot process catalog or database search requests. This inability adds fuel to the arguments that only CQ can scale up to the e-catalog task.4

In sum, no examples exist of NLQ working in large-scale e-commerce catalog shopping sites, and it is tempting to believe the proponents of CQ rather than NLQ. The research results we report here are the first large-scale test offering evidence to counter pure-CQ proponents. Our results show that NLQ is an effective complement to CQ in shopping Web sites.

System architecture and algorithm

Now let’s take a look at how our agent, called EQUIsearch, performs NLQ. We assume that a Markov decision process is suitable to analyze the semantics and morphosyntactics of the user’s query. Specifically, at any moment in time, the agent is in one of a finite number of states (s = 1, S) and must choose one of a finite set of actions (a = 1, A) to transition to the next state. More specifically, optimizing a Markov decision process is a dynamic programming problem that maximizes E(U), the expected discounted rewards across future periods, as

$$\max V^* = E \left[ \sum_{t=1}^{T} U(s_t, a_t) \right]$$

where $V^*$ is the optimum value point (in terms of precision and recall) and $U(\cdot)$ is the reward function or utility from selecting action $a_t$ at state $s_t$.

We can solve Equation 1 if we loop across iterations ($t = 1, T$) and for each iteration loop across all states ($s = 1, S$) to find the action or set of actions that maximizes both current and future rewards to avoid local optima. We capture this expansion by finding the maximal value of the following function after testing all possible actions, $a = 1, A$:
$Z_t(s, a) = U(s_t, a_t) + \delta \sum_{s_{t+1}} \pi(s_t, a_t, s_{t+1}) V_{t+1}(s_{t+1})$ \hspace{1cm} (2)

Here, $\pi(s_t, a_t, s_{t+1})$ is the transition probability of being in state $s_{t+1}$ immediately after taking action $a_t$ from state $s_t$. $V_{t+1}(s_{t+1})$ is $Z_{t+1}(s_{t+1}, \text{argmax}_a (Z_{t-1}(s_t, a)))$, where argmax$_a$ finds the maximal action.

Thus, recursive Equation 2 summarizes the standard computable value iteration formulation. To use this, we must define the permissible states and actions, the reward function, the transition probabilities, and other terms of the equation for catalog search problems. We provide an overview of these items in what follows and in Figure 1; readers can find the details elsewhere.\(^9\)

As a language parser, the agent iteratively labels each stripped and stemmed token of the query. Then, it modifies these labels by applying a sequence of transformation rules to reduce the remaining ambiguities and residual errors left by the previous rules. In doing so, it labels the state of each term in the query, where state is defined within a subgrammar. The subgrammar we created here is called OAV Triplet Grammar. OAV triplets, or object-attribute-value triplets, are metadata that describe what catalogs contain in much the same way as resource description formats (RDFs) describe the Web’s content.\(^10,11\)

Specifically, in product catalog domains, users usually search for objects with attributes of a certain value. This corresponds to searching for a noun phrase that includes adjectives. For example, some triplets in various orderings might be “mini sized amps” or “hammer colored red.” The agent must discover the order of the triplets. Thus, a common order variant occurs where the agent initially seeks the object (the O in OAV) and then uses the AVs for comparison and search refinement—for example, bolt cutter followed by size, price, and availability. Even more common are searches for one or more V of a given type of O where the A is suppressed—for example, AA Eveready batteries, 1/2" no. 8 slotted screws, or desk chair. (In the last example, the first of the two consecutive nouns functions as an adjective.)

In addition to the subgrammar’s rules, we obviously must also derive a vocabulary for any given instantiation or catalog. In shopping domains, we can derive the vocabulary for objects and another for attribute-value pairs by crawling the relational catalog database’s fields and extracting all unique types: item names are object names, attributes are parameter names, and parameter settings are attribute values. We show this tool for crawling the catalog and extracting the knowledge-base values at the bottom of Figure 1. Using such a tool, we can readily construct the subgrammar and its vocabulary for any given electronic catalog. The KB lookup tables (at the base of the agent in Figure 1) store the result so the phrase parser’s transformation rules can use it to infer and insert O, A, and V labels onto the various terms of any given search phrase. At the bottom right of the agent box, Figure 1 shows other rules ($\Phi_r$) that the agent uses to infer OAV meanings from queries (for example, the rule about consecutive nouns).

It is not sufficient that the catalog’s extracted vocabulary covers 100 percent of a given shopping site’s lexicon if the site’s users are unaware of that lexicon or prone to misusing it. To help overcome such semantic impasses, the agent first stems the tokens in the query string, expands them for synonyms, and ultimately checks them for spelling (see bottom left of the agent in Figure 1). The dictionaries and KBs across the base of Figure 1 support these tasks.

### Comparison testing EQUIsearch

In Figure 1, we showed EQUIsearch as complementing commercial search engines for online catalogs. To illustrate that point, we empirically evaluated query performance both with and without the EQUIsearch agent for a major commercial search engine relevant to e-commerce catalogs.

In general, there are four ways to measure search engine performance: retrieval effectiveness metrics (such as recall and precision), user satisfaction measures, transaction log analysis, and the critical incident technique. Another article examines the last three of these.\(^12\) In this study, we used effectiveness metrics for our comparison.
Measuring query effectiveness

The literature on information retrieval defines effectiveness as a measure of a system’s ability to satisfy the user in terms of the retrieved items’ relevance or pertinence. Pertinence means aboutness and appropriateness. Ultimately, the user determines whether a document is pertinent; in this case, we do. The ubiquitous contingency table, Table 1 in this article, delineates the categories for retrieved and unretrieved items in a catalog.

For our purposes, we will use ratios of hits/retrieved and hits/pertinent. We define the ratio of hits to items retrieved as precision. We term the ratio of hits to pertinent items the recall. From Table 1, we see that

\[
\text{Precision} = \frac{\text{RP}}{\text{RP} + \text{RI}}
\]

and

\[
\text{Recall} = \frac{\text{RP}}{\text{RP} + \text{NP}}.
\]

Scale-up test results

To test whether NLQ can scale up, we deployed it at a market exchange. EqualFooting (www.equalfooting.com) is a B2B online marketplace for the maintenance, repair, and operations (MRO) sector. Basically, this means that EqualFooting sells industrial and construction supplies—something like a Home Depot for small contractors, but with an order of magnitude more products. The company’s official launch date was February 2000, and by June 2000 it was handling one million hits per day (by about 23,000 separate users daily). At this writing, EqualFooting’s catalog integrates almost 450,000 products from more than 2,000 sellers.

The company stores its catalog internally in an Oracle database. Users at this Web site now use the EQUIsearch NLQ agent, which uses a spell checker. Oracle’s database integrates almost 450,000 products from more than 2,000 sellers.

We performed these steps manually for the test site during the fall of 2000 and deployed the agent on 17 November 2000. Since then, it has been in continuous operation—24 hours, 7 days a week.

Comparison test results

For comparison testing, we ran three types of query strings as listed in Table 2. Nouns or objects are the item names or synonyms of those names. Noun-adjective pairs (object-attribute pairs) cover cases in which the user seeks to narrow the choices returned. Lastly, although most users are unaccustomed to using sentence format, we thought it was an important search category because we hope future users will be unaware of search engines’ previous constraints.

Statistics on current search habits indicate that users type about 2.3 words per query, with nouns being the most common search, noun-adjective pairs next most common, and multiword phrases being least common. We designed our test bank, shown in Table 2, to approximate this distribution of 2.3 words per query on average: 146 words + 63 queries = 2.3 words/query.

Table 3 compares effectiveness and timing statistics for the CQ search (via Oracle EQUIMedia) with those for the NLQ agent. We collected these results over EqualFooting’s intranet, so they don’t include Internet latency, but that would be identical for both methods. Inspecting the last row of Table 3 shows that, on average, the CQ search was nearly twice as fast as NLQ (0.5 versus 0.9 seconds per query). However, CQ tended to retrieve nearly three times as many hits, although its recall was not as good as NLQ’s (0.8 versus 1.0) and its precision was only half as good (0.5 versus 1.0). Furthermore, the time advantage of CQ search is relatively inconsequential, because most users can’t detect a half second of difference.

Consider also the first row of the Table 3, noun search, where we see NLQ holding a clear advantage across the board. Not only is NLQ more precise, it’s also faster. The only place CQ holds its own is in recall, where both methods are perfect in bringing back all
the pertinent items. This story changes for noun-adjective pair search (row 2), at least in terms of speed. Here, NLQ takes almost four times as long as CQ, although it still wins on precision. It should surprise no one that the NLQ is faster than CQ in noun search and slower in noun-adjective search. In noun or object search, CQ must dynamically sort and index the entire munge (many subfields denormalized), whereas NLQ need only sort through the item-name field, because it narrows the search by labeling the token. Likewise, in object-attribute search, CQ behaves the same as before, but NLQ causes Oracle interMedia to sort an additional set of fields (all those with attributes) and then do a soft join and eliminate items not in all parts of the join.

NLQ handles sentences (row 3) more quickly than word pairs (row 2), further confirming that the parsing processes are not very significant determinants of delay. A query’s specific terms drive the process. In this case, the noun-adjective pairs that the parser extracted from the sentences proved to be less common items in the database; hence the Oracle CQ engine could process them faster.

Results analysis
We intend the NLQ agent presented in this article to complement and improve search engines for online e-commerce shopping catalogs. Our results show that the agent improves the precision and recall of the search with no significant overall impact on response time. Several of the lessons we learned in deploying and testing EQUIsearch bear further discussion:

• Scaling up NLQ. To scale up NLQ, we did away with some of the less practical NLQ proposals in the literature, such as conversational feedback and explaining the query translation to be used prior to executing the query. Instead, EQUIsearch converses with the user in the fashion that CQ systems use: after the query, by displaying either hits or canned, limited explanations of query failures. Also, we implemented our agent as a Markovian decision process. The agent continues to work at a relatively large-scale market exchange, which previously operated with CQ search alone.

• CQ paves the way for domain-specific NLQ. Relational DBMSs ship with CQ features, but each enterprise must enable and fill in the stripping, synonym, and spelling dictionaries before CQ can work for its shopping catalog. The good news is that these are the same three dictionaries

<table>
<thead>
<tr>
<th>Search</th>
<th>Total retrieved</th>
<th>Precision</th>
<th>Recall</th>
<th>Two-run average time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CQ</td>
<td>NLQ</td>
<td>CQ</td>
<td>NLQ</td>
</tr>
<tr>
<td>31 nouns</td>
<td>2,965</td>
<td>1,091</td>
<td>0.6</td>
<td>1.0</td>
</tr>
<tr>
<td>18 noun-adjective pairs</td>
<td>160</td>
<td>66</td>
<td>0.6</td>
<td>1.0</td>
</tr>
<tr>
<td>14 sentences</td>
<td>1</td>
<td>71</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Average (for 63 queries)</td>
<td>1,580</td>
<td>571</td>
<td>0.5</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 3. Timing and effectiveness results for CQ search versus the NLQ agent.
that NLQ needs. If a site has already developed them and deployed CQ, it takes only minimal extra effort to deploy NLQ.

- **Data-cleansing obstacles remain for any search method.** Numerous types, missing item names and other data, and poor-quality attribute information constitute possibly the most serious obstacle to scale up a search in any unified shopping catalog. This obstacle is not unique to NLQ, but equally plagues CQ and keyword search methods.

- **Speed differences are irrelevant in most cases.** It seems that NLQ is faster for noun search, but slower than CQ for noun-attribute pair searching. But in either case, the time differences are not statistically significant.

- **NLQ agent provides precision and recall improvement.** Our results to date reflect about a 50 percent improvement in precision when NLQ is added to CQ. This means that users experience shortened retrieval sets and that the items retrieved include far fewer false positives. Additionally, there are fewer false negatives or relevant items omitted.

- **NLQ agents offer parsing services CQ can’t provide.** Many shopping sites have begun to add chatterbots, such as Dell’s AskDudley (http://support.dell.com/us/en/askdudley), that provide navigation help and answer site or content questions in natural-like language. The results to date indicate that users like this type of self-service help. When it is present, they build up a higher expectation that the catalog search will behave in a similarly naturalistic way, and they no longer limit their queries to the short keyword format (the 2.3-word average mentioned earlier). They pose English-like sentences and questions to the catalog search engine, and it seems they are frustrated by the CQ search engines’ inability to parse their questions. NLQ agents such as EQUI-search appear to be the answer for shopping sites facing this dilemma.

The difficulty of searching online shopping catalogs is the difficulty of the Web in general: content created for human consumption poses a challenge for machine interpretation and use by other agents. To address this larger challenge, DARPA and the W3C have significant efforts under way to create the Semantic Web—tagging the semantics of the Web so that content created by laypersons for use by other people is automatically and transparently marked up for machine and agent interpretation and use. The process draws on tools such as the W3C’s RDF, the DARPA agent markup language (DAML), and the ontology interface language (OIL). Although we did not use this tool set in the work we report here, having it available and scaled up for industrial use could integrate catalogs into the Semantic Web. Potentially, this could ease the content interpretation of market catalogs.

The Semantic Web efforts do not include research on NLQ, so it would also be interesting and challenging to explore how to extend natural language agents such as those we describe in this article to work in the broader domain of the Semantic Web. In general, such agents operate in narrow domains only, so the challenge would be to populate the Web with many such special-purpose agents.

**References**


