Additional Experiments on Negative Rules

Steven Euijong Whang — Stanford University
Omar Benjelloun — Google Inc.
Hector Garcia-Molina — Stanford University

{euijong, hector}@cs.stanford.edu
benjello@google.com

We implemented the General and Enhanced algorithms as described in our technical report [3] and conducted extensive experiments. We ran our experiments on a comparison shopping dataset provided by Yahoo!. In this application, hundreds of thousands of records arrive on a regular basis from different online stores and must be resolved before they are used to answer customer queries. Because of the volume of data, we used blocking techniques to partition the data into independent clusters and then applied our algorithms on each cluster. In our experiments, we used a partition containing records with the sub-string "iPod" in their titles; we will call these iPod-related records from now on.

The algorithms were implemented in Java, and our experiments were run on a 1.8GHz AMD Opteron processor with 20.4GB of memory. Though our server had multiple processors, we did not exploit parallelism.

1 Metrics

We use two metrics to determine the performances of the General and Enhanced algorithms. The "action costs" such as the time taken by the match and merge rules reflect the systems aspect of performance. The "human effort" made by the solver to choose records reflects the human part of performance.

1.1 Action Costs

Actions include evaluating the match, merge, and negative rules and comparing records for domination. Since actions are black-box functions that can potentially be expensive, it is desirable to minimize the number of actions performed.

The number of actions performed largely depends on four factors. The first factor is the match selectivity, which captures the "selectivity" of the application. Given the initial set of records I, the match selectivity is defined as the ratio between the number of matching pairs of records in I and the number of all the possible record pairs in I. For example, if five records match with each other among ten records, the match selectivity is $\binom{5}{2}/\binom{10}{2}$ = 10/45 $\approx$ 0.202.

The match selectivity affects the invocations of various actions. First, it is obvious that the match selectivity directly affects the invocations of the match and merge rules. However, the match selectivity also affects the invocations of the negative rules and domination comparisons. For instance, if the records are more likely to match, then the merge closure size increases. An increased merge closure will affect the number of records the solver chooses as well as the number of records compared with the chosen records for binary inconsistencies and dominations. Consequently, the frequencies of invoking the two rules also change. The other factors we introduce below also affect various actions for similar reasons.

The second factor is the unary density, which captures the probability of a record being internally inconsistent. Given the merge closure $\bar{I}$, the unary density is defined as the ratio between the number of internally inconsistent records in $\bar{I}$ and the number of records in $\bar{I}$. For example, suppose we have 20 records in the merge closure. If five records are internally inconsistent, the unary density is 0.25.

The third factor is the binary density, which captures the probability of a pair of records being inconsistent. Given the merge closure $\bar{I}$, the binary density is defined as the ratio between the number of inconsistent record pairs in $\bar{I}$ and the number of all the possible record pairs in $\bar{I}$. The binary density can be calculated just like the match selectivity, except that the records now come from the merge closure.

The last factor is the degree of domination, i.e., the probability of one record dominating another. The probability depends on the partial order relation among records.

In summary, the number of action invocations depends on several factors. Among them, we are interested in the match selectivity, unary density, and binary density. In our experiments, we show how some of these factors individually impact the system runtime.

1.2 Human Effort

The second metric for performance is the human effort made by the solver (domain expert) for choosing records. Of course, human effort is very hard to model and is seldom quantified in our database community. Nevertheless, because the human solver plays a key role in entity resolution with negative rules, we feel important to analyze human effort, even if our metric is far from perfect.

In the General algorithm, the solver is repeatedly given a set of non-dominated records from which she has to choose the most desirable record. The effort made to choose such a record will vary depending on the strategy used by the solver. For example, one could take a look at all the records before making a decision while another could pick any "reasonable" record that appears first without doing an entire scan of the candidate records.

Since it is difficult to predict the behavior of the solver, we use the following simple model as a surrogate of the human effort. The cost of selecting one record from a set of records ndS (step 7 of the General Algorithm [3]) is $|ndS|$. The total human cost is then the sum of costs for all such selections. For example, given a set of ten records with no inconsistencies or domination relationships, the total human effort for choosing all the ten records is $10 + 9 + ... + 2 = 54$. Notice that we do not count the effort for choosing the last record.

We caution that the human effort values we present, by themselves, are not very useful. However, we believe they can be helpful in comparisons. For instance, if in Scenario A the cost is 10 times that in Scenario B, then we can infer that the solver will be significantly more loaded in Scenario A.

Note that we can improve the human effort by automatically choosing non-dominated records that will be "chosen anyway." Consider the records in ndS that are consistent with every record in the set S. (Recall that S is the subset of the merge closure from which non-dominated records are chosen; see General Algorithm [3].) Such records are guaranteed to eventually be chosen by the solver because the only cases for a record not to be chosen is when it is inconsistent with or dominated by a chosen record. On the other hand, there is a loss in system performance because we need to do more inconsistency checks to tell whether a non-dominated record is indeed consistent with all the records in S. Specifically, we need to do a pair-wise comparison for all the records in $\bar{I}$ using the binary negative rule. Hence, the improvement is a tradeoff between the human effort and system performance.

2 The Rules

The rules we used in our experiments are based on the scenario where the solver imposes constraints on the match and merge rules defined by the application writer. Suppose that the application writer is using match and merge rules that compare records using the title and price attributes. However, suppose that the solver notices two problems for the match and merge rules. One problem is that a record in the solution may have very different prices (we will later show how this problem can occur). The other problem is that the solution may contain two records having titles that are "too similar" for the records to be different. Notice that the first problem can be solved by imposing a unary negative rule while the second problem needs a binary negative rule. In this case, the binary negative rule can force the human solver to examine a pair of non-matching records that are suspiciously similar.

The merge rule is simply a union operator for merging attributes of two records. That is, instead of having a single value, an attribute of the merged record retains all the distinct attribute values of the base records.

The match rule compares two records using the title and price attributes and returns true only if both attributes match closely. For each attribute, the match rule does a pairwise comparison between all the possible attribute values from each record and looks for an existing match. Titles are compared using the Jaro-Winkler similarity measure [1] [2], to which a threshold t_M from 0 to 1 is applied to get a yes/no answer. Prices are compared based on their numerical distance and matched when the ratio of the lower price x to the higher price y is at least p_M (i.e., $x/y \geqslant p_M$). It is easy to show that the union operation for merging and existential comparison for matching guarantee the ICAR properties to hold.

We can determine reasonable thresholds for the match rule by comparing iPod-related records. Figure 1 shows the titles and prices of ten of the iPod-related records we used for our experiments. Figure 2 shows the title similarities (using the Jaro-Winkler similarity measure) and price similarities (using the numerical distance) for five pairs of records from Figure 1. Records 1 and 2 can be considered very similar, since they have nearly identical titles and the same prices. Hence, t_M should be lower than 0.983, the similarity of titles 1 and 2. On the other hand, records 3 and 4 are not identical, since they are products with different colors. Since the title similarity is 0.943, we can reasonably assume that the title similarity should be at least 0.95 for two titles to match and thus set t_M to 0.95. Records 5 and 6 have a high similarity for titles (0.983), but are actually different products because of their different disk capacities (4G vs 6G). Since the difference in capacity is better reflected in the different prices, we can use the price similarity to distinguish the two records. Since the price similarity between the two records is 0.799, we

[1]The Jaro-Winkler similarity measure returns a similarity score in the 0 to 1 range base on many factors, including the number of characters in common and the longest common substring.

RecID	Title	Price
1	“iPod mini 6GB Green”	249
2	“iPod Mini 6GB, Green”	249
3	“Apple iPod Mini 4 GB Music Player in Green”	199.95
4	“Apple iPod Mini 4 GB Music Player in Pink”	199.95
5	“Apple iPod Mini 4 GB Music Player in Pink”	199.95
6	“Apple iPod Mini 6 GB Music Player in Pink”	249.99
7	“Apple iPod Mini 4GB Pink”	179.99
8	“Apple iPod mini 6GB Blue”	249
9	“Protective case for iPod mini”	13.26
10	“PROTECTIVE CASE FOR IPOD MINI”	19.99

Figure 1: iPod-related records

RecID Pair	Title Similarity	Price Similarity
$\langle 1, 2\rangle$	0.983	1.0
$\langle 3, 4\rangle$	0.943	1.0
$\langle 5, 6\rangle$	0.983	0.799
$\langle 7, 8\rangle$	0.861	0.722
$\langle 9, 10\rangle$	0.988	0.663

Figure 2: Title and price similarities

set p_M to a higher 0.8. Finally, records 7 and 8 show different products where neither the titles nor prices match. After considering many other pairs of records in our data set, we indeed settled on the thresholds above to be used as the base settings in our experiments.

The unary negative rule checks if the possible prices in a record are “too far apart.” Specifically, a record is internally inconsistent if the ratio of the lowest price to the highest price is smaller than p_N. (Notice that a record with only once price is always consistent). The unary negative rule prevents the case where a sequence of records can merge together in a certain order due to the similarity of adjacent records, but the first and last records are too different to be identical. For example, although three records with prices 8, 10, 12 can merge by comparing 8 with 10 and then $8|10$ with 12, 8 and 12 may be considered too different to coexist in the final record now.

The binary negative rule checks if the titles of two records are “too similar” for the products to be distinct. That is, the binary negative rule does a pairwise title comparison between all the possible titles of the two records, looking for existing matches. Just like the match rule, we use the Jaro-Winkler similarity measure, to which a threshold t_N from 0 to 1 is applied to get a yes/no answer. It is also easy to see that the two negative rules above satisfy the commutativity and persistence properties.

We can also determine reasonable thresholds for the negative rules by comparing more iPod-related records. The threshold p_N is determined by the belief that the maximum price of a product should never be more than three times the minimum price. Hence, p_N is set to 0.33. The threshold t_N can be determined by comparing records 9 and 10 in Figure 1. Although the titles of the records are almost identical, the records are still considered different due to the different prices. Since we do not want the products in our solution to have very similar titles, we set t_N to 0.98, which is slightly lower than the title similarity 0.988 of the two records shown in Figure 2. Hence, records 9 and 10 cannot co-exist in a solution (without intervention by the solver).

The thresholds for the rules derived above make our algorithms produce “correct” solutions for the iPod-related data. Using the thresholds, we observed a match selectivity of 1.46E-4, a unary density of 0 (i.e., there were no internally inconsistent records), and a binary density of 7.86E-4. However, different applications might have different selectivities and densities. To study these potentially different applications, we varied the thresholds of our various rules, obtaining different match selectivities and densities. These different settings may not be appropriate for our particular scenario, but our goal is to “emulate” how other applications may perform. Throughout our experiments, any threshold whose value is not varied will be set to our derived values as a default.

It should be noted that, in our application scenario, the match and binary negative rules are the most expensive operators and usually dominate the overall runtime of the algorithms. Both rules use expensive string similarity comparisons, which are far more expensive than numerical price comparisons. Hence, we will mainly focus on the number of invocations of the match and binary negative rules when explaining our experimental results.

3 Match Selectivity Impact

We measured the performances of the General and Enhanced algorithms by varying the match selectivity on 1,000 random iPod-related records. Varying the thresholds of the match rule affects the match selectivity. In our experiments, we only varied the title threshold t_M to capture a variety of match criteria. The lower the t_M, the higher the match selectivity because more records tend to match and merge.

As the match selectivity increases, the runtime of the General algorithm slowly increases until there is a surge of work at a certain point. Figure 3 shows the decomposition of the runtime. Each subplot shows the runtime for a certain type of action. For example, the match plot shows the runtime for invoking the match rule. The number of match rule invocations increases because more records match and merge. The number of binary negative rule invocations also increases because, for every record chosen by the solver, more binary inconsistency checks must be done between the chosen record and the records in the set S. The abrupt runtime increase around a selectivity of 0.0004 is because multiple similar records start to match

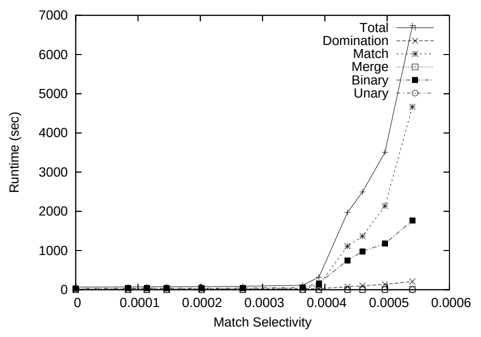

Figure 3: Match selectivity impact (General)

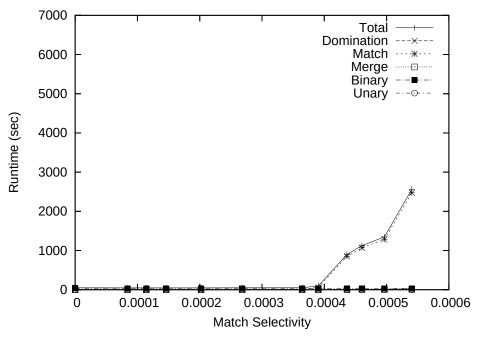

Figure 4: Match selectivity impact (Enhanced)

with each other, causing an explosion in record comparisons.

The runtime of the Enhanced algorithm (Figure 4) does not change much until the analogous surge in work. Unlike the General algorithm, however, the runtime due to binary negative rule invocation stays almost constant due to two competing factors. As the merge closure increases, the average package size becomes larger, increasing the number of binary rule invocations for each component; however, there are now fewer packages, reducing the cost of constructing the connected components. (Constructing connected components requires, in the worst case, a pairwise comparison between all the packages using the binary negative rule.) The sharp increase of runtime at the end of the plot is due to the "skewed" data more than the nature of the algorithm. Although the idea of the Enhanced algorithm is to run the General algorithm on small components, the components themselves were not evenly sized. For example, during a point where the Enhanced algorithm was taking 60 seconds to run, there happened to be nine records in the same package having very long and similar titles. (In fact, they were the same products only differing in color). These records together produced a merge closure of size 150, meaning that many composite records having several long titles were created. Comparing such composite records takes longer than comparing single records because of the existential title comparisons we used. Running the General algorithm on the package took about 45 seconds, contributing to most of the algorithm runtime. In an ideal situation where the components sizes are small and fixed, the Enhanced algorithm would have a quadratic runtime behavior.

In summary, the runtimes for the General and Enhanced algorithms do not change much until there is an explosion of matching records. The explosion is due to many similar records matching with each other. Although the Enhanced algorithm has a quadratic behavior in an ideal situation, the concentration of records in certain packages caused the Enhanced algorithm to also have an abrupt runtime increase.

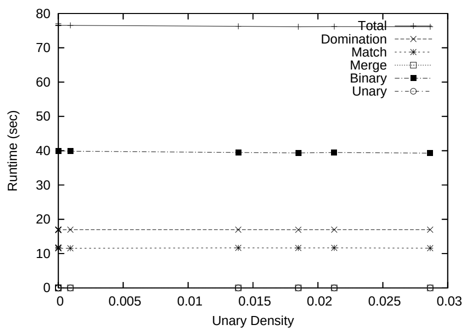

Figure 5: Unary density impact (General)

4 Unary Density Impact

We measured the performances of the two algorithms by varying the unary density on 1,000 random iPod-related records. Varying the price threshold p_N of the unary negative rule affects the unary density; a higher p_N results in a higher unary density because the prices of a record have a higher chance of being different enough for the record to be internally inconsistent.

The total runtime decreases as the unary density increases. The reason is that, as more records are internally inconsistent, fewer binary negative rules are invoked between the records chosen and the records in the set S. Figures 5 and 6 show the results of running the General and Enhanced algorithms as the unary density varies. Note that unary density has little impact on performance. The main reason for the low density was that many records in the merge closure were single records having single prices. Even if there were composite records, they usually consisted of near-identical prices. We suspect that in applications with higher unary densities than what we were able to achieve here, the performance impact would be more marked.

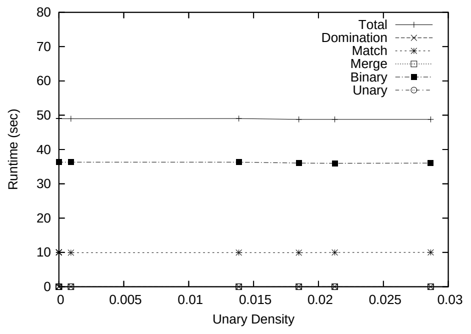

Figure 6: Unary density impact (Enhanced)

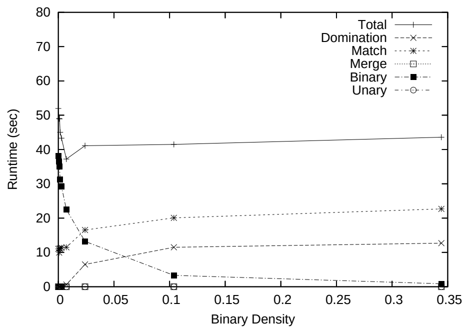

Figure 8: Binary density impact (Enhanced)

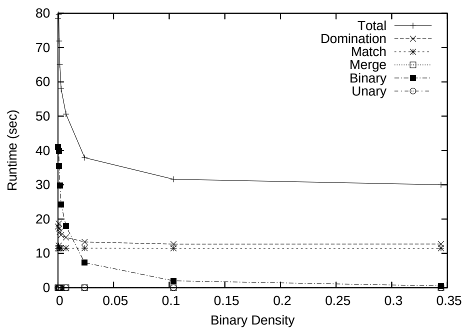

Figure 7: Binary density impact (General)

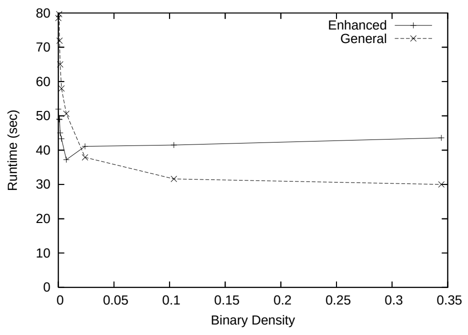

Figure 9: Binary density impact (General vs. Enhanced)

5 Binary Density Impact

We measured the performances of the two algorithms by varying the binary density on 1,000 random iPod-related records. Varying the title threshold t_N of the binary negative rule affects the binary density; a lower t_N results in a higher binary density because a pair of records is more likely to be inconsistent.

The runtime of the General algorithm decreases inversely to the binary density (Figure 7). As the binary density increases, more records in S tend to be inconsistent with the records chosen by the solver and are discarded early. That is, fewer records need to be processed in the future, reducing the number of binary negative rule invocations. The time for invoking the match rule stays constant because the merge closure itself does not change.

The Enhanced algorithm is faster than the General algorithm for small binary densities, but is actually slower for larger densities. Figure 8 shows the runtime decomposition of the Enhanced algorithm while Figure 9 shows the total runtime comparison between the General and Enhanced algorithms. Initially, the Enhanced algorithm is faster because the component sizes are small, minimizing the time for running the General algorithm on each component. As the binary density increases, however, the components get larger, and running the General algorithm on them takes longer. For high binary densities, the benefits of the Enhanced algorithm disappear because the merge closure is no longer partitioned into smaller components. Fortunately, our binary density for the default thresholds is very small (7.86E-4), making the Enhanced algorithm about 38% faster than the General algorithm.

In summary, the Enhanced algorithm outperforms the General algorithm for low binary densities. For high binary densities, the Enhanced algorithm loses its benefit of partitioning the merge closure into smaller components. Note that, in our base case, our binary density is very small (7.86E-4), making the Enhanced algorithm 38% faster than the General algorithm.

6 Human Effort

We measured the total human efforts for the two algorithm on 500 to 2,500 random iPod-related records. In our runs, the decisions of the solver were simulated using a heuristic that finds the "largest" record among the non-dominated records. Specifically, any record that is a combination of the largest number of base records was chosen. For instance, given a set of non-dominated records $\{r_1, r_{23}, r_{45}\}$, the system chooses either r_{23} or r_{45}. This approach requires a linear scan on the non-dominated records, which is consistent with our notion that human effort is related to the number of records examined. We did not use the optimization mentioned in Section 1.2.

The total human effort for the General algorithm in-

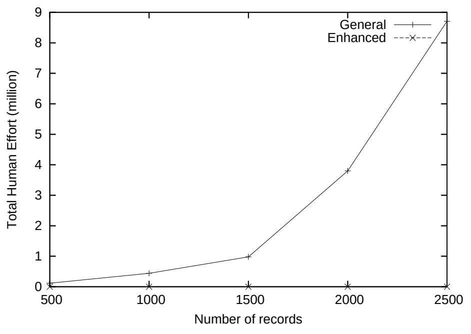

Figure 10: Human effort

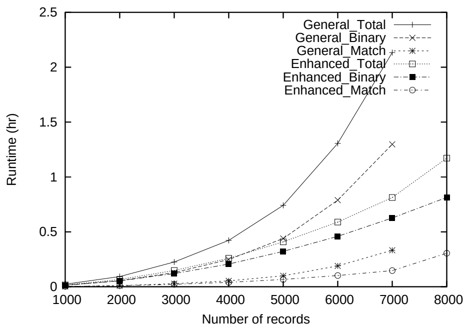

Figure 11: Runtime scalability

creases rapidly as the number of records increases (Figure 10). As the number of initial records increases, the merge closure becomes larger, resulting in more records for the solver to view. The significantly larger human effort for the General algorithm compared to the Enhanced algorithm is due to the highly redundant records views, which can be illustrated by the following example. Suppose that a set of initial records has a merge closure size of 100. Moreover, suppose that the initial records form ten connected components where each component has a merge closure size of 10. For simplicity, we ignore the inconsistency and domination relationships among records. For the General algorithm, the solver must view $\sum_{i=2}^{100} i = 5499$ records; for the Enhanced algorithm, the solver only needs to view $10 \times \sum_{i=2}^{10} i = 540$ records, which is about one-tenth the effort of the General algorithm effort. Although the merge closure is the same for both algorithms, the Enhanced algorithm saves a lot of redundant views because it partitions the merge closure into many smaller independent components.

On the other hand, the total human effort for the Enhanced algorithm is negligible compared to the General algorithm. Although not clearly shown in the plot, the human effort of the Enhanced algorithm for 2,500 records is 170 (versus 8.7 million for the General algorithm). In addition to the advantage that the merge closure was partitioned into smaller components, many components consisted of single records, saving a great amount of human effort. (Recall that we do not count the effort to choose a record from a component consisting of a single record.)

As we argued in Section 1.2, the absolute values in Figure 10 are not meaningful. For instance, the 8 million value (with 2,500 input records) does not mean the solver will examine 8 million records. Faced with this scenario, a solver will most likely use an imperfect strategy, e.g., selecting records based on intuition, and not examining all the choices. However, what we can infer from Figure 10 is that for the same style of decision-making, the Enhanced algorithm will require significantly less human effort.

7 Scalability

We conducted scalability tests for the General and Enhanced algorithms on 1,000 to 8,000 records randomly selected from the entire comparison shopping dataset provided by Yahoo!. We used the entire dataset, which is a strict superset of the iPod-related data, in order to test on larger data sets. Since the data was not specialized like the iPod-related data, the match selectivity was lower than our default value.

Figure 11 shows the runtime scalability for each algorithm. The General algorithm cannot handle more than 7,000 records in a reasonable time while the Enhanced algorithm can handle more records. The largest runtime bottleneck for both of the algorithms was the cost of invoking the binary negative rules. For applications that use less expensive rules, the scalability will improve accordingly.

In summary, although the Enhanced algorithm outperforms the General algorithm in scalability, Entity Resolution with Negative Rules is an inherently expensive process and thus only relatively small sets of records can be handled. Thus, large data sets need to be partitioned into smaller sets that can be resolved in detail. How large a data set can be exhaustively resolved depends on the application. It is also possible to distribute the ER-N computations across multiple processors, in order to handle larger data sets. We can use techniques similar to the ones in [1] to distribute the data and computation among processors.

References

[1] O. Benjelloun, H. Garcia-Molina, H. Kawai, T. E. Larson, D. Menestrina, and S. Thavisomboon. D-swoosh: A family of algorithms for generic, distributed entity resolution. In *ICDCS*, 2007.

[2] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[3] S. E. Whang, O. Benjelloun, and H. Garcia-Molina. Generic entity resolution with negative rules. Technical report, Stanford University, 2007. Available at http://dbpubs.stanford.edu/pub/2007-17.