Abstract

We start by comparing two popular k-medoids algorithms, clarans (Ng et al., 2005) and vik (Park and Jun, 2009), and find that clarans consistently results in better clusterings. We provide an explanation for this finding in terms of local minima of their respective loss functions. We then discuss how clarans can be accelerated by using the triangle inequality and early proposal rejection, making it a viable algorithm for large datasets. The observation that clarans finds better minima than vik, along with the similarity between vik and the standard k-means algorithm, suggests that clarans may be an effective k-means initialiser. We show that this is indeed the case, with clarans outperforming other popular seeding algorithms such as k-means++ (Arthur and Vassilvitskii, 2007) on 20/20 datasets with a mean decrease in energy of 2.2%.

1 Introduction

We introduce the k-means and k-medoids problems and the most popular algorithms for solving them. We then discuss k-means initialisation, and finally describe our contributions.

1.1 k-means

The k-means problem is to find $K$ centroids to minimise the mean distance between samples and their nearest centroids. Specifically, given $N$ training samples $X = \{x(i) : i \in \{1, \ldots, N\}\}$ in a vector space $V$, the task is to find $Y = \{y(k) : k \in \{1, \ldots, K\}\}$ in $V$ to minimise energy $E$ defined by,

$$E(Y) = \frac{1}{N} \sum_{i=1}^{N} \|x(i) - y(a^1(i))\|^2,$$  

(1)

where $a^1(i) = \arg \min_{k \in \{1, \ldots, K\}} \|x(i) - y(k)\|$. In general the k-means problem is NP-hard, and so a trade off must be made between low energy and low run time in finding an approximate solution. The k-means problem arises in data compression, classification, density estimation, and many other areas.

We will refer to the standard algorithm for k-means (Lloyd, 1982) as lloyd. It relies on a two-step iterative refinement technique. In the assignment step, each sample is assigned to the cluster whose centroid is nearest to it. Then in the update step, cluster centroids are updated in accordance with assigned samples, that is

$$y(k) \leftarrow \frac{\sum_{i : a(i) = k} x(i)}{\|i : a(i) = k\|}, \ k \in \{1, \ldots, K\}. \quad (2)$$

Lloyd’s algorithm was voted as one of the top 10 algorithms in machine learning in Wu et al. (2008).

1.2 k-medoids

Given $N$ samples, the k-medoids problem in its most general form is to select $K$ indices $C = \{c(k) : k \in \{1, \ldots, K\}\} \subset \{1, \ldots, N\}$ so as to minimise energy $E$ now defined by,

$$E(C) = \frac{1}{N} \sum_{i=1}^{N} f(x(i), x(c(a^1(i)))),$$  

(3)

where function $f$ measures dissimilarity, and $a^1(i) = \arg \min_{k \in \{1, \ldots, K\}} f(x(i), x(c(k)))$. Function $f$ has no constraint, and does not need to satisfy any of the four requirements of a metric, namely non-negativity, identity, symmetry and the triangle inequality. Unlike k-means, samples need not belong to a vector space. In practice $f$ is often not defined explicitly, and an $N \times N$ matrix of dissimilarities is provided by the user. Given such an $N \times N$ matrix, the k-medoids task corresponds...
to selecting $N$ matrix entries under the constraint that there is one entry selected in each column and zero entries selected in $N - K$ rows, so as to minimise the sum of selected entries.

In the case where data belongs to a metric space and $f(x(i), x(i')) = \|x(i) - x(i')\|^2$, $k$-medoids corresponds to $k$-means under the constraint that centers coincide with points. As with $k$-means, the $k$-medoids problem is NP-hard and heuristic algorithms are used. A simple two-step iterative $k$-medoids algorithm which is very similar to $lloyd$ is that of Park and Jun (2009). We will refer to the algorithm of Park and Jun (2009) as $vik$ (voronoi iteration $k$-medoids) as they do not provide a name for it. The update step of $vik$ consists of computing cluster medoid indices,

$$c(k) \leftarrow \arg \min_{i:a^1(i) = k, i':a^1(i') = k} \sum f(x(i), x(i')),$$

as opposed to cluster means as in (2) in $lloyd$, otherwise $lloyd$ and $vik$ are the same. The computation of the medoid (4) is quadratic in cluster size, although when $f$ is a metric the algorithm of Newling and Fleuret (2016b) can be used to improve the complexity under certain constraints.

Precursors of $vik$ are all swap-based algorithms. In contrast to $vik$ which updates all cluster medoids at every iteration, swap-based algorithms only update one medoid per iteration. This is done by swapping one of the $K$ medoids with one of the $N - K$ non-medoids. There are a total of $K(N - K)$ possible such swaps, one for each (medoid, non-medoid) pair. As all swaps between medoids and non-medoids are permitted, it is possible for medoids to move much further with swap-based algorithms than with $vik$, allowing them to escape from local minima.

The first swap-based algorithm, $pam$ of Kaufman and Rousseeuw (1990), considers all $K(N - K)$ possible swaps at each iteration, and implements the one resulting in the greatest reduction in energy. Other swap-based algorithm are $clara$ (Kaufman and Rousseeuw 1990), and $clarans$ (Ng et al. 2005) which proceeds by iteratively proposing a swap, computing the total change in energy resulting from the proposed swap, and then immediately implementing the swap if and only if the change is negative. Swap-based algorithms will be presented in more detail in Section 2.1.

It should be noted that all published $k$-medoids algorithms take as input a dissimilarity matrix, which means that they are all $O(N^2)$ in memory. This renders them unusable for even moderately large datasets ($N \sim 10^5$). However, it is straightforward to adapt the algorithms $vik$ and $clarans$ to the setting where distances are computed as needed, which we will refer to as the on-the-fly setting. It does not make sense to adapt $pam$ and $clara$ to the on-the-fly setting, as they both require a quadratic number of distances at every iteration anyway.

We finally note that the only published comparison between $vik$ and a swap-based algorithm is that in Park and Jun (2009), where only extremely small ($N \leq 360, K \leq 3$) low-dimensional real datasets are considered in their comparison of $vik$ with $pam$.

### 1.3 $k$-means initialisation

A recent comparative study of initialisation schemes for $k$-means is that of Celebi et al. (2013), where 8 methods are compared across a large number of datasets. The comparison is done in terms of both speed (time to complete initialisation and $lloyd$ run) and energy (final SSE). The study finds that 3 of the 8 considered schemes should never be used. One of the 3 poor schemes is uniform initialisation, henceforth $uni$, where $K$ samples are randomly selected to initialise the centroids. Of the remaining schemes, there is no clear winner, with results varying between datasets, but the authors suggest that the algorithm of Bradley and Fayyad (1998), which we will call $bf$, is in general a good choice. It should be noted that the experiments in Celebi et al. (2013) are all in the relatively low $K$-regime ($K < 50, N/K > 100$).

The $bf$ scheme of Bradley and Fayyad (1998) works as follows. Samples are separated into $J$ (= 10) partitions. $lloyd$ with $uni$ is performed on each of the partitions, providing $J$ centroid sets of size $K$. A superset of $JK$ elements is created by concatenating the $J$ center sets. $lloyd$ is then run $J$ times on the superset, initialised at each run with a distinct center set. The center set which obtains the lowest SSE on the superset is taken as the final initialiser for the final run of $lloyd$ on all $N$ samples.

Probably the most widely implemented initialisation scheme other than $uni$ is $k$-means++ (Arthur and Vassilvitskii 2007). Its popularity stems from its simplicity, theoretical guarantees, and strong experimental support. The algorithm works by sequentially selecting $K$ seeding samples. At each iteration, a sample is selected with probability proportional to the square of the distance to the nearest previously selected sample. Selection probabilities are initially uniform.

All of the initialisation schemes considered by Bradley and Fayyad (1998) are greedy, in the sense that seedings are never refined. Once selected as an initialising center in $k$-means++, a sample cannot be removed. This is in contrast to $k$-medoids algorithms, which suggests the utility of $k$-medoids for $k$-means initialisation.
Our second contribution is to accelerate the on-the-fly setting. We have chosen the \textit{pam} and \textit{vik} of any previous studies presenting this useful finding. We are unaware of any previous studies in the \textit{first choice for k-medoids clustering}. We are unaware of any previous studies presenting this useful finding. We are unaware of any previous studies presenting this useful finding.

Specifically, when considering \( c(2) = 7 \), a swap-based algorithm will consider for example set-\( \{1\} \cup \{4\} \) and \( \{2\} \cup \{4\} \) exchanges. This is described with an example in Section 2.2.

Our third and most significant contribution is to conduct experiments showing that \textit{clarans} provides better initialisation for \textit{lloyd} than other popular schemes such as \textit{k-means++}. We show, on 20 publicly available datasets, that for any reasonable time budget, initialisation with \textit{clarans} always results in improved final clusterings, with lower energy than those obtained using \textit{bf} or \textit{k-means++}.

Our final contribution is to release our open source \texttt{C++} implementation of the accelerated \textit{clarans} algorithm, which can be used with a wide range of data types. Our multi-threaded implementation can be used directly as a shared library with header files, or through our Python (and R, coming soon) libraries.

\section{k-medoids algorithms}

We present the notation required to describe the algorithms \textit{vik} \cite{park2009accelerating}, \textit{pam} \cite{kaufman1990finding} and \textit{clarans} \cite{ng2005accelerating}. As before, we will let \( N \) be the number of samples which we want to partition into \( K \) clusters be \( x(1), \ldots, x(N) \). Let \( t \in \{1, \ldots, \infty\} \) denote the current round of the algorithm. Let \( c_t(k) \in \{1, \ldots, N\} \) be the index of the sample chosen as the center of cluster \( k \in \{1, \ldots, K\} \) at iteration \( t \), so that \( x(c_t(k)) \) is the center of cluster \( k \) at iteration \( t \). Let \( C_t = \{c_t(k) \mid k \in \{1, \ldots, K\}\} \subseteq \{1, \ldots, N\} \) denote all such center indices. We again let \( a_t(i) \) be the cluster of sample \( i \), that is

\begin{equation}
    a_t(i) = \arg \min_{k \in \{1, \ldots, K\}} f(x(i), x(c_t(k))).
\end{equation}

Let \( \psi_t(k) \) denote the sum of the dissimilarities of elements in cluster \( k \) at iteration \( t \), also referred to as the energy of cluster \( k \), so that

\begin{equation}
    \psi_t(k) = \sum_{i: a_t(i) = k} f(x(i), x(c_t(k))).
\end{equation}

Let \( \psi_t = \sum_k \psi_t(k) \) be the total energy, the quantity which we ultimately wish to minimise. With this notation in hand, we present \textit{vik} in Alg. 1.\footnote{Manuscript under review by AISTATS 2017}
The first swap-based algorithm for k-medoids, \texttt{pam}, was introduced by \cite{kaufman1990finding}. It works by iteratively changing a single cluster center at a time, until no energy reducing changes can be found. Specifically, at iteration $t$ all possible ways of replacing one of the $K$ centers with one of the $N - K$ non-centers, are considered. In total there are $K(N - K)$ such swaps to consider, where each swap involves computing the total energy with centers $C_t \setminus \{c_t(k_p)\} \cup \{i_p\}$, which in itself incurs a cost $O(N - K)$. Thus \texttt{pam} costs $O(KN^2)$ to change a single cluster, which is very expensive and makes \texttt{pam} unusable for moderately large datasets. Pseudocode for \texttt{pam} is presented in Alg. 5 in SM-C.

The poor scaling of \texttt{pam} motivated \cite{kaufman1990finding} to propose the algorithm \texttt{clara}, where only a subset of the $N$ elements are considered as possible centers. An improvement over \texttt{clara} came about with the \texttt{clarans} algorithm of \cite{ng2005clarans}, where all $N$ elements are considered as possible centers, but instead of waiting for the best possible swap to be found, any swap which results in a reduction in energy is automatically accepted.

If the number of consecutive swaps rejected exceeds some threshold $N_r$, \texttt{clarans} halts. The full \texttt{clarans} algorithm also proposes using several random initialisations, although this is not an idea specific to \texttt{clarans}. The core of \texttt{clarans} is presented in Alg. 2

\begin{algorithm}
\caption{\texttt{vik}, the two-step iterative k-medoids algorithm of \cite{park2009two} (2009). It is the k-medoids equivalent of \texttt{lloyd} for k-means.}
\begin{algorithmic}
  \State $t \leftarrow 0$
  \State Initialise $C_0 \subset \{1, \ldots, N\}$.
  \While{$t = 0$ or $C_t \neq C_{t-1}$}
    \For{$i \in \{1, \ldots, N\}$}
      \State $a_i^t(i) \leftarrow \arg \min_{k \in \{1, \ldots, K\}} f(x(i), x(c_t(k)))$
    \EndFor
    \For{$k \in \{1, \ldots, K\}$}
      \State $c_{t+1}(k) \leftarrow \arg \min_{i : a_i^t(i) = k} \sum_{i' : a_i^t(i') = k} f(x(i), x(i'))$
    \EndFor
    \State $t \leftarrow t + 1$
  \EndWhile
\end{algorithmic}
\end{algorithm}

\subsection{2.2 Contribution : accelerating \texttt{clarans}}

In this section we outline how we accelerate \texttt{clarans}, with a full analysis and pseudocode found to SM-D.2

We assume here that dissimilarity can be decomposed as in Eqn. [3], which will enable the use of the triangle inequality. Let $d_{t1}^i(i)$ be the distance at iteration $t$ of sample $i$ to its nearest center, that is

$$d_{t1}^i(i) = \min_{i' \in C_t} \text{dist}(x(i), x(i')).$$

Under assumption [3], we now have [6] taking the form,

$$a_i^t(i) = \arg \min_{k \in \{1, \ldots, K\}} \text{dist}(x(i), x(c_t(k))),$$

so that $d_{t1}^i(i) = \text{dist}(x(i), x(c_t(a_i^t(i))))$. In the same way as we use $a_i^t(i)$ and $d_{t1}^i(i)$ for the nearest center, we will use $a_i^t(i)$ and $d_{t2}^i(i)$ for the second nearest center, that is

$$d_{t2}^i(i) = \min_{i' \in C_t \setminus \{c_t(a_i^t(i))\}} \text{dist}(x(i), x(i')),$$

$$a_i^t(i) = \arg \min_{k \in \{1, \ldots, K\} \setminus \{a_i^t(i)\}} \text{dist}(x(i), x(c_t(k))),$$
so that $d^2_t(i) = \text{dist}(x(i), x(c_t(a^2_t(i))))$. The energy of a sample is now defined as the energy of the distance to its nearest center, so that at iteration $t$ the energy of sample $x(i)$ is $\psi(d^2_t(i))$. Finally, let the margin of sample $i$ be defined as $m_t(i) = \psi(d^2_t(i)) - \psi(d^2_t(i))$.

Some cluster specific quantities which are required in the accelerated algorithm are,

\begin{align*}
N_t(k) &= |\{i : a^1_t(i) = k\}|, \\
D^1_t(k) &= \max_{i : a^1_t(i) = k} d^1_t(i), \\
D^2_t(k) &= \max_{i : a^1_t(i) = k} d^2_t(i), \\
M^*_t(k) &= \frac{1}{N_t(k)} \sum_{i : a^1_t(i) = k} m_t(i).
\end{align*}

The key triangle inequality results used to accelerate \texttt{clarans} evaluations are now presented. Firstly,

$$\text{dist}(x(i_p), x(c_t(k_p))) \geq D^1_t(k_p) + D^2_t(k_p) \implies \text{change in energy of cluster } k_p \text{ is } N_t(k_p)M^*_t(k_p),$$

which says that if the new center $x(i_p)$ of cluster $k_p$ is sufficiently far from the old center $x(c_t(k_p))$, then all old elements of cluster $k_p$ will migrate to their old second nearest clusters, and so their change in energies will simply be their margins, which have already been computed. The second inequality used is,

$$k \neq k_p \land \text{dist}(x(c_t(k)), x(i_p)) \geq 2D^1_t(k) \implies \text{no change in energy of cluster } k,$$

which states that if cluster $k$ is sufficiently far from the new center of $k_p$, there is no change in its energy as the indices of samples assigned to it do not change.

These implications allow changes in energies of entire clusters to be determined in a single comparison. Clusters likely to benefit from these tests are those lying far from the new proposed center $x(i_p)$. The above tests involve the use of $\text{dist}(x(c_t(k)), x(i_p))$, but the computations of this quantity can sometimes be avoided by using the inequality,

$$\text{dist}(x(c_t(k)), x(i_p)) \geq cc_t(a^1_t(i_p), k) - D^1_t(i_p),$$

where $cc_t$ is the $K \times K$ matrix of inter-medoid distances at iteration $t$. To accelerate the update step of \texttt{clarans}, the following bound test is used,

$$\min(\text{dist}(x(c_t(k_p)), x(c_t(k))), \text{dist}(x(i_p), x(c_t(k)))) \geq D^1_t(k) + D^2_t(k) \implies \text{no change in cluster } k.$$

We also use a per-sample version of the above inequality for the case of failure to eliminate the entire cluster. Full proofs, descriptions, and algorithms incorporating these triangle inequalities can be found in \texttt{SM-D.2}.

The triangle inequality based accelerations do not affect the final clustering of \texttt{clarans} in any way, so that for a given sequence of random proposals, the final clustering produced with or without the triangle inequality accelerations are the same. We now discuss a further acceleration which does modify the final clustering for a given sequence of random proposals. The idea is that, if a proposal is bad, one may be able to determine this with a small sub-sample of data.

We estimate the post-swap energy on small sub-samples of data, only proceeding to larger sub-samples if the estimated energy is less than the pre-swap energy. This approach, described in \texttt{SM-D.2} occasionally results in good swaps being rejected, but can mean far less time spent in computing energies of bad swaps.

### 3 Results: k-medoids algorithms

We compare \texttt{clarans} with and without the accelerations discussed in Section 2.2 and \texttt{vik}. Experiments are performed on four synthetic datasets, and then on four real datasets. Experiments are run given a fixed time budget, or until convergence. Across all runs with all eight datasets, we observe that \texttt{clarans} obtains significantly better clusterings (lower energies) than \texttt{vik}. This is as predicted in the discussion accompanying Figure 1. We also observe that the accelerated version of \texttt{clarans} often results in significant speed-ups.

The synthetic datasets which we have generated are described in Table 1 with the real datasets described in Table 2. We have attempted to cover a wide range of settings to illustrate the diversity of applications of k-medoids, using text and genomic sequence data, and sparse and dense vector data.

One observation is that in all 8 experiments, \texttt{clarans} does not appear to be sensitive to initialisation. The energies obtained using both the accelerated and unaccelerated versions appear to converge to a common minimum energy. This is true even when the initial energy varies greatly, as is the case with the rcv1 dataset. This suggests that, given a fixed amount of time, it is better to use the entire time budget on a single run of \texttt{clarans} than to run several shorter runs of \texttt{clarans} with distinct initialisations. In contrast, \texttt{vik} converges to local minima of varying quality, being strongly dependent on the initial medoids selected.

### 4 Results: k-means initialisation

We consider using \texttt{clarans} to initialise \texttt{lloyd}. Specifically, we consider setting the initial centroids of \texttt{lloyd} to be the final medoids of \texttt{clarans}. One can consider running \texttt{clarans} for a specified length of time during
Table 1: Synthetic datasets used for comparing k-medoids algorithms (Figure 2). **syn-1**: Each of the cluster centers is a random binary sequence of 16 bits. In each of the clusters, 50 elements are generated by applying 2 mutations (insert/delete/replacement) to the center, at random locations. **syn-2**: Each of the centers is a vector in $\mathbb{R}^{10^6}$, and is non-zero at exactly 5 indices, with the 5 values drawn from $N(0, 1)$. Each sample is a linear combination of two centers, with coefficients 1 and $Q$, where $Q \sim U[-0.5, 0.5]$. **syn-3**: Centers are integer co-ordinates of an $11 \times 11$ grid. For each center, 50 samples are generated, each sample being the center plus Gaussian noise of identity covariance. **syn-4**: Data are points drawn uniformly from $[0, 1]^2$. In this experiment we use the k-medoids algorithms in an attempt to cover a unit square with 100 squares of diameter 0.1, a task with a unique lattice solution. Points not covered have energy 1, while covered points have energy 0.

| N   | K   | type  | metric | $\psi(d)$ |
|-----|-----|-------|--------|-----------|
| syn-1 | 20000 | 40    | sequence | Levenshtein | $d$ |
| syn-2 | 200000 | 100   | sparse-v | $l_2$ | $d^2$ |
| syn-3 | 25800 | 144   | dense-v | $l_1$ | $e^d$ |
| syn-4 | 20000 | 100   | dense-v | $l_\infty$ | $I_{d>0.05}$ |

Table 2: Real datasets used for comparing k-medoids algorithms (Figure 3), with hyper-links to data in SM.

| N   | K   | type  | metric       | $\psi(d)$ |
|-----|-----|-------|--------------|-----------|
| rcv1 | 23149 | 400   | sparse-v     | $l_2$ | $d^2$ |
| genome | 400000 | 1000 | sequence     | n-Levensh. | $d^2$ |
| mnist | 10000 | 400   | dense-v      | $l_2$ | $d^2$ |
| words | 354983 | 1000 | sequence     | Levenshtein | $d^2$ |

Figure 3: Results on real datasets. Vertical axes are energies relative to the lowest energy found. We observe that vik performs very poorly on sequence datasets (right), failing to find clusterings significantly better than the random initialisations. While an improvement over the initial seeding is obtained using vik on the vector datasets (left), the energies obtained using clarans are significantly lower. Runs with accelerated clarans appear to converge to a common energy solution, even when the initial energies vary greatly, as uis the case in sparse dataset rcv1. The majority of runs with vik converge to a local minimum before the allotted time limit of $2^{10}$ seconds. We observe a speed-up using the accelerated version of clarans (cl(+) ), especially on the the sequence datasets.
As in Celebi et al. (2013), we find that uni is consistently worse than k-means++. We also find that bf outperforms uni but is generally not as good as k-means++. These findings are presented in SM-A in Figures 5, 6, 7 and 8 and establish k-means++ as the best baseline. We will therefore focus on presenting results comparing k-means++ with cl-1 and cl-3.

In Figure 4 the key results of this paper are presented. For each dataset, energies of all experiments are plotted at their cumulative finishing times, with curves tracking the lowest energies found so far with each initialisation scheme. While initialisation with k-means++ results in more runs in time limit TL than with cl-1, the energies of obtained with cl-1 are consistently significantly lower, meaning that a single run with cl-1 in general outperform the best run of k-means++. On all 20/20 datasets, the energy at time limit TL with the best cl-1 is lower than that with k-means++. The mean improvement is 2.2%, with a Table of improvements by dataset in SM-A.

We see that using cl-1 results in many more runs in time limit TL than using cl-1, but that the final energies found using cl-1 are significantly worse. Thus it appears that performing more swaps with clarans guarantees better initialisation.

## 5 Conclusion and Future Works

In this paper, we have demonstrated that the algorithm clarans outperforms vik on a diverse range of k-medoids clustering tasks, establishing it as a good default choice for k-medoids. We have described techniques for accelerating clarans, and shown that it works very effectively as an initialiser for lloyd, outperforming other initialisation schemes on a wide range of datasets.

An interesting extension of clarans may be to consider two or more swaps simultaneously, enabling clarans to find better configurations. Another idea is to initialise clarans with k-means++. A third possibility could be to use importance sampling to better estimate energy, although the elimination of clusters with the triangle inequality can be considered a form of importance sampling. Finally, it may be beneficial to weight sample proposals by cluster energy, so as to favour the removal of under-populated clusters.

| dataset   | N  | dim | K  | TL  |
|-----------|----|-----|----|-----|
| a1        | 3000 | 2  | 40 | 0.37 |
| a2        | 5250 | 2  | 70 | 0.53 |
| a3        | 7500 | 2  | 100| 0.69 |
| birch1    | 100000 | 2  | 200| 10.58 |
| birch2    | 100000 | 2  | 200| 6.97 |
| birch3    | 100000 | 2  | 200| 9.48 |
| ConfLong  | 164860 | 3  | 22 | 22.41 |
| dim032    | 1024 | 32 | 32 | 0.36 |
| dim064    | 1024 | 64 | 32 | 0.40 |
| dim1024   | 1024 | 1024 | 32 | 1.72 |
| europe    | 169308 | 2  | 1200| 128.24 |
| housec8   | 34112 | 3  | 400| 10.87 |
| KDDCUP04  | 145751 | 74  | 600| 990.76 |
| mnist     | 10000 | 784 | 300| 105.94 |
| Mopsi     | 13467 | 2  | 100| 0.86 |
| s1        | 5000  | 2  | 30 | 0.42 |
| s2        | 5000  | 2  | 30 | 0.39 |
| s3        | 5000  | 2  | 30 | 0.54 |
| s4        | 5000  | 2  | 30 | 0.47 |
| yeast     | 1484  | 8  | 40 | 0.43 |

Table 3: The 20 datasets used to compare k-means initialisers. They are all publicly available at [https://cs.joensuu.fi/sipu/datasets/](https://cs.joensuu.fi/sipu/datasets/) with detailed descriptions. Column ‘TL’ is the time allocated for running with each of initialisation schemes: no new run may start after TL seconds have elapsed.
Figure 4: Tracking the minimum energy using each of the 3 initialisation schemes, \texttt{k-means++}, \texttt{cl-1} and \texttt{cl-3.0}. Each point represents a run of \texttt{lloyd} until convergence, with each series of points corresponding to a different initialisation scheme. The horizontal axis is time, relative to the time limit TL (see Table 3). The vertical axis is the logarithm of the energy, relative to the best energy over all runs ($E_{min}$) for each dataset. The solid step-like curves track the minimum found so far with each scheme. On 20/20 datasets, we observe that initialisation with \texttt{cl-3} results in the lowest energy clustering in time limit TL. As runs with \texttt{cl-3} are significantly slower than those with \texttt{k-means++} and \texttt{cl-1}, it takes longer to complete a single run, however the energy of the first run with \texttt{cl-3} is better than all runs with \texttt{k-means++} on 19/20 datasets (ConfLong is the exception). While \texttt{cl-1} is better than \texttt{k-means++} at time limit TL on 15/20 datasets, it is never as good as \texttt{cl-3}, so it is important to run \texttt{clarans} with at least 3K swaps for optimal performance. The same figure but including \texttt{bf} and \texttt{uni} is presented in SM-A as Figure 5.
Acknowledgements

James Newling was funded by the Hasler Foundation under the grant 13018 MASH2.

References

Arthur, D. and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’07, pages 1027–1035, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Bradley, P. S. and Fayyad, U. M. (1998). Refining initial points for k-means clustering. In Proceedings of the Fifteenth International Conference on Machine Learning, ICML ’98, pages 91–99, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Celebi, M. E., Kingravi, H. A., and Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. Expert Syst. Appl., 40(1):200–210.

Kaufman, L. and Rousseeuw, P. J. (1990). Finding groups in data : an introduction to cluster analysis. Wiley series in probability and mathematical statistics. Wiley, New York. A Wiley-Interscience publication.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. JOURNAL OF MACHINE LEARNING RESEARCH, 5:361–397.

Lloyd, S. (1982). Least squares quantization in pcm. IEEE Trans. Inf. Theor., 28(2):129–137.

Newling, J. and Fleuret, F. (2016a). Fast k-means with accurate bounds. In Proceedings of the International Conference on Machine Learning (ICML), pages 936–944.

Newling, J. and Fleuret, F. (2016b). A sub-quadratic exact medoid algorithm. arXiv:1605.06950.

Ng, R. T., Han, J., and Society, I. C. (2005). Clarans: A method for clustering objects for spatial data mining. IEEE Transactions on Knowledge and Data Engineering, pages 1003–1017.

Park, H.-S. and Jun, C.-H. (2009). A simple and fast algorithm for k-medoids clustering. Expert Syst. Appl., 36(2):3336–3341.

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G., Ng, A., Liu, B., Yu, P., Zhou, Z.-H., Steinbach, M., Hand, D., and Steinberg, D. (2008). Top 10 algorithms in data mining. Knowledge and Information Systems, 14(1):1–37.

Yujian, L. and Bo, L. (2007). A normalized levenshtein distance metric. IEEE Trans. Pattern Anal. Mach. Intell., 29(6):1091–1095.
SM-A Further initialisation result figures

In Figure 5, the experiments with \texttt{bf} and \texttt{uni} are added to Figure 4. On the majority of datasets we see that initialisation with \texttt{bf} and \texttt{uni} result in significantly worse energies than initialisation with \texttt{k-means++} and \texttt{clarans}. We believe that the poor performance of \texttt{bf} relative to \texttt{k-means++}, which appears to be in disagreement with the results in Celebi et al. (2013), is due to the larger $K$s and smaller $N/K$s which we consider.

In Figure 8, the results are presented in a different manner. We again have each marker representing a full run of k-means, with each curve representing all runs initialised (not completed, as in 5) within time limit TL with a given initialisation scheme, but these plots hide the time component of performance. We see that there are fewer markers on curves with cl-3 than with cl-1, as expected as initialisation with 2.5$K$ swaps takes longer than initialisation with $K$ swaps. The horizontal axis is final energy after Lloyd has completed, relative to the lowest energy across all Lloyd runs. The vertical axis is fraction of runs above a certain energy, so that for example, fraction $f = 1$ corresponds to the best run and fraction $f = 0$ corresponds to the worst.
Figure 5: Tracking the minimum using each of the 5 initialisation methods over a series of runs, as in Figure. We observe that uni always performs poorly, while bf generally performs poorly too.
Figure 6: Illustrating the color coding in Figure 7 for dataset yeast. On top, each series of coloured markers denotes a series of runs of Lloyd with each of the five initialisers, with energy on the vertical axis and fraction of time limit TL elapsed on the horizontal axis. Below, the initialiser corresponding to the lowest energy obtained so far. When several initialisers are equally good to within factor 1.01, they are presented in parallel. The colours are as per Figure 8.

Figure 7: The initialisation schemes resulting in the lowest energies at different epochs, coloured as in Figure 8 and as explained in Figure 6.

Figure 8: Initialisation experiment results for the 20 datasets in Table 3. Each marker (‘x’ or ‘+’) corresponds to one run of k-means. The horizontal axis is energy, relative to the lowest energy obtained across all runs. The vertical axis is fraction of experiments, so that a marker at height $0 \leq f \leq 1$ has lower energy than fraction $f$ of all runs using the same initialisation scheme. The best run with cl-3 in time limit TL always results in lowest energy, and in most experiments a single run with cl-3 is enough to find an as good or better energy than with any other scheme.
Table 4: Relative energies of best runs with \texttt{k-means++} and c1-3 in time limit TL. Specifically, the values are \((\text{min energy over all runs with \texttt{k-means++}}) / (\text{min energy over all runs with c1-3})\). The mean ratio is 1.0223, so that the mean improvement using c1-3 is 2.23\%, as presented in the paper.

| Dataset | Energy |
|---------|--------|
| a1      | 1.0259 |
| a2      | 1.0153 |
| a3      | 1.0320 |
| birch1  | 1.0073 |
| birch2  | 1.0378 |
| birch3  | 1.0138 |
| ConfLong| 1.0000 |
| dim032  | 1.0204 |
| dim064  | 1.0500 |
| dim1024 | 1.0331 |
| europe  | 1.0310 |
| housec8 | 1.0085 |
| KDDCUP04| 1.0013 |
| mnist   | 1.0095 |
| Mopsi   | 1.1090 |
| s1      | 1.0114 |
| s2      | 1.0160 |
| s3      | 1.0061 |
| s4      | 1.0085 |
| yeast   | 1.0097 |
SM-B  The task

We restate the k-medoids task in the setting where dissimilarity is an increasing function of a distance function. Given a set of $N$ elements, \( \{ x(i) : i \in \{1, \ldots, N\} \} \), with a distance defined between elements,

\[
\begin{align*}
dist(x(i), x(i')) & \geq 0, \\
dist(x(i), x(i)) & = 0, \\
dist(x(i), x(i')) & = \text{dist}(x(i'), x(i)), \\
dist(x(i), x(i'')) & \leq \text{dist}(x(i), x(i')) + \text{dist}(x(i'), x(i'')),
\end{align*}
\]

and given an energy function \( \psi : \mathbb{R}^+ \rightarrow \mathbb{R}^+ \) satisfying,

\[
\begin{align*}
\psi(0) & = 0, \\
v_1 \leq v_2 & \iff \psi(v_1) \leq \psi(v_2),
\end{align*}
\]

The task is to find indices \( \{ c(k) : k \in \{1, \ldots, K\} \} \subset \{1, \ldots, N\} \), to minimise,

\[
\sum_{i=1}^{N} \min_{k \in \{1, \ldots, K\}} \psi(\text{dist}(x(i), x(c(k)))).
\]

SM-C  The pam algorithm

Algorithm 3 The pam algorithm of Kaufman and Rousseeuw [1990]. At lines 4 and 5, one loops over all possible (medoid, non-medoid) swaps, recording the energy obtained with each swap. At line 9, the best of all possible swaps is chosen. At line 10, if the best found swap results in decrease in energy, proceed, otherwise stop.

1: \( t \leftarrow 0 \)
2: Initialise \( C_0 \subset \{1, \ldots, N\} \).
3: while true do
4:    for \( i_p \in \{1, \ldots, N\} \setminus C_t \) do
5:        for \( k_p \in \{1, \ldots, K\} \) do
6:            \( \psi_{t+1}(i_p, k_p) \leftarrow \sum_{i'=1}^{N} \min_{i' \in C_t \setminus \{c_t(k_p)\} \cup \{i_p\}} \psi(\text{dist}(x(i), x(i'))) \)
7:        end for
8:    end for
9:    \( i^*_p, k^*_p \leftarrow \arg \min_{i_p, k_p} \psi_{t+1}(i_p, k_p) \)
10:   if \( \psi_{t+1}(i^*_p, k^*_p) < 0 \) then
11:      \( C_{t+1} \leftarrow C_t \setminus \{c_t(k^*_p)\} \cup \{i^*_p\} \)
12:   else
13:      break
14:   end if
15:   \( t \leftarrow t + 1 \)
16: end while

SM-D  clarans in detail, and how to accelerate it.

SM-D.1 Some additional notation

Consider a proposed update for centers at iteration \( t + 1 \), where the center of cluster \( k_p \) is replaced by \( x(i_p) \). Let \( \delta_t(i \mid k_p \bowtie i_p) \) denote the change in energy of sample \( i \) under such an update, that is

\[
\delta_t(i \mid k_p \bowtie i_p) = \text{energy after swap} - \text{energy before swap} = \min_{i' \in C_t \setminus \{c_t(k_p)\} \cup \{i_p\}} \psi(\text{dist}(x(i), x(i'))) - \psi(d^1_t(i)).
\]
We choose subscript ‘p’ for \( k_p \) and \( i_p \), as together they define a proposed swap. We will write \( a^{12}d_t^{12}(i) = \{a_1^1(i), a_1^2(i), d_t^1(i), d_t^2(i)\} \) throughout for brevity. Finally, let

\[
D_t^1(k) = \max_{i : a_1^1(i) = k} d_t^1(i),
\]

\[
D_t^2(k) = \max_{i : a_1^2(i) = k} d_t^2(i).
\]

**Algorithm 4** One round of clarans. The potential bottlenecks are the proposal evaluation at line 2 and the update at line 6. The cost of proposal evaluation, if all distances are pre-computed, is \( O(N) \), while if distances are not pre-computed it is \( O(dN) \) where \( d \) is the cost of a distance computation. As for the update step, there is no cost if \( \Delta_t \geq 0 \) as nothing changes, however if the proposal is accepted then \( C_{t+1} \neq C_t \), and all data whose nearest or second nearest center change needs updating.

1: Make proposal \( k_p \in \{1, \ldots, K\} \) and \( i_p \in \{1, \ldots, N\} \setminus C_t \).
2: \( \Delta_t(k_p < i_p) \leftarrow \frac{1}{N} \sum_{i=1}^{N} \delta_t(i | k_p < i_p) \) \hspace{1cm} \triangleright \text{The assignment evaluation step, see Alg. 5}
3: if \( \Delta_t < 0 \) then
4: \( C_{t+1} \leftarrow C_t \setminus \{c_t(k_p)\} \cup \{i_p\} \)
5: for \( i \in \{1, \ldots, N\} \) do
6: \( \text{Set } a^{12}d_{t+1}^1(i) \)
7: end for
8: else
9: \( C_{t+1} \leftarrow C_t \)
10: for \( i \in \{1, \ldots, N\} \) do
11: \( a^{12}d_{t+1}^2(i) \leftarrow a^{12}d_t^2(i) \)
12: end for
13: end if

**Algorithm 5** Standard approach (level 0) with clarans for computing \( \delta_t(i | k_p < i_p) \) at iteration \( t \), as described in [Ng et al.](2005).

1: \( d \leftarrow \text{dist}(x(i), x(i_p)) \)
2: if \( a_1^1(i) = k_p \) then
3: if \( d \geq d_t^2(i) \) then
4: \( \delta_t(i | k_p < i_p) \leftarrow \psi(d_t^2(i)) - \psi(d_t^1(i)) \)
5: else
6: \( \delta_t(i | k_p < i_p) \leftarrow \psi(d) - \psi(d_t^1(i)) \)
7: end if
8: else
9: if \( d \geq d_t^1(i) \) then
10: \( \delta_t(i | k_p < i_p) \leftarrow 0 \)
11: else
12: \( \delta_t(i | k_p < i_p) \leftarrow \psi(d) - \psi(d_t^1(i)) \)
13: end if
14: end if

**SM-D.2 Accelerating clarans**

We discuss how to accelerate the proposal evaluation and the cluster update. We split our proposed accelerations into 3 levels. At levels 1 and 2, triangle inequality bounding techniques are used to eliminate distance calculations. At level 3, an early breaking scheme is used to quickly reject unpromising swaps.

**SM-D.2.1 Basic triangle inequalities bounds**

We show how \( \delta_t(i | k_p < i_p) \) can be bounded, with the final bounding illustrated in Figure 9. There are four bounds to consider: upper and lower bounds for each of the two cases \( k_p = a_1^1(i) \) (the center being replaced is
Algorithm 6 Simple approach (level 0) with clarans for computing $a_{t+1}^{12}(i)$

1: // If the center which moves is nearest or second nearest, complete update required
2: if $a_1^i(i) = k_p$ or $a_2^i(i) = k_p$ then
3: Get $\text{dist}(x(i), x(c_{t+1}(k)))$ for all $k \in \{1, \ldots, K\}$
4: Use above $k$ distances to set $a_{t+1}^{12}(i)$
5: else
6: // $d_1^i(i)$ and $d_2^i(i)$ are still valid distances, so need only check new candidate center $k_p$
7: $d \leftarrow \text{dist}(x(i), x(i_p))$
8: Use the fact that $\{d_{t+1}^1(i), d_{t+1}^2(i)\} \subset \{d_1^i(i), d_2^i(i), d\}$ to set $a_{12}^i(i)$
9: end if

the center of element $i$) and $k_p \neq a_1^i(i)$ (the center being replaced is not the center of element $i$). We will derive a lower bound for the two cases simultaneously, thus we will derive 3 bounds. First, consider the upper bound for the case $k_p \neq a_1^i(i)$,

$$\delta_t(i \mid k_p \triangleleft i_p) = \min_{i' \in C \setminus \{c_i(k_p)\} : i_p} \psi(\text{dist}(x(i), x(i'))) - \psi(d_t^i(i)),$$

and thus we have

$$k_p \neq a_1^i(i) \implies \delta_t(i \mid k_p \triangleleft i_p) \leq 0. \quad (8)$$

Implication [8] simply states the obvious fact that the energy of element $i$ cannot increase when a center other than that of cluster $a_1^i(i)$ is replaced. The other upper bound case to consider is $k_p = a_1^i(i)$, which is similar,

$$\delta_t(i \mid k_p \triangleleft i_p) = \min_{i' \in C \setminus \{c_i(k_p)\} : i_p} \psi(\text{dist}(x(i), x(i'))) - \psi(d_t^i(i)),$$

and thus we have

$$k_p = a_1^i(i) \implies \delta_t(i \mid k_p \triangleleft i_p) \leq M_t(i). \quad (9)$$

Implication [9] simply states the energy of element $i$ cannot increase by more than the maximum margin in the cluster of $i$ when it is the center of cluster $a_1^i(i)$ which is replaced. We now consider lower bounding $\delta_t(i \mid k_p \triangleleft i_p)$ for both the cases $a_1^i(i) = k_p$ and $a_1^i(i) \neq k_p$ simultaneously. We choose to bound them simultaneously as doing so separately arrives at the same bound,

$$\delta_t(i \mid k_p \triangleleft i_p) = \min_{i' \in C \setminus \{c_i(k_p)\} : i_p} \psi(\text{dist}(x(i), x(i'))) - \psi(d_t^i(i)),$$

and $d_p(k)$ denote the distance between the elements in the proposed swap,

$$d_p(k) = \text{dist}(x(c_t(k)), x(i_p)).$$
The triangle inequality guarantees that,
\[ \text{dist}(x(i), x(i_p)) \geq \begin{cases} 0 & \text{if } d_p(a_1^i(i)) \leq D_1^1(a_1^i(i)), \\ d_p(a_1^i(i)) - D_1^1(a_1^i(i)) & \text{if } D_1^1(a_1^i(i)) < d_p(a_1^i(i)). \end{cases} \]  
(11)

Using (11) in (10) we obtain,
\[ \delta_t(i \mid k_p < i_p) \geq \begin{cases} -\psi(D_1^1(a_1^i(i))) & \text{if } d_p(a_1^i(i)) \leq D_1^1(a_1^i(i)) \\ \psi(d_p(a_1^i(i)) - D_1^1(a_1^i(i))) - \psi(D_1^1(a_1^i(i))) & \text{if } D_1^1(a_1^i(i)) < d_p(a_1^i(i)) \leq 2D_1^1(a_1^i(i)) \\ 0 & \text{if } 2D_1^1(a_1^i(i)) < d_p(a_1^i(i)). \end{cases} \]  
(12)

These are the lower bounds illustrated in Figure 9. Define \( \Delta_t(k \mid k_p < i_p) \) to be the average change in energy for cluster \( k \) resulting from a proposed swap, that is,
\[ \Delta_t(k \mid k_p < i_p) = \frac{1}{N_t(k)} \sum_{i: a_1^i(i) = k_p, k_p < i_p} \delta_t(i \mid k_p < i_p). \]

Let the average of the change in energy over all data resulting from a proposed swap be \( \Delta_t(k_p < i_p) \), that is
\[ \Delta_t(k_p < i_p) = \sum_k p_t(k) \Delta_t(k \mid k_p < i_p). \]

One can show that for \( k = k_p \),
\[ d_p(k_p) \geq D_1^1(k_p) + D_1^2(k_p) \iff \Delta_t(k \mid k_p < i_p) = M_t^*(k_p). \]  
(13)

The equality (13) corresponds to a case where the proposed center \( x(i_p) \) is further from every point in cluster \( k_p \) than is the second nearest center, in which case the increase in energy of cluster \( k_p \) is simply the sum of margins. It corresponds to the solid red horizontal line in Figure 9 left.

**SM-D.2.2 Level 1 proposal evaluation accelerations**

What we wish to evaluate when considering a proposal is the mean change in energy, that is,
\[ \frac{1}{N} \sum_{i=1}^{N} \delta_t(i \mid k_p < i_p) = \frac{1}{N} \left( \sum_{k:k_p \neq k} \sum_{i:a_1^i(i) = k} \delta_t(i \mid k_p < i_p) + \sum_{i:a_1^i(i) = k_p} \delta_t(i \mid k_p < i_p) \right). \]  
(14)

Where in (14) we define \( \Delta_t^-(k_p < i_p) \) as,
\[ \Delta_t^-(k_p < i_p) = \frac{1}{N - N_t(k_p)} \sum_{k:k_p \neq k} \sum_{i:a_1^i(i) = k} \delta_t(i \mid k_p < i_p). \]

From SM-D.2.1 we have the result, corresponding to the solid line in Figure 9 that
\[ a_1^i(i) = k \land k \neq k_p \land \text{dist}(x(c_t(k)), x(i_p)) \geq 2D_1^1(k) \implies \delta_t(i \mid k_p < i_p) = 0. \]  
(15)
We use this result to eliminate entire clusters in the proposal evaluation step: a cluster \( k \) whose center lies sufficiently far from \( x(\i_p) \) will not contribute, as long as \( k \neq k_p \),

\[
\Delta_i^-(k_p < i_p) = \frac{1}{N - N_t(k_p)} \sum_{k:k \neq k_p} \sum_{i:a_t^+(i) = k} \delta_i(i \mid k_p < i_p).
\]

Implication \( 13 \) corresponding to the solid line in Figure 9, left, can be used in the case \( k = k_p \) to rapidly obtain the second term in \( 14 \) if \( \text{dist}(x(\i_t(k_p)), x(\i_p)) \geq D^1_t(k) + D^2_t(k) \).

The level 1 techniques for obtaining whole cluster sums require the distances from the second term in (14) if \( \text{dist}(x(\i_t(k_p)), x(\i_p)) \geq D^1_t(k) + D^2_t(k) \).

The level 1 techniques for accelerating the proposal are presented in Alg. 7.

A second layer of element-wise triangle inequality tests is included for the case where the test on an entire cluster fails. Although in Section SM-D.2.4 (level 2) we show how even these distance calculations can sometimes be avoided. A second layer of element-wise triangle inequality tests is included for the case where the test on an entire cluster fails.

These level 1 techniques for accelerating the proposal are presented in Alg. 7.

**Algorithm 7 CLARANS-1-EVAL**: proposal evaluation using level 1 accelerations. We call subroutines for processing the cluster \( k_p \) (CLARANS-12-EVAL-P) and all other clusters (CLARANS-1-EVAL-N-P). The expected complexity for the full evaluation is \( O(d(K + N/K)) \). The expected complexity for CLARANS-12-EVAL-P assumes that the probability that cluster \( k_p \) is not processed using \( 13 \) is \( O(1/K) \).

1: // Set distances from proposed center \( x(\i_p) \) to all current centers \( \mathcal{C}_t \) \( \triangleright O(dK) \)
2: for \( k \in \{1, \ldots, K\} \) do
3: \( d_c(k) \leftarrow \text{dist}(x(\i_p), x(\i_t(k))) \)
4: end for
5: \( d_{pp} \leftarrow d_c(k_p) \)
6: // Process cluster \( k_p \) \( \triangleright O(dN/K^2) \)
7: CLARANS-12-EVAL-P()
8: // Process all other clusters \( \triangleright O(dN/K) \)
9: CLARANS-1-EVAL-N-P()

**Algorithm 8 CLARANS-12-EVAL-P**: adding the contribution of cluster \( k_p \) to \( \Delta_t(k_p < i_p) \). The key inequality here is \( 13 \), which states that if \( \i_p \) is sufficiently far from the center of cluster \( k_p \), then elements in cluster \( k_p \) will go to their current second nearest center if the center of \( k_p \) is removed.

1: // Try to use \( 13 \) to quickly process cluster \( k_p \)
2: if \( d_{pp} \geq D^1_t(k_p) + D^2_t(k_p) \) then
3: \( \Delta_t(k_p < i_p) = \Delta_t(k_p < i_p) + p_t(k_p)M_t^p(k_p) \)
4: else
5: // Test \( 13 \) failed, enter element-wise loop for cluster \( k_p \)
6: for \( i \in \{i' : a_t^+(i') = k_p \} \) do
7: // Try tighter element-wise version of \( 13 \) to prevent computing a distance
8: if \( d_{pp} \geq d^1_t(i) + d^2_t(i) \) then
9: \( \Delta_t(k_p < i_p) = \Delta_t(k_p < i_p) + m_t(i)/N \)
10: else
11: // Test failed, need to compute distance
12: \( d \leftarrow \text{dist}(x(\i_p), x(i)) \)
13: \( \Delta_t(k_p < i_p) = \Delta_t(k_p < i_p) + \min(d, m_t(i))/N \)
14: end if
15: end for
16: end if

**SM-D.2.3 Level 1 cluster update accelerations**

If a proposal is accepted, the standard CLARANS uses Alg. 6 to obtain \( a^{12}d_{t+1}^2(i) \), where every element \( i \) requires at least 1 distance calculation, with those elements for which cluster \( k_p \) is the nearest or second nearest at \( t \)
Algorithm 9 CLARANS-1-EVAL-N-P: adding contributions of all clusters $k \neq k_p$ to $\Delta_t(k_p < i_p)$. The key inequality used is (15), which states that if the distance between $x(i_p)$ and the center of cluster $k$ is large relative to the distance from the center of cluster $k$ to its most distant member, then there is no change in energy in cluster $k$.

1: for $k \in \{1, \ldots, K\} \setminus \{k_p\}$ do
2: // Try to use (15) to quickly process cluster $k$
3: if $d_c(k) < 2D_1^t(k)$ then
4: // Test (15) failed, enter element-wise loop for cluster $k$
5: for $i \in \{i' : a_1^t(i') = k\}$ do
6: // Try tighter element-wise version of (15) to prevent computing a distance
7: if $d_a(k) < 2d_1^t(i)$ then
8: // Test failed, need to compute distance
9: $d \leftarrow \text{dist}(x(i_p), x(i))$
10: if $d < d_1^t(i)$ then
11: $\Delta_t(k_p < i_p) = \Delta_t(k_p < i_p) + (d - d_1^t(i))/N$
12: end if
13: end if
14: end for
15: end if
16: end for

The inequality to eliminate an entire cluster is,

$$
\min(\text{dist}(x(c_t(k_p)), x(c_t(k))), \text{dist}(x(i_p), x(c_t(k)))) > D_1^t(k) + D_2^t(k)
$$

$$
\Rightarrow \text{no change in cluster } k.\tag{16}
$$

While the inequality used to eliminate the distance calculation for a single sample is,

$$
\min(\text{dist}(x(c_t(k_p)), x(c_t(k))), \text{dist}(x(i_p), x(c_t(k)))) > d_1^t(i) + d_2^t(i)
$$

$$
\Rightarrow \text{no change for sample } i.\tag{17}
$$

Note that the inequalities need to be strict, ‘$\geq$’ would not work. The test (16) is illustrated in Figure 10, left. These bound tests are used in Alg. (10). The time required to update cluster related quantities $(D_1, D_t, M^*)$ is negligible as compared to updating sample assignments, and we do not do anything clever to accelerate it, other than to note that only clusters which fail to be eliminated by (16) potentially require updating.

**SM-D.2.4 Level 2 proposal evaluation accelerations**

We now discuss level 2 accelerations. Note that these accelerations come at the cost of an increase of $O(K^2)$ to the memory footprint. The key idea is to maintain all $K^2$ inter-center distances, denoting by $cc_t(k, k') = \text{dist}(x(c_t(k)), x(c_t(k'))) \text{dist}(x(c_t(k)), x(c_t(k')))$ the distance between centers of clusters $k$ and $k'$. At level 1, all distances $\text{dist}(x(i_p), x(c_t(k)))$ for $k \in \{1, \ldots, K\}$ are computed up-front for proposal evaluation, but here at level 2 we use

$$
\text{dist}(x(i_p), x(c_t(k))) \geq cc_t(a_1^t(i_p), k) - d_1^t(i_p),\tag{18}
$$

to eliminate the need for certain of these distances. Combining (18) with (15) gives,

$$
a_1^t(i) = k \land k \neq k_p \land cc_t(a_1^t(i_p), k) - d_1^t(i_p) \geq 2D_1^t(k) \Rightarrow \delta_t(i \mid k_p < i_p) = 0.\tag{19}
$$

**SM-D.2.5 Level 2 cluster update accelerations**

The only acceleration added at level 2 for the cluster update is for the case $k_p \in \{a_1^t(i), a_2^t(i)\}$, where at level 1, $a^2d_{t+1}^2$ is set from scratch, requiring all $K$ distances to centers to be computed. At level 2, we use $cc_t$ to
Algorithm 10 CLARANS-1-UPDATE : cluster update using level 1 accelerations. Inequalities \[16\] and \[17\] are used to accelerate the updating of \(a^1(i)\) for \(i : a^1(i) \neq k_p\). Essentially these inequalities say that if neither the old center of \(k_p\) nor its new center \(x(i_p)\) are near to an element (or all elements in a cluster), then the nearest and second element of that element (or all elements on a cluster) will not change.

1: // Set distance from centers to the nearer of new and old cluster center \(k_p\) \(\Rightarrow O(dK)\)
2: for \(k \in \{1, \ldots, K\}\) do
3: \(d_.(k) \leftarrow \min(\text{dist}(x(c_i(k_p)), x(c_i(k))), \text{dist}(x(i_p), x(c_i(k))))\)
4: \((= \min(\text{dist}(x(c_i(k_p)), x(c_i(k))), \text{dist}(x(c_{i+1}(k_p)), x(c_i(k))))))\)
5: end for
6: // Process elements in cluster \(k_p\) from scratch
7: for \(i \in \{i' : a^1(i') = k_p\}\) do
8: Obtain \(a^1D_{t+1}(i)\) from scratch
9: end for
10: // Process all other clusters
11: for \(k \in \{1, \ldots, K\} \setminus \{k_p\}\) do
12: // Try to use \[16\] to quickly process cluster \(k\)
13: if \(d_.(k) \leq D^1_k + D^2_k\) then
14: for \(i \in \{i' : a^1(i') = k\}\) do
15: // Try to use \[17\] to quickly process element \(i\)
16: if \(d_.(k) \leq d^1_k + d^2_k\) then
17: if \(a^1(i) = k_p\) then
18: Obtain \(a^1D_{t+1}(i)\) from scratch
19: else
20: \(d \leftarrow \text{dist}(x(i), x(i_p))\)
21: Use \(\{d^1_{t+1}(i), d^2_{t+1}(i)\} \subset \{d^1_{t}(i), d^2_{t}(i), d\}\) as in \[13\]
22: end if
23: else
24: \(a^1D_{t+1}(i) \leftarrow a^1D_{t}(i)\)
25: end if
26: end for
27: end if
28: end for
29: Update cluster statistics for \(t + 1\) where necessary
Algorithm 11 CLARANS-2-EVAL-N-P : add contribution of all clusters \( k \neq k_p \) to \( \Delta_t(k_p < i_p) \). In addition to the bound tests used at level 1, inequality (19) is used to test if a center-center distance needs to be calculated.

1: \textbf{for} \( k \in \{1, \ldots, K\} \setminus \{k_p\} \ \textbf{do} \\
2: \null \quad \text{// Try to use (19) to quickly process cluster } k \\
3: \null \quad \textbf{if } cc_t(a_1^t(i_p), k) - 2D_1^t(k) < d_1^t(i_p) \ \textbf{then} \\
4: \null \quad \null \quad \text{// Test (19) failed, computing } d_c(k) \text{ and resorting to level 1 accelerations...} \\
5: \null \quad \null \quad d_{pk} \leftarrow \text{dist}(x(i_p), x(c_t(k))) \\
6: \null \quad \null \quad \textbf{if } d_{pk} < 2D_1^t(k) \ \textbf{then} \\
7: \null \quad \null \quad \null \quad \text{// Test (13) failed, enter element-wise loop for cluster } k \\
8: \null \quad \null \quad \null \quad \textbf{for } i \in \{i' : a_1^t(t') = k\} \ \textbf{do} \\
9: \null \quad \null \quad \null \quad \quad \text{// Try tighter element-wise version of (13) to prevent computing a distance} \\
10: \null \quad \null \quad \null \quad \quad \textbf{if } d_{pk} < 2d_1^t(i) \ \textbf{then} \\
11: \null \quad \null \quad \null \quad \quad \quad \text{// Test failed, need to compute distance} \\
12: \null \quad \null \quad \null \quad \quad \quad d \leftarrow \text{dist}(x(i_p), x(i)) \\
13: \null \quad \null \quad \null \quad \quad \quad \textbf{if } d < d_1^t(i) \ \textbf{then} \\
14: \null \quad \null \quad \null \quad \quad \quad \quad \Delta_t(k_p < i_p) \leftarrow \Delta_t(k_p < i_p) + (d - d_1^t(i))/N \\
15: \null \quad \null \quad \null \quad \quad \textbf{end if} \\
16: \null \quad \null \quad \null \quad \textbf{end if} \\
17: \null \quad \null \quad \textbf{end for} \\
18: \null \quad \textbf{end if} \\
19: \textbf{end if} \\
20: \textbf{end for}

Algorithm 12 CLARANS-2-EVAL : Proposal evaluation using level 2 accelerations. Unlike at level 1, not all distances from \( x(i_p) \) to centers need to be computed up front.

1: \( d_{pp} \leftarrow \text{dist}(x(i_p), x(c_t(k_p))) \) \quad \triangleright O(dN/K^2) \\
2: \null \quad \text{// Process cluster } k_p \\
3: \text{CLARANS-12-EVAL-P()} \\
4: \null \quad \text{// Process all other clusters} \quad \triangleright O(dN/K) \\
5: \text{CLARANS-2-EVAL-N-P()}
eliminate certain of these distances using \textsc{WarmStart}, which takes in the distances to 2 of the $K$ centers and uses the larger of these as a threshold beyond which any distance to a center can be ignored.

\textbf{Algorithm 13 CLARANS-2-UPDATE} : update using level 2 accelerations. The only addition to level 1 accelerations is the use of \textsc{WarmStart} to avoid computing all $k$ sample-center distances for elements whose nearest or second nearest is $k_p$.

1: For $k \in \{1, \ldots, K\} \setminus \{k_p\}$: compute $dist(x(i_p), x(c_t(k)))$ (= $dist(x(c_{t+1}(k_p)), x(c_{t+1}(k)))$ and set $cc_{t+1}$ accordingly (in practice we don’t need to store $cc_t$ and $cc_{t+1}$ simultaneously as they are very similar).

2: for $k \in \{1, \ldots, K\}$ do
3: \hspace{1em} $d_c(k) \leftarrow \min(cc_t(k, k), cc_{t+1}(k, k))$
4: end for
5: // Process elements in cluster $k_p$ from scratch
6: for $i \in \{i' : a^1_t(i') = k_p\}$ do
7: \hspace{1em} $d \leftarrow dist(x(i), c_{t+1}(k_p))$
8: \hspace{1em} Obtain $a^{12}_t d^{12}_{t+1}(i)$, using \textsc{WarmStart} with $d$ and $d^2_t(i)$.
9: end for
10: // Process all other clusters
11: for $k \in \{1, \ldots, K\} \setminus \{k_p\}$ do
12: \hspace{1em} // Try to use [10] to quickly process cluster $k$
13: \hspace{2em} if $d_c(k) \leq D^1_t(k) + D^2_t(k)$ then
14: \hspace{3em} for $i \in \{i' : a^1_t(i') = k\}$ do
15: \hspace{4em} \hspace{1em} // Try to use [17] to quickly process element $i$
16: \hspace{5em} \hspace{2em} if $d_a(k) \leq d^1_t(k) + d^2_t(k)$ then
17: \hspace{6em} \hspace{3em} $d \leftarrow dist(x(i), c_{t+1}(k_p))$
18: \hspace{6em} \hspace{3em} if $a^2_t(i) = k_p$ then
19: \hspace{7em} \hspace{4em} Obtain $a^{12}_t d^{12}_{t+1}(i)$, using \textsc{WarmStart} with $d$ and $d^1_t(i)$.
20: \hspace{6em} else
21: \hspace{7em} \hspace{4em} Use $\{d^1_{t+1}(i), d^2_{t+1}(i)\} \subset \{d^1_t(i), d^2_t(i), d\}$ as in [6]
22: \hspace{6em} end if
23: \hspace{5em} else
24: \hspace{6em} $a^{12}_td^{12}_{t+1}(i) \leftarrow a^{12}_t d^{12}_{t+1}(i)$
25: \hspace{6em} end if
26: \hspace{3em} end if
27: end for
28: end for
29: Update cluster statistics for $t+1$ where necessary

\textbf{SM-D.3 Level 3}

At levels 1 and 2, we showed how \texttt{clarans} can be accelerated using the triangle inequality. The accelerations were exact, in the sense that for a given initialisation, the clustering obtained using \texttt{clarans} is unchanged whether or not one uses the triangle inequality.

Here at level 3 we diverge from exact acceleration. In particular, we will occasionally reject good proposals. However, the proposals which are accepted are still only going to be good ones, so that the energy strictly decreases. In this sense, it is not like stochastic gradient descent, where the loss is allowed to increase.

The idea is to the following. Given a proposal swap : replace the center of cluster $k_p$ with the element indexed by $i_p$, use a small sample of data to estimate the quality of the swap, and if the estimate is bad (increase in energy) then immediately abandon the proposal and generate a new proposal. If the estimate is good, obtain a more accurate estimate using more ($2 \times$) elements. Repeat this until all the elements have been used and the exact energy under the proposed swap is known: if the exact energy is lower, implement the swap otherwise reject it.

The level 1 and 2 accelerations can be used in parallel with the acceleration here. The elements sub sampled at level 3 are chosen to belong to clusters which are not eliminated using level 1 and 2 cluster-wise bound tests.
Suppose that there are $\hat{K}$ clusters which are not eliminated at level 2, we choose the number of elements chosen in the smallest sub sample to be $30\hat{K}$. Thereafter the number of elements used to estimate the post-swap energy doubles.

Let the number of elements in the $\hat{K}$ non-eliminated clusters be $n_A$ and the number sampled be $n_S$, so that $n_S = 30\hat{K}$. Supposing that $n_A/n_S$ is a power of 2. Then, one can show that the probability that a good swap is rejected is bounded above by $1 - n_S/n_A$. Consider the case $n_A/n_S = 2$, so that the sample is exactly half of the total. Suppose that the swap is good. Then, if the sum over the sample is positive, the sum over its complement must be negative, as the total sum is negative. Thus there at least as many ways to draw $n_S$ samples whose sum is negative as positive.

If $n_A/n_S = 4$, then consider what happens if one randomly assign another quarter to the sample. With probability one half the sum is negative, thus by the same reasoning with probability at least $1/2 \times 1/2 = 1/4$ the sum over the original $n_S$ samples is negative.

**Algorithm 14** Level 3: Schemata of using sub sampling to quickly eliminate unpromising proposals without computing an exact energy. This allows for more rapid proposal evaluation.

1. Determine which clusters are not eliminated at level 2, define to be $U$.
2. $\hat{K} \leftarrow |U|$.
3. $N_T \leftarrow \sum_{k \in U} N_t(k)$
4. $N_S \leftarrow 30\hat{K}$
5. $S \leftarrow$ uniform sample of indices of size $N_S$ from clusters $U$
6. $\hat{\Delta}_t(k_p < i_p) \leftarrow \infty$
7. while $N_S < N_T$ and $\hat{\Delta}_t(k_p < i_p) < 0$ do
   8. \[ \hat{\Delta}_t(k_p < i_p) \leftarrow \frac{1}{N_S} \sum_{i \in S} \delta_t(i | k_p < i_p) \]
   9. $N_S \leftarrow \min(N_T, 2N_S)$
10. $S \leftarrow S \cup$ uniform sample of indices so that $|S| = N_S$.
11. end while
12. if $N_S < N_T$ then return reject
13. else
14. Compute $\Delta_t(k_p < i_p)$
15. if $\Delta_t(k_p < i_p) < 0$ then return accept
16. else return reject
17. end if
18. end if

**SM-E** Links to datasets

The rcv1 dataset: [http://jmlr.csail.mit.edu/papers/volume5/lewis04a/a13-vector-files/lyrl2004_vectors_train.dat.gz](http://jmlr.csail.mit.edu/papers/volume5/lewis04a/a13-vector-files/lyrl2004_vectors_train.dat.gz)

Chromosone 10: [http://ftp.ensembl.org/pub/release-77/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.chromosome.10.fa](http://ftp.ensembl.org/pub/release-77/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.chromosome.10.fa)

English word list: [https://github.com/dwyl/english-words.git](https://github.com/dwyl/english-words.git)

**SM-F** Local minima formalism

**Theorem SM-F.1.** A local minimum of clarans is always a local minimum of vik. However, there exist local minima of vik which are not local minima of clarans.

**Proof.** The second statement is proven by the existence of example in the Introduction. For the first statement, suppose that a configuration is a local minimum of clarans, so that none of the $K(N - K)$ possible swaps results in a decrease in energy. Then, each center must be the medoid of its cluster, as otherwise we could swap the center with the medoid and obtain an energy reduction. Therefore the configuration is a minimum of clarans. $\Box$
SM-G  Efficient Levenshtein distance calculation

The algorithm we have developed relies heavily on the triangle inequality to eliminate distances. However, it is also possible to abort distance calculations once started if they exceed a certain threshold of interest. When we wish to determine the 2 nearest centers to a sample for example, we can abort a distance calculation as soon as we know the distance being calculated is greater than at least two other centers.

For vectorial data, this generally does not result in significant gains. However, when computing the Levenshtein distance it can help enormously. Indeed, for a sequence of length $l$, without a threshold on the distance the computation cost of the distance is $O(l^2)$. With a threshold $m$ it becomes $(lm)$. Essentially, only the diagonal of is searched while running the dynamic Needleman-Wunsch algorithm. We use this idea at all levels of acceleration.

SM-H  A Comment on Similarites used in Bioinformatics

A very popular similarity measure in bioinformatics is that of Smith-Waterman. The idea is that similarity should be computed based on the most similar regions of sequences, and not on the entire sequences. Consider for example, the sequences $a = 123123898989$, $b = 454545898989$, $c = 123123012012$. According to Smith-Waterman, these should have $\text{sim}(a, b) = \text{sim}(a, c) \gg \text{sim}(b, c)$. This is not possible to turn into a proper distance, as one would need $\text{dist}(a, b) = \text{dist}(a, c) \ll \text{dist}(b, c)$, which is going to break the triangle inequality. Thus, the triangle inequality accelerations introduced cannot be applied to similarities of the Smith-Waterman type.

SM-I  Acceleration results : levels 1,2,3

Todo (12/09/2016). In this section I will describe and present experiments comparing clarans at levels 0,1,2 and 3. There will be comparisons of performance with flag ‘patient’ active and deactve, and with ‘rooted’ true and false.