

# Match Graph Generation for Symbolic Indirect Correlation

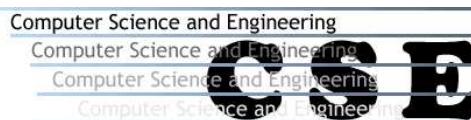
*Daniel Lopresti    George Nagy    Ashutosh Joshi*

November 2005

Technical Report LU-CSE-06-021

Department of Computer Science and Engineering  
Lehigh University  
Bethlehem, PA 18015 USA

<http://www.cse.lehigh.edu/>



# Match Graph Generation for Symbolic Indirect Correlation\*

Daniel Lopresti<sup>1</sup>      George Nagy<sup>2</sup>      Ashutosh Joshi<sup>2</sup>

<sup>1</sup> Department of Computer Science and Engineering,  
Lehigh University, Bethlehem, PA 18015  
`lopresti@cse.lehigh.edu`

<sup>2</sup> Department of Electrical, Computer, and Systems Engineering,  
Rensselaer Polytechnic Institute, Troy, NY 12180  
`nagy@ecse.rpi.edu`

November 2005

## Abstract

Symbolic indirect correlation (SIC) is a new approach for bringing lexical context into the recognition of unsegmented signals that represent words or phrases in printed or spoken form. One way of viewing the SIC problem is to find the correspondence, if one exists, between two bipartite graphs, one representing the matching of the two lexical strings and the other representing the matching of the two signal strings. While perfect matching cannot be expected with real-world signals and while some degree of mismatch is allowed for in the second stage of SIC, such errors, if they are too numerous, can present a serious impediment to a successful implementation of the concept. In this paper, we describe a framework for evaluating the effectiveness of SIC match graph generation and examine the relatively simple, controlled cases of synthetic images of text strings typeset, both normally and in highly condensed fashion. We quantify and categorize the errors that arise, as well as present a variety of techniques we have developed to visualize the intermediate results of the SIC process.

## 1 Introduction

*Symbolic indirect correlation* (SIC) is a new approach for bringing lexical context into the recognition of unsegmented signals that represent words or phrases in printed or spoken form [NSML03, NJK<sup>+</sup>04, JN05]. It is applicable wherever segments of lexically labeled reference signals can be compared to unlabeled signals. The only requirement is that the signal preserve sufficient ordering of the alphabetic units within a word, and of words within a phrase. Beyond this one constraint, SIC is general in the sense that it does not depend on the signal representation or signal-matching algorithm. It shares some similarities both to

---

\* Presented at *Document Recognition and Retrieval XIII (IS&T/SPIE International Symposium on Electronic Imaging)*, January 2006, San Jose, CA.

Nearest Neighbor classification as well as to Hidden Markov Models (HMM's), but appears to offer substantial benefits over each of these well-known techniques.

*Symbolic* means that the label of the unknown signal is determined by comparing the signal-level matches with lexical matches (the lexical matches between the labels of the reference signals and a lexicon of admissible words are precomputed). *Indirect* means that the unknown signals can represent words for which no labeled signals are available. *Correlation* refers to both the lower-level signal and lexical comparisons.

The nature of the underlying features is immaterial, subject to the previously mentioned order constraint. Errors at the feature level can be compensated for by extending the reference signal to increase the number of potential matches for each segment of the unknown signal. Different features can be used for on-line handwriting, scanned documents, and speech.

Like Nearest Neighbor, SIC is a non-parametric classifier that requires only a labeled reference list. Nearest Neighbor cannot be applied to unsegmented signals, however, or to reference data that do not contain at least one prototype of every class; SIC suffers from no such limitations. Symbolic Indirect Correlation also promises benefits over Hidden Markov Models since SIC avoids parameter estimation through unstable Expectation Maximization and can use reference samples more efficiently than HMM's, immediately incorporating new inputs into the recognition process.

Fig. 1 presents a high-level overview of SIC. Working from left to right, an appropriate feature string representation is first created for the 1-D signal input and reference. These are compared using a methodology capable of identifying matching subsegments. The output from this stage might, say, take the form of a lattice graph as we shall show shortly. In the same way, the transcript of the signal reference is compared to the lexicon entries. Entering the second stage, second-level features are extracted, for example, the starting positions of all subsegment matches that surpass a given threshold. These are then compared to determine a classification for the input. Further details, including preliminary studies suggestive of the efficacy of SIC, appear in earlier papers [NSML03, NJK<sup>+</sup>04, JN05].

One way of viewing the SIC problem is to find the correspondence, if one exists, between two bipartite graphs, one representing the matching of the two lexical strings and the other representing the matching of the two signal strings. This is illustrated in Figs. 3 and 4, where the former illustrates a situation where the match is perfect (each edge in the lexical match graph on the left of the figure corresponds precisely with an edge in the signal match graph on the right), while the latter illustrates the two kinds of errors that may arise, thereby complicating the situation: edges that are missed (the edge from “in” in “morphines” to “in” in “invention”) and spurious edges that are added (*e.g.*, the edge from “ne” in “morphines” to “he” in “mother”).<sup>1</sup> While perfect matching cannot be expected with real-world signals and while some degree of mismatch is allowed for in the second stage of SIC, such errors, if they are too numerous, can present a serious impediment to a successful implementation of the concept.

In this paper, we describe a framework for evaluating the effectiveness of SIC match

---

<sup>1</sup>All examples appearing in this paper reflect real instances of SIC match graphs and were generated during the experimental evaluations to be described later.

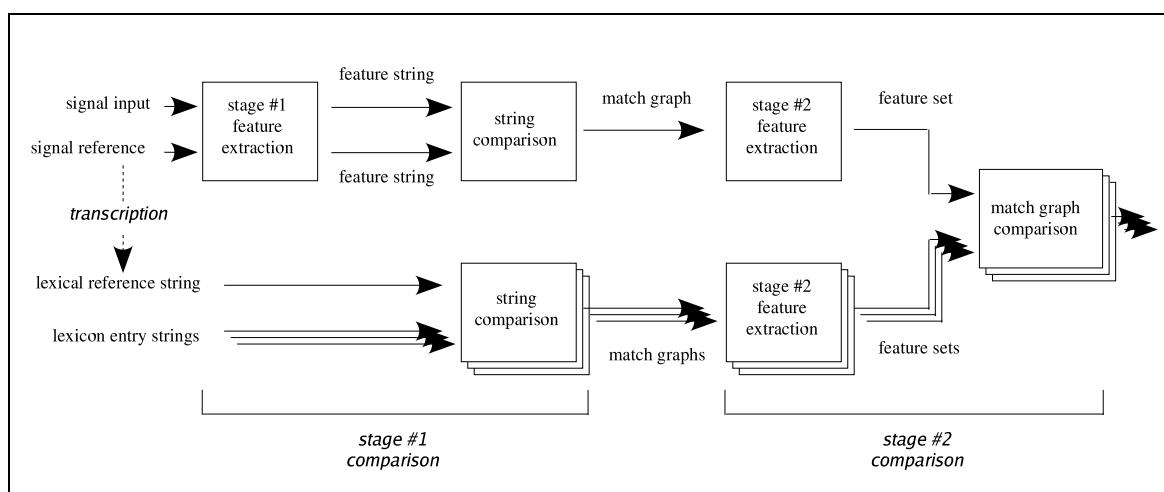


Figure 1: Overview of Symbolic Indirect Correlation (SIC).

graph generation and examine the relatively simple, controlled cases of synthetic images of text strings typeset, both normally and in highly condensed fashion. We quantify and categorize the errors that arise, as well as present a variety of techniques we have developed to visualize the intermediate results of the SIC process.

## 2 Approximate String Matching for SIC

SIC can be regarded as a comparison of string comparison results – in effect, a “meta” string comparison. The particular formulation we showed based on matching bipartite graphs is just one way to cast the problem. In its most general statement, SIC is a two-stage classification process. In the first stage, an unknown input signal is compared to a reference signal for which the lexical transcript is known. This comparison characterizes, in some fashion, subsegments of the unknown input that match well with subsegments of the reference signal (they need not be identical). Valuable information is encoded in the lengths, strength, and ordering of these subsegment matches. An analogous comparison is made between each entry in a target lexicon and the lexical transcript of the reference signal (this part of the computation can be performed in advance, offline, to save time). The outputs from the first stage comparison are input to a second stage that determines when two sets of results are similar enough that the lexicon entry in question likely corresponds to the unknown input.

There are in fact a number of ways one can consider performing the comparisons in question. A goal of our research is to investigate existing techniques and develop new approaches in the context of specific applications of interest. In addition to classification accuracy, our preliminary experiments suggest that computational efficiency will almost certainly take on a fundamental importance. We plan to explore the various tradeoffs for different choices of algorithms at each stage.

In the case of the signal strings, we cannot count on exact matches. Even in the case

of the lexical strings, this might be a limiting assumption because of transcription errors. Hence, the string comparison techniques we require must allow for approximate matching. Fortunately, there is a vast literature on this subject, beginning with three seminal papers in information theory, molecular biology, and computer science [Lev66, NW70, WF74].

One potential model for computing subsegment matching between strings makes use of an extension of the dynamic programming algorithm for string edit (or evolutionary) distance that is due to Smith and Waterman [SW81]. This algorithm builds a two-dimensional matrix that, in effect, encodes the starting and ending positions and weights of all possible substring matches for a given assignment of basic editing costs. For two strings  $S = s_1s_2\dots s_m$  and  $T = t_1t_2\dots t_n$ , the computation can be formulated as a dynamic programming problem. Define  $dist_{i,j}$  to be the distance corresponding to the best match ending at the  $i^{th}$  symbol of  $S$  and the  $j^{th}$  symbol of  $T$ . This match is permitted to start anywhere earlier in the two strings. The initial conditions are:

$$\begin{aligned} dist_{0,0} &= 0 \\ dist_{i,0} &= 0 & 1 \leq i \leq m, 1 \leq j \leq n \\ dist_{0,j} &= 0 \end{aligned} \tag{1}$$

and the main recurrence is:

$$dist_{i,j} = \min \begin{cases} 0 \\ dist_{i-1,j} + c_{del}(s_i) \\ dist_{i,j-1} + c_{ins}(t_j) \\ dist_{i-1,j-1} + c_{sub}(s_i, t_j) \end{cases} \quad 1 \leq i \leq m, 1 \leq j \leq n \tag{2}$$

Here deletions, insertions, and mismatches are charged positive costs, and exact matches are accorded negative costs. The computation builds a matrix of distance values working from the upper left corner ( $dist_{0,0}$ ) to the lower right ( $dist_{m,n}$ ). Note that the Smith-Waterman formulation is fundamentally different from the more widely known Wagner-Fischer (Needleman-Wunsch) formulation for edit (evolutionary) distance in that it allows for multiple substring matches that can start and end anywhere. This is precisely what is required for SIC.

By maintaining the decisions used to obtain the minimum in each step of the computation, it becomes possible to backtrack from any given point in the distance matrix and obtain the optimal substring matching that ends at that position. Such a backtrack matrix is shown on the left side of Fig. 2. In this formulation, the output from the first stage of SIC can be viewed as a weighted lattice graph, with the substring matchings corresponding to various backtracking paths as illustrated in the figure. Note that for the proper choice of editing costs and thresholds,<sup>2</sup> it is possible to require that the matches be limited to shared n-grams between the two strings; this is simply a special case of the more general problem.

The second stage of SIC takes the comparison results for an unknown signal input versus a signal reference, computed as above, and a set of results for lexicon entries compared against the corresponding lexical reference string. Then it finds the most similar results

---

<sup>2</sup>In particular, making the deletion and insertion costs so large that those two operations are always excluded.

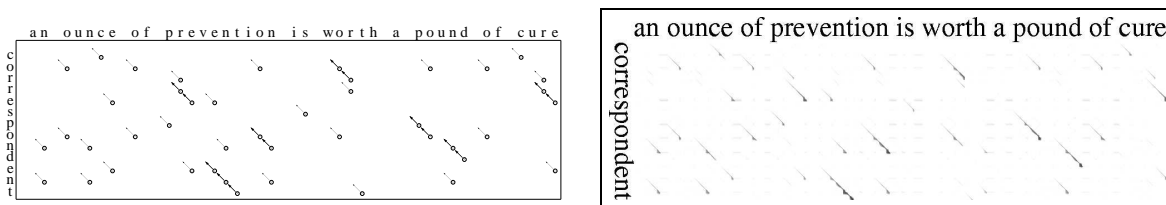


Figure 2: Graphical representations of lexical (left) and signal (right) edit distance computations.

according to predetermined criteria. One alternative is to extract all substring matches that exceed a certain threshold and represent the comparison as a bipartite graph. The problem then becomes one of determining the correspondence between two such graphs, one representing the comparison of lexical strings and the other the comparison of signal strings. For example, we might encounter the two match graphs displayed in Fig. 3. The graph on the left was produced as output when the lexical query string “correspondent” was compared to the lexical reference string “an ounce of prevention is worth a pound of cure.” The graph on the right, on the other hand, was computed by comparing feature strings extracted from bitmap images of the text in question: the unknown input query, which is on top, and the previously-transcribed reference string, on the bottom. Note that these two match graphs correspond perfectly, with each of the seven edges in one graph mapping to exactly one edge – at the appropriate location – in the other, a best-case scenario.

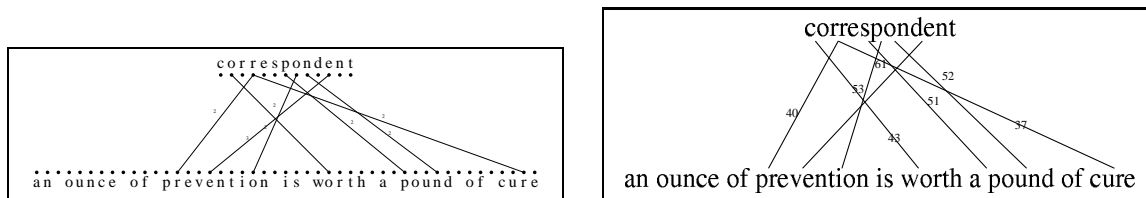


Figure 3: Lexical (left) and signal (right) match graphs.

Unfortunately, this matching process is not always perfect. Edges can be missed in the signal graph and spurious edges may be added. These problems are shown in Fig. 4. Such errors can arise for a variety of reasons, including noise in the signal input, poor quality pre-processing or feature extraction, and the inherent confusability of the basic symbols making up the strings. In this case, we might suspect that the edge from “in” in “morphines” to “in” in “invention” was missed because “i” is a very thin character and does not contribute much weight to the matching under the particular scheme we used. On the other hand, the two spurious edges from “ne” in “morphines” to “he” in “the” and “mother” can be explained by the physical similarity between the right portions of the characters “h” and “n.”

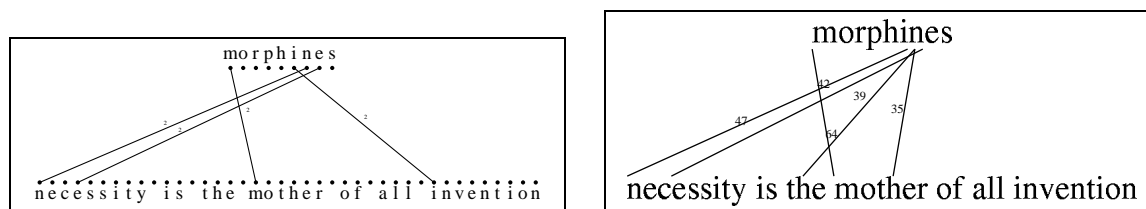


Figure 4: Lexical (left) and signal (right) match graphs exhibiting edge deletion and insertion errors.

### 3 Evaluating Match Graph Accuracy

As a precursor to full-scale implementation of the SIC concept, we have created a framework for evaluating the accuracy of match graph extraction. Our initial tests are performed in a highly controlled environment allowing us to identify unambiguously all edges present in the lexical and signal match graphs and to determine which edges correspond to one another, as well as to tabulate the edges that have been missed and spurious edges that have been added. The goal is to develop a better understanding of the first stage of SIC, where it might fail, and what can be done to ameliorate such problems.

For the initial tests we report in this extended abstract, we synthesized TIF bitmaps of known text strings and computed the first-stage SIC matchings using the Smith-Waterman algorithm (Equations 1 and 2). For our reference strings, we collected 100 random proverbs from various websites. These ranged from 40 to 60 characters long, with an average length 47.59 characters. Our query words were chosen from “Yet Another Word List” (YAWL) [YAW05]. We selected 100 words randomly that did not occur anywhere within the reference string collection. The minimum length for the query words was 8 characters, the maximum was 16, and the average was 10.76.

All punctuation was stripped and uppercase characters were converted to lowercase. We used PostScript to render the text strings, along with ghostscript and built-in Linux utilities to convert the PostScript to bitmap form. The synthesized text was typeset in 12-point Times and rendered as though it were scanned at 300 dpi. Font metric information was collected separately to allow us to build a complete ground-truth for evaluating the detection of match graph edges. This approach presents an advantage over working with scanned images in that it allows definitive precision and recall measurements without requiring laborious labeling of a ground-truth. Nonetheless, we plan to expand our evaluation to include real scanned data as well as online handwriting in the near future.

For features, we employed the same set of four reported by Manmatha and Rath in the context of their work on word-spotting in offline handwriting [MR03] – namely, a representation of the lower and upper contours of the text, the density of black pixels in each column, and a count of the 0-1 crossings in each column. This feature set is illustrated graphically for one of our query words in Fig. 5.

As noted earlier, we used the Smith-Waterman algorithm for matching both the lexical and the signal strings. Naturally, the cost functions in the two cases differed. In the lexical case, the costs were set so that all bigram matches and longer were extracted. In the signal

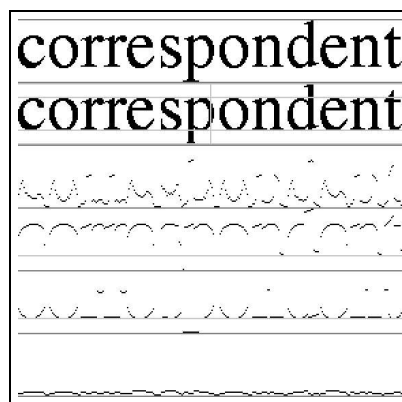


Figure 5: Illustration of the feature set used in our experiments.

case, we varied the threshold on the strength of the match to yield curves illustrating various tradeoffs. For signal strings, deletions and insertions were permitted so that the “warping” required for more robust matching was possible.

Since the focus of this work is entirely on the problem of building the match graphs and ignores the last stage of SIC, we chose to use recall (the percentage of graph edges present in the lexical comparison that are found in the signal comparison) and precision (the percentage of graph edges found in the signal comparison that are present in the lexical comparison) as our performance measures.

For our 10,000 ( $=100 \times 100$ ) comparisons, the total number of edges in all of the ground-truth graphs was 16,965, which implies an average of roughly 1.7 edges per match graph. Our results showing the tradeoff between recall and precision appear in Fig. 6. In this case, the edge detection accuracy at the equal error rate is approximately 80% and occurs at a threshold value of 35. One might wonder why the performance does not approach 100% since the input is noise-free. It is important to keep in mind that despite the lack of real-world degradations, there is still variability in the query and reference signals due to differences in typesetting (kerning, rendering)<sup>3</sup> and the inherent similarity between parts of non-identical characters. The former effect can cause edges to be missed, while the latter may induce spurious edges.

Looking more closely at the threshold which achieves the equal error rate (35), in Table 3 we tabulate the 10 most-frequent edge effects in each of three categories: edges that are correctly determined by SIC in this instance, those that are missed, and spurious edges between substrings that should not be matched. Keeping in mind our earlier observations that very thin characters can lead to missed edges, and characters that share physically similar segments can result in spurious edges, the data reported in the table should come as no surprise.

One of the compelling arguments in favor of SIC is its ability to handle hard-to-segment inputs. We probed this issue by creating a parallel experiment to our first, with the text strings typeset in an extremely condensed fashion so that adjacent characters overlapped

<sup>3</sup>The spatial sampling effect comes to mind [SNZL98].

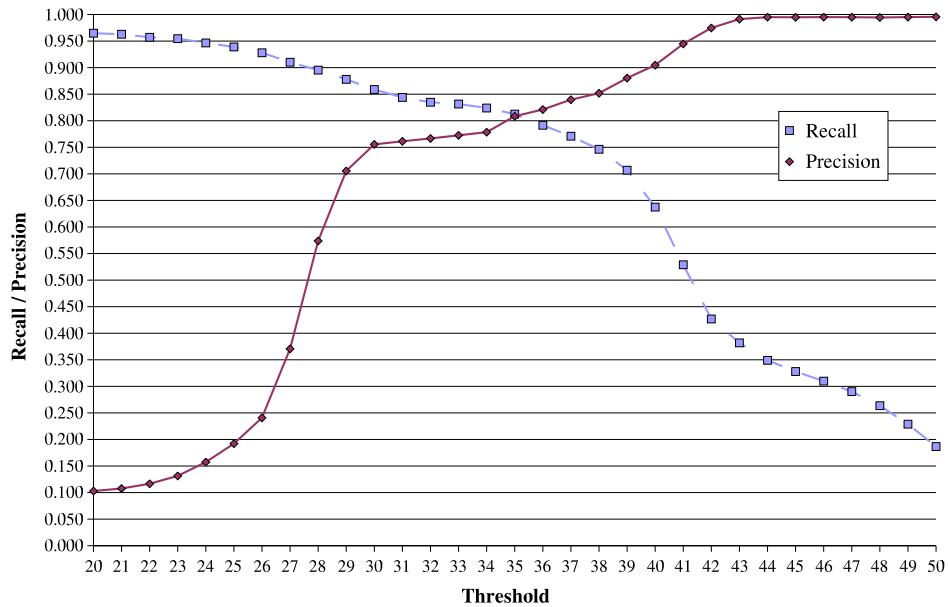


Figure 6: Match graph edge detection results for 10,000 comparisons.

severely. An example is shown in Fig. 7 (we adjusted the placement of each character in the PostScript file to achieve this effect). The same set of queries and reference strings was used as before.

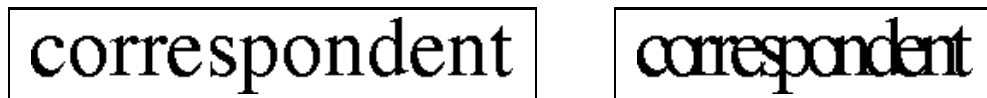


Figure 7: Normal (left) versus condensed (right) text.

Results for our condensed text experiment are depicted in Fig. 8. While clearly worse than the results reported for our original test – in this case the accuracy at the equal error rate is approximately 30% – this is perhaps not unexpected given how the identifying characteristics of each symbol begin to break down under such conditions. Still, SIC did not break down completely. As this work is still very much in progress, we hope to be able to improve the match graph generation accuracy for such hard inputs by studying the effects of the features we have chosen, tuning the string matching algorithm, and incorporating additional context.

Rank	Correct Edges		Missed Edges		Spurious Edges	
	Count	Pattern(s)	Count	Pattern(s)	Count	Pattern(s)
1	1,056	“es”	331	“it”, “es”	37	“nes” ↔ “her”
2	971	“th”	219	“st”	34	“me” ↔ “ma”
3	957	“in”	209	“ti”	32	“nes” ↔ “he m”
4	827	“er”	199	“li”	30	“mo” ↔ “ma”
5	537	“re”	178	“is”	25	“mo” ↔ “me”
6	471	“at”	141	“re”	24	“me” ↔ “mo”
7	426	“on”	124	“ll”	22	“nes” ↔ “he b”
8	394	“an”	95	“te”	21	“nes” ↔ “he h”
9	391	“he”	82	“er”, “at”	19	“nes” ↔ “he d”, “owe” ↔ “wi”, “mo” ↔ “mi”
10	380	“nt”	57	“en”	18	“me” ↔ “mi”

Table 1: Most-frequent edge effects in our experiment.

## 4 Discussion

In this work, we have focused our attention on the first stage of the symbolic indirect correlation process: generation of the lexical and signal match graphs that attempt to capture and represent the inherent dissimilarities between different inputs to the system. We have shown that a dynamic programming algorithm based on the Smith-Waterman paradigm provides reasonably good results, at least on synthesized images of text strings, although certain improvements seem possible. In the near term, we intend to examine the performance of match graph generation for appropriately ground-truthed online handwriting and scanned image data. We also plan to relate accuracy for the first stage of SIC to the final system accuracy on end-to-end pattern recognition tasks.

In addition to the studies reported in this paper, we have recently begun exploring different formulations for the second stage matching in SIC, including an intriguing connection to the multiple complete digest mapping problem from computational biology [FJK<sup>+</sup>97]. We are also interested in identifying different techniques for visualizing the results of SIC, as depicted in Fig. 9. Here the overlaps between the query string (the word “splashiness”) and each of 10 reference strings are displayed concisely in a single figure.

## References

- [FJK<sup>+</sup>97] Daniel P. Fasulo, Tao Jiang, Richard M. Karp, Reuben Settergren, and Edward C. Thayer. An algorithmic approach to multiple complete digest mapping. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*, pages 118–127, Sante Fe, NM, 1997.

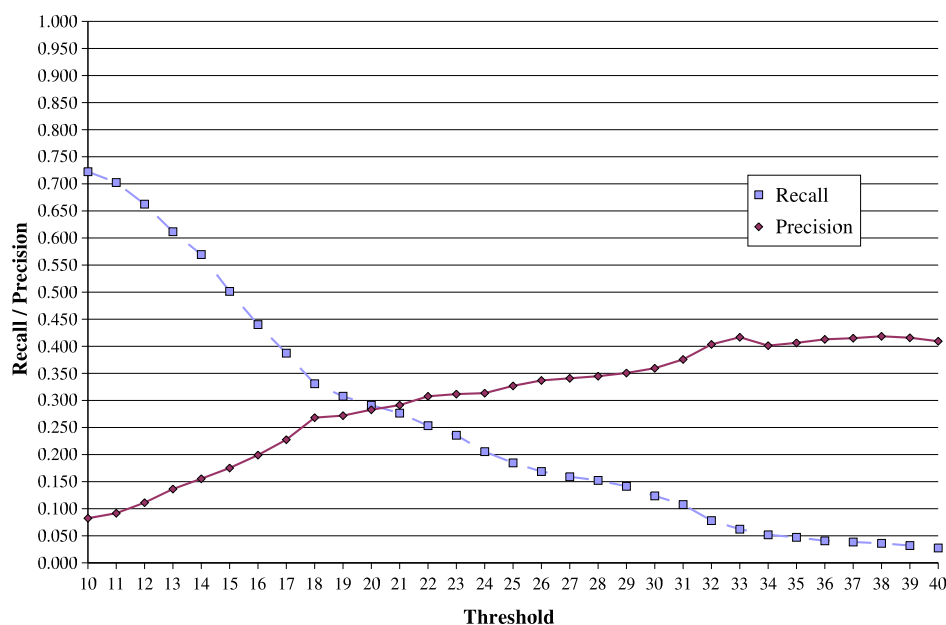


Figure 8: Match graph edge detection results for condensed input (10,000 comparisons).

- [JN05] A. Joshi and G. Nagy. Online handwriting recognition using time-order of lexical and signal co-occurrences. In *Proceedings of the 12th Conference of the International Graphonomics Society*, Salerno, Italy, June 2005.
- [Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1966.
- [MR03] R. Manmatha and T. Rath. Indexing handwritten historical documents – recent progress. In *Proceedings of Symposium on Document Image Understanding (SDIUT03)*, pages 77–85, Greenbelt, MD, April 2003.
- [NJK<sup>+</sup>04] George Nagy, Ashutosh Joshi, Mukkai Krishnamoorthy, Yu Lin, Daniel Lopresti, Shashank Mehta, and Sharad Seth. A nonparametric classifier for unsegmented text. In Elisa H. Barney Smith, Jianying Hu, and James Allan, editors, *Proceedings of Document Recognition and Retrieval XI (IS&T/SPIE Electronic Imaging)*, volume 5296, pages 102–108, San Jose, CA, January 2004.
- [NSML03] G. Nagy, S. Seth, S. Mehta, and Y. Lin. Indirect symbolic correlation approach to unsegmented text recognition. In *Proceedings of the Workshop on Document Image Analysis and Retrieval*, Madison, WI, June 2003.

- [NW70] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino-acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.
- [SNZL98] Prateek Sarkar, George Nagy, Jiangying Zhou, and Daniel Lopresti. Spatial sampling of printed patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):344–351, March 1998.
- [SW81] T. F. Smith and M. S. Waterman. Identification of common molecular sequences. *Journal of Molecular Biology*, 147:195–197, 1981.
- [WF74] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21:168–173, 1974.
- [YAW05] YAWL (yet another word list), July 2005. <http://www.ibiblio.org/pub/linux/libs/>.

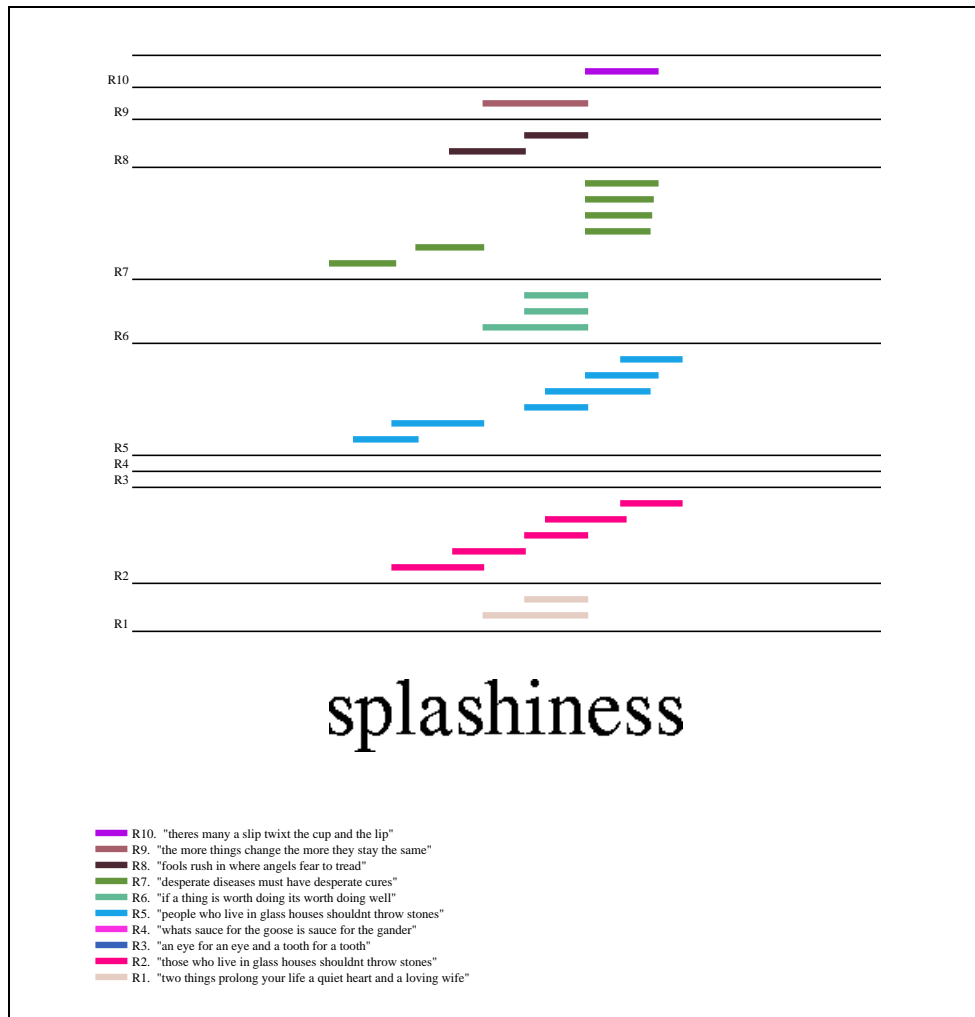


Figure 9: Visualization of SIC first-stage matching results for a single query versus 10 reference strings.