# AN APPLICATION OF MACHINE LEARNING TECHNIQUES TO INTERACTIVE, CONSTRAINT-BASED SEARCH

A Thesis Presented to the Faculty of the Graduate School
University of Missouri – Columbia

In Partial Fulfillment
Of the requirements for the degree

Master of Science

By

CHRISTOPHER W. HARBERT

Dr. Yi Shang, Thesis Supervisor

DECEMBER 2005

© Copyright by Christopher W. Harbert 2005
All Rights Reserved

The undersigned, appointed by the Dean of the Graduate School, have examined the thesis entitled,

AN APPLICATION OF MACHINE LEARNING TECHNIQUES TO INTERACTIVE, CONSTRAINT-BASED SEARCH

Presented by Christopher W. Harbert

A candidate for the degree of Master of Science

And hereby certify that in their opinion it is worthy of acceptance.

Dr. Yi Shang

Dr. Hongchi Shi

Dr. Dominic Ho

## ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Yi Shang, for being both an excellent teacher and great mentor; Dr. Hongchi Shi and Dr. Dominic Ho for evaluating my work; and Karen Kraus, for providing motivation, constructive criticism, and sound eBay knowledge.

AN APPLICATION OF MACHINE LEARNING TECHNIQUES TO INTERACTIVE, CONSTRAINT-BASED SEARCH

Christopher W. Harbert

Dr. Yi Shang, Thesis Supervisor

## ABSTRACT

Search engine users frequently place additional constraints on search results that are not included in the user's original query. To respond to these additional constraints, search engine designers frequently add an "advanced search" page. On these pages, the user supplies a set of constraints for the result items. While this is certainly more useful, it relies on two assumptions: that the user knows these constraints prior to the search, and that the constraints are independent. This is not always the case.

This work presents a method to use an existing search engine to create an interactive, constraint-based search: the Query Expansion and Refinement Process (QUERP). In addition, this work provides an example of the method as applied to the popular eBay auction site. The experimental results show that using QUERP to provide an interactive, constraint-based search has the potential to provide higher precision and recall than the original search engine.

## LIST OF FIGURES

## LIST OF TABLES

## CONTENTS

# 1 Introduction

Most eBay users know what they are trying to find. They may be looking for a "green lamp", a "Nolan Ryan baseball card", or almost anything else a person can sell. So typically, the eBay user visits the site, types in "green lamp", and then reads through the results, one-by-one. Frequently the user has additional constraints on the item that are not expressed in the original query. These additional constraints may be price, location, who the seller is, or something else. Because the search engine does not allow the user to easily express these constraints, the user is forced to wade through many results that do not meet his or her criteria.

One approach to this problem is to provide the user with an "advanced search". In an advanced search, the user specifies additional criteria to be placed on the query. Some examples of advanced searches are Google's "Advanced Search" and eBay's "Search: Find Items" [1] [2]. However, existing advanced search interfaces rely on two assumptions: that the user knows what these constraints are *prior* to performing the search, and that the constraints are *independent*. There are many cases where these assumptions are not valid.

Prior to the search, many users are not sure what specific constraints to place on the search. What the user considers a fair price for an item may depend on the price of similar items. In fact prior to the search a user may think one price is fair, and then after seeing the going price for similar items, *change his or her mind* about what constitutes a fair price.

The advanced search also assumes that search constraints are independent of each other. This is not always the case. On eBay sellers with highly positive feedback can generally sell items for a higher price than sellers with negative feedback or no feedback at all. This is likely because buyers feel more comfortable placing higher bids on auctions that have a lower amount of perceived risk. The result of this behavior is that an individual buyer may be searching for expensive items from reputable sellers and *at the same time* searching for inexpensive items from risky sellers. In this case the search constraints of “price” and “seller feedback” are not independent. Figure 1 displays the options the advanced search makes available to users. In the advanced search, the seller may choose to find (Q1) only inexpensive items, (Q2) only items from risky sellers, (Q3) only expensive items, (Q4) only items from reputable sellers, (Q5) only items from risky sellers at a low price, or (Q6) only items from reputable sellers at a high price. While this seems like a fair number of options, consider the case where the user is searching for both inexpensive items from risky sellers as well as more expensive items from reputable sellers. Then if relevant items are evenly distributed between these two classifications, all six queries can hope at best to achieve 50% recall, regardless of the constraints chosen. Moreover, if the user opts not to provide any constraints at all, the user can hope at best to achieve 50% precision (since expensive items from risky sellers and inexpensive items from reputable sellers are both included). Admittedly it is unlikely that a user would not want inexpensive items from reputable sellers, but this is just one example of how ignoring constraint dependencies can automatically decreases either precision or recall.

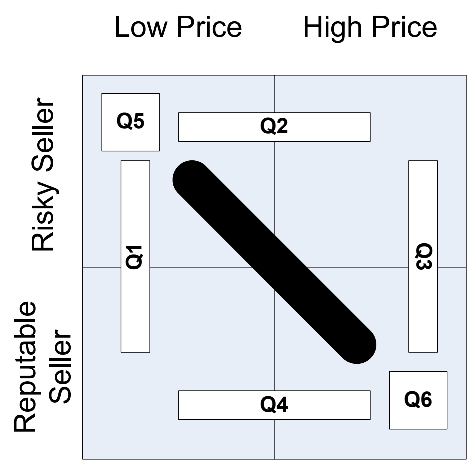


**Figure 1 – Possible Advanced Search Queries**

A third drawback of the traditional information retrieval search is that user supplied queries are often ambiguous. One user may search for “bike” and really mean “bicycle”, while another user may search for “bike” and really mean “motorcycle”. The traditional search must either give the user results for both “motorcycle” and “bicycle”, or give the user results for only the more popular sense of the term “bike”.

In the interactive, constraint-based search (ICBS), the user’s constraints are not specified or assumed prior to the search. Instead the user is presented with a set of result items that is representative of the various senses of the query. In the example of “bike”, the user may be presented with the more popular matches for “motorcycle” and the more popular matches for “bicycle”. However, once these options are presented, the user selects those items that are more relevant. Through this interactive process, the search engine is able to learn the user’s constraints *during* the searching process.

This work presents a general method to use a search engine’s original results to create an interactive, constraint-based search. This method is called the Query Expansion and Refinement Process (QUERP). QUERP consists of two phases: (1) query expansion and generation, and (2) query refinement. In the query expansion and generation phase, the user's original query is translated into multiple semantically related queries. A semantically related query is a query that is closely related in meaning to the original query. For example, "red bike" may be converted to "red bicycle", "red motorcycle", "maroon bike", etc. In the query refinement phase, the user continually interacts with the search engine to provide user feedback. The user feedback is used to provide more relevant results, and the more relevant results are then used for the next round of feedback. So if the user selects a "red motorcycle" as an interesting item, the new results for the user will more likely be motorcycles instead of bicycles.

In addition to QUERP, this work provides an example of the technique as applied to the popular eBay Auction Site. The Semantic Auction Metasearch Engine (SAME) is a desktop application that uses QUERP in order to provide the user with more relevant auction items. The experimental results from testing SAME will demonstrate that using QUERP to provide an interactive, constraint-based search has the potential to provide higher precision and recall than the original search engine.

Specific cases where a QUERP-based engine may outperform the original search engine are:

1. The constraints the user has placed on the search have interdependencies (e.g. seller feedback influences the user's notion of a fair price).
2. The user's query is not the more common sense of the word (e.g. for most people bike means a bicycle, but the user is actually looking for a motorcycle).

3. The user places extra value on items that were not included in the original search engine’s results (e.g. a hard to find item on eBay is less likely to be bid on, so the user may get a better price).
4. The user’s preferences are not constant. After seeing some results, the user changes his or her mind about what’s interesting (e.g. after seeing the average price for an item, the user changes his or her mind on what constitutes a fair price).

# 2 Related Work

## 2.1 Decision Trees

The induction of decision trees has proven to be a highly flexible, robust, and efficient means of classification. The purpose of decision tree induction is to solve the general classification problem. That is, to find a function, $f : E \to C$, where E is the set of examples to be classified and C is the set of possible classifications, such that $f(e) = c$ is the proper classification, c, of an example, e. In order to do this, a tree structure is built from a set of training examples. This tree structure is then used to classify each example.

A *decision tree* is a tree consisting of non-leaf nodes, called attribute nodes, leaf nodes called classification nodes, and edges labeled with predicates. Each attribute node is labeled with an attribute that will be used for classification. Each edge coming from the attribute node represents a predicate test for that attribute. The classification nodes (leaf nodes) are labeled with a classification. This classification applies to all examples that satisfy the predicates in the path from the root node to the classification node. Figure 2 shows an example of how the decision tree is used for classification. In this case, e is the example being classified. The attributes used for classification are A1, A2, A3, and A4. The corresponding values of the attributes for e are b, d, a, and e. The classification begins at the root. Since A1=b is true, the classification continues at A2. Following the edges from root node to classification node, the final classification of e is found to be C2.

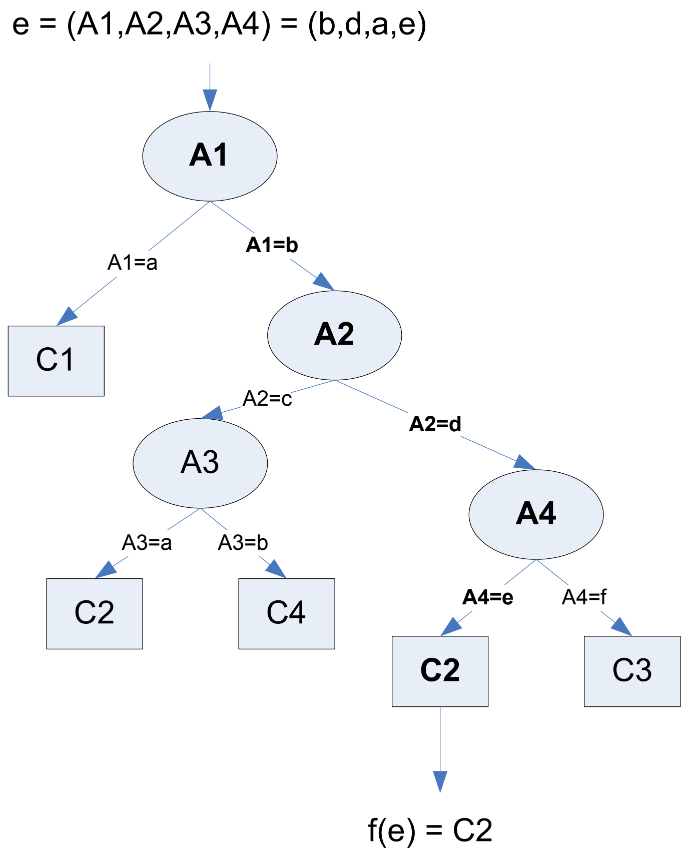


**Figure 2 – Classification of an Example Using a Decision Tree**

There are a variety of methods available for the induction of such decision trees, such as ID3 and C4.5 [3] [4]. In Quinlan's ID3 algorithm, the attribute with the highest information gain is chosen to be the root of the decision tree. This has the desirable attribute of splitting the tree into roughly equal parts. Information gain is defined as

$$\mathrm{gain}(\mathrm{a_i}) = I(p,n) - E(a_i)$$

where I(p,n) is the entropy of the total set of examples, and $E(a_i)$ is the expected information contained by attribute, $a_i$. The expected information contained by attribute, $a_i$, is defined as

$$E(a_i) = \sum_{j=1}^{|V_i|} \frac{p_{ij} + n_{ij}}{p+n} \cdot I(p_{ij}, n_{ij}) .$$

The entropy of two values x and y is defined as,

$$I(x,y) = \begin{cases} \text{0, if x=0} \\ \text{0, if y=0} \\ -\frac{x}{x+y}\log\left(\frac{x}{x+y}\right) - \frac{y}{x+y}\log\left(\frac{y}{x+y}\right) \end{cases} .$$

After an attribute is selected to be the root of the decision tree, the examples are divided into groups based on the predicates of the selected attribute. The algorithm is then performed recursively on the newly created branches. The pseudo-code of the ID3 algorithm is shown in Figure 3.

```
//
// The ID3 algorithm is a recursive algorithm that builds the tree
// based on entropy values.
//
ID3.BuildTree(attributes, examples)
{
  splitAttribute = attribute with highest gain();

  foreach possible value of splitAttribute
  {
    branchExamples = all examples that have the value for split attr.;
    remainingAttributes = attributes - splitAttribute;

    if (branchExamples are of same classification)
    {
      Return leaf node labeled with classification;
    }
    else
    {
      branchTree = ID3.BuildTree(remainingAttributes, branchExamples);
    }

    Add branchTree to root;
  }

}
```

**Figure 3 – The ID3 Algorithm**

In Quinlan's ID3 algorithm, the entire set of training examples used to construct the tree. Once the tree is learned, it is used to classify new examples. However, in the *online* classification problem, the user continually provides new training examples, and the decision tree must be updated to reflect the new examples. An obvious approach to using ID3 to solve the online classification problem is to add the new training example to the original set of training examples, and then completely reconstruct the ID3 decision tree. Schlimmer and Fischer call this method $\mathrm{ID3}'$ and recommend using it as a benchmark for online decision tree construction [5].

Utgoff presents an online decision tree building approach, ID5R, which produces equivalent decision trees to ID3 and is faster than $\mathrm{ID3}'$ in the worst case [6]. The ID5R algorithm differs from ID3 in that the counts of each positive and negative example are retained in the nodes of the decision tree. As each new example is learned, the aggregate data is then used to determine whether or not the decision tree needs to be restructured.

In QUERP, a decision tree is constructed from user feedback in order to help determine whether or not a search result will be interesting to the user. For simplicity of implementation, the Semantic Auction Metasearch Engine (SAME) uses the ID3′ algorithm for online decision tree induction. Although ID5R would be a better choice of implementation, ID3′ was chosen because of its simplicity and similarity to ID5R. Since both algorithms produce equivalent trees, the experimental results retain the same values (although they may take longer to retrieve).

## 2.2 Semantics Research

In May of 2001, Berners-Lee, Hendler, and Lassila published an article in Scientific American entitled "The Semantic Web" [7]. The general motivation for the Semantic Web is that information on the Internet is currently structured so that it is easily interpretable by people, but that if content were structured for machines instead, applications would be able to provide better information to users.

In traditional Internet documents, the text must be parsed and interpreted in order to gain meaning from the document. Through the use of Extensible Markup Language (XML) and Resource Description Frameworks (RDF), software applications are able to get meaning and access to data without interpreting the unstructured text. Using these technologies, web designers can encode meaning into their documents. Increasingly, companies are beginning to provide XML Web Services to their data.

What makes QUERP successful in the auction domain is that eBay provides additional semantic information about each auction item. This information takes the form of price, user feedback, time left in the auction, categories, etc. As the Semantic Web evolves, semantic information for more documents will become available. This additional semantic information will make QUERP applicable to more domains.

The Semantic Web is not the first time researchers have attempted to take advantage of the semantic meaning between concepts. WordNet is a lexical database created at Princeton that stores key relationships among words [8]. The WordNet database contains the various senses, synonyms, hypernyms, and hyponyms for each word in its database. For example, WordNet synonyms for “ball” include “globe” and “orb”; hypernyms include more generic terms like “game equipment”, “shot”, and “sphere”; and hyponyms include more specific types of balls like “baseball”, “basketball”, and “billiard ball”.

According to Mandala, Takenobu, and Hozumi, previous attempts to use WordNet for information retrieval purposes have generally been unsuccessful [9]. The authors claim these attempts to use WordNet failed because (1) interrelated terms may have different parts of speech, but are unrelated in WordNet; (2) many relationships between two words are not in WordNet; and (3) some terms are not included in WordNet (e.g. proper names). Their work shows that the automatic construction of a thesaurus may improve the recall and precision of IR systems that use WordNet.

This suggests that the automatic construction of a thesaurus may improve the results of QUERP, which uses the WordNet 2.1 database in order to generate semantically related queries for the original query.

## *2.3 Query Expansion/Refinement*

Query refinement is the process of changing an original query in order to improve the results. Adding additional terms to the search query or using relevance feedback are two ways to perform query refinement. Query expansion is the process of adding terms to the original query in order to refine the search results. Voorhees provides a method for query expansion using lexical-semantic relationships [10]. Voorhees's method uses the WordNet database to refine the query for use on the TREC database.

Voorhees's method for query expansion uses an extended vector space model for computing the similarity between documents and queries. In this model, the expanded query is represented as a vector, $Q = \langle Q_1, Q_2, ..., Q_k \rangle$, where each $Q_i$ represents the terms related to the query by a particular WordNet relationship. For example, the expanded query for "computer" could be

$$\begin{aligned} Q &= \langle Q_{synonyms}, Q_{hyponyms}, Q_{hypernyms} \rangle \\ &= \left\langle \begin{array}{l} \langle \text{computing machine, computing device} \rangle, \\ \langle \text{laptop, notebook} \rangle, \\ \langle \text{machine} \rangle \end{array} \right\rangle . \end{aligned}$$

The similarity between the document and the expanded query is then computed as,

$$sim(D,Q) = \sum_c \alpha_i \times D \cdot Q_i$$

where D is the document vector, $Q_i$ is the i$^{\text{th}}$ subvector of Q, and $\alpha_i$ normalizes the concept types by using the length of the original query terms. Voorhees's general findings were that query expansion, which is generally regarded as a recall enhancing technique, does increase recall, but at the cost of reducing precision.

QUERP uses WordNet for query expansion, but instead of generating a single expanded query, the expanded query is treated as multiple traditional queries. Voorhees's work suggests that this method should increase recall at the expense of precision, which is the intended behavior. The results are then merged, and machine learning is used to filter the merged results. The merging of results and addition of machine learning techniques improves both the precision and recall.

## *2.4 Interactive Searches*

In the interactive search, users provide feedback, which is in turn used to provide better search results. Dennis, Bruza, and McArthur provide an experimental study of three types of searches [11]. The work studies the traditional query search, keyword-based search, and Query-By-Navigation. In the traditional query search, the user enters a search query and receives results. In keyword-based search, the user instead navigates a directory of common keywords. In Query-By-Navigation, the user is presented with more discriminatory terms for the original query. The experimental study found that Query-by-Navigation does significantly increase the relevance of returned documents. This is similar to word sense disambiguation, which is another method of query refinement. Recent work has also shown that word sense disambiguation is a promising way to increase precision [12].

One previous attempt to use machine learning techniques for information retrieval is Inductive Query by Examples (IQBE) [13]. In IQBE, the user supplies a set of example documents and asks for similar documents. Chen, Shankaranararayanan, and She studied the use of ID3, Genetic Algorithms, and Neural Networks. In each of these examples, the user-provided documents are used, which represent a query, are used as a training set for the classifier. The classifier is then used to help retrieve relevant documents. The authors claim that both Genetic Algorithms and Neural Networks performed better than ID3. This suggests that SAME may be improved by using other machine learning techniques such as genetic algorithms or neural networks.

In QUERP, an interactive search is used to filter the expanded search results. QUERP differs from IQBE in that the training examples do not need to be provided up front, which makes QUERP more practical in many situations.

## *2.5 Automated Query Refinement*

Another approach to Query Refinement other than interactive searches is Automated Query Refinement (AQF). In Automated Query Refinement, the query is automatically changed in order to provide better results.

Carmel, Farchi, Petruschka, and Soffer present a method to perform Automatic Query Refinement (AQF) that uses lexical affinities [14]. A *lexical affinity* (LA) is a pair of closely related terms that uses a single query term. Let L be the set of all lexical affinities. That is, $L = \{(t_1, t_2) \mid (t_1 \in q \vee t_2 \in q) \wedge \neg(t_1 \in q \wedge t_2 \in q)$. Then the method of Carmel, et. al. selects the LA with the highest information gain. Information gain for lexical affinities is defined as,

$$IG(l) = H(D) - \left[ \frac{|D^+|}{|D|} H(D^+) + \frac{|D^-|}{|D|} H(D^-) \right]$$

where H(D) is the entropy of the set of documents, D, $D^+$ is the set of documents containing the lexical affinity, and $D^-$ is the remaining documents. Note that selection of lexical affinities is by the same method that’s used to select attributes in the ID3 decision tree algorithm.

QUERP is similar to this method of AQR in that maximal information gain is used to divide the document set in both methods. The key difference is that QUERP uses domain-specific attributes (e.g. price, time left in auction, seller feedback), whereas the lexical affinities method uses related terms that appear in the same document.

# 3 The Query Expansion and Refinement Process (QUERP)

## *3.1 Problem Specification*

Since existing search interfaces are not adequate for many situations, the problem is to find a way to build an interactive, constraint-based search on top of an existing search engine in order to deal with these situations.

Let I be the total set of items contained by the search engine. Then the goal of an information retrieval system is to find the relevance function, $r : I \rightarrow \{0,1\}$, such that $R = \{i, r(i) = 1\}$ is the set of all items the user finds relevant. Note that this is a specific example of the general classification problem.

When building on top of an existing search engine, the set of all items, I, contained by the search engine is often only partially accessible. This is due to a limitation on the number of items that a single query can obtain. That is, the items can only be obtained as subsets, $I_q$, where q is a query submitted to the original search engine.

The Query Expansion and Refinement Process is proposed as a solution to the problem of building an interactive, constraint-based search The pseudo-code for QUERP is listed in Figure 4.

```
// The Query Expansion and Refinement Process
QUERP(query)
{
  Initialize tree;
  Initialize examples;

  // Query Expansion and Generation
  queries = GenerateSemanticallyRelatedQueries(query);

  // Submit generated queries to original engine.
  expandedResults = RetrieveExpandedResults(queries);

  // Continuously learn User Interest Decision Tree and filter results.
  While (query is active)
  {
    interestingItems = GetMostInterestingItems(tree,
                           expandedResults, 10);
    Display interestingItems to user;
    User labels an item as "Interesting" or "Not Interesting";
    examples = LearnUserExample(item);
    tree = UpdateUserInterestDecisionTree(examples);
  }
}
```

**Figure 4 – The QUERP Process**

## 3.2 Query Expansion and Generation

The first phase in QUERP is query expansion and generation. The goal of the query expansion and generation phase is to increase the recall of the original search engine. In order to do this, the original query is expanded into multiple semantically related queries. A semantically related query is a query that has the same or similar meaning to the original query. For example, a query for "ball" is semantically related to "football", "soccer ball", "baseball", etc.

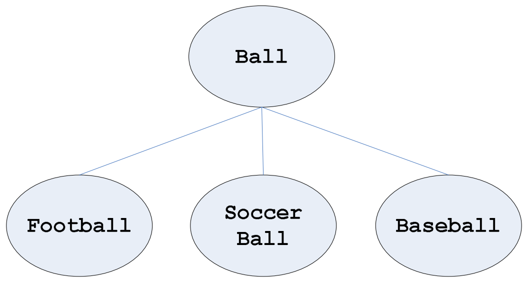


**Figure 5 – Query Expansion and Generation**

One method of generating these semantically related queries is to take advantage of the WordNet 2.1 database. The pseudo-code for generating the semantically related queries is listed in Figure 6.

```
//
// Returns the semantically related queries.
//
GenerateSemanticallyRelatedQueries(query)
{
  words = parsed words from query;
  // Generate all permutations of semantically related terms.
  Queries = GetPhrases("", words);
}
```

**Figure 6 – Generating Semantically Related Queries**

The query is first broken up into its individual terms. The array of terms is then passed to the GetPhrases() method, which gets all possible permutations of the query terms. GetPhrases() is implemented as follows,

```
GetPhrases(currentPhrase, remainingWords)
{
  if (remainingWords == 0)
  {
    return currentPhrase;
  }
  else
  {
    remainingWords.Remove(0);
    subphrases = GetPhrases(currentPhrase, remainingWords);
    relatedWords = GetRelatedWordsFromWordNet();

    foreach (word in relatedWords)
    {
      foreach (phrase in subphrases)
      {
        phrases.Add(word + " " + phrase);
      }
    }
}
```

**Figure 7 – Generating Related Phrases**

If there are no remaining terms, then the currently constructed phrase is returned as a possible phrase. Otherwise, the method removes the first term from the current set of terms and then recursively gets all the possible phrases for the remaining terms. The result is that all possible combinations of the search query are returned. While this is by no means a perfect method of query expansion, it does achieve the aim of increasing the recall of the search results. Although methods that rely on combinations tend to be inefficient, in practice this query expansion method is feasible.

Let m be the number of terms in the query, and let k be the maximum number of related words for any term in the query. Then the computational complexity of computing the semantically related queries is $O(m \cdot k^m)$. In practice, most user queries are relatively short, and the number of related words is also small. The number of queries generated is $O(k^m)$.

Once the expanded queries have been generated, the results for each of these queries are retrieved from the original search engine. The pseudo-code for retrieving the results is listed in Figure 8.

```
//
// Returns the expanded results for all semantically related queries.
// Results are interlaced, so that the best items of each query are
// displayed first.
//
RetrieveExpandedResults(queries)
{
  initialize results;
  initialize expandedResults;
  for I = 1 to |queries|
  {
    results[i] = GetResultsFromSearchEngine(queries[i]);
    if (|results[i]| >= maxResultCount)
    {
      maxResultCount = |results[i]|;
    }
  }

  for I = 1 to maxResultCount
  {
    for (j = 1 to |queries|)
    {
      expandedResults.Add(results[i][j]);
    }
  }

  return expandedResults;
}
```

**Figure 8 – Retrieving Expanded Results from Search Engine**

Each of the result sets is retrieved, and then the results are interlaced. Interlacing the results will give the user the top selections from each query. If the user selects an item from one query, it will become more likely that the user receives more items from the same query. This allows for query refinement.

Let n be the maximum number of results returned from any query. Since the number of queries generated is $O(k^m)$, then the computational complexity of retrieving the expanded results is $O(nk^m)$.

Consider the case where a query is ambiguous. Then, in order to provide a reasonable level of precision among search results and without knowing the user's preferences, the original search engine is limited to returning a certain number of results from each sense of the query. This is generally done by computing some metric that evaluates each item's popularity. The query expansion and generation step increases the recall (simultaneously decreasing precision) by combining the results from multiple senses of the ambiguous query. This is achieved by expanding the original query into a set of semantically related queries.

Let $I_q$ be the set of items returned by the original query, q. We first find, $Q = \{q_i, q_i \cong q\}$, which is the set of queries that are semantically related to q. Then we construct a set $\bar{I}$, such that $\bar{I} = \bigcup_{q_i \in Q} I_{q_i}$. This set represents the set of all semantically related items. The assumption is that $\bar{I}$ will in many cases contain some relevant items that are not in $I_q$.

Voorhees uses "query expansion" to refer to expanding the terms of a single query in order to make the query more precise. Voorhees has shown that this step alone does not increase precision or recall [10]. In QUERP, "query expansion and generation" is similar to Voorhees's notion of query expansion; however, the aim is not to make the queries more precise. In QUERP, the goal of the query expansion and generation phase is to actually make the results *less precise* in order to make the results have *higher recall*. Performing query expansion and generation alone would not suffice in an information retrieval system. The result would be a low precision rate and thus overwhelm the user with too many irrelevant items. Instead of stopping here, the expanded results are continuously filtered through an interactive machine learning approach.

## 3.3 User Feedback

Initially, the user is presented with the top ranking items from each query. The user then selects the options which he or she finds interesting. The user may choose to “watch” or “not watch” each item. If the user opts to “watch” an item, the item is moved to the user’s watch list. If the user instead opts to “not watch” an item, the item is removed from the search results. In each case, a training example is sent to the Decision Tree Ranking Engine. When an item is removed, a new item is retrieved from the ranking engine to replace the removed item. A “watch” item is considered a positive example for the decision tree, and a “not watch” item is considered a negative example. The user feedback is continuously used to provide better refinement of the expanded query results. Figure 9 shows the pseudo-code for processing learned examples.

```
// Performed after a user labels the item as “Interesting” or
// “Not Interesting”.
LearnUserExample(item)
{
  If (user labeled item as “Interesting”)
  {
    Add a positive example to the training set;
  }
  Else
  {
    Add a negative example to the training set;
  }

  UpdateTree();
}
```

**Figure 9 – Learning User Examples**

The algorithm for constructing the tree is shown in Figure 10.

```
//
// Rebuilds the User Interest Decision Tree using Quinlan's ID3
// algorithm.
//
UpdateTree(examples)
{
  return ID3.BuildTree(examples);
}
```

**Figure 10 – Rebuilding the Decision Tree**

If ID5R were used, the UpdateTree() method in Figure 10 would be replaced with ID5R's tree update method.

The computational complexity of rebuilding the tree is as follows. The ChooseSplitAttribute() method is $O(kn)$, where k is the number of attribute values, and n is the number of examples learned from the user. If d is the maximum number of values an attribute may have, there are at most d branches in the worst case. According to [6] the worst case for iteratively constructing the ID3 decision tree is

$$\sum_{i=1}^{d} i \cdot n \quad = n \cdot \frac{k \cdot (k+1)}{2} = O(n \cdot k^2) .$$

Since k is constant in QUERP, the running time is $O(n)$.

## 3.4 Query Refinement

In order to increase the precision of the expanded results, the User Interest Decision Tree filters the expanded results. The User Interest Decision Tree is maintained by the Decision Tree Ranking Engine. As each training example is received from the user, the entire ID3 decision tree is reconstructed from all previously seen examples. Completely reconstructing the tree takes a negligible amount of time and helps prevent over-fitting and bias of the tree construction. In the current implementation, the constructed ID3 tree provides a classifier based on seller feedback, time left in the auction, and price. Future implementations may include query and other attributes. Figure 11 is an example of what a User Interest Decision Tree may look like.

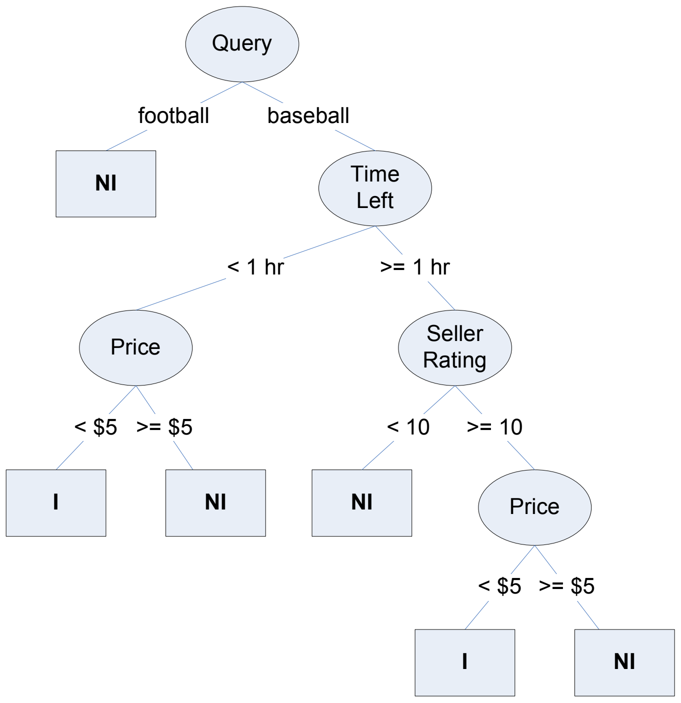


**Figure 11 – Possible User Interest Decision Tree**

Although related work suggests that other machine learning techniques may be more appropriate for information retrieval, ID3 was chosen because of its classical roots and simple implementation [13]. It was also chosen because the attributes provided in semantic data play an extremely large role in item selection. Decision trees are most effective in domains where appropriate attributes and their values are easily selected. Figure 12 shows the pseudo-code for retrieving the expanded search results from the original search engine.

```
//
// Filters the expanded results in order to display the n most
// “interesting” items.
//
GetMostInterestingItems(tree, expandedResults, n)
{
  Initialize mostInterestingItems;
  For i=1 to n
  {
    classification = tree.ClassifyItem(expandedResults[i]);
    if (classification is “interesting”)
    {
       mostInterestingItems.Add(expandedResults[i]);
    }
  }

  Return mostInterestingItems;
}
```

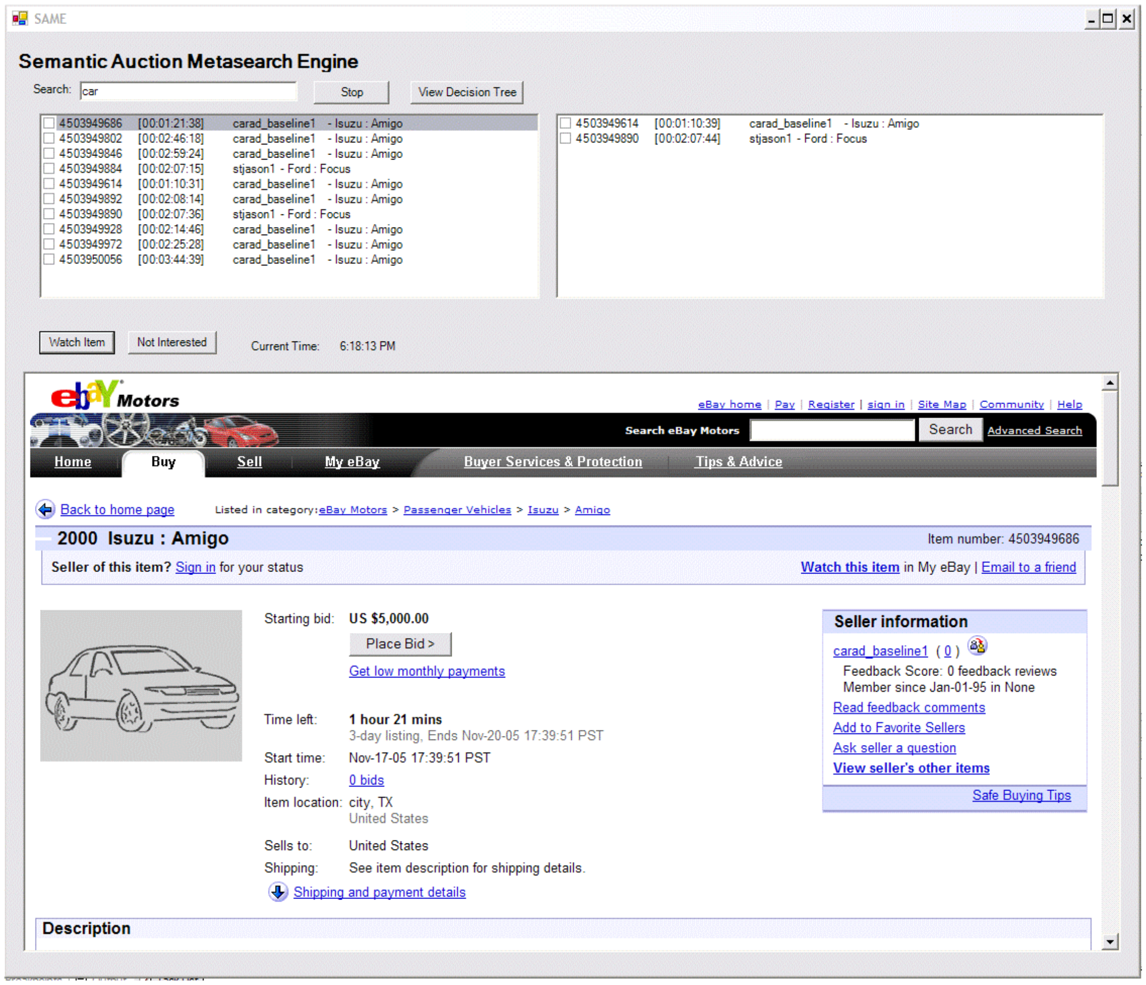
**Figure 12 – Getting the Interesting Items**

Decision tree classification is $O(\log(b))$, where b is the maximum number of values for an attribute. Therefore the computational complexity of GetMostInterestingItems() is $O(n \cdot \log(b))$. But since n is a constant (e.g 10 most interesting items) and b is a constant, the complexity is $O(1)$.

# 4 The Semantic Auction Metasearch Engine: An Application of QUERP

## *4.1 User Interface*

The user interface for the Semantic Auction Metasearch Engine (SAME) is a desktop application that allows the user to search for items on eBay (Figure 13). First, the user enters his or her search query and then clicks “Search”. This begins the search process, which uses QUERP to retrieve results from the eBay server. The ten “most interesting” items are shown to the user in the box on the left. Clicking on an item will load the eBay auction site for the item in the bottom window. For each item, the user can then choose to “Watch” the item, or select “Not Interested”.



SAME

Semantic Auction Metasearch Engine

Search: car

Stop | View Decision Tree

```
4503949686  [00:01:21:38]  carad_baseline1  - Isuzu : Amigo
4503949802  [00:02:46:18]  carad_baseline1  - Isuzu : Amigo
4503949846  [00:02:59:24]  carad_baseline1  - Isuzu : Amigo
4503949884  [00:02:07:15]  stjason1 - Ford : Focus
4503949614  [00:01:10:31]  carad_baseline1  - Isuzu : Amigo
4503949892  [00:02:08:14]  carad_baseline1  - Isuzu : Amigo
4503949890  [00:02:07:36]  stjason1 - Ford : Focus
4503949928  [00:02:14:46]  carad_baseline1  - Isuzu : Amigo
4503949972  [00:02:25:28]  carad_baseline1  - Isuzu : Amigo
4503950056  [00:03:44:39]  carad_baseline1  - Isuzu : Amigo
```

```
4503949614  [00:01:10:39]  carad_baseline1  - Isuzu : Amigo
4503949890  [00:02:07:44]  stjason1 - Ford : Focus
```

Watch Item | Not Interested | Current Time: 6:18:13 PM

ebay Motors

eBay home | Pay | Register | sign in | Site Map | Community | Help

Search eBay Motors | Search | Advanced Search

Home | Buy | Sell | My eBay | Buyer Services & Protection | Tips & Advice

Back to home page    Listed in category: eBay Motors > Passenger Vehicles > Isuzu > Amigo

**2000 Isuzu : Amigo**    Item number: 4503949686

**Seller of this item?** Sign in for your status    **Watch this item** in My eBay | Email to a friend

Starting bid: **US $5,000.00**

Place Bid >

Get low monthly payments

Time left: **1 hour 21 mins**
3-day listing, Ends Nov-20-05 17:39:51 PST

Start time: Nov-17-05 17:39:51 PST

History: 0 bids

Item location: city, TX
United States

Sells to: United States

Shipping: See item description for shipping details.

Shipping and payment details

**Seller information**

carad_baseline1 ( 0 )
Feedback Score: 0 feedback reviews
Member since Jan-01-95 in None

Read feedback comments
Add to Favorite Sellers
Ask seller a question
**View seller's other items**

Safe Buying Tips

**Description**

**Figure 13 – The Semantic Auction Metasearch Engine User Interface**

Clicking "Watch Item" will move the item to the list on the right (the interesting items list). Clicking "Not Interested" will remove the item from both lists. The user's selection of "Watch Item" or "Not Interested" is then used as a training example for the decision tree used in QUERP. The user interface is a Windows application that was developed in Microsoft C#.

In addition to the above interface, a Windows console application was also developed. This console application was used to obtain the experimental results.

## *4.2 System Architecture*

The Semantic Auction Metasearch Engine was developed using a layered architecture (Figure 14). The layered architecture is a commonly chosen architecture pattern in software development. This architecture was chosen to promote flexibility and reusability in the design. In a layered architecture, objects in one layer may only communicate with objects in adjacent layers. For example, an object in the User Interface Layer cannot call an object in the SAME Database directly. Both the SameExperiment Console Application and the SAME Desktop Application use the same Cwh.Same Class Library. Since all of the functionality is captured in Cwh.Same, different interfaces can easily share the same functionality.

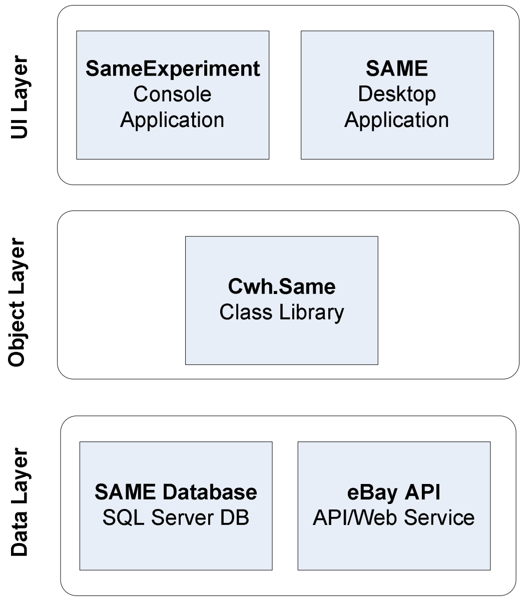


**Figure 14 – SAME Architecture**

The three layers of the Semantic Auction Metasearch Engine (SAME) are (1) the User Interface Layer, (2) the Object Layer, and (3) the Data Layer. The User Interface Layer consists of the SameExperiment Console Application and the SAME Desktop

Application. The console application is used to run experiments that test the implementation. The SAME Desktop Application provides the user interface shown in Figure 13.

The Object Layer contains the Cwh.Same class library. This class library encapsulates the functionality of SAME. The more important classes in Cwh.Same are shown in the simplified class diagram (Figure 15).

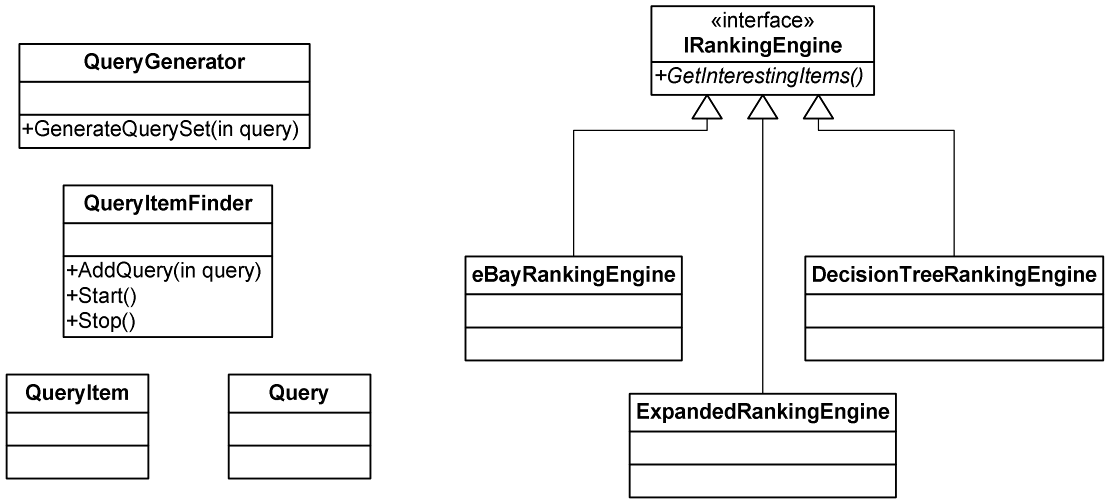


**Figure 15 - Simplified Class Diagram of Cwh.Same**

QueryGenerator implements the pseudo-code shown in Figures 6 and 7. The original query is supplied to the GenerateQuerySet() which uses WordNet to generate additional queries for the original query. QueryItemFinder implements a thread that will call the eBay API to search items and then store them in the SAME Database. The AddQuery method is used to add to the list of queries the QueryItemFinder will search. Query represents actual queries and QueryItem represents the results for a particular query.

The various ranking engines implement the IRankingEngine interface. The IRankingEngine interface requires a ranking engine to return interesting items in the GetInterestingItems() method. The EBayRankingEngine simply takes the results that return for the original query from the eBay API. The ExpandedRankingEngine takes the results that return from the original query in addition to the results that return from the semantically related queries. The DecisionTreeRankingEngine implements both phases of QUERP and returns only the results from the ExpandedRankingEngine that are classified as “interesting” by the User Interest Decision Tree.

The Data Layer contains the SAME Database and the eBay API. The SAME Database contains tables for persisting the seller information, query information, and query item information. This information is persisted using NHibernate, an open-source Object Relational Mapping solution for .NET [15].

# 5 Experimental Results

## 5.1 Experimental Setup

Ten common queries where chosen at random from a list of common eBay keywords [16]. Then for each query, results from the Decision Tree Ranking Engine, eBay Ranking Engine, and Expanded Results Engine were compared. The metrics used to evaluate each engine were precision and recall.

Precision is defined as

$p = \frac{|R_{rel}|}{|I|}$, where $R_{rel}$ is the set of relevant items returned, and I is the set of all items returned.

Recall is defined as

$r = \frac{|R_{rel}|}{|R|}$, where $R_{rel}$ is the set of relevant items that were returned and R is the set of all relevant items.

For each query, the expanded result set was divided into those items with an even item id and those items with an odd item id. This roughly divides the expanded result set into two sets, which were then used for the training set and the testing set. In order to simulate user input, a random ground truth decision tree was created for each query. The tree was then used to assign ground truth labels to each item in the testing set. The ground truth tree represents a potential strategy that a buyer may have. The randomly generated tree is similar to the tree shown in Figure 11.

## *5.2 User Interest Decision Tree Induction*

Each item in the training set was processed individually. At each step, the ground truth tree is used to assign a label to the training set item. If a “watch” label is provided, a positive example is sent to the decision tree ranking engine. If a “not watch” label is assigned, a negative example is sent. The currently learned decision tree is then reevaluated for precision and recall against the testing set. Note that the learned decision tree may have a different tree structure from the ground truth tree. The induction of the decision tree for a query, “GMC”, is shown in Table 1. As each positive or negative example is learned, the precision and recall for the top n items is shown. Note that as the first and second negative examples are learned, the precision and recall both improve upon the original search results. The final results for all three engines are shown in Table 2.

**Query Text:** GMC

**Precision and Recall While Building the Decision Tree**

| Pos. Ex. | Neg. Ex. | P5 | R5 | P10 | R10 | P15 | R15 | P20 | R20 | P25 | R25 | P30 | R30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1.00 | 0.28 | 0.90 | 0.50 | 0.93 | 0.78 | 0.90 | 1.00 | 0.90 | 1.00 | 0.90 | 1.00 |
| 2 | 0 | 1.00 | 0.28 | 0.90 | 0.50 | 0.93 | 0.78 | 0.90 | 1.00 | 0.90 | 1.00 | 0.90 | 1.00 |
| 3 | 0 | 1.00 | 0.28 | 0.90 | 0.50 | 0.93 | 0.78 | 0.90 | 1.00 | 0.90 | 1.00 | 0.90 | 1.00 |
| 3 | 1 | **1.00** | **0.28** | **0.90** | **0.50** | **0.93** | **0.78** | **0.95** | **1.00** | **0.95** | **1.00** | **0.95** | **1.00** |
| 4 | 1 | 1.00 | 0.28 | 0.90 | 0.50 | 0.93 | 0.78 | 0.95 | 1.00 | 0.95 | 1.00 | 0.95 | 1.00 |
| 5 | 1 | 1.00 | 0.28 | 0.90 | 0.50 | 0.93 | 0.78 | 0.95 | 1.00 | 0.95 | 1.00 | 0.95 | 1.00 |
| 5 | 2 | **1.00** | **0.28** | **1.00** | **0.56** | **1.00** | **0.83** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |
| 6 | 2 | 1.00 | 0.28 | 1.00 | 0.56 | 1.00 | 0.83 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 7 | 2 | 1.00 | 0.28 | 1.00 | 0.56 | 1.00 | 0.83 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 8 | 2 | 1.00 | 0.28 | 1.00 | 0.56 | 1.00 | 0.83 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 9 | 2 | 1.00 | 0.28 | 1.00 | 0.56 | 1.00 | 0.83 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 9 | 3 | 1.00 | 0.28 | 1.00 | 0.56 | 1.00 | 0.83 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 9 | 4 | 1.00 | 0.28 | 1.00 | 0.56 | 1.00 | 0.83 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 10 | 4 | 1.00 | 0.28 | 1.00 | 0.56 | 1.00 | 0.83 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 11 | 4 | 1.00 | 0.28 | 1.00 | 0.56 | 1.00 | 0.83 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 12 | 4 | 1.00 | 0.28 | 1.00 | 0.56 | 1.00 | 0.83 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |

**Table 1 - Induction of user interest tree for "GMC" query.**

| Engine | P5 | R5 | P10 | R10 | P15 | R15 | P20 | R20 | P25 | R25 | P30 | R30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Decision Tree Engine | 1.00 | 0.28 | 1.00 | 0.56 | 1.00 | 0.83 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| eBay Engine | 1.00 | 0.28 | 0.90 | 0.50 | 0.93 | 0.78 | 0.90 | 1.00 | 0.90 | 1.00 | 0.90 | 1.00 |
| Expanded Results Engine | 1.00 | 0.28 | 0.90 | 0.50 | 0.93 | 0.78 | 0.90 | 1.00 | 0.90 | 1.00 | 0.90 | 1.00 |

**Table 2 - Final Results for "GMC" query.**

## *5.3 Precision and Recall*

In most cases, the experimental results confirm the hypothesis that the expanded results have lower precision, but higher recall. The results also confirm that the Decision Tree Engine produces both higher precision and higher recall. Table 3 shows the final results for the "heart" query. While the original search engine retrieves only relevant items in the top 25 and top 30 results, it also has lower recall than the expanded results. This is because the original search engine actually retrieves less than 25 items for the original query, while the expanded results engine continues to find relevant items. The expanded results engine also includes some irrelevant items, but the Decision Tree Engine is able to replace these irrelevant items with relevant ones, increasing both precision and recall. Similar results are shown for the "yarn" query in Table 4.

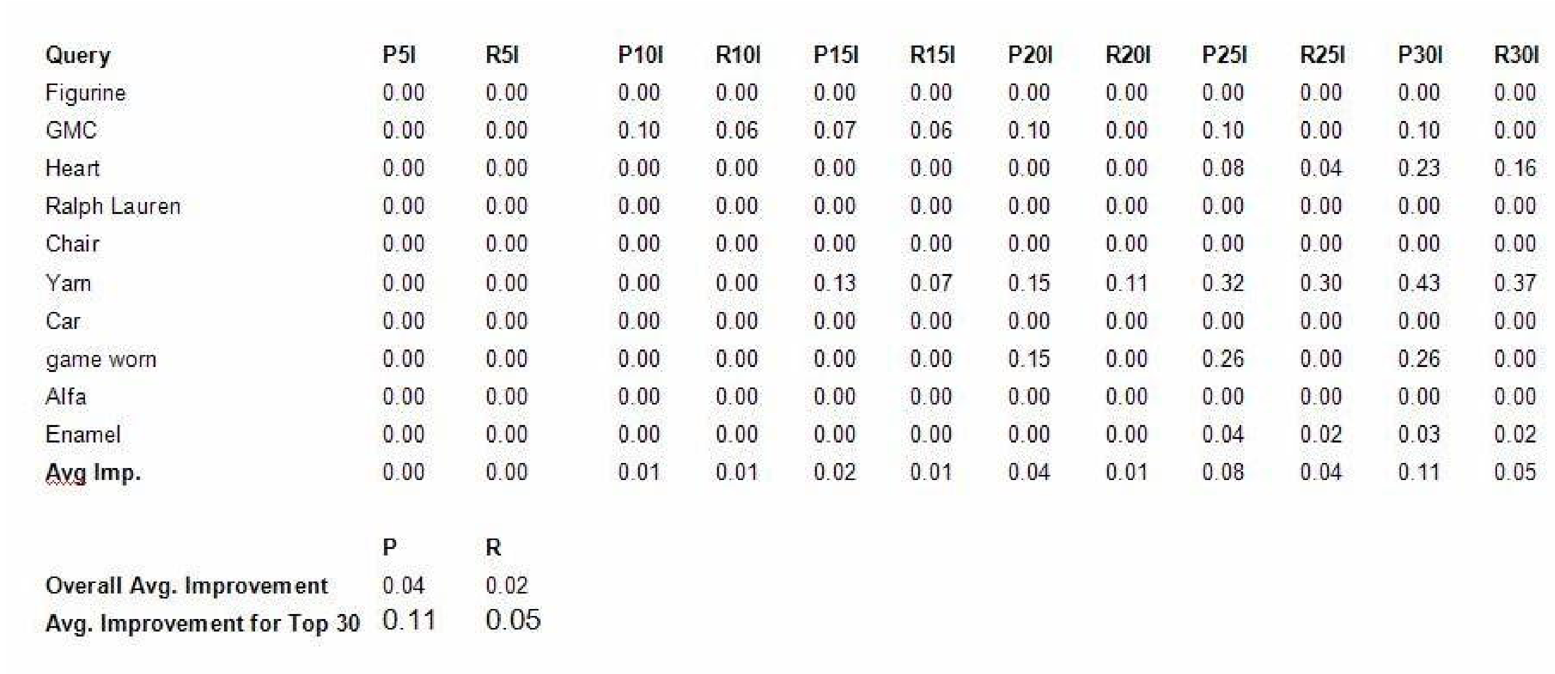
| Query Text: | heart |  |
|---|---|---|
| **Engine** | **P5** | **R5** |
| Decision Tree Engine | 1.00 | 0.11 |
| eBay Engine | 1.00 | 0.11 |
| Expanded Results Engine | 1.00 | 0.11 |

**Table 3 - Final Results for "heart" query.**

| Query Text: | yarn |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Engine** | **P5** | **R5** | **P10** | **R10** | **P15** | **R15** | **P20** | **R20** | **P25** | **R25** | **P30** | **R30** |
| Decision Tree Engine | 1.00 | 0.19 | 1.00 | 0.37 | 1.00 | 0.56 | 1.00 | 0.74 | 1.00 | 0.93 | 1.00 | 1.00 |
| eBay Engine | 1.00 | 0.19 | 1.00 | 0.37 | 1.00 | 0.48 | 1.00 | 0.48 | 1.00 | 0.48 | 1.00 | 0.48 |
| Expanded Results Engine | 1.00 | 0.19 | 1.00 | 0.37 | 0.87 | 0.48 | 0.85 | 0.63 | 0.68 | 0.63 | 0.57 | 0.63 |

**Table 4 - Final Results for the "yarn" query.**

The improvements over the original search engine for the ten randomly selected queries are shown in Table 5. Some queries showed no improvement over the existing engine (“figurine”, “Ralph Lauren”, and “alfa”). In the case of “figurine”, all 30 results from the original engine were found to be relevant. The Decision Tree Ranking Engine in this case could not improve on the results and performed just as well. In the cases of “Ralph Lauren” and “alfa”, no WordNet entries were available for the terms, so the result set was not expanded.



| Query | P5I | R5I | P10I | R10I | P15I | R15I | P20I | R20I | P25I | R25I | P30I | R30I |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Figurine | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| GMC | 0.00 | 0.00 | 0.10 | 0.06 | 0.07 | 0.06 | 0.10 | 0.00 | 0.10 | 0.00 | 0.10 | 0.00 |
| Heart | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.08 | 0.04 | 0.23 | 0.16 |
| Ralph Lauren | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Chair | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Yarn | 0.00 | 0.00 | 0.00 | 0.00 | 0.13 | 0.07 | 0.15 | 0.11 | 0.32 | 0.30 | 0.43 | 0.37 |
| Car | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| game worn | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.15 | 0.00 | 0.26 | 0.00 | 0.26 | 0.00 |
| Alfa | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Enamel | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.04 | 0.02 | 0.03 | 0.02 |
| **Avg Imp.** | 0.00 | 0.00 | 0.01 | 0.01 | 0.02 | 0.01 | 0.04 | 0.01 | 0.08 | 0.04 | 0.11 | 0.05 |

|  | P | R |
|---|---|---|
| **Overall Avg. Improvement** | 0.04 | 0.02 |
| **Avg. Improvement for Top 30** | 0.11 | 0.05 |

**Table 5 - Improvements over Original Search Engine**

Overall, the average improvement for the top n items was 4% precision and 2% for recall. The average improvement for the top 30 items was 11% precision and 5% recall.

## *5.4 Evaluation of Experimental Results*

These results do show an improvement over the original search engine. And on average, the user will be able to find relevant items that were not included in the original results. It is probably not the best testing methodology to use a randomly generated decision tree. This methodology was chosen because it is practical. The methodology relies on the assumption that a user's interests can be captured by a decision tree in the first place. Although relying on this assumption, these results do suggest that further evaluation may be valuable. In addition to further testing, the results may also be improved by using the query as an attribute in the decision tree.

# 6 Conclusion

## 6.1 Summary of Work

This work presents a new method of using an existing search engine to create an interactive, constraint-based search. While the results of this work are not conclusive enough to suggest that this method will outperform existing information retrieval techniques in every case, they are significant enough to justify further work along these lines. The two major conclusions to be drawn from this work are (1) that the addition of semantic information to Internet documents warrants a reevaluation of machine learning techniques for information retrieval purposes, and (2) search engine designers should consider implementing interactive, constraint based searches in addition to advanced search pages.

## 6.2 Future Work

### 6.2.1 Replacing ID3′ with Another Machine Learning Technique

ID3′ was chosen because of its ease of implementation. In order to improve the speed of the engine, ID3′ can be replaced with ID5R, which builds equivalent trees in less time [6]. Additionally, Chen, Shankaranararayanan, and She have shown that other machine learning techniques may outperform ID3 for the purposes of information retrieval [13]. Future work may consider implementing SAME or another QUERP-based engine using a different machine learning technique.

### 6.2.2 Replacing WordNet with another Semantic Word Database

While WordNet has been very useful in many research projects, the database is limited by the number of terms it contains. Many companies, brands, and specific item details are not included. The construction of another semantic word database that contains terms more relevant to eBay could further increase the precision and recall of SAME.

### 6.2.3 Using AJAX for an Online User Interface

Asynchronous JavaScript Technology and XML, better known as AJAX, has led to a new paradigm in web development. A good overview of AJAX is available at [17]. In AJAX web development, requests are periodically made to the server in the background. This allows additional information to be retrieved from the server, based on a user's actions, without refreshing the page. AJAX would be an excellent choice of technology for implementing an online, QUERP-based search engine.

# BIBLIOGRAPHY

[1] *Advanced Search.* 2005. Google. 26 Nov. 2005 <http://www.google.com/advanced_search?hl=en>.

[2] *Search: Find Items*. 2005. eBay. 26 Nov. 2005 <http://search.ebay.com/ws/search/AdvSearch>.

[3] Quinlan, J. R. “Induction of Decision Trees.” *Machine Learning* 1 (1986): 81-106.

[4] Quinlan, J.R. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kauffman, 1993.

[5] Schlimmer, J.C. and Fisher, D. “A Case Study of Incremental Concept Induction.” *Proceedings of the Fifth National Conference on Artificial Intelligence* Philadephia, PA: Morgan Kaufmann, 1986. 496-501.

[6] Utgoff, P. “Incremental Induction of Decision Trees” *Machine Learning* 4.2 (1989): 161-186.

[7] Berners-Lee, T., Hendler, J., Lassila, O. “The Semantic Web”, *Scientific American* 17 May 2001: 34-43.

[8] Miller, G.A., Beckwith, R., Felbaum, C., Gross, D., and Miller, K. “Introduction to WordNet: An On-line Lexical Database.” *International Journal of Lexicography* 3.4 (1990): 235-244.

[9] Mandala, R., Takenobu, T., and Hozumi, T. “The Use of WordNet in Information Retrieval” *Proceedings of the COLING/ACL Workshop on*

*Usage of WordNet in Natural Language Processing Systems, Montreal*. 1998. 31-37.

[10] Vorhees, E. “Query Expansion using Lexical-Semantic Relations” *Proceedings of the 17th Annual International AC SIGIR Conference on Research and Development in Information Retrieval, Dublin, Ireland*. 1994. 61-69.

[11] Dennis, S., Bruza, P., and McArthur, R. “Web Searching: A Process-Oriented Experimental Study of Three Interactive Search Paradigms” *Journal of the American Society for Information Science and Technology* 53.2 (2001): 120-133.

[12] Stokoe, C., Oakes, M. and Tait, J. “Word Sense Disambiguation in Information Retrieval Revisited.” *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Toronto, Canada 28-1 Aug. 2003*.

[13] Chen, H., Shankaranararayanan, G., She, L. “A Machine Learning Approach to Inductive Query by Examples: An Experiment Using Relevance Feedback, ID3, Genetic Algorithms, and Simulated Annealing.” *Journal for the American Society of Information Science* 49.8 (1998): 693-705.

[14] Carmel, D., Farchi, E., Petruschka, Y., Soffer, A., “Automatic query refinement using lexical affinities with maximal information gain.” *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Tampere, Finland 11-15 Aug. 2002.*

[15] *NHibernate Home*. 2005. Confluence. 23 Nov. 2005 <http://nhibernate.org>.

[16] *eBay Keywords*. 2005. eBay. 23 Nov. 2005 <http://buy.ebay.com/>.

[17] Murray, G. "Asynchronous JavaScript Technology and XML (AJAX) With Java 2 Platform, Enterprise Edition" 9 June 2005. *Sun Developer Network*. 26 Nov. 2005 <http://java.sun.com/developer/technicalArticles/J2EE/AJAX/index.html?cid=59754>.