A New Data Structure For Fast Approximate Matching

Andrew P. Berman
University of Washington
Department of Computer Science
aberman@cs.washington.edu

March 2, 1994

Abstract

Given a set of objects S and a metric D, we describe how to represent S as a new data structure, the *triangulation trie*. This data structure can be used to search through S quickly to find approximate matches to a given object. Using the triangle inequality, the search tree is repeatedly pruned to reduce the number of object comparisons required. Much of the work is done within the tree using integer comparisons. This method can result in very fast database searches in applications where object comparisons are traditionally costly. Furthermore, the data structure seems to be applicable to a very wide variety of object types. The trie is unusual in its construction in that objects are partitioned according to their respective distances from a common set of "key" objects.

1 Introduction

When searching through databases, people are often interested in close but not necessarily exact matches. This problem of finding an "approximate" match is a common one, occurring in fields as varied as spell checking, fingerprint analysis, voice recognition, image understanding, and DNA sequence matching. In general, the standard problem consists of a set of objects S drawn from some universal set of objects U with a *distance measure* D on $U \times U \rightarrow \Re$. Given integer k and $x \in U$, the goal is to find all $y \in S$ such that $D(x, y) \leq k$.

It can be expensive to find $D(x, y)$ for certain U and D. For example, in the *sequence alignment problem*[NW70], finding $D(x, y)$ takes time proportional to $|x||y|$. Thus, in database searching, there is a need for sublinear searches, where most of the database is never directly examined. Some current methods of pruning database searches include using some subroutine other than the original distance measure D defined on U. For example, the BLAST[AGM+90] algorithm for genetic databases uses comparisons on short subsequences

of the DNA in question. While BLAST is very fast, this method of using subsequences cannot be applied to other pattern-matching problems where the term "subsequence" may have no meaning.

The *Triangulation Trie* defined in this paper can be used to create and search databases with only the given distance measure as a tool. The only requirement is that the given distance measure satisfy the triangle inequality. An algorithm described below can be used on the trie to return a subset of objects in the trie, thus avoiding the cost of comparing a test object to each object in the database. The filtering process can result in false positives which can later be eliminated through direct comparison with the test object. However, the algorithm will never fail to report a correct match. The data structure and algorithm have been tested on string sequences using two different distance measures, *edit distance* and *Hamming distance*; and on sets of graphs testing for isomorphism. In tests on these cases, we were to eliminate many object comparisons, sometimes resulting in a substantial time savings.

In Sections 2 and 3 we give algorithms for constructing and searching through the trie. Section 4 analyzes the behavior of the data structure and search on binary strings using the Hamming distance measure. Section 5 discusses some experiments on searching through a set of graphs for isomorphic and "close to isomorphic" graphs. In Section 6, we show how this data structure could be used to organize and search through a database of dresses in a department store catalog.

We have recently become aware that a paper containing similar results will be presented at the June Combinatorial Pattern Matching Conference[BYCMWar].

2 Constructing The Triangulation Trie

We make the simplifying assumption that distances are integer-valued. For objects with real-valued distance measures, some mapping into an integer-valued distance measure must be made.

The construction of the trie is simple: We first choose a set of *key objects* from our universal set. Most of our experiments were done using random sets of key objects, but the key objects can be chosen according to arbitrary convenient criteria. Note that the performance of searches in the trie depends on the choice of the key objects. We create a vector for each object in the database consisting of the ordered set of distances to the key objects. These vectors are then combined into a *trie*[Fre60]. The trie is a compact representation of the distances from the objects in our database to this set of "keys". Each path from the root to a leaf represents the ordered set of distances from the keys to the objects in that leaf. Note that each object is in precisely one leaf but a leaf may contain more than one object. The latter occurs when two or more objects are the same distance from each of the keys.

Formally, let $S = (x_1, ...x_n)$ be our set of objects in the database. Let $key_1, ...key_j$ be another set of objects, known as "key objects". For each x_i in S compute the vector $v_i = (D(x_i, key_1), D(x_i, key_2), \ldots, D(x_i, key_j))$. Then combine the vectors $v_1, \ldots, v_n$ into a

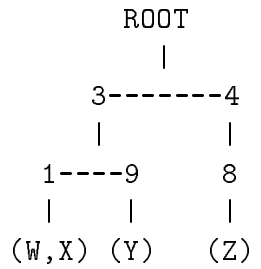

Figure 1: An Example Trie with Four Objects and Two Keys

trie, with x_i being placed on the leaf reached by following the path represented by v_i.

2.1 Constructing the Trie: An Example

Let $S = (W, X, Y, Z)$ be our objects and (key_1, key_2) be our set of keys. Let $v_W = (3, 1)$, $v_X = (3, 1)$, $v_Y = (3, 9)$, and $v_Z = (4, 8)$. A depiction of the trie is shown in Figure 1.

3 Searching The Trie For An Approximate Match

The search is based on the following mathematical equivalent of the triangle inequality:

$$\forall x, y, z \in U, D(x, y) \geq |D(x, z) - D(y, z)|$$

Suppose we are given object x and integer k and wish to find all objects in our database with a distance from x of not more than k. Now, consider a node p at level l with a value of d. Every object at leaves descendant from p has a distance of d from the key object key_l. Thus, if $|d - D(x, key_l)|$ is greater than k, then we know from the triangle inequality that $D(x, s')$ is greater than k for all object s' which are descendants of p. Thus, we can safely prune the search at node p.

The algorithm for searching the database is straightforward. Compute the distances from x to each key: $D(x, key_1), \ldots, D(x, key_j)$. Perform a depth-first search of the trie, pruning the search at node p at level l with value d if $|d - D(x, key_l)|$ is greater than k.

When a leaf is reached, measure the distance from x to every object in the leaf and return those objects s for which $D(x, s)$ is less than or equal to k.

3.1 Searching the Trie: An Example

We continue with the example from Section 2.1. Suppose we wish to search our database for a close match to object V where our maximum allowed distance to V is 1. We compute v_V by calculating $D(V, key_1)$ and $D(V, key_2)$. We discover that $v_V = (3, 8)$. We then perform

our depth first search. At the top level, we only search nodes with a value within 3 ± 1. At the second level, we only search children of those nodes with a value within 8 ± 1. Figure 2 shows the trie with asterisks placed on those nodes and leaves which were examined. It shows that Y and Z are returned as potential matches, while X and W are eliminated. The final step is to compute $D(V, Y)$ and $D(V, Z)$. Note that $D(V, X)$ and $D(V, W)$ are never computed.

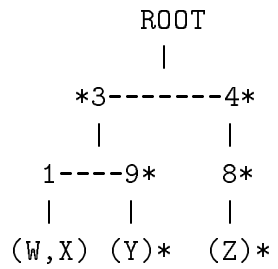

Figure 2: Searching the Example Trie for Matches

Notice that in this example the *nature* of the objects is never stated. The flexibility of the structure is such that it can handle any type of object as long as there is a distance measure defined on the object space that satisfies the triangle inequality.

Suppose that Y was a close match to V but Z was not. Additional keys would give additional chances for eliminating Z from the returned set. However, additional keys increase the depth of the trie, increasing the cost of searching the internal nodes. Thus the optimum depth of the trie is highly dependent both on the cost of comparing two objects and the behavior of the distance measure on the objects.

4 Random Binary Strings and Hamming Distance: An Example

Suppose we wish to find approximate matches to a set S of random binary strings of length p. Assume that our distance measure D simply counts the number of bits which differ between two strings. This is known as the *Hamming distance*. We show that a triangulation trie with logarithmic depth will enable us to do very fast searches within distance k, where k can grow with $O(\sqrt{p})$.

To achieve this result we must first examine the behavior of the Hamming distance function on random binary strings. For the remainder of this section we assume all strings are of length p. Let $D(x, y)$ be the Hamming distance on strings x and y. We need to understand the behavior of two strings with distance d relative to a third random string. The following lemma gives us this information:

Lemma 1: Let x and y be arbitrary strings, and let r be a random string. Let $R(d)$ be a random variable representing the final distance to the origin of a simple random walk of length d on the line. Then:

$$P(|D(x,r) - D(y,r)| \leq k | D(x,y) = d) = P(R(d) \leq k)$$

Proof: Let x_i, y_i, and r_i represent the i^{th} bits of x,y, and r respectively. Let x'_i, y'_i, and r'_i represent the i^{th} prefixes of x, y, and r. Consider a point on the line which starts on the origin at time 0, and at time t moves $+1$ if $x_t \neq r_t$ and $y_t = r_t$, -1 if $x_t = r_t$ and $y_t \neq r_t$, and does not move at all if $x_t = y_t$. Thus, the position of the point increases by $+1$ at time t when $D(x'_t, r'_t) - D(y'_t, r'_t) = D(x'_{t-1}, r'_{t-1}) - D(y'_{t-1}, r'_{t-1}) + 1$. It decreases by -1 at time t when $D(x'_t, r'_t) - D(y'_t, r'_t) = D(x'_{t-1}, r'_{t-1}) - D(y'_{t-1}, r'_{t-1}) - 1$. And it doesn't move when $D(x'_t, r'_t) - D(y'_t, r'_t) = D(x'_{t-1}, r'_{t-1}) - D(y'_{t-1}, r'_{t-1})$. This last claim is seen when one observes that when $x_t = y_t$, either $D(x'_t, r'_t)$ and $D(y'_t, r'_t)$ both increase by one from the previous time step, or they both remain the same, depending on whether $x_t = y_t = r_t$.

Thus, the movement of the point represents a walk where steps are taken only when $x_i \neq y_i$. The position of the point after p steps is $D(x,r) - D(y,r)$. To show this movement is a simple random walk of length d, we now only need to point out that since r is a random string, on those bits i where $x_i \neq y_i$, we have:

$$P(x_i = r_i) = P(y_i = r_i) = \frac{1}{2}$$

The above lemma shows that the relative distance of two strings to a random third string is equivalent to a random walk on the bits that are different between the two strings. If we are searching for close matches to a string x, and some string s in our database is "far" from x, we want $|D(x,r) - D(s,r)|$ to be large for some key string r—large enough to eliminate s from our search. The following lemma describes the probability of eliminating s:

Lemma 2: Let $R(d)$ be a random variable representing the length of a simple random walk on the line of length d. Then the following approximation holds:

$$P(R(d) \leq k) \rightarrow \Phi(\frac{k}{\sqrt{d}}) - \Phi(\frac{-k}{\sqrt{d}})$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal random variable.

Proof: We use the DeMoivre-Laplace Limit Theorem[Fel70]. $R(d)$ is no greater than k if the difference between rightward and leftward moves is no greater than k. This means that there are between $\frac{d-k}{2}$ and $\frac{d+k}{2}$ rightward moves. Thus, if X_d denotes the number of rightward moves in d independant moves, we have:

$$P(R(d) \leq k) = P(\frac{d-k}{2} \leq X_d \leq \frac{d+k}{2})$$

$$= P(\frac{\frac{d-k}{2} - \frac{d}{2}}{\sqrt{\frac{d}{4}}} \leq \frac{X_d - \frac{d}{2}}{\sqrt{\frac{d}{4}}} \leq \frac{\frac{d+k}{2} - \frac{d}{2}}{\sqrt{\frac{d}{4}}})$$

$$= P(\frac{-k}{\sqrt{d}} \leq \frac{X_d - \frac{d}{2}}{\sqrt{\frac{d}{4}}} \leq \frac{k}{\sqrt{d}}) \rightarrow \Phi(\frac{k}{\sqrt{d}}) - \Phi(\frac{-k}{\sqrt{d}})$$

We note that $\Phi(\frac{k}{\sqrt{d}}) - \Phi(\frac{-k}{\sqrt{d}})$ is less than 7/10 when $\frac{k}{\sqrt{d}}$ is less than or equal to 1. This fact leads us to the following lemma:

Lemma 3: Let $c = 10/7$. Let a triangulation trie T be composed of a set S of random binary strings of length p with $\lceil \log_c |S| \rceil$ keys chosen randomly and uniformly from all binary strings of length p. Suppose a string x is our test string and we wish to search through T for all strings $s \in S$ such the distance from x to s is less than or equal to some integer k. Then we make the following two claims:

Claim 1: We expect to examine not more than one string in S with a distance from x of greater than k^2.

Claim 2: We expect to examine not more than $O(J \log_c(|S|) + |S|^{1-1/\log(c(2k+1))})$ internal nodes of T where J is the total number of strings in S we do examine.

Proof of Claim 1: Consider T from the point of view of a given string $s \in S$ in the tree with distance from x of greater than k^2. The path to s covers $\lceil \log_c(|S|) \rceil$ integers, representing the distance from s to the randomly chosen keys. Let this sequence of integers be $(d_1, ...d_{\lceil \log_c(|S|) \rceil})$. Lemma 2 tells us that if $D(s,x) > k^2$ for a string s, then with probability greater than $1 - 1/c^{\lceil \log_c(|S|) \rceil} \geq 1 - 1/|S|$, there will be at least one integer i such that $|D(x, r_i) - d_i| > k$, which will eliminate s from the search. Since there are fewer than $|S|$ strings total of distance greater than k^2 from x, it is expected that at most one such string will survive the pruning and be returned.

To give some perspective on this, let us consider a database of $1,000,000$ strings of length 60. If we create a triangulation trie with 40 keys, then a search for matches within a distance of 3 to some string x is expected to yield not more than 1 string whose distance to x is greater than 9. If the original strings are random, we expect that no strings have a distance to x of less than 10. Thus, we have eliminated all but one of the strings in our set, at a cost of a mere 40 key comparisons at the outset! If we search for strings within a distance of 4 to x, then Lemma 3 says that we can expect to have fewer than 200 strings returned as potential candidates. If we search through a distance of 5 the lemma only tells us that we can expect at least 13% of the strings will be eliminated. However, remember that the lemma places a lower bound on the expected number of strings that are eliminated from the search. In actual practice, we should get even better results. See section 4.1 for an example.

Proof of Claim 2: We ignore floors and ceilings for the purpose of exposition, noting that we are after an asymptotic value. The triangle inequality restricts the maximum number of children searched per node to $2k + 1$, giving a potential of $(2k + 1)^d$ nodes searched at level d. However, the number of nodes searched at any level is limited to the number of still viable strings at that level. From Lemma 2, the number of viable strings at level d is less than $J + |S|/c^d$. Thus, if L_d is the number of nodes searched at level d,

Max Distance Allowed	Number of Nodes Searched	Number of Strings Returned As Potential Matches
3	1572	0
4	5200	0
5	13908	3
6	31839	37
7	64304	247
8	116229	1002
9	188149	2779
10	273937	5794

Table 1: Average Number of internal nodes searched and strings returned in a triangulation trie with 25,000 random strings of length 60 with a trie depth of 29.

then $L_d \leq \min((2k+1)^d, J + |S|/c^d)$. Assuming J is much smaller than $|S|$, the maximum value of L_d occurs when $(2k+1)^d = |S|/c^d$, which, after some calculations, yields $d_{max} = \log(|S|)/\log(c(2k+1))$, giving approximately $|S|^{1-1/\log(c(2k+1))} + J$ nodes searched at that level. $L_d - J$ is constrained by reducing factors of $1/(2k+1)$ for $d < d_{max}$ and by reducing factors of c for $d > d_{max}$, giving a limit of $O(J \log_c(|S|) + |S|^{1-1/\log(c(2k+1))})$ on the total number of nodes searched.

4.1 Test Results:

Testing was done on binary strings of length 60. The set size was 25000, and a tree depth of 29 was used ($\log_{(10/7)}(25000) \approx 28.4$). As is shown in Table 1, the experimental results are significantly better than the theoretical prediction. For example, note that even searching within a distance of 8, an average of 96% of the strings were removed from consideration, while Lemma 3 doesn't promise any filtering.

5 Finding "Almost Isomorphic" Graphs

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a permutation π such that $\forall i \forall j (i,j) \in E_1 \iff (\pi(i), \pi(j)) \in E_2$. We can further define a distance measure D over the set of graphs by letting $D(G_1, G_2)$ be equal to the minimum number of edges that need to be removed or added to G_1 to make G_1 isomorphic to G_2. It is not known if the question of whether two graphs are isomorphic is NP-complete[GJ79], but so far, nobody has come up with a polynomial time solution. Since learning $D(G_1, G_2)$ answers the question of whether G_1 and G_2 are isomorphic, it is safe to say that no polynomial-time method of computing D is currently known. This makes D an excellent distance measure to test the

		Max Distance Allowed				
		0	1	2	3	4
Tree Depth	5	0.41	43.16	205.33	359.03	463.73
	10	0.24	21.59	148.34	314.11	443.69
	15	0.24	12.87	111.73	268.31	421.62
	20	0.24	12.45	106.57	257.55	409.36
	25	0.24	11.82	100.19	248.52	405.84
	30	0.24	11.30	95.59	239.63	402.74
	50	0.24	11.00	91.84	234.16	399.79

Table 2: Average number of graphs returned as potential close isomorphisms while searching through a set of 500 random graphs of 6 vertices.

		Max Distance Allowed				
		0	1	2	3	4
Tree Depth	5	5	102	393	662	836
	10	6	216	1102	2093	2826
	15	7	272	1600	3296	4712
	20	8	324	2062	4437	6553
	25	9	374	2497	5530	8352
	30	10	421	2908	6581	10135
	50	14	602	4480	10667	17213

Table 3: Average number of nodes searched in trie, rounded to the nearest integer while searching through a set of 500 random graphs of 6 vertices.

triangulation trie, since minimizing computations of D would be a very high priority if one had to find close matches within a large set of graphs to another graph. We created a set of 500 random graphs of 6 vertices, where the probability of an edge between two vertices was 1/2 and each edge was chosen independently. We then created several triangulation tries with different depths and looked for approximate matches to another set of 100 graphs, with maximum distances allowed ranging from 0 to 4. Table 2 shows our results. Even a modest trie depth of 5 is able to eliminate over 90% of the graphs from consideration when searching for isomorphisms within a distance of 1. When searching for exact isomorphisms, a tree depth of 5 could eliminate all but 1 graph from consideration most of the time. A single comparison is so expensive that larger tree depths seem to be very worthwhile even if the added depth eliminates just a few nodes from consideration. In our example, when searching for graphs within a distance of 3, a depth of close to 30 seems to be the optimum choice.

Table 3 shows the average number of internal nodes examined. When the cost of measuring the distance between two objects is very high, even moving through hundreds or thousands of internal nodes may not add much to the total time. In this example, each computation of D took an average of more than a tenth of a second compared to faster than thousandths of a second to move through a node in the trie.

6 Finding The Almost Perfect Dress

The purpose of this section is to show the flexibility of the data structure in its ability to deal with objects which may otherwise be difficult to organize. Consider the following problem: A woman comes into a department store and asks the salesperson to help her find a dress. She describes what she considers to be the "perfect" dress for herself, but will accept something that is "close to perfect." Dresses vary in many different ways including, but not limited to, size, fabric type, skirt length, sleeve length, buttons (size, type, number and placement), and fabric patterns. The question arises— how can the salesperson quickly find a set of good candidate dresses to show the customer?

Standard methods might provide an answer by having the customer provide a "range" of acceptable variance for each variable. For example, instead of saying she wants to spend \$100, she can say \$80 to \$120. The problem is that a database search using this method might return a dress which is just within the range on every variable, and not return a dress which costs \$121 but is perfect in every other way.

The triangulation trie enables a solution to this problem. One can construct a distance measure over the set of dresses. For example, one simple measure would be to take each variable such as size or skirt length in turn, and count the number of differences between two dresses. A more complicated measurement might weight the variables according to what people are usually willing to give up. For example, a customer is probably more willing to be flexible on the button placement than on the skirt size. In any case, once the distance measure is created, a triangulation trie of dress descriptions can be used to search for approximate matches to some ideal dress.

7 Conclusion

In this paper we have introduced a new data structure, the triangulation trie. This data structure is easy to construct, and our theoretical and experimental data suggests that the trie can be a powerful tool in organizing and approximate searching over a wide variety of object types, especially in cases where individual object comparisons are expensive.

8 Acknowledgments

I'd like to thank Larry Ruzzo for guiding me in this research and for editing. I'd like to thank my wife Debbie Berman, Rex Jakobovits, and Brendan Mumey for help in editing and many suggestions. I'd like to also thank Kevin Mounts for suggesting the dress example.

References

[AGM+90] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[BYCMWar] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Combinatorial Pattern Matching*, June 1994 (to appear).

[Fel70] W. Feller. *An Introduction to Probability Theory and its Applications.* John Wiley & Sons, third, revised edition, 1970.

[Fre60] E. Fredkin. Trie memory. *Communications of the ACM*, (3):490–499, 1960.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[NW70] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.