V-Twin: A Lightweight Engine for Interactive Use

Daniel E. Rose Curt Stevens

Apple Research Laboratories
Apple Computer, Inc.
One Infinite Loop, MS 301-4A
Cupertino, CA 95014

{rose|curt}@apple.com

Abstract

This paper describes V-Twin, an information access toolkit designed to provide indexing and search capabilities for a variety of applications. We discuss the phenomenon of very short queries generated by users of interactive search services, and summarize a new technique we are using in V-Twin to handle these queries more effectively. We then present some results based on V-Twin's performance at the TREC-5 ad hoc task. V-Twin achieved a high level of performance despite having much lower index overhead and memory footprint than other systems participating in TREC.

1. V-Twin Background

V-Twin is the code name of the Apple Information Access Toolkit (AIAT), originally created in Apple Research Labs and now available for licensing by Apple developers. The toolkit consists of a library of C++ classes designed to be linked to a front-end application. V-Twin is based on the standard vector model of information retrieval, with a variant of *tf x idf* weighting and length normalization. V-Twin was developed not only to facilitate our own research on experimental information access systems but also to be used in Apple's products[1] and those of our developers. In order to meet the needs of this diverse community, it was designed to be scalable with respect to speed, disk usage, and in particular, memory usage.

One of the primary applications we envisioned for V-Twin was full-text relevance-ranked searching as part of the Macintosh Finder (the file system interface). This meant that it had to work effectively on all the configurations of Macintosh offered by Apple. At the time of development this included laptops and portables based on both RISC-based PowerPC and the older Motorola 68K families of microprocessors, and as little as 8 MB of memory.

[1] More information about V-Twin product plans is available on the world-wide web at http://www.research.apple.com/research/tech/V-Twin.

Index overhead had to be kept low (between 15 and 30% of the source text, depending on which features of the system were used), and the indexes had to support in-place updating. Since low-end users of Macintosh computers typically do not use virtual memory, V-Twin-based applications are able to constrain the amount of memory used for indexing; a larger memory allocation produces better indexing performance.

2. Optimizing for Short Queries

V-Twin was intended at the outset to be used in interactive, real-time environments by ordinary users with little or no previous experience with searching. This meant not only designing the APIs to provide easy access to features we wanted to encourage (such as relevance feedback), but also designing interfaces that facilitated their use. One of the applications we built, an indexing and search CGI for MacOS-based web servers called "Apple e.g.", was freely distributed on the Internet[2] and tested on several of Apple's web sites. Like many V-Twin-based applications, Apple e.g. shows a graphical score bar for each hit as well as a list of query terms found in the document, ordered by their importance.

In looking at the query logs and listening to feedback from users, we were struck by two things. (Note: A detailed discussion of these issues may be found in another paper [Rose & Cutting, 1996]). First, out of over 10,000 queries issued by about 4500 distinct users, the majority of queries (53%) consisted of only one word, and nearly all the queries were three words or less. We later learned that these results are consistent with (if somewhat more extreme than) those found by other search services, as shown in Table 1. In every case, at least 87% of the queries were three words or less.

Second, when a document containing one query term scored higher in relevance than a document containing two query terms, users considered it an error. One reaction to this might be to conclude that users don't have experience with ranked searching, and try to educate them to set their expectations. However, since the goal of a retrieval system is to estimate how much each item is relevant *to the user,* we decided to adjust the system's scoring to be more consistent with users' expectations. Specifically, we wanted to reorder the hits without adversely affecting system performance.

Note that we did not conclude that users really wanted a Boolean query or assumed that a Boolean AND operator was implied. They were content to have some hits with all the query terms and some hits with only a subset. It was the ordering of certain hits they objected to. It was clear that for very short queries, the coordination level (the number of matching query terms) of a document was of primary importance. Nevertheless, for longer queries, and for relevance feedback queries, the coordination level was increasingly

[2] http://cybertech.apple.com/apple_eg.html

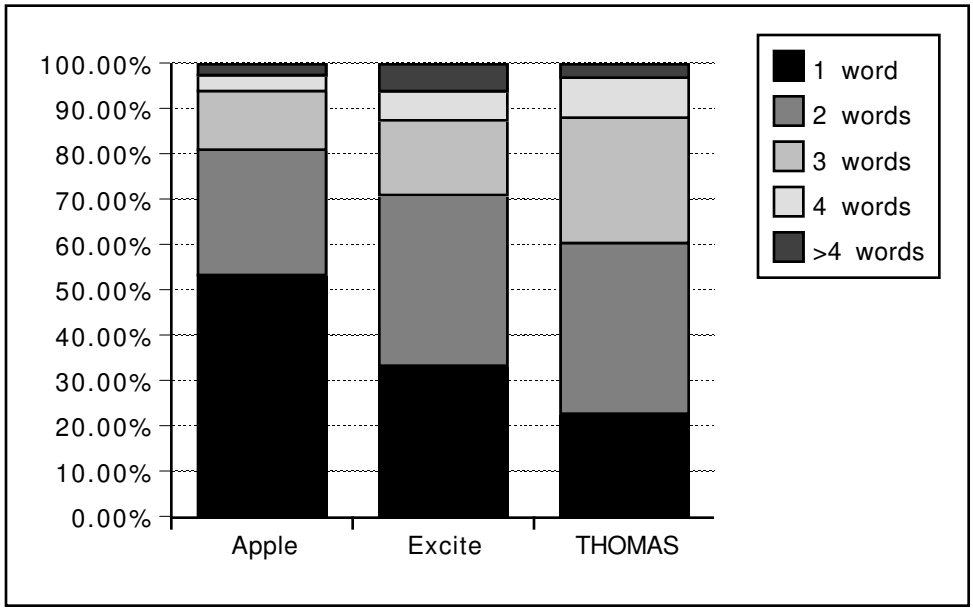

Table 1: Proportion of user queries of different lengths, from three interactive ranked search systems.

irrelevant.

To resolve this conflict, we designed a family of functions that took the raw relevance scores from an arbitrary[3] similarity matching function and transformed it into an adjusted score that took coordination level into account. What makes this function unusual is that it is parametrized by the query length, so that for very short queries, the "boost" due to coordination level dominates the raw score, while for longer queries, the coordination boost gradually becomes insignificant.

One function that meets this requirement is:

$$f(s) = \left[s + \frac{(v-1)}{(q-\partial)^2} \right] \cdot \frac{(q-1)}{q}$$

where s is the raw score from the original similarity ranking method, v is the "overlap," or number of terms in common, between query and document (i.e. the coordination level), q is the query length, and ∂ is a real-valued parameter in the range [0,1) that controls the strength of the coordination effect. We call the function SQR, for Short Query Ranking. (Note that SQR is only used when there are at least two terms in the query.)

While we knew that the SQR approach would address the users' concerns

[3] Our assumption is that the raw function's range is [0,1].

collection	# terms	R-prec. baseline	R-prec. SQR	change
TREC-4	all	0.2318	0.2501	7.9%
	4	0.1980	0.2216	11.9%
	3	0.1748	0.1929	10.4%
	2	0.1519	0.1843	21.3%

Table 2: Effect of SQR method on TREC-4 ad hoc queries truncated to various numbers of terms.

about ranking, we weren't sure whether it might also have a harmful effect on retrieval performance, as measured by traditional tests. It might have been the case that although users preferred the results, the system was retrieving fewer relevant documents. So we designed an experiment to measure the SQR effect. We truncated queries from standard test collections by keeping the *n* best terms with the highest product of weight and maximum term frequency. Then we ran the system with and without SQR.

The results for the TREC-4 collection are shown in Table 2. We expected that SQR would improve precision for the shortened queries, and it did: 21.3% for the two-word versions. (Note that the precision of the truncated queries will be significantly less than the original queries.) What surprised us was that SQR also improved performance on the original, full-length queries (which averaged 16 words in length) by a modest amount as well, 7.9%.

3. The Road to TREC

When we entered TREC-5, our focus was on the interactive track. We intended to experiment with various interaction capabilities that would help users explore the information space more effectively, including some we had investigated earlier in other contexts [Rose & Belew, 1991; Rose et al., 1993]. We did not expect to score that well at the ad hoc task, since our engine was much lighter weight than most of the other entrants and was designed and optimized for interactive use. However, as first time participants, we underestimated the time it would take to build tools for analyzing the collection. Ultimately we were unable to participate in the interactive track.

Despite this, we were pleased with the initial results we obtained from testing the system on the TREC-4 ad hoc collection. After some tuning, we were achieving what we considered to be good performance levels, as shown in Table 3.

Query construction	Baseline		Feedback		Change	
	Avg. prec.	R-prec.	Avg. prec.	R-prec.	Avg. prec.	R-prec.
automatic	0.2101	0.2692	0.2667	0.3084	26.9%	14.6%
manual	0.2643	0.3155	0.2701	0.3156	2.2%	0.0%

Table 3: Results of running V-Twin on the TREC-4 collection.

	Text size	# Documents	# Terms	Index size	Overhead	Index time
TIPSTER Disk 2	864 MB	231,219	411,864	199 MB	23.0%	11:14:37
TREC Disk 4	1194 MB	293,958	748,921	263 MB	22.0%	17:24:44
TOTAL	2058 MB	525,177		462 MB	22.4%	28:39:21

Table 4: Some statistics about V-Twin's indexes.

We noted that the use of automatic relevance feedback for query expansion had a striking effect on performance, producing average precision levels for fully automatic runs that surpassed our (admittedly inexpert) manually constructed queries.

V-Twin supports both multiple index searching (with accurate combination of relevance scores incorporating corpus-wide statistics) and index merging. We ran the TREC-4 tests both with a single index and with two separate indexes, one for each disk. Although the merged index was slightly smaller (since nearly 20% of the terms were found in both subcollection indexes), there was no other advantage.

4. TREC-5 Performance

Based on our experience with the TREC-4 data, we felt that V-Twin would perform reasonably well on the TREC-5 ad hoc task. We indexed each disk separately and searched both indexes simultaneously. Some index statistics are shown in Table 4.

We entered two automatic runs. The first, vtwnA1, used just the topic descriptions; the second, vtwnB1, used the entire topic. (We refer to these as the "short query" and "full query" runs.) Both runs used automatic relevance feedback on a preliminary search to generate the final list of matching documents. Both used a stop word list (a slightly modified version of the SMART list), and a heuristic stemmer.

Although the TREC-5 task proved to be more difficult than TREC-4's, we were very pleased with V-Twin's performance, especially considering that this was our first participation in TREC. As shown in Table 5, our overall average precision was 0.1894 for the shorter queries (topic descriptions only) and 0.2065 for the longer queries. Despite their similar average, the two runs were

	All 50 topics		47 topics with > 2 rel. docs	
	Avg. Prec.	R-Prec.	Avg. Prec.	R-Prec.
Short queries	0.1894	0.2315	0.2002	0.2463
Full queries	0.2065	0.2268	0.2189	0.2413

Table 5: Results of two V-Twin runs on the TREC-5 ad hoc task. All queries were automatically constructed.

quite different. On the shorter queries, V-Twin produced better-than-median responses on 30 of the 50 queries, while our response on the longer queries — while worse than the median on 27 of the queries — produced the best average precision of all systems on 4 of the queries.

Our results were especially satisfying to us due to the very low resource requirements of our system in comparison to most other TREC participants. We initially tested indexing on a low-end desktop computer (a Power Macintosh 7200/90, which features a 90 MHz PowerPC 601 processor) with 16 MB of memory, just to demonstrate the scalability of V-Twin. Our final indexing runs were conducted on what would now be considered a mid-performance desktop system (a Power Macintosh 9500/120 with a 120 MHz PowerPC 604); 35 MB of memory (out of a total of 48) were available to the indexing process. In contrast, most of the other entrants used upwards of 128 MB of RAM, with some much higher. Similarly, our index occupied just 462 MB of disk space, while others used as much as 2 GB.

5. Factors Affecting Performance

5.1 Examining Individual Results

By looking at the actual documents retrieved for queries on which we did poorly, we can identify certain areas of the system that need improvement. In certain cases, overly aggressive stemming and case normalization appears to have hurt performance. V-Twin does not contain a built-in stemmer or tokenizer; it provides an abstract class for text translation that allows applications to generate and modify tokens any way they please. For the purposes of TREC-5, we were using a Porter-style heuristic stemmer that predated the rest of the system by several years.

An example of the tokenization and stemming problem can be found by examining the topic at which we did worst (as measured by difference in average precision from median) on the short query run, topic 262. This topic discussed seasonal affective disorder, or SAD. The tokenizer we were using transformed “SAD” to “sad,” while the stemmer turned “affective” into “affect” and “seasonal” into “season.” As a result, what should have been the highest-weighted terms (based on an IDF weighting) were in fact among the

lowest. When the full queries are used, giving contextual clues like "darkness" and "daylight," our system performs quite well. Prior to these experiments, we were already developing a less aggressive stemmer; these results seem to confirm the wisdom of that approach. (It is also interesting to note that one of the four relevant documents for this query, AP881125-0098, is a digest of two stories that contain just one occurrence of each of the terms in "seasonal affective disorder," while containing several mentions of "llama pregnancy.")

Perhaps the most interesting failures are those catastrophic examples where there were many relevant documents, yet we failed to retrieve any of them. These seem to argue against the use of automatic relevance feedback. One example of these was topic 269, "Define instances of the use of foreign trade as an instrument to achieve national and foreign policy objectives." Unfortunately, the stemmed forms "define," "instance," and "object" were among the highly weighted terms, and these three terms co-occur much more frequently, and with more focused content, in an entirely different context: object-oriented programming. Thus V-Twin found several good matches about such things as the Common Lisp Object System, and then proceeded, through the diligent use of relevance feedback, to eliminate almost anything but these from contention. (The word "foreign" was not sufficiently distinguishing, perhaps because programming environments often have "foreign function" capabilities.) The narrative added similarly generic terms such as "goal" and "order," so the full query version performed no better.

A second example is topic 297, "Right To Die - Pros and Cons." Here, three of the query terms in the topic description are stop words, leaving just "pro," "con," and "die." V-Twin's best matches included a list of addresses from the Federal Register (FR940527-0-00146) that included the abbreviation "CONS" repeated many times, a brief resolution from the Congressional Record (CR93H-16258) in which the speaker **pro** tempore reports on resolution S. **Con**. Res. 33. about adjourning "sine **die**," and election results from Britain (FT922-13588), with party affiliations (such as Lab for Labour and Con for Conservative) listed after each of a few dozen parliamentary seats. It's hard to imagine how to solve this problem in an automatic system as long as "right" is a stop word. (The full query version, containing the term "euthanasia," had no such problems.)

5.2 When Query Expansion Hurt

For both short and long versions of the topics, we used automatic relevance feedback to expand the queries. As shown in Table 6, this had a significant positive effect on performance. However, as noted above, we suspected that relevance feedback was actually causing problems on some of the topics. To test this hypothesis, we examined the effect of feedback by individual topic. We found that in 18 of the 50 topics, feedback was harming performance.

	Baseline		Feedback		Change	
topics	Avg. prec.	R-prec.	Avg. prec.	R-prec.	Avg. prec.	R-prec.
short	0.1584	0.2009	0.1864	0.2220	17.7%	10.5%
long	0.1901	0.2235	0.2065	0.2268	8.6%	1.5%

Table 6: Effect of query expansion by automatic relevance feedback.

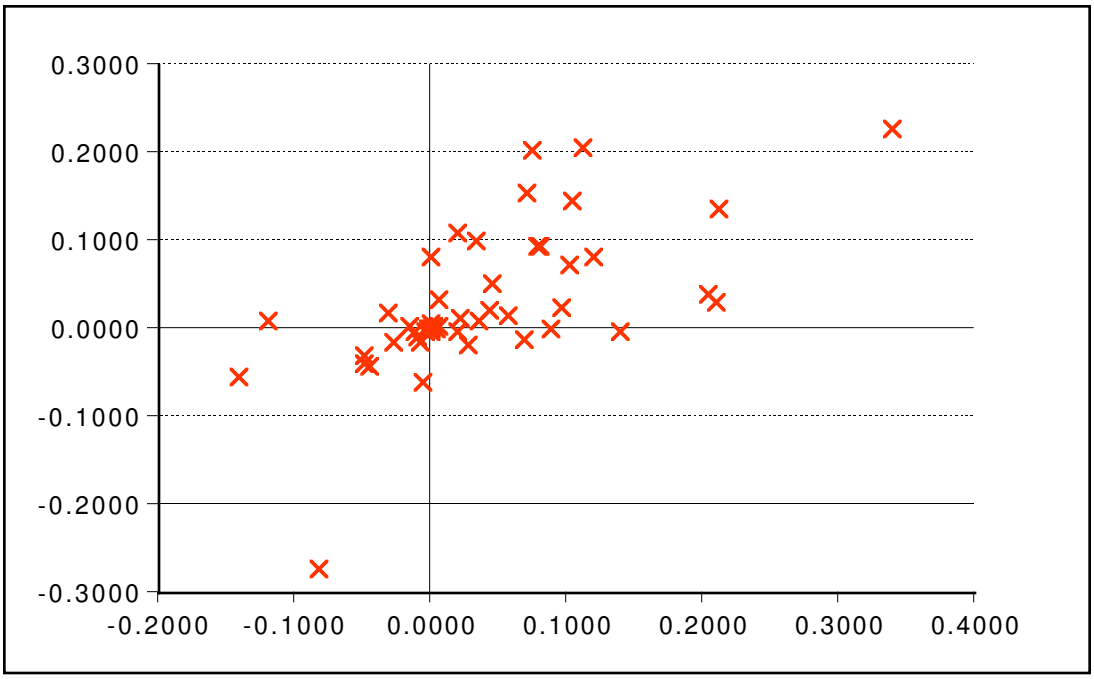

Figure 1: This scatterplot shows the correlation between each query's deviation from the median for all groups (on the X axis) and the affect of relevance feedback on its score (on the Y axis).

Furthermore, in 13 of the 17 short query topics where we did worse than the median (including the three topics discussed in the previous section), feedback had a negative effect.

Figure 1 shows a scatterplot where each query is represented by how our score compared with the median for all groups, and how feedback affected the score. There is a significant correlation between these two measures (r = 0.64). This suggests a promising line of inquiry for future experiments: if we can assess the quality of the initial retrieval, we may be able to "cut our losses" rather than compounding the problem by using inappropriate documents for feedback.

5.3 When Stemming Hurt

We performed a similar analysis to assess whether stemming and stop word removal were hurting performance in certain queries. However, the actual

	No stemming		Stemming		Change	
topics	Avg. prec.	R-prec.	Avg. prec.	R-prec.	Avg. prec.	R-prec.
short	0.1352	0.1690	0.1864	0.2220	37.9%	31.4%
long	0.1537	0.1748	0.2065	0.2268	34.4%	29.7%

Table 7: Effect of stemming & stop word elimination.

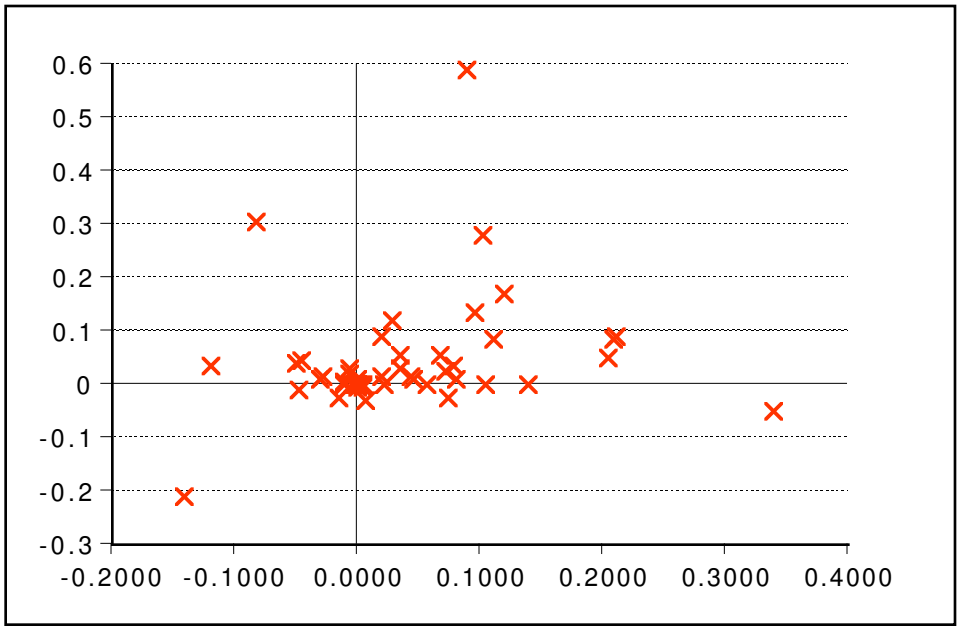

Figure 2: Scatterplot showing each query's deviation from the median for all groups (on the X axis) and the effect of stemming on that query (on the Y axis).

comparison performed was between two different text processing modules which had slightly different tokenizers in addition to differing by the presence or absence of stemming and stop word removal. For convenience, we will refer to the more powerful text processing module as "the stemmer."

As Table 7 indicates, the stemmer was much more effective than the simpler module. Given the uncertainty in the literature about the effects of stemming on performance, we are withholding judgment on these results until we can isolate stemming as a factor.

In any case, we then examined how the stemmer affected individual queries. Again, we found that stemming actually harmed performance on 13 queries. As we predicted, one of these was topic 262 (about Seasonal Affective Disorder), where the stemmed query performed 78% worse than the unstemmed query. However, there was little correlation (r = 0.20) in general between topics where stemming hurt and topics where we did poorly, as shown in Figure 2. Only three of the topics on which we scored below the median had a negative effect due to stemming.

# terms	Baseline		SQR		Change	
	Avg. prec.	R-prec.	Avg. prec.	R-prec.	Avg. prec.	R-prec.
all	0.1584	0.2009	0.1595	0.2013	0.7%	0.2%
4	0.1371	0.1801	0.1388	0.1812	1.2%	0.6%
3	0.1277	0.1575	0.1303	0.1582	2.0%	0.4%
2	0.1181	0.1449	0.1224	0.1482	3.6%	2.3%

Table 8: Effect of SQR on TREC-5 short queries, automatically truncated to various lengths. (No relevance feedback for query expansion.)

# terms	Baseline		SQR		Change	
	Avg. prec.	R-prec.	Avg. prec.	R-prec.	Avg. prec.	R-prec.
all	0.1864	0.2220	0.1890	0.2322	1.4%	4.6%
4	0.1423	0.1683	0.1566	0.1839	10.0%	9.3%
3	0.1389	0.1645	0.1398	0.1752	0.6%	6.5%
2	0.1198	0.1484	0.1210	0.1521	1.0%	2.5%

Table 9: Effect of SQR technique on TREC-5 short queries, automatically truncated to various lengths and run with relevance feedback for query expansion.

6. SQR Revisited

We discovered after submitting our official runs that, due to an error in one of our configuration files, we had not used the SQR technique in the official runs. Because of the large amount of tuning we had done to improve performance, we discovered that the SQR adjustment would have had a relatively insignificant impact on the final results.

We repeated the original SQR experiment described earlier, this time with the TREC-5 corpus. That is, we automatically truncated the queries to two, three, and four words and measured the effect of the SQR technique. In addition, we constructed two-, three-, and four-word manual queries. These results are shown in Tables 8 and 9. As with our previous SQR experiment summarized in Table 2, we found again that SQR did not have a detrimental effect. In fact, it again improved performance, with the largest gains generally coming, as expected, on the most truncated (i.e. shortest) queries.[4] In contrast to the previous experiment, however, we found this time that the overall performance effect was almost insignificant.

At first, we believed this might be due to the fact that the queries were artificially truncated in the SQR tests. To test this hypothesis, we manually

[4] The exception, the 4-term test in Table 9, can be explained by the fact that query expansion had a negative effect on this condition; R-precision for the expanded 4-term query was actually worse than R-precision for the unexpanded one.

	Baseline		SQR		Change	
# terms	Avg. prec.	R-prec.	Avg. prec.	R-prec.	Avg. prec.	R-prec.
4	0.1630	0.2113	0.1642	0.2120	0.7%	0.3%
3	0.1597	0.1947	0.1623	0.1964	1.6%	0.9%
2	0.1296	0.1750	0.1332	0.1761	2.8%	0.6%

Table 10: Effect of SQR technique on manually truncated TREC-5 short queries,run without query expansion.

generated an alternate set of 2-, 3-, and 4-word queries. As shown in Table 10, the basic results are the same: While overall performance of manually truncated queries is better than that of automatically truncated queries, the SQR technique has only a small effect on the result.

What might account for such a dramatic difference between the earlier experiment and the later ones in the effect of the SQR method? We believe the primary explanation lies in the low baseline we used when performing our original SQR experiment on TREC-4 data. This baseline predated our participation in TREC and used less effective term weighting and normalization algorithms. Thus our current assessment of the SQR method is that first, it appears to solve a usability problem without harming retrieval performance, and second, in cases of low absolute performance it may have a significant positive effect. To put it another way, the SQR technique can be used to partially compensate for weaknesses in a retrieval system's ranking algorithm.

7. Conclusions

Our work in TREC-5 has suggested several promising areas on which we may focus in the future, such as ways to detect when relevance feedback will harm performance. We have also learned a great deal about tuning our engine for the TREC task. In particular, we now believe that we may have overfit to the TREC-4 data when preparing for TREC-5.

We found further evidence that our SQR technique, which solves a problem with users' perception of ranking, has no detrimental effect on performance, and in fact seems to improve precision slightly. The results also reaffirmed our belief that searching is most effective when treated as an interactive process rather than a batch event. Perhaps most importantly, we successfully demonstrated that a lightweight engine such as V-Twin could effectively handle a larger scale search task, outperforming systems with much greater disk and memory requirements.

References

Rose, D. E. and Belew, R. K. (1991). "Toward a Direct-Manipulation Interface for Conceptual Information Retrieval Systems." In Martin Dillon, ed., *Interfaces for Information Retrieval and Online Systems: The State of the Art,* pp. 39–54, Greenwood Press, Westport, CT.

Rose, D. E., Mander, R., Oren, T., Ponceleón, D. B., Salomon, G., Wong, Y. (1993). "Content Awareness in a File System Interface: Implementing the 'Pile' Metaphor for Organizing Information. *SIGIR-93,* pp. 260–269.

Rose, D. E. and Cutting, D. R. (1996). "Ranking for Usability: Enhanced Retrieval for Short Queries." Apple Technical Report #163.