

0.1 Changes 3/14 v1

I completed and rewrote some sections on the sample based iterative clustering after the experiment got completed.

- 4.6.2 Sample Based Iterative Clustering
- 6.1.5 Experiment 1.5: Sample Based Iterative Clustering
- 9.2 Future Work - A paragraph was inserted (highlighted)

In summary by Theorem 3, as soon as the algorithm detects a row p in the recurrence table such that $\frac{\min_row(p)}{\max\{\|seq_i\|, \|seq_j\|\}} > \max\{dist_k(seq_i), dist_k(seq_j)\}$, it is clear that $dist(seq_i, seq_j) = dist(seq_j, seq_i) = \infty$. At this point, the recurrence table calculation can stop and simply return ∞ . Checking for the condition $\frac{\min_row(p)}{\max\{\|seq_i\|, \|seq_j\|\}} > \max\{dist_k(seq_i), dist_k(seq_j)\}$ at the end of each row takes negligible time and space when $k \ll N$ and $k \ll L$.

Hueristically, setting up the recurrence table such that the shorter of the two sequences goes across the recurrence table and the longer sequence goes down can save more time. This is because the algorithm has to finish calculating a full row before checking for the stop condition. Thus intuitively, recurrence tables with shorter rows are likely to find the stopping condition faster. The experimental results in section 6.1.4 show that in practice, such optimization can speedup time by a factor of upto 40%. The running time becomes almost linear with respect to L_{seq} (Figure 6.4).

4.6.2 Sample Based Iterative Clustering

The $O(N_{seq}^2)$ for the clustering step can be improved by using an iterative partitioning method similar to the well known kmediods⁴ clustering methods. However, there are two major difficulties with using the kmediods clustering methods. First, without proper initialization, it is impossible to find the right clusters. Thus, finding a reasonably good starting condition is crucial for kmediods methods to give good results. Second, the general kmediods method requires that the user input the number of clusters in the data. However, the proper number of partitions is not known in our application.

To overcome these problems but still benefit from the efficiency in time, we introduce a sample based iterative clustering method. It involves two main steps. The first step finds the clusters and its representative sequences based on a small random sample of the data, \mathcal{D}' , using the density based k -nearest neighbor method discussed in section 4.2.2. Then in the second step, the number of clusters and the representative sequences are used as the starting condition to iteratively cluster and recenter the full database until the algorithm converges. The full algorithm is given in Algorithm 3.

When $\|\mathcal{D}'\| \ll \|\mathcal{D}\|$, the time complexity for the clustering method is obviously $O(t \cdot N_{seq})$ where t is the number of iterations needed to converge. Our experiments show that the algorithm converges very quickly. Figure 4.8 shows that for most datasets it take from 3 to 6 iterations. The experimental results in section 6.1.5 show that in practice, this optimization can speedup time considerably at the cost of a slight reduction in accuracy (Figure 6.6(b)) and larger memory requirements. Given enough memory the running time is reduced to about 10%-15% and becomes almost linear with respect to N_{seq} (Figures 6.6(c) and 6.6(d))

⁴kmediods clustering is exactly the same as the more popular kmeans algorithm, but it works with the representative points in clusters rather than the means of clusters.

ALGORITHM 3. (SAMPLE BASED ITERATIVE CLUSTERING)

Input: a set of sequences $\mathcal{D} = \{seq_i\}$, sampling percentage α , and number of neighbor sequences k' for the sampled dataset;

Output: a set of clusters $\{C_j\}$, where each cluster is a set of sequences;

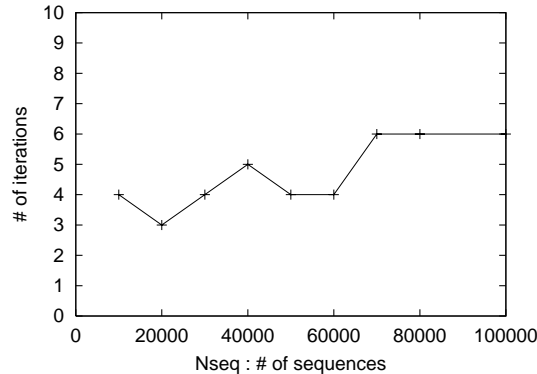
Method:

1. **Randomly sample the database \mathcal{D} into \mathcal{D}' using α .** The size of \mathcal{D}' will be a trade off between time and accuracy. In our experiments, we found that at a minimum $\|\mathcal{D}'\|$ should be 4000 sequences for the default $k' = 3$. We also found that roughly 10% of the data will give you comparable results when $N_{seq} \geq 40,000$.
2. **Run uniform kernel density based k -NN clustering (Algorithm 1) on \mathcal{D}' with parameter k' .** The output is a set of clusters $\{C'_s\}$
3. **Center: Find the representative sequence for each cluster C'_s .** The representative sequence for each cluster is the one which minimizes the sum of the intra-sequence cluster distance. That is, the representative sequence, seq_{sr} , for a cluster, C'_s , is chosen such that $\sum_j dist(seq_{sr}, seq_{sj})$ for all sequences, seq_{sj} , in cluster C'_s is minimized.
4. **Initialization: Initialize each cluster, C_s , with the representative sequence, seq_{sr} ,** found in the previous step.
5. **Cluster: Assign all other sequences in the full database, \mathcal{D} , to the closest cluster.** That is assign sequence seq_i such that $dist(seq_i, seq_{sr})$ is minimum over all representative sequences, seq_{sr} .
6. **Recenter: Find the representative sequence for each cluster C_s .** Repeat the centering step in 3 for all clusters C_s formed over the full database.
7. **Iteratively repeat Initialization, Cluster, and Recenter.** Steps 5 through 7 are repeated until no representative point changes for any cluster or a certain iteration threshold, $MAX_LOOP = 100$, is met.

- *optimized (all)*⁵). Even when there is not enough memory, in the worst case the running time is still reduced to about 25%-40% (Figures 6.6(c) and 6.6(d) - *optimized (none)*).

There is a few implementation details to note. First, when using a small sample of the data, k for k -nearest neighbor algorithm has to be smaller than what is used on the full database to achieve the clustering at the same resolution because the k -nearest neighbor in the sampled data is most likely $(k + \alpha)$ -nearest neighbor in the full database. Again this

⁵*optimized (all)* is a simple hash table implementation with all proximity values stored and *optimized (none)* is the implementation with none of the proximity values stored.

Figure 4.8: Number of iterations

might be best understood if you think of a digital image. When less pixels are sampled, blurring has automatically occurred from the sampling and less manual blurring (averaging of the sampled points) need to happen for a certain resolution of the picture. In **ApproxMAP**, since the default value for k is 5, the default for k' in the sample based iterative clustering method is 3.

Second, in the iterative clustering method much more memory is required in order to fully realize the reduction in running time because the $N_{seq} * N_{seq}$ proximity matrix needs to be stored in memory. In the normal method, although we need to calculate the full $N_{seq} * N_{seq}$ proximity matrix, the information can be processed one row at a time and there is no need to return to any value. That is, we only need to maintain the k nearest neighbor list without keeping the proximity matrix in memory. However in the iterative clustering method, it is faster to store the proximity matrix in memory over different iterations so that the distance calculation does not have to be repeated. When N_{seq} is large, the proximity matrix is quite large. Hence, there is a large memory requirement for the fastest optimized algorithm.

Nonetheless, we should note that the proximity matrix becomes very sparse when the number of clusters is much smaller than N_{seq} ($\|C_s\| \ll N_{seq}$). Thus, much space can be saved by using a hash table instead of a matrix. Our initial implementation of a simple hash table ran upto $N_{seq} = 70,000$ with 2GB of memory. A more efficient hash table could easily improve the memory requirement. Furthermore, a slightly more complicated scheme of storing only upto the possible number of values and recalculating the other distances when needed (much like a cache) will reduce the running time compared to the normal method. Efficient hash tables are a research topic on its own. Our initial implementation of the hash table demonstrates the huge potential for reduction in time well.

In order to study the full potential we also measured the running time when no memory was used to store the proximity matrix. That is, distances values were always recalculated whenever needed. We found that even in this worst case the running time was reduced significantly to about 25%-40% (Figures 6.6(c) and 6.6(d) - *optimized (none)*). Thus, a good

implementation would give running times in between the *optimized (all)* and the *optimized (none)* line in Figure 6.6(c). More efficient hash table implementations will be studied in future research.

The loss of accuracy is from the difficulty kmedioids clustering algorithms have with outliers and clusters of different sizes and non-globular shapes [17]. As with most iterative clustering methods, this sampled based algorithm tries to optimize a criterion function. That is the sequences are partitioned so that the intra-cluster distance is minimized. This tends to build clusters of globular shape around the representative sequences regardless of the natural shape of the clusters. Thus, the clusters are not as accurate as when they were built around similar sequences following the arbitrary shape and size of the cluster as done in the pure density based k -nearest neighbor algorithm.

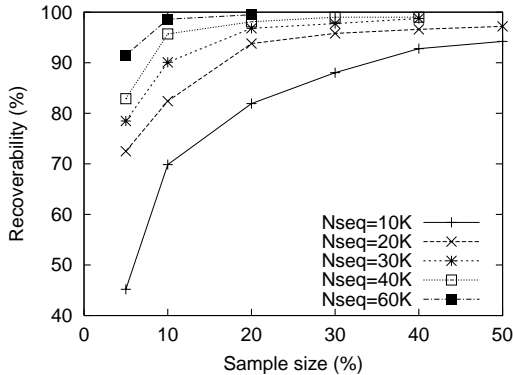
Moreover, it is not robust to outlier sequences. Although kmedioids methods based on representative points are more robust to outliers than kmeans methods, the outliers will still influence the choice of the representative points. This in turn, will change how the data is partitioned arbitrarily.

Fortunately, the steps following the clustering step makeup for most of the inaccuracy introduced in it. That is, multiple alignment of each cluster into weighted sequences and then summarization of the weighted sequences into consensus sequences is very robust to outliers in the cluster. In fact by the time we get to the final results, the consensus sequences, only the general trend in each cluster remain. In practice, the core sequences that form the main trend do not change regardless of the different clustering methods because these sequences will be located close to each other near the center. Thus, the clustering step only needs to be good enough to group these core sequences together in order to get comparable consensus sequences.

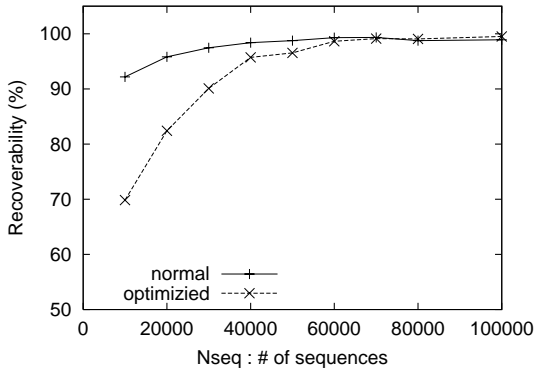
accuracy of the results.

6.1.5 Experiment 1.5: Sample Based Iterative Clustering

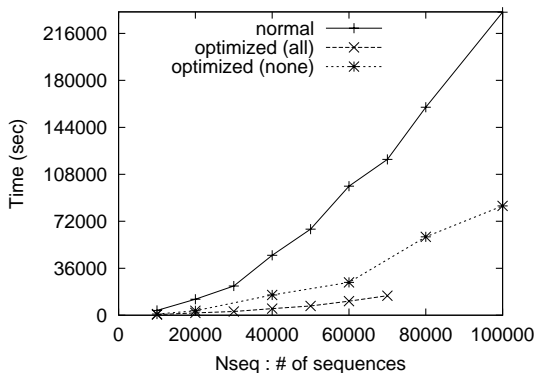
Figure 6.6: Sample based iterative clustering



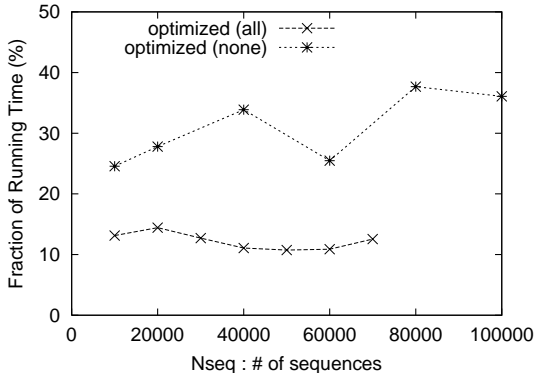
(a) Recoverability w.r.t. sample size



(b) Recoverability w.r.t. N_{seq} (sample=10%)



(a) Running time w.r.t. N_{seq}



(b) Fraction of running time w.r.t. N_{seq}

In section 4.6.2 we discussed how to speedup the clustering step by using a sample based iterative partitioning method. Such change can optimize the time complexity with respect to $O(N_{seq}^2)$ at the cost of some reduction in accuracy and larger memory requirements. Obviously, the larger the sample size the better the accuracy with slightly less gain in running time. In this section, we study the tradeoff empirically to determine the appropriate sample size. The experiments also suggests when such optimizations should be used.

Figure 6.6(a) presents recoverability with respect to sample size for a wide range of datasets. All runs use the default $k' = 3$. We see that when $N_{seq} \geq 40,000$, recoverability levels off at 10% sample size. When $N_{seq} < 40,000$, recoverability levels off at a larger sample size. However, when $N_{seq} = 40,000$ it takes less then 13 hours even without the optimization. Thus, the experiment suggests that the optimization should be used for datasets when $N_{seq} \geq 40,000$ with sample size 10%. For datasets with $N_{seq} < 40,000$, as seen in Figure 6.6(b), sample size 10% is too small. Thus, a larger sample size should be used if the

normal algorithm is not fast enough for the application. The experiments indicate that the sample size should have at least 4000 sequences to get comparable results. As expected, in the smaller datasets with $N_{seq} < 40,000$, the running time is fast even when using a larger sample size. When $N_{seq} = 30,000$ and sample size=20% the running time was only 90 minutes.

Figure 6.6(b) and (c) show the gain in running time and the loss in recoverability with respect to N_{seq} with the optimization (sample size=10%, $k' = 3$). *optimized (all)* is a simple hash table implementation with all proximity values stored and *optimized (none)* is the implementation with none of the proximity values stored. The results clearly show that the optimization can speedup time considerably at the cost of negligible reduction in accuracy. Figure 6.6(d) show that the optimization can reduce running time to roughly 10%-40% depending on the given conditions.

6.2 Experiment 2: Effectiveness of ApproxMAP

In this section, we study the effectiveness of ApproxMAP using the full evaluation method discussed in chapter 5 on some manageable sized datasets. We ran many experiments with various synthetic data sets. The trend is clear and consistent. The evaluation results reveal that ApproxMAP returns a succinct but accurate summary of the base patterns with few redundant or spurious patterns. It is also robust to both noise and outliers in the data.

6.2.1 Experiment 2.1: Spurious patterns in random data

In this section, we study empirically under what condition spurious patterns are generated from completely random data ($\|Z\| = 100, N_{seq} = 1000, L_{seq} = 10, I_{seq} = 2.5$). For random data, because there are no base patterns embedded in the data, evaluation criteria recoverability, precision, and number of redundant patterns do not apply. The only important evaluation measure is the number of spurious patterns, N_{spur} , generated by the algorithm. We include the total the number of result patterns returned, N_{total} , for completeness. Since there are no base patterns in the data $N_{total} = N_{spur}$.

Recall that there are two parameters, k and θ in ApproxMAP. We first study the cutoff parameter θ . To determine the threshold at which spurious patterns will be generated, T_{spur} , we ran 8 experiments varying θ for a particular k . Each individual experiment has a constant k from 3 to 10.

Here we first report the full results of the experiment with the default value $k = 5$. When $k = 5$, there are 90 clusters of which only 31 clusters had 10 or more sequences. That means that the remaining 59 clusters will not return a consensus pattern no matter how low the cutoff is set because $min_DB_strength = 10$ sequences. Recall that when generating pattern consensus sequences, the greater of the two cutoff, θ or $min_DB_strength$ is used. Therefore, the following theorem shows that the lowest value of θ that can introduce the most spurious

research, policy analysis, business analysis, career analysis, web mining, and security.

Our extensive evaluation demonstrates that **ApproxMAP** will effectively extract useful information by organizing the large database into clusters as well as give good descriptors (weighted sequences and consensus sequences) for the clusters using multiple alignment. We demonstrate that together the consensus patterns form a succinct but comprehensive and accurate model of the sequential data. Furthermore, **ApproxMAP** is robust to its input parameters, robust to noise and outliers in the data, scalable with respect to the size of the database, and in comparison to the conventional support model **ApproxMAP** can better recover the underlying patterns with little confounding information under most circumstances.

In addition, our case study on social welfare service patterns illustrates that approximate sequential pattern mining can find general, useful, concise, and understandable knowledge and thus is an interesting and promising direction.

9.2 Future Work

This dissertation is our first step towards the study of effective sequential pattern mining. Following the approximate frequent pattern mining model, many interesting research problems need to be solved.

First, more recent advances in multiple alignment come from Gibbs sampling algorithms, which use hidden Markov models [57]. These methods are better for local multiple alignment. Local multiple alignment is to find substrings of high similarity. Formally, given a set of strings, local multiple alignment first selects a substring from each string and then finds the best global alignment for these substrings. Since DNA sequences are very long, finding local similarity has many benefits. One possible future direction would be to expand **ApproxMAP** to do local alignment and investigate the benefits of local multiple alignment for sequences of sets in KDD applications.

Second, in the optimization of sample based iterative clustering the hash table implementation needs to be explored further. The optimization is made in order to speed up the running time for large datasets. But the current hash table implementation has a large memory requirement for large datasets. In our experiments, we ran out of memory for datasets $N_{seq} > 70000$ given 2GB of memory. We already know that there are other more efficient implementations of hash tables. But ultimately, to make our method practically scalable, we need to explore an implementation that stores only the possible proximity values (limited by the given memory size), and recalculate the other distances when needed. This will make the application work with any memory size and still give a significant reduction in time (Figure 6.6(d)).

The most interesting future direction is to expand the distance metric to be more comprehensive. First, it could be expanded to handle sequences of multisets or sets with quan-

titative information. Many of the data mining applications have sets that have more than one of the same item (multiset). For example, people buy many packs of diapers at once. If **ApproxMAP** could be expanded to handle multisets, it can find quantitative sequential patterns.

Second, user specified taxonomies could be used to customize the replacement cost. For example, two toys should be considered more similar to each other than a toy and a piece of furniture. Under the current model, {doll}, {crib}, and {ball} are all equally distant. If a user specified a taxonomy tree putting doll and ball under the same ancestor and crib in a separate branch, the distance metric could be expanded to a weighted distance metric which can incorporate this information.

Last, a practical improvement to **ApproxMAP** would be to automatically detect the best strength threshold, θ , for each cluster of sequences. An interesting approach could be analyzing the distribution of the item weights dynamically. Initial investigation seems to suggest that the item weights may follow the Zipf distribution. Closer examination of the distribution might give hints for automatically detecting statistically significant cutoff values customized for each cluster. When presenting an initial overview of the data, such approach could be quite practical.