Efficient Joins With Reference Set

Andreas Villscheider
Free University of Bozen

Supervisor: Nikolaus Augsten
Free University of Bozen

September 14, 2010

Contents

1 Introduction **4**

2 Related Work **6**

3 Preliminaries **7**

4 Problem Definition **8**

5 Search Space Reduction **9**

6 Choosing A Reference Set **10**
6.1 Draw a sample set 11
6.2 Cluster the sample set 12
6.3 Draw reference points 13

7 Experiments **14**
7.1 The data sets 14
7.2 Size of sample set 14
7.2.1 Evaluation concepts 15
7.2.2 Unclustered 16
7.2.3 Regular clustered 17
7.2.4 Unregular clustered 18
7.2.5 Average case 19
7.3 Varing threshold 20
7.3.1 Unclustered 20
7.3.2 Unregular clustered 21
7.3.3 Regular clustered 22
7.3.4 Average clustered 23
7.4 Varing threshold and input range 24
7.5 Experiment with real data 25

8 Conclusion **26**

9 Appendix **28**
9.1 Application 28

Abstract

How similar are two objects? Are they almost equal or are they too different. This is a question of importance in many sciences such as genetic research, crime scene investigations, linguistic sciences, computer sciences and many more. In computer science this could be of relevance in case like a join of data out of different sources.

A practical scenario could be the join of databases of inhabitants of a city to enrich information. The join could be performed over the addresses. Performing an exact join over the addresses may end up with poor results because of the lack of common keys, a different 14way of naming the same real world object or misspelled addresses. To perform an effective join, we have to match also similar pairs. Finding similar pairs involves evaluating a distance function which is often very expensive to compute. In the brute force approach the distance must be computed for each pair of objects from both sets leading to very expensive joins. Our goal is to minimize these direct comparison using a reference set. With this reference set we will be able to compute upper and lower bounds which we will use as filters.

Instead of directly comparing every element of the original sets, we compare each of the original sets with each element in the reference set. Since the reference set is much smaller than the original sets there are much less comparisons to do. Now, applying the triangle inequality, we are able to save comparisons between some elements because they either match or do not match for sure.
An important thing is to find a good size for the reference set. Small reference sets provide little overhead, because additional elements in the reference set require additional distance computations at query time. Large reference sets provide more effective filters thus save comparisons.

In this thesis I will show for which cases the approach with the reference sets works well and for which cases it works less effective. I will show when the traditional brute force method would be a better choice. For that I will experiment with the size and the clustering of the input sets as with the size of the reference set to find a optimum. Further I will show the impact of the threshold on the comparisons we save. The experiments will be done on syntactical and real data.

1 Introduction

In many applications the quenstion of similarity of elements is fundamental. In genetic research as in crime scene investigations we have to find matches of similar DNA sequences or finger prints. In linguistic sciences the similarity of words is of interest.
An other application scenario is data integration. For example a municipal wants to enrich information about its inhabitants. It takes its own geographic data sources and tries to join them with the data of the province. Due to the lake of a common way to store street names (see Figure 1, 2), an exact join ends up with poor results, and a approximate matching technique must be applied.

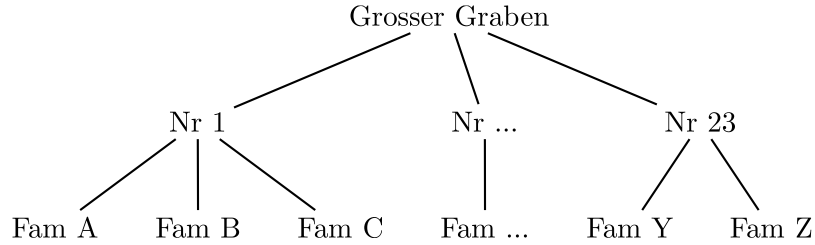


Figure 1: Naming convention of municipal

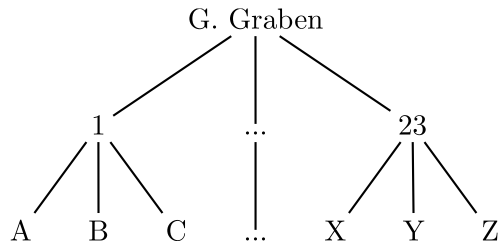


Figure 2: Naming convention of province

Definition 1 (Distance Function) *Given two sets of elements, A and B, a distance function for A and B maps each pair (a,b) $\in A \times B$ to a positive real number (including zero).*

$$\delta : A \times B \rightarrow \mathbb{R}_0^+$$

Definition 2 (Approximate Join) *Given two sets of elements, A and B, a distance measure, dist(a,b), between the elements $a \in A$ and $b \in B$, and a threshold τ. The approximate join computes all pairs (a,b) $\in A \times B$, such that dist(a,b) $\leq \tau$.*

The distance function depends on the elements we want to join. In the case we want to join strings we could use the "string edit distance" or the "longest common subsequence". For points in the Cartesian plan the "euklidean distance" would work well as approach. In the municipal scenario (Figure 1, 2), the items are structured as a tree like in XML documents. Here the standard way for this is the so called "tree edit distance". It defines the distance between two trees as the minimal number of edit operations (node insertion, relable, deletion) required to transform one tree into the other.
Since in this paper I will not have a look at a specific distances function I introduce the term DIST() and will use this for every kind of distance computation. The computation time of most distance functions is very high[1]. This is not a big problem when we have to compare just small sets. But since the brute force approximate join requires $|A| \times |B|$ distance computations with larger sets we have to do a lot of expensive computations. For example with 2 sets of 1000 elements we already have to do 1 mio distances computations. To address this problem we have to keep the number of expensive DIST() as low as possible. This means, we first have to reduce the search space some how.

In this paper I will show an approach proposed by Guha et al. [11, 10] which tries to reach this reduction introducing a third set the so called reference set. This set consists of some elements of set 1 and set 2. I will show and anaylse the proposed approach, that does not compute the distance functions of the cross product immediately. It pre-computes the distances from each element of set 1 and set 2 to each element of the reference set. Knowing the distances between a,b (a $\in$ set 1, b $\in$ reference set) and b,c (b $\in$ reference set, c $\in$ set 2) with the triangle inequality an upper and a lower bound can be computed on DIST(a,c) (a $\in$ set 1, c $\in$ set 2). The upper and lower bounds can be used to prune the search space.
It is evident that the size of the reference set is an important factor for this approach. It should not be to large since the size of the set is proportional to the all over work we have to do. Neither it should be to small because it provides just effective filters when it is large enough. I will show how according Guha et al. [11, 10] we find this optimum.
Further I will show the results of some experiments. One experiments states for which thresholds the reference set approach works fine and why we reach our average worst case at half way of the largest distance between the elements. An other experiment shows how the "reference set approach" deals with differently clustered input data. The experiments are done on syntactical and real data with different cluster types and varying sizes of the input sets.

[1]Worst case for the "tree edit distance algorithm" is $O(N^4)$ where N is the sum of the nodes of tree 1 and tree 2

2 Related Work

Matching approximate elements is a problem of central interest in many applications. In crime scene investigations for example the investigators have to compare dusted finger prints [16] with their large databases to catch the criminals. In genetic research a possible scenario could be the extraction of a special gene of a DNA sequence [15] to determine whether it is responsible for a given disease. Also here we have to compare a lot of different DNA sequences among themselves.
With different approaches it is possible to map the differences between the compared elements into the metric space [4, 7]. This paper assumes such metric distances. Such distances were already introduced for different complex data types such as strings [14, 9][Levensthein 1966], pairs of trees [Zhang and Shasha 1997], ordered labeled trees [2][Tree Edit Distance Apostolico and Galil 1992; Sankoff and Kruskal 1983] like XML documents [1, 3, 13, 8, 5, 6] and so on. These distances are often combined with high costs such as string edit distance and tree edit distance. For that we have to keep the number of these distance computations as low as possible.

The algorithm for approximate joins which is presented and evaluated in this paper is proposed by Guha et. al [11] in the paper "Integrating XML Data Sources Using Approximate Joins". The algorithm minimizes the pairwise distance computations using an attribute of triangle inequality. With that approach introducing a new set, the so called reference set, it is possible to determine an upper and a lower bound for each distance between the elements. Applying these bounds as filter it we are able to prune some pairs of elements with out directly computing the distance between them. To get these bounds we first have to sample the input sets. This sample becomes clustered [12] and out of the largest n clusters we pic our reference set elements.
The proposed approach was to implement nearly straight forward. Just the choice of the size of the sample set required foreknowledge about the numbers of clusters in the sample before executing the method. Since that is not always the case I varied the method of choice. More about that later in section "Draw a sample set".

3 Preliminaries

A data set is a collection of elements. In our case a set for example can consist of XML-documents, strings or coordinates.
The reference set approach works with an attribute of the triangle inequality. The triangle inequality in general says that one side of a triangle can be at most so long as the sum of the other two sides.
Formally this means $c < a + b$. The graphical mean is shown in Figure 3.

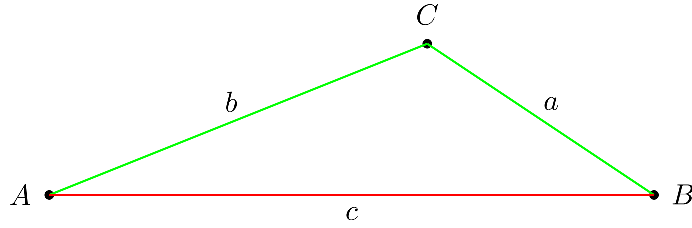


Figure 3: $c < a + b$

Because of symmetric reasons $a < c + b$ is true and we can derive $|a - b| \leq c$ or analog $|b - a| < c$ (see Figure 4).

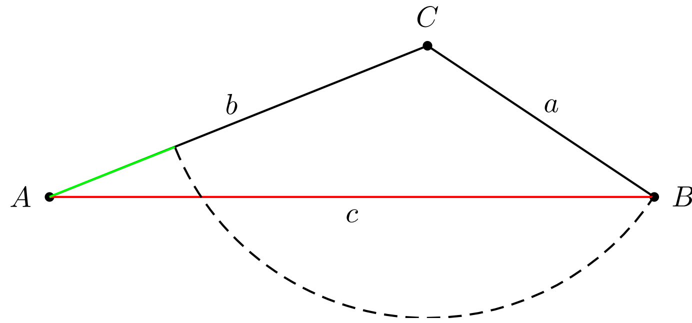


Figure 4: $|b - a| < c$

Further we distinguish two special cases. One when the sum of a and b is exactly as long as c ($b + a = c$). The other when the difference of a and b is as long as c ($|b - a| = c$). Taking all cases together we can sum up:

$$|b - a| \leq c \leq a + b$$

Applying this formula we are able to calculate an upper and an lower bound for the length of c knowing the length of a and b. For the approximate join with reference set we can use this attitude as follows. Knowing the distances from the elements in set 1 and set 2 to the elements in the reference set we

can calculate an upper and an lower bound for the distances of the elements in set set 1 to the elements in set 2.

$$|DIST(x,y) - DIST)(x,z)| \leq DIST(x,z) \leq DIST(x,y) + DIST)(x,z)$$
$$\text{where } x \in \text{set } 1,\ y \in \text{reference set and } z \in \text{set } 2$$

4 Problem Definition

Given two sets of size n and m. A distance computation of each element of one set, to each element of the other will end up in n*m DIST() (see Figure 5). So the runtime complexity is O(n^2). Filter algorithms are special algo-

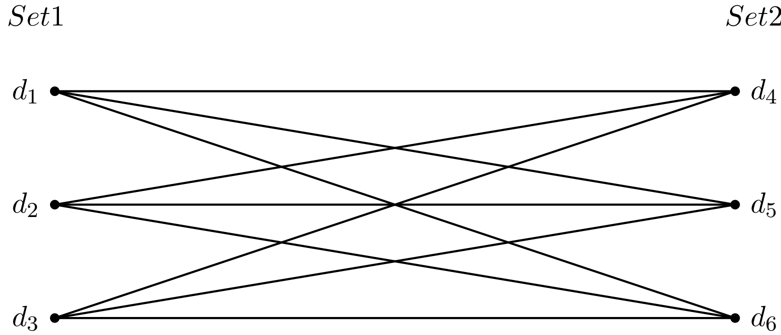


Figure 5: Cross Product (9 comparisons)

rithms which are used to reduce the search space of a given problem. The run time complexity of such algorithms has to be smaller than the DIST() algorithm, otherwise it would not make sense to apply them. For the "tree edit distance" such a filter algorithm is the traversal string. It can be used as lower bound for the "tree edit distance". An upper bound for it provides the "constrained edit distance". To address our data integration problem we have also to apply such a filter algorithm to reduce the search space.
To get a filter for the data integration problem according the approach proposed in the paper “Integrating XML Data Sources Using Approximate Joins“ by Guha et al. [11] we first choose at random some elements out of set 1 and set 2. These elements form a sample set which becomes clustered. Next we sort the clusters in decreasing order of the number of sample elements contained in them and form a reference set of size k by picking one random element out of the largest k clusters. Now we have to calculate the distances function of each element of the input sets to each element in the reference set. Knowing these distances appling the triangle inequality we are able to calculate an lower and an upper bound for the distances between

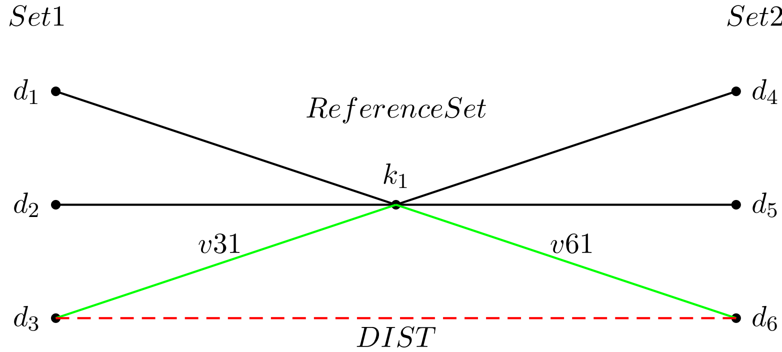
the elements of set 1 and set 2. We use these bounds as filter and are so able to prune some pairs because they either match or do not match for sure. The expensive distance computations has just do be applied for the remaining pairs.
It is obvious that with the number of the elements in the reference set also the number of over all distance computations rises. So we have to keep the reference set as small as possible. But just when the reference set is large enough it provides an effective filter. How Guha et al. [11, 10] propose to find the optimal balance I will discuss later.

5 Search Space Reduction

Let S_1 and S_2 be two sets of elements from different data sources between we wish to compute a join. Let $K \subset S_1 \cup S_2$ be a chosen set of elements which will refer as a reference set. Let $d_i \in S_1$ and $d_j \in S_1$ be two elements. We will compute for each element in S_1 and S_2 the distance function DIST() to the elements in the reference set K. These distances become stored in corresponding vectors $v_i(v_j)$. The dimensionality of the vectors depends on $|K|$. Let $k_l, ...k_{|K|}$ be the elements in K. Further let v$_{il}$ = DIST(d_i, k_l), 1 $\leq l \leq |K|$; similarly $v_{jl} = DIST(d_j, k_l), 1 \leq l \leq |K|$. Since DIST() is a metric the following is true by application of the triangle inequality:

$$\forall 1 \leq l \leq |K| |v_{il} - v_{jl}| \leq DIST(d_i, d_j) \leq v_{il} + v_{jl}$$

Example with one item in the reference set:



$$|v_{31} - v_{61}| \leq DIST(d_3, d_6) \leq v_{31} + v_{61}$$

Figure 6: Comparison with reference set

Assume we want to find all pairs of elements of S_1 and S_2 which are closer than a specific threshold, say τ. We can take $|v_{il} - v_{jl}|$ as lower bound and

$v_{il} + v_{jl}$ as upper bound for the distance of the two elements d_i and d_j (see Figure 6).
In the case when the upper bound is smaller than the threshold τ it is obvious that the two elements (d_i and d_j) are within the given threshold and we can add the pair to the result set without calculating the expensive DIST().
Another case is when the lower bound is bigger than τ. In this case we know for sure that d_i and d_j are not within the threshold and we can prune the pair. The DIST() function has just to be applied for pairs where τ is between the upper and lower bound.
The importance of good bounds becomes here quite evident. How these according Guha et al. [11] can be found I will explain in the next sections. As an example, we will consider points in the Cartesian plan.

6 Choosing A Reference Set

In our example as input we get 100 points for set 1 and 100 points for set 2. The points are spread over the Cartesian plain in the space from 0 to 100 in x- and y-direction. As we can see in Figure 7 for the example I have chosen an input of clusters of different size, because with such an input some later operations are easier to show and the whole process becomes clearer. Of course each distribution would be a valid input.

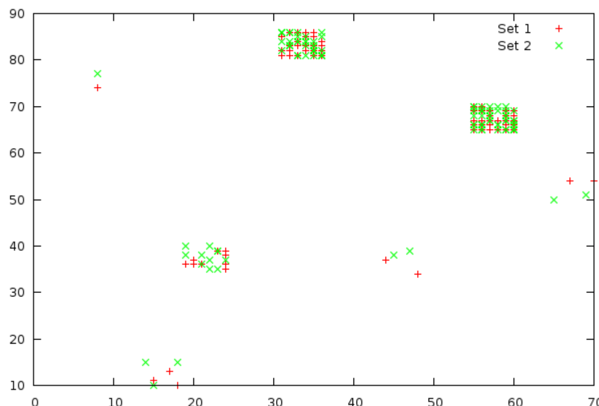


Figure 7: Experiment Input

To estimate the distance between the points I use the euklidean distance:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}.$$

Note: depending on the input type the distance function can vary. For XML documents we could use the tree edit distance and for strings the string edit distance or the longest common subsequence. It is not evident which distance estimation is applied, it has just to map the distance into the metric space.
To get good bounds we first have to find good reference points. The size of the reference set should be as small as possible since it is proportional to the amount of work we have to do, yet it should provide bounds that are as decisive as possible. According the paper "Integrating XML Data Sources Using Approximate Joins" our strategy will be first to sample the input and than to clusters the sample into n clusters. Out of the larges k cluster I will pick a point and add it to the reference set.

6.1 Draw a sample set

First we have to draw a sample out of the two input sets S_1 and S_2. For that we form a super set S (S $\in S_1 \cup S_2$). Out of this new set S we randomly pick items for our sample set. The size of the sample set depends on the size of S_1 and S_2. According the approach proposed by Guha et al. [11, 10] a sample size of

$$12\frac{\sqrt{k|S|}}{\epsilon}log|S|$$

is enough to identify all k clusters with high probability. This formula assumes that we already know the number of clusters in our input sets which is not always the case for our inputs. For that I decided to make some experiments with different input- and sample sizes on differently clustered sets to identify what actualy is a good size for the sample set. According my experiments for an input of clusters of different size[2], like in our example a good choice are 3 percent of the super set S. I will explain more on this experiment later.
In our example it is enough when we pick 6 points for the sample set, because S_1 consists of 100 points as also S_2. So the super set S contains 200 points.

$$|K| = |S_1| + |S_2|/100 * 3$$
$$|K| = |S|/100 * 3$$
$$|K| = 200/100 * 3$$
$$|K| = 6$$

Consider the two input sets in Figure 7. Figure 8 shows the sample that were chosen at random.

[2] in term of number of contained elements

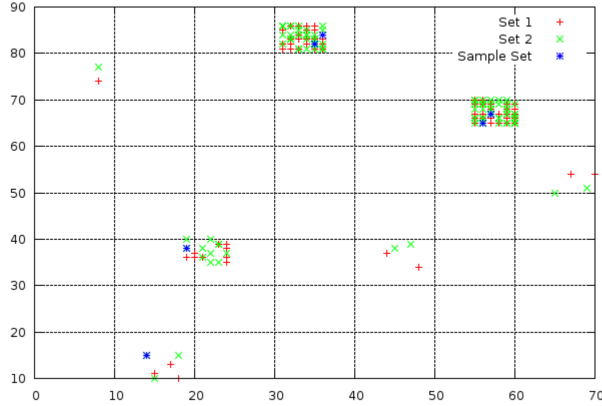


Figure 8: Sample Set

6.2 Cluster the sample set

The next step will be the clustering of the sample set. We will cluster the set with a cluster size of half threshold ($\tau/2$). This has the consequence that inside a cluster the distances between the points are at most τ.

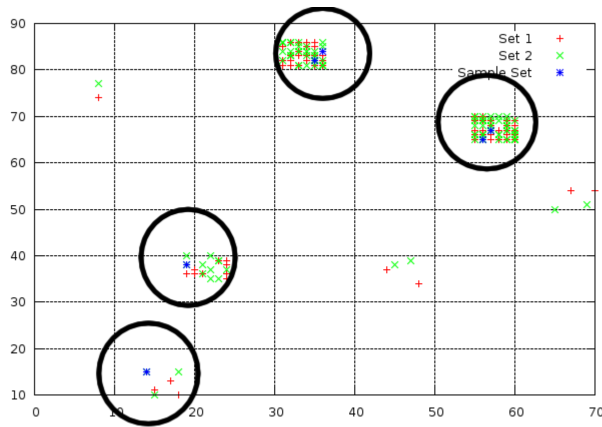


Figure 9: Clustered Sample

We call every point which does not belongs to a cluster a free point. Initially every point is a free point. So we pick a point at random and compute the distance function DIST() to each other point in the sample set. Every point which is inside the radius $\tau/2$ comes in this first cluster. Then we pick our next free point and compute the distances to the other remaining free points and put them in the next cluster. We repeat this until no free point is remaining. At the end of this procedure we have a clustered sample set

where each point belongs to exactly one cluster. Figure 9 shows the resulting clusters (the clustered points are circled)

6.3 Draw reference points

Finally we can draw some reference points. Since the sample set is clustered, we could pick out of each cluster a reference point. But as we already know, is the size of the reference set proportional to the amount of work we have to do. So we pick just one point out of the largest i clusters. We compute f_i as the fraction of points in the first i clusters. As i increases, we will be comparing $(1 - f_i)^2 n^2$ pairs. Thus the number of comparisons decreases by ratio $(1 - f_{i+1})^2/(1 - f_i)^2$ and the size of the reference set increases by $1 + 1/i$ in size. We balance these two and choose $k \geq i \geq 2$ such that:

$$\frac{(1-f_{i+1})^2}{(1-f_i)^2} > \frac{i}{i+1}$$

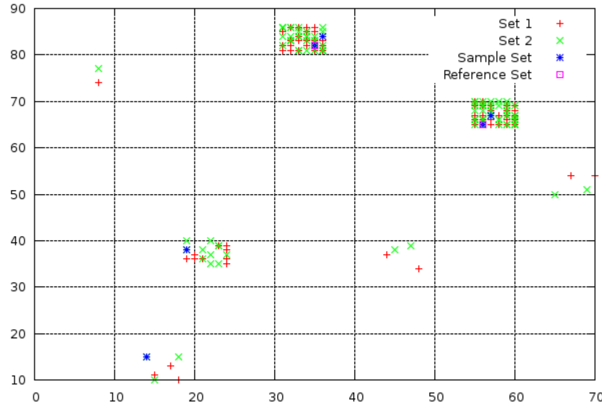


Figure 10: Reference points

1. Turn: i=0; $f_i = \frac{0}{6}$; $f_{i+1} = \frac{2}{6}$;
$\frac{(1-\frac{2}{6})^2}{(1-\frac{0}{6})^2} > \frac{0}{0+1} \rightarrow 0.44 > 0$

2. Turn: i=1; $f_i = \frac{2}{6}$; $f_{i+1} = \frac{4}{6}$;
$\frac{(1-\frac{4}{6})^2}{(1-\frac{2}{6})^2} > \frac{1}{1+1} \rightarrow 0.25 > 0.5$

We reach the balance already at the second turn where i is only 1. Since $k \geq i \geq 2$ we choose for our example 2 reference points. We pick them out of the largest 2 clusters.

7 Experiments

7.1 The data sets

For expressive experiments we need different distribution in the input data. For my experiments I dived non-, regular- and unregular-clustering.

Unclustered means that the items are distributed uniformly over the space (see Figure 11 left).
Regular clustered means that the items are clustered in equal large[3] clusters (see Figure 11 right).
Unregular clustered means that the items are clustered in varying large[4] clusters (see Figure 12).

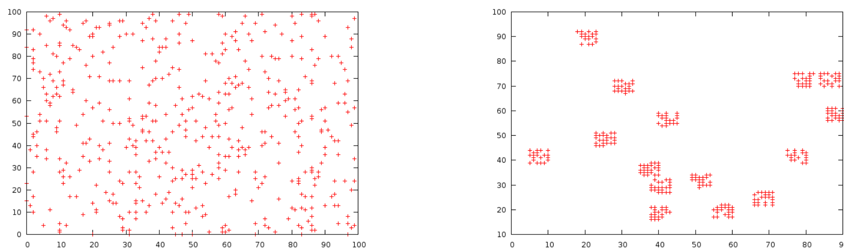


Figure 11: Unclustered (left), Regular Clustered (right)

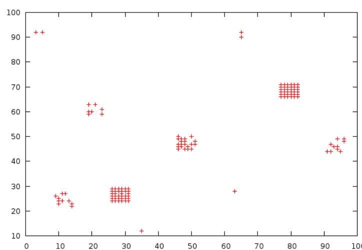


Figure 12: Unregular clustered

7.2 Size of sample set

In this experiment we want to find out what actually is a good size for a sample set and on which parameters it depends. For that I take different clustered input sets with different sizes, increase the sample size from 0 to 100 percent and count the saved distance computations at each step.

[3]size in term of points in a cluster
[4]size in term of points in a cluster

The inputs are the following: unclustered, regular and unregular clustered points of the sizes[5] 20, 100, 400, 800 and 1000.
In my first experiment I increase the size of the sample set at each step by 5 percent. I started at 1 percent and went up to 100 percent. The results showed that an actual good sample size for each distribution is to find below 20 percent of the input sets.
So I started a second experiment in which I increased the size at each step by 1 percent from 1 percent to 20 percent.

7.2.1 Evaluation concepts

Let me introduce first some concepts. S_1 and S_2 are the input sets, K is the reference set, O is the number of distance computations required to cluster the sample set, R is the number of DIST() computations using the reference set, SR is the number of saved distance computations using the reference set, C is the candidate set, i.e, the subset of $S_1 \times S_2$ for which DIST() must be computed after filtering. The amount of distance computations are calculate as follows:

$$|R| = |S_1| * |K| + |S_1| * |K| + |O| + |C|$$

Subtracting that number from the cross product $|S_1| * |S_2|$ we get the number of saved computations:

$$|SR| = |S_1| * |S_2| - |R|$$

Following graphs illustrate the outcome of the experiments. On the x-axis I put the percentage of the sample set and on the y-axis the percentage of the saved distance computations.

[5] The size is given as the number of points in the input set 1 plus the number of points in the input set 2

7.2.2 Unclustered

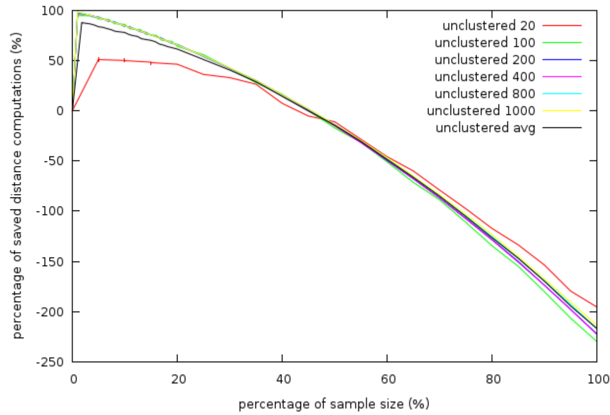


Figure 13: Normal clustered (sample set 0-100 percent)

In Figure 13 we can see how much the size of the sample size influences the number of distance computations for a uniform distributed input. It shows us dramatically that the choice of the sample size is important. If the size is too small we do not reach the optimal performance, if it is too big, using reference sets is less efficient than the brute force algorithm that computes the cross product (see negative values in Figure 13)

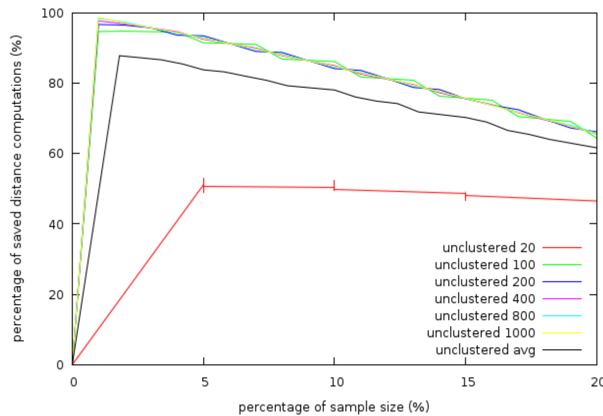


Figure 14: Unclustered (sample set fist 20 percent)

A nearer look to the first 20 percent of the experiment (see Figure 14) shows us that a good average sample size for a unclustered input would be 1 percent of the actual size of the two input sets.

7.2.3 Regular clustered

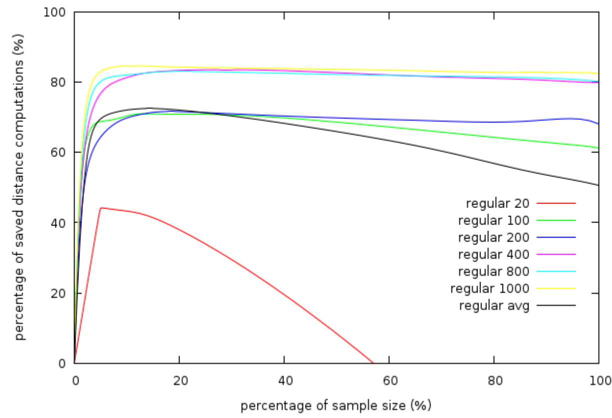


Figure 15: Regular clustered (sample set 0-100 percent)

Figure 15 shows us the impact of the reference set size on a regular clustered input. We can see that such an input is much tolerant in respect to saved distance computations than an unclustered. In other words, also with a big sample size we save some DIST() computations in almost all settings.

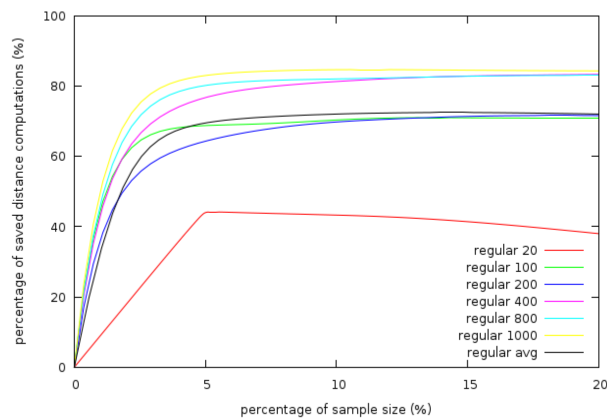


Figure 16: Regular clustered (sample first 20 percent)

But also in this case we reach the optimum below 20 percent. Figure 16 shows that this point is reached at about 15 percent for regular clusters.

7.2.4 Unregular clustered

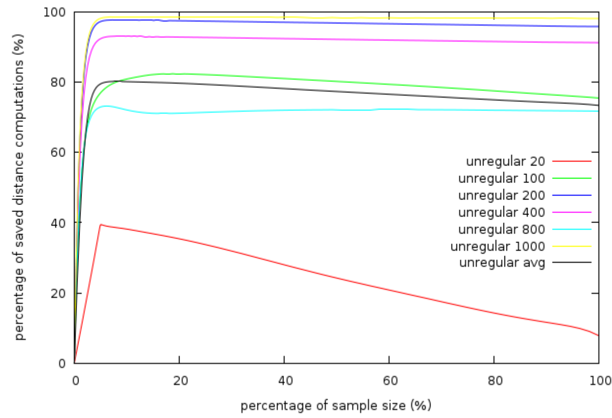


Figure 17: Unregular clustered (0-100 percent)

Our third cluster type is also quite tolerant with large input sets. This is so because the more points we have in the sample set, the higher the probability that we cover each cluster (see Figure 17).

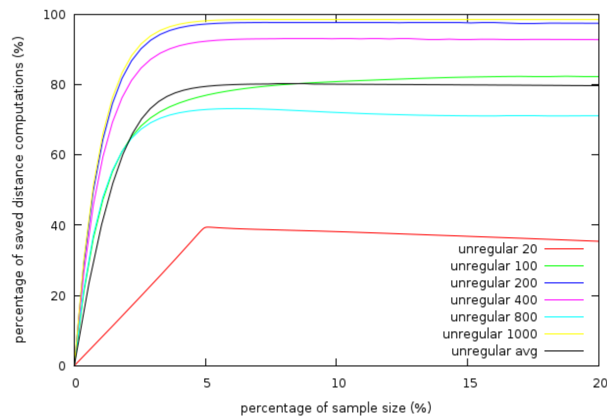


Figure 18: Unregular clustered (sample set first 20 percent)

But since we do not need necessarily to identify each cluster we already reach our optimal point at 3 percent (see Figure 18).

7.2.5 Average case

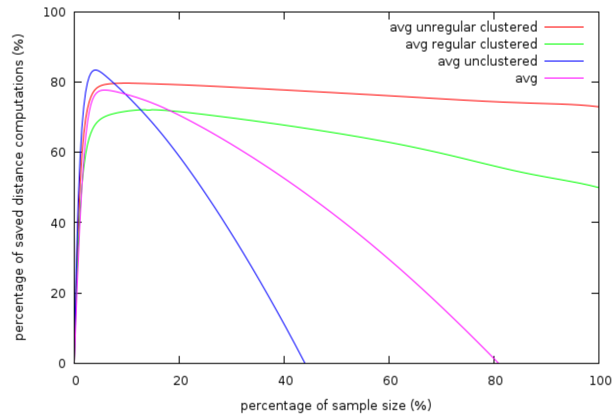


Figure 19: Average clustered (sample set 0-100 percent)

The previous results showed us that the optimal percentage for the sample size varies for each type of input cluster. Since we do not know always how our inputs are clustered we need an average case which works some how for each type of clustering. To find this point I took the average of each previous result and build the overall average (see Figure 19).

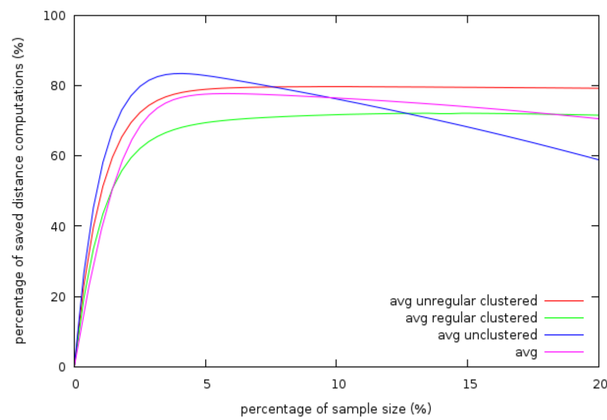


Figure 20: Avarage clustered (sample set first 20 percent)

As we can see in Figure 20 the optimal point for unknown clustering is reached at 5 percent of the sum of the elements in the input sets.

7.3 Varing threshold

In the second experiment I will vary the threshold to see what is its impact on the number of saved distance computations, overall computations and overhead. Further i will observe the size of the matching as also the pruning set.
As in the first experiment I will differ the inputs in size as in clustering. The points are distributed over the Cartesian plane from 0 to 100 in x- as in y direction. After some preliminary experiments I noticed that it is enough when I vary the threshold from 0 up to 250. To be able to compare the results of the different clustered inputs I have to take the same percentage of sample sizes for each distribution. For that I have chosen the average percentage of 5 percent for my experiment.

7.3.1 Unclustered

The threshold has a big influence on the number of distances computations we have to do as we can see in the Figure 21. Is the threshold small we save many computations because most of the pairs can be pruned. The larger the threshold becomes the larger the candidate set becomes. (see Figure 22) Since we have to check each candidate whether it matches, the number of overall calculations rises.

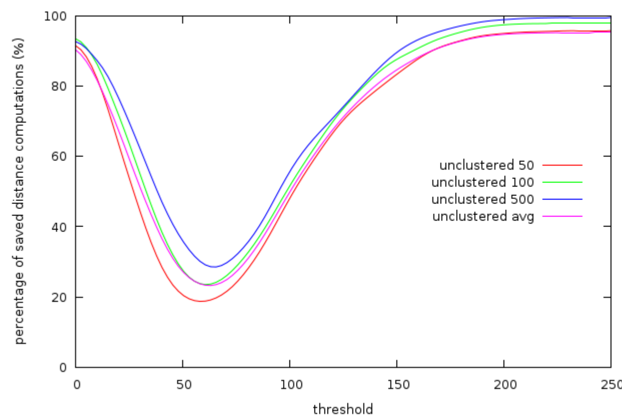


Figure 21: Unclustered (threshold 0-250)

But we reach a point from that onwards we save again more an more calculation increasing the threshold further. This point in our experiment with unclustered inputs is reached at a threshold of round 70, independent of the number of points in the input sets. Since the input set is uniformly distributed we reach the inflection point at half of the maximal distance of the points ($\frac{\sqrt{100^2+100^2}}{2} \approx 70$). The graphical representation of the saved

distance computation is an inversion of a classical bell shape (see Figure 21, 22).

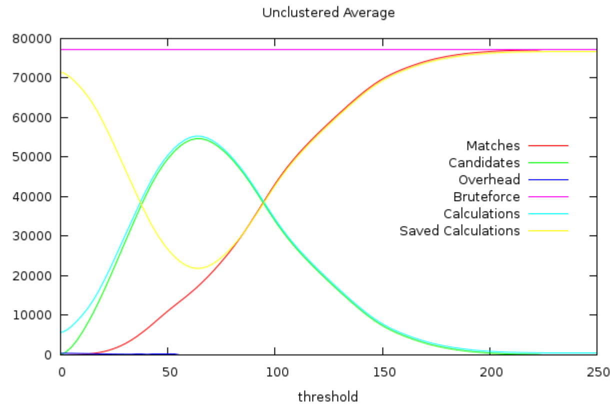


Figure 22: Unclustered

7.3.2 Unregular clustered

Feeding our experiment with clusters of different sizes we get some similar results. The graph of saved DIST() is again an inverted bell shape. But on an closer look we can see that we have not a equable falling/rising graph. At some part the slop of the graph is larger than on other parts.

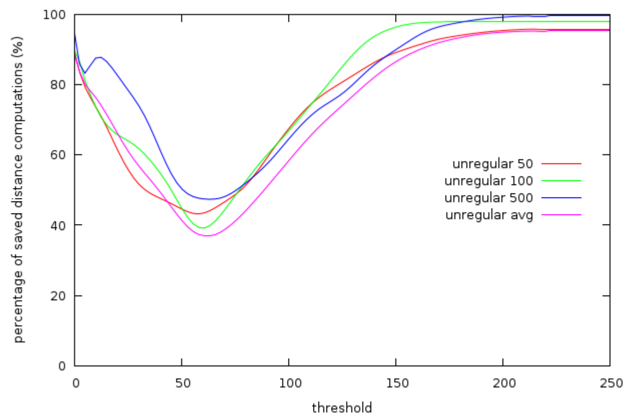


Figure 23: Unregular clustered (threshold 0-250)

The reason is that the input is clustered. That means on some threshold value we are able to prune many pairs at a time. But after this point we have to increase the threshold a lot till we find the next cluster to be able to

prune new pairs. Since the clusters are not so far away from each other we

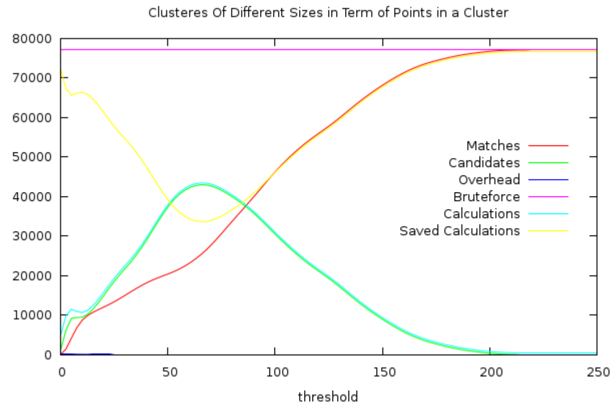


Figure 24: Avg clustered (0-100)

have the inflection point again at nearly the half of the maximal distance of points (around 70).

7.3.3 Regular clustered

Our next inputs will be regular clustered points. Also here I got a quite similar outcome as before. The graph (Figure 25) of the saved distance computations is an inverted bell shape with some anomalies.

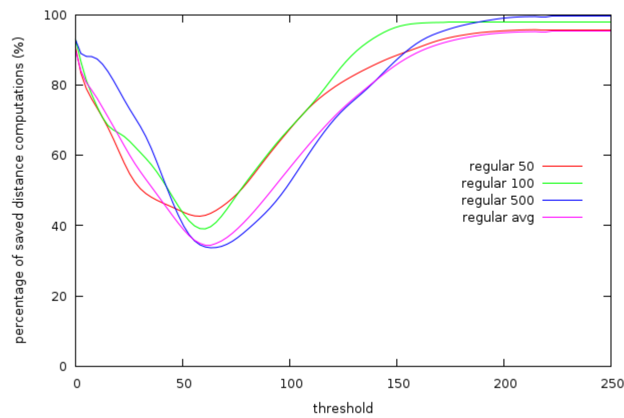


Figure 25: Regular clustered (threshold 0-250)

The only difference to unregular clustered trees is that the graph is a bit smoother because of the regularity of the clusters.

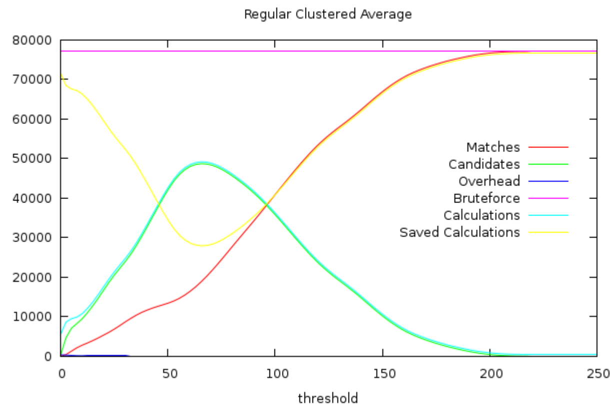


Figure 26: Regular clustered (threshold 0-250)

7.3.4 Average clustered

Of course, also in the average case I got a bell shape with some anomalies (see Figure 27). Comparing the cluster types directly we see that with an

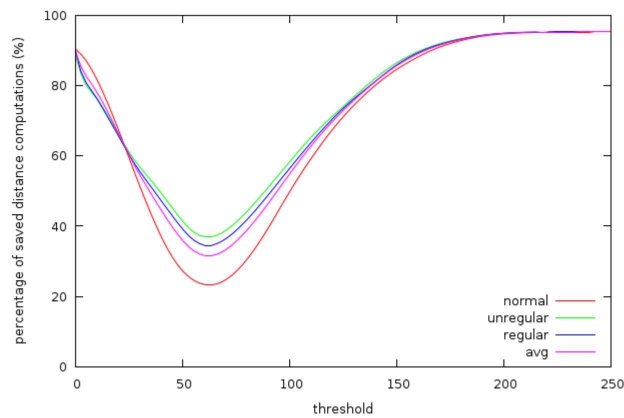


Figure 27: Avg clustered (threshold 0-250)

unclustered input in the worst case we save just 20 percent of the computations we had to do with the brute force approach. Instead with regular and unregular clustered inputs we are able to save about 40 percent even in the worst case. This is so because with regulare and unregular clustered inputs the filters work most effective.

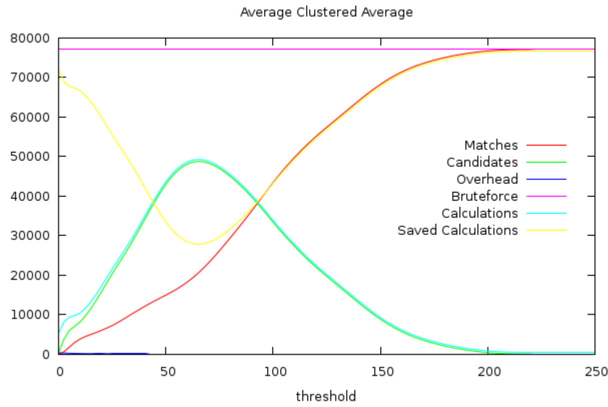


Figure 28: Average clustered

7.4 Varing threshold and input range

To find out when actually the point of inflection is reached I started a third experiment in which I increment the threshold on different input ranges[6]. Since we have seen in the pervious experiment that the inflection points of clustered inputs depends on their clustering, for that experiment I will take unclustered inputs only. As we can see in Figure 29, we always get our inverted bell shape with the reflection point at half of the maximal distance between the points. For example with a range[7] of 300 we get the inflaction point at $\frac{\sqrt{300^2+300^2}}{2} \approx 212$) (range 1000 $\rightarrow$ 707...)

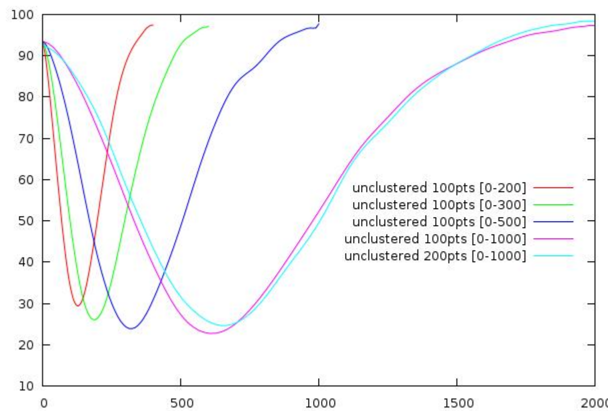


Figure 29: Varing range and threshold

[6] maximal distance between the points
[7] maximal distance between the points

To show that the size in term of points in the input has no impact on the outcome, I started the experiment with range 1000 twice. One time with 100 points in the input and a second one with 200 points. Also that behavior we can see in Figure 29.
Figure 30 shows why we get the inverted bell shape for the saved distance computations increasing the threshold. Is the threshold small the upper bound is realy strong and we are able to prune many pairs with it. The increasing of the threshold brings the effectiveness of the upper bound down and we save at each step less computations. But as the effectiveness of the upper bound sinks, the stronger the lower bound becomes (see Figure 30). So we reach a point from that onwards we again are able to prune more and more pairs. From now on with the lower bound. The worst case of the reference set approach is reached when neither the upper nor the lower bound are able to prune many pairs. Depending on the clustering this point in average is reached at half of the maximal distance between the elements.

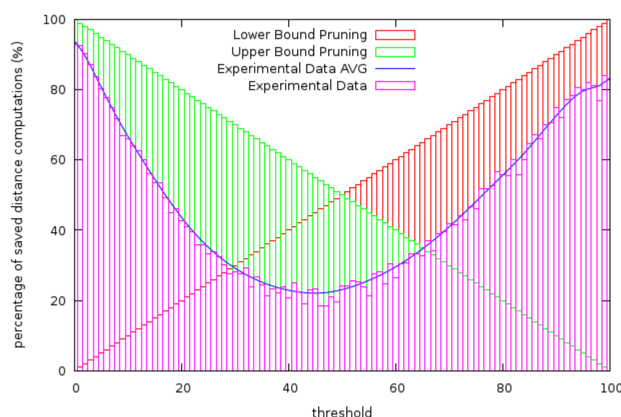


Figure 30: Bound Pruning

7.5 Experiment with real data

To varify that the approach does not work only with syntactical data I started some experiments on real data. The data consists of streets of the municipal Bozen. One representation of the streets is the streetname like "cesare abba strasse“. The other is a kind of tree. For example, 30:{cesare abba strasse{1}{2}{3{{1}{3}}}{11}} is the address tree with ID 30, its root node has the label "cesare abba strasse" and the children of the root are labeled 1, 2, 3, 11; 3 has a child with an empty string label, which in turn has two children with labels 1 and 3. Since my application does not supports trees like that I applied for both representations the "String Edit Distance"

as distance computation method. For the tree representation the “String Edit Distance“ can be seen as lower bound like the tree traversal. In Figure 31 we can see the outcome of the experiments with street names only. As on syntactical data also in this experiment we get an inverted bell shape for the saved distance computations. The maximal distance between the elements is 29. Since the input is not that much clustered we get also with real data the inflaction point nearly at half of the maximal distance (13).

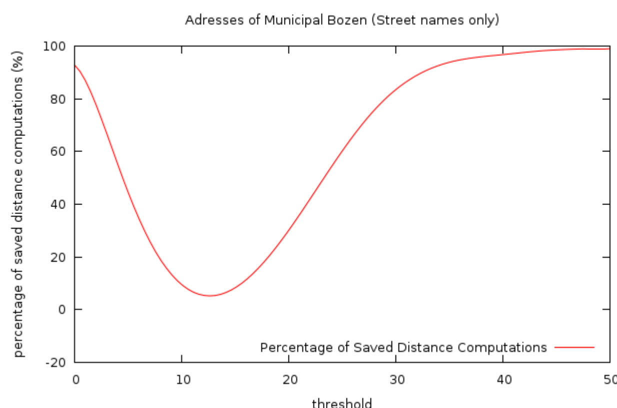


Figure 31: Street names representation

Things change in the second real data experiment (see Figure 32). Here the outcome at a first look, seams strange. But again it shows that with small and high[8] thresholds we are able to prune more pairs than with an intermediate threshold. Because the input data is realy widespread clustered the filters work quite effective and so we save in the worst case even more than 70 percent of the overall computations we had to do appling the brute force methode. But also because of the clustering in spite of kowing the maximal distance between the elements is 7298 a prediction of how many calculations we have to do at a given threshold, is realy hard to provide with out executing the experiment almost once.

8 Conclusion

After many experiments under different conditions I can say that the approach to find similar elements of two sets by pruning elements with an reference set is very efficient. Under optimal conditions we are able to save up to 98 percent with respact to the brute force algorithm that computes the cross product. Only if the input sets are very small (for example each contains just 5 elements) the traditional brute force approach is superior.

[8]high in respact to the maximal distance between elements

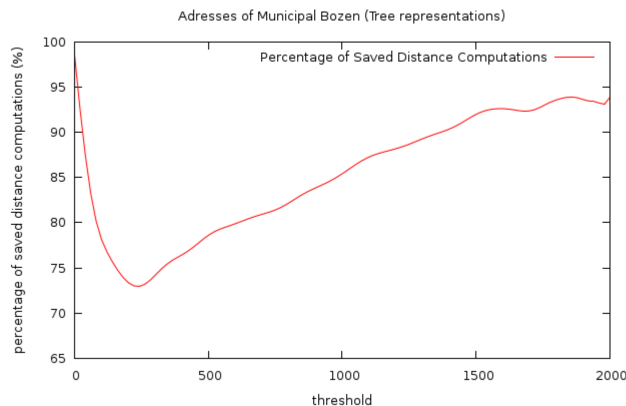


Figure 32: Tree representation

Is the amount of input data big enough we even save some distance computation under for the approach worst conditions. These conditions depend mainly on the size of the sample set and the threshold of the approximate join.
If the sample set is to large, we produce a lot of overhead, clustering it and the approach becomes unprofitable. But the experiments showed that it is never necessary to choose the sample set larger than 15 percent of the size[9] of the input sets. Depending on the clustering of the input data, the optimal percentage varies between 1 and 15 percent of the number of elements in the input sets.
I found out that for unclustered inputs a sample set size of 1 percent is enough to form a adequate sample set. Are the input sets unregular clustered this percentage is about 3 percent. The highest percentage is necessary when the inputs are regular clustered. In that case we need 15 percent of the input elements to form a good sample set. In our experiments 5 percent gave good results for all data sets and we recommend this value if the distribution of the input data is unknown.
The experiments showed that also the threshold has a big impact on the number of calculations we are able to save with the reference set method. If it is small enough, we save many of calculations pruning pairs with the upper bound. The larger the threshold becomes the weaker the upper bound becomes and the less calculations we save until the lower bound becomes strong enough to prune more and more pairs. From that point on we again save more calculations if we further increase the threshold. We conclude that the average worst case for the reference set approach is when the threshold is at half way of the largest distance between two elements. But even in this

[9]size in term of elements in the set

case we save calculations. For unclustered inputs more than 20 percent and for regular and unregular clustered inputs even more than 35 percent.

9 Appendix

9.1 Application

To experiment with the reference set approach I have implemented it in C++ with an Qt-Gui. The application accepts XML-files, n-dimensional points and strings as input. One can define the desired threshold and the sample set percentage. The sample set percentage can be entered directly or by choosing the clustering in a combo box. The reference set can be chosen manually or is computed automatically. If no reference set is chosen, the traditional brute force method is applied. After the match finding procedure is finished, the results are shown on the main window (see Figure 33). It shows the matches as also the candidates in the final set. Also the number of matches, candidates, needed calculations, saved calculations and overhead calculations are printed out. It shows also the maximal found distance of the elements during sampling. When the inputs are not to much widespread clusterd with this distance we are able to predict how many calculation will be saved before we start an experiment. Further the applications after the pre-calculations provides the posibility to check which candidates are matches. So finally we exactly know how many matches we have an which they are.
The application supports two experiment modes. One is the experiment with varying sample percentage and the other with varying threshold.
The user can decide from which value (percentage/threshold) the experiment starts and until where it goes. Further he/she can set the step size which defines how much the percentage/threshold increases each step.
The user is also able to determine how often the experiment is executed. This is important to get authentic results without great variations. The results become stored on the hard drive in csv files and so he/she is able to calculate the average of the experiment files to get a good final result.
In the csv files following data is stored: threshold, size of set1, set2, reference set, sample set, match set, candidate set, number of overhead calculations and saved calculations. How many calculation we would have to do with the brute force method. How many we have done with the reference set method.

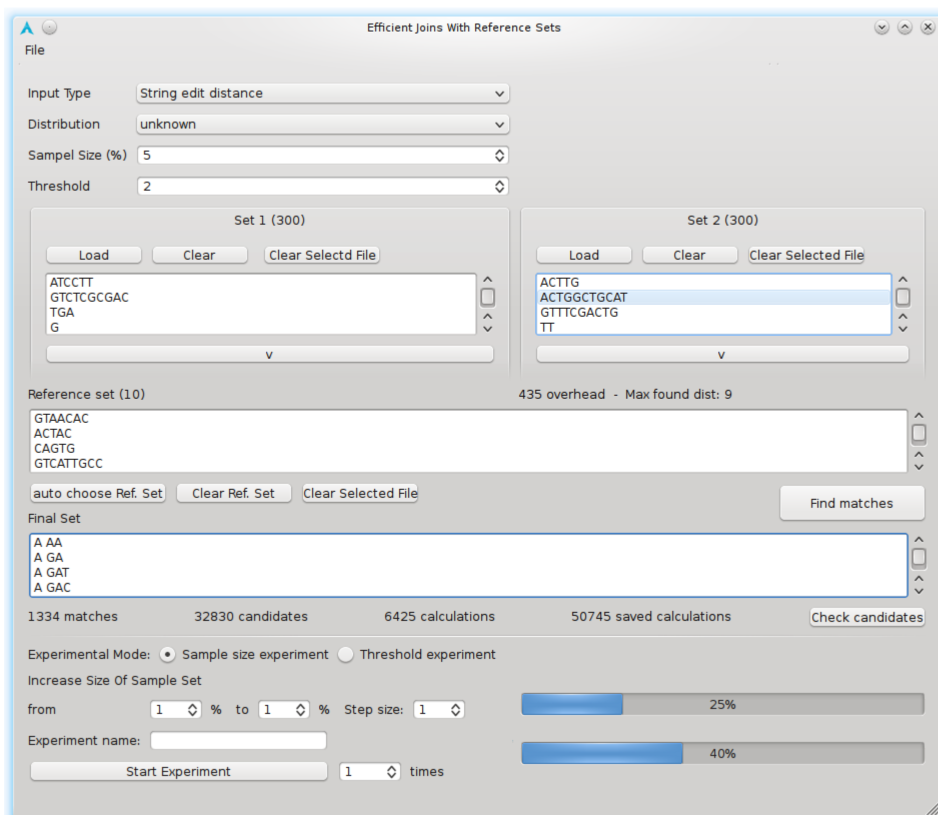


Figure 33: Main Application

References

[1] S. AL-KHALIFA, H.V. JAGADISH, N. KOUDAS, J. M. PATEL, D. SRIVASTAVA, and Y. WU. Structural joins: a primitive for efficient XML query pattern matching. 2000.

[2] P. BILLE. Tree edit distance, alignment distance and inclusions. Technical report, 2003.

[3] N. Bruno, N. KOUDAS, and D. SRIVASTAVA. Holistic twing joins: Optimal XML pattern matching. 2002.

[4] E. CHAVEZ, G. NAVARRO, R. BEAZA-YATES, and J. MARROWUIN. Searching in metric spaces. 2001.

[5] S. CHAWATHE, H. MOLINA, and J. WIDOM. Meaningful change detection in structured data. 1997.

[6] S. CHAWATHE, A. RAJARAMAN, H. MOLINA, and J. WIDOM. Change detection in hierachical structured information. 1996.

[7] P. CIACCIA, M. PATELLA, and P ZEZULA. An efficient access method for similarity search metric spaces. 1997.

[8] G. COBENA, S. ABIDEBOUL, and A. MARIAN. Detecting changes in XML documents. 2002.

[9] L. GRAVANO, P. IPERIROTIS, H. V. JAGADISH, N. KOUDAS, S. MUTHUKRISHNAN, and D. SRIVASTAVA. Approximate strings joins in a database (almost) for free. 2001.

[10] S. GUHA, D. JAGADISH, N. KOUDAS, D. SRIVASTAVA, and T. YU. Approximate XML joins. 2002.

[11] S. GUHA, D. JAGADISH, N. KOUDAS, D. SRIVASTAVA, and T. YU. Integrating XML data sources using approximate joins. 2004.

[12] S. GUHA, R. RASTOGI, and K. SHIM. Cure: An efficient clustering algorithm for large databases. 1998.

[13] A. MARIAN, S. ABIDEBOUL, G. COBENA, and L. MIGNET. Change centric management of versions in an XML warehouse. 2001.

[14] G. NAVARRO. A guided tour to approximate string matching. 2001.

[15] F. SANGER, S. NICKLEN, and A R. COULSON. Dna sequencing with chain-terminating inhibitors. 1977.

[16] M. TICO and P. KOUSMANEN. Fingerprint matching using an orientation-based minutia descriptor. 2003.