Novel Algorithms For Clustering

THÈSE N^{O} 8375 (2018)

PRÉSENTÉE LE 15 FÉVRIER 2018
À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
LABORATOIRE DE L'IDIAP
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

James Peter NEWLING

acceptée sur proposition du jury:

Prof. R. West, président du jury
Dr F. Fleuret, directeur de thèse
Prof. S. Marchand-Maillet, rapporteur
Prof. R. Sznitman, rapporteur
Prof. M. Jaggi, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2018

Acknowledgements

This dissertation would not have been possible without the support of several people, to whom I am extremely grateful.

First and foremost, to my PhD supervisor *François Fleuret*, for his unwavering support. François is an inspiring computer scientist, committed team leader, and kind mentor.

To my thesis committee for their constructive feedback, and interest in my research.

To the EPFL Doctoral School. Completing my dissertation was smoother than I had anticipated, in large part thanks to their support and planning.

To the administrative team at the Idiap Research Institute, committed to making a productive research environment. My 4+ years in Switzerland have been remarkably hassle free, and I have been able to focus almost entirely on interesting scientific questions.

To past and present members of ML Group; *Charles Dubout*, *Leonidas Lefakis*, *Olivier Canévet*, *Tatjana Chavdarova*, *Angelos Katharopoulos*, *Suraj Srinivas* and *Alain Rossier*. Our discussions and sharing of research ideas has been stimulating and rewarding.

I am glad to have started at the same time as *Cijo Jose*, who kept us all focused with his frequent ‘What is the most important problem in machine learning?’ question. We may not have a found definitive answer, but I think we have made progress.

While my internship in the Radeon Open Compute team at AMD was not directly related to this thesis, thank you to everyone in Austin who taught me about GPU programming, and expanded my knowledge of the field.

To everyone at Idiap, past and present, for making working and living in Martigny so enjoyable. There are too many people to thank here for all the fun times.

To all the kind people in Lausanne and Martigny who I have been fortunate enough to meet and interact with, at CAS, at the pool, elsewhere. You have all contributed to making this thesis possible.

Finally, to the Hasler Foundation for funding this research through the grant 13018.

Martigny, 18 January 2018 jnewling@idiap

Abstract

Clustering is a method for discovering structure in data, widely used across many scientific disciplines. The two main clustering problems this dissertation considers are K-means and K-medoids. These are NP-hard problems in the number of samples and clusters, and both have well studied heuristic approximation algorithms. An example is Lloyd's algorithm for K-means, which is so widely used that it has become synonymous with the problem it attempts to solve.

A large part of this dissertation is about accelerating Lloyd's algorithm, and its mini-batch and K-medoids variants. The basic tool used to achieve these accelerations is the triangle inequality, which can be applied in a multitude of ways to eliminate costly distance calculations between data samples, as well as to reduce the number of comparisons of these distances.

The first effective use of the triangle inequality to accelerate K-means was by Elkan [2003], with novel refinements appearing more recently in Hamerly [2010]. In Chapter 1 we extend these approaches. First, we show that by using centers stored from previous iterations, one can greatly reduce the number of sample-center distance computations, with substantial improvements in algorithm execution time. We then present an improvement over previous triangle inequality based algorithms for low-dimensions, which uses inter-center distances in a novel way.

Chapter 2 considers the use of the triangle inequality to accelerate the mini-batch variant of Lloyd's algorithm [Sculley, 2010]. The main difficulty of incorporating triangle inequality bounding in this setting is that clusters can move significantly during the iterations in which a sample is unused, which makes triangle inequality bounding ineffective. We propose a modified sampling scheme to reduce the length of these periods of dormancy, and present an algorithm which achieves an order of magnitude acceleration over the standard mini-batch algorithm.

We then turn attention to the K-medoids problem. In Chapter 3 we focus on the specific problem of determining the medoid of a set. With N samples in $\mathbb{R}^d$, we present a simple algorithm of complexity $O(N^{3/2})$ to determine the medoid. It is the first sub-quadratic algorithm for this problem when $d > 1$. The algorithm makes use of the triangle inequality to eliminate all but $O(N^{1/2})$ samples as potential medoid candidates.

Finally, in Chapter 4 we compare different K-medoids algorithms, and find that `clarans` [Ng and Han, 1994], which iteratively replaces randomly selected centers with non-centers, avoids the local minima of other popular K-medoids algorithms. This motivates the use of `clarans` for initializing Lloyd's algorithm for K-means, which results in improved final energies as compared to K-means++ seeding. We use the triangle inequality to offset the increased computation required by `clarans`.

Keywords: clustering, triangle inequality, k-means, k-medoids, medoid, mini-batch

Résumé

Le partitionnement de données est une méthode d'apprentissage non-supervisée qui vise à diviser un ensemble de données en des groupes dans lesquels les données partagent des caractéristiques communes. Les deux problèmes de partitionnement principaux considérés dans cette thèse sont le K-means (K-moyennes) et le K-medoids (K-médoïdes). Ce sont des problèmes NP-difficiles dans le nombre de données et de partitions, pour lesquels il existe des algorithmes heuristiques bien étudiés. Le plus connu de ces algorithmes est celui de Lloyd, que l'on appelle parfois simplement l'algorithme K-means.

Une grande partie de cette thèse porte sur l'accélération de l'algorithme de Lloyd et ses variantes. L'outil de base pour réaliser ces accélérations est l'inégalité triangulaire, qui peut être utilisée de multiples façons pour éliminer des calculs de distance coûteux entre les données, ainsi que pour réduire le nombre de comparaisons de ces distances.

La première utilisation efficace de l'inégalité triangulaire pour accélérer l'algorithme de Lloyd est due à Elkan [2003], avec des améliorations proposées plus récemment par Hamerly [2010] et Ding et al. [2015]. Dans le chapitre 1 nous proposons des améliorations supplémentaires. Tout d'abord, nous montrons qu'en utilisant des centres estimés lors d'itérations précédentes, il est possible de réduire considérablement le nombre de calculs de distances. Ensuite, nous présentons un algorithme qui utilise les distances entre les centres d'une manière nouvelle, qui permet de grandes accélérations en basses dimensions par rapport aux algorithmes existants.

Le chapitre 2 considère l'utilisation de l'inégalité triangulaire pour accélérer la variante *mini-batch* de l'algorithme de Lloyd. La difficulté principale de l'incorporation de l'inégalité triangulaire ici est que les centres peuvent se déplacer de manière significative au cours des itérations dans lesquelles une donnée est inutilisée, ce qui rend inefficace la technique de délimination. Nous proposons une méthode d'échantillonnage modifiée pour réduire cette période d'inactivité, et nous présentons un algorithme avec une accélération d'un ordre de grandeur par rapport à l'algorithme original du mini-batch.

Nous nous concentrons ensuite sur le problème K-medoids. Dans le chapitre 3, nous abordons le problème spécifique de la détermination du medoid d'un ensemble de N données. Nous présentons un algorithme simple qui est de complexité $O(N^{3/2})$ dans $\mathbb{R}^d$, le premier algorithme sous-quadratique pour $d > 1$. L'algorithme utilise l'inégalité triangulaire pour éliminer tous les données sauf $O(N^{1/2})$ comme candidats medoids.

Enfin, dans le chapitre 4, nous comparons différents algorithmes de K-medoids, et nous montrons que `clarans` [Ng and Han, 1994], qui échange aléatoirement des centres et des non-centres, évite les minima locaux d'autres algorithmes. Cela nous encourage à utiliser `clarans` pour initialiser l'algorithme de Lloyd pour le K-means, ce qui donne des énergies finales améliorées par rapport à K-means++ [Arthur and Vassilvitskii, 2007]. Nous utilisons l'inégalité triangulaire pour réduire la complexité de calcul de `clarans` dans ce contexte.

Mots clés : partitionnement, inégalité triangulaire, k-moyennes, k-médoïdes, médoid, mini-batch

Contents

Acknowledgements **iii**

Abstract **v**

List of figures **xii**

List of tables **xiii**

List of algorithms **xv**

Introduction **1**
- 0.1 Energy minimization clustering problems 2
- 0.2 Dissertation outline and contributions 3

1 Fast K-means with Accurate Bounds **5**
- 1.1 Chapter introduction 5
  - 1.1.1 Approximate K-means 5
  - 1.1.2 Accelerated exact K-means 6
  - 1.1.3 Our contribution 6
- 1.2 Notation and baselines 7
  - 1.2.1 Standard algorithm (`sta`) 7
  - 1.2.2 Simplified Elkan's algorithm (`selk`) 7
  - 1.2.3 Elkan's algorithm (`elk`) 8
  - 1.2.4 Hamerly's algorithm (`ham`) 9
  - 1.2.5 Annular algorithm (`ann`) 10
  - 1.2.6 Simplified Yinyang (`syin`) and Yinyang (`yin`) algorithms 10
- 1.3 Contributions and new algorithms 11
  - 1.3.1 Exponion algorithm (`exp`) 11
  - 1.3.2 Improving bounds (sn to ns) 13
  - 1.3.3 Simplified Elkan's algorithm-ns (`selk-ns`) 14
  - 1.3.4 Changing bounds for other algorithms 15
- 1.4 Experiments and results 15
  - 1.4.1 Single core experiments 16
  - 1.4.2 Multicore experiments 20

1.5 Chapter conclusion and future work 21

2 Nested Mini-Batch K-means 23
2.1 Chapter introduction 23
2.1.1 Previous approximate K-means algorithms 23
2.1.2 Contribution of this chapter 24
2.2 Related works 25
2.2.1 Exact acceleration using the triangle inequality 25
2.2.2 Mini-batch K-means 26
2.3 Nested mini-batch K-means : `nmbatch` 26
2.3.1 Modifying cumulative sums to prevent duplicity 27
2.3.2 Balancing premature fine-tuning and redundancy 27
2.3.3 A note on parallelization 30
2.4 Results 30
2.5 Chapter conclusion and future work 31
2.6 Comparing Baseline Implementations 33

3 A Sub-Quadratic Exact Medoid Algorithm 35
3.1 Chapter introduction 35
3.1.1 Medoid algorithms and our contribution 35
3.1.2 K-medoids algorithms and our contribution 36
3.2 Previous works 37
3.2.1 A related problem: the geometric median 37
3.2.2 Medoid algorithms : `TOPRANK` and `TOPRANK2` 38
3.2.3 K-medoids algorithm : `KMEDS` 39
3.3 Our new medoid algorithm : `trimed` 40
3.3.1 On the assumptions in theorem 3.3.2 42
3.3.2 Sketch of proof of theorem 3.3.2 43
3.4 Accelerated K-medoids algorithm : `trikmeds` 44
3.5 Results 44
3.5.1 Medoid algorithm results 45
3.5.2 K-medoids algorithm results 47
3.6 Chapter conclusion and future work 48

4 K-medoids For K-means Seeding 49
4.1 Chapter introduction 49
4.1.1 K-means initialization 50
4.1.2 Our contribution and chapter summary 51
4.1.3 Other related works 51
4.2 Two K-medoids algorithms 52
4.3 A simple simulation study 53
4.4 Complexity and accelerations 53
4.5 Results 57

4.5.1 Baseline performance . . . 58
4.5.2 `clarans` performance . . . 60
4.6 Chapter conclusion and future works . . . 60

5 Conclusions and Future Works 63

A Appendix for Chapter 1 65
A.1 Proofs . . . 65
A.1.1 Proof of correctness of Elkan's algorithm update . . . 65
A.1.2 Proof of correctness of Elkan's algorithm inter-centroid test . . . 66
A.1.3 Proof of correctness of Annular algorithm test . . . 66
A.1.4 Proof of correctness of Exponion algorithm test . . . 66
A.1.5 Proof that ns upper-bound is tighter than sn upper-bound . . . 67
A.2 Detailed descriptions . . . 68
A.2.1 The inner Yinyang test . . . 68
A.2.2 SMN, MSN, MNS . . . 68
A.3 Results tables . . . 69

B Appendix for Chapter 2 73
B.1 There are more first time visits than revisits in the first epoch . . . 73
B.2 Showing that two expectations are approximately the same . . . 73
B.3 Time-energy curves with various doubling thresholds . . . 75
B.4 On algorithms intermediate to `mbatch` and `nmbatch` . . . 77
B.5 Premature fine-tuning . . . 77

C Appendix for Chapter 3 79
C.1 On the difficulty of the medoid problem . . . 79
C.2 `medlloyd` pseudocode . . . 79
C.3 `RAND`, `TOPRANK` and `TOPRANK2` pseudocode . . . 79
C.3.1 On the number of anchor elements in `TOPRANK` . . . 80
C.3.2 On the parameter α' in `TOPRANK` and `TOPRANK2` . . . 80
C.3.3 On the parameters specific to `TOPRANK2` . . . 80
C.4 On the proof that `TOPRANK` returns the medoid with high probability . . . 83
C.4.1 Probability that the medoid is returned . . . 83
C.5 On the initialization of Park and Jun [2009] . . . 85
C.6 Scaling with α, N, and dimension d . . . 85
C.7 Proof of The Main Theorem . . . 86
C.8 Pseudocode for `trikmeds` . . . 92
C.9 Datasets . . . 97
C.10 Scaling with dimension of `TOPRANK` and `TOPRANK2` . . . 97
C.11 Example where geometric median is a poor approximation of medoid . . . 98
C.12 Miscellaneous . . . 98

D Appendix for Chapter 4 **101**
- D.1 Generalised K-medoids results 101
- D.2 The task 101
- D.3 The `pam` algorithm 104
- D.4 `clarans` In detail with accelerations 104
  - D.4.1 Review of notation and ideas 106
  - D.4.2 Accelerating `clarans` 108
  - D.4.3 Level 3 117
- D.5 Links to datasets 117
- D.6 Local minima formalism 118
- D.7 Efficient Levenshtein distance calculation 118
- D.8 A comment on similarities used in bioinformatics 119
- D.9 Pre-initializing with `km++` 119
- D.10 Comapring the different optimizations levels, and kmlocal 119

Bibliography **127**

Curriculum Vitae **129**

List of Figures

1.1 Comparing sn- and ns- bounds 13

2.1 Centroid based definitions of redundancy and premature fine-tuning . . . 29
2.2 Mean energy on validation data relative to lowest energy 32
2.3 Relative errors on validation data 33

3.1 Using the triangle inequality to eliminate medoid candidates 41
3.2 Illustration of the constants used in Theorem 3.3.2 43
3.3 Comparison of `TOPRANK` and `trimed` on simulated data 45

4.1 Illustrating the importance of good initialization 50
4.2 Illustrating the swap technique 53
4.3 Simulated data in $\mathbf{R}^2$ and solutions with different K-medoids algorithms . 54
4.4 Results on simulated data 54
4.5 The number of consecutive swap proposal rejections 55
4.6 Initialization and final MSEs 58

B.1 Time-energy curves for various ρ 75
B.2 Time-energy curves with bounds disabled 76
B.3 Performace of intermediate algorithms 78

C.1 Number of points computed on simulated data 86
C.2 Summing uniformly distributed cones 87
C.3 Illustrating parameters α, β and ρ 88
C.4 Type 1 elimination 99

D.1 Results on synthetic datasets 103
D.2 Results on real datasets 103
D.3 Illustrating the bounds 110
D.4 Illustrating a bound test 113
D.5 Comparing `km++`+`clarans`+`lloyd` and `km++`+`lloyd` 119
D.6 Comparing the different optimization levels 120
D.7 Sscaling in K of `clarans` performance gain 121

List of Tables

1.1 Datasets used in exact K-Means experiments 16
1.2 Comparison of K-Means implementations 18
1.3 Comparing `yin` and `elk` with simplfied versions 19
1.4 Mean runtimes and number of calculations using low-d algorithms 19
1.5 Counts of times each sn-algorithm is fastest 19
1.6 Measuring the effect of using ns-bounds 20
1.7 Multicore speedups 21

2.1 Comparing implementations of `mbatch` on INFMNIST and RCV1 34

3.1 Comparison of `TOPRANK`, `TOPRANK2` and `trimed` on publicly available data 47
3.2 Relative numbers of distance calculations and final energies 48

4.1 Algorithm complexities at different levels of optimization 56
4.2 Number of distance calculations and time required on simulated data . . . 56
4.3 The datasets used in K-medoids experiments 57
4.4 Summary of results on the 23 datasets 59

A.1 Full names of datasets 69
A.2 Full results with $k = 100$ 70
A.3 Full results with $k = 1000$ 71

C.1 Comparison the initialization schemes from Park and Jun [2009] 85
C.2 Table Of Notation For `trikmeds` 96

D.1 Synthetic datasets used for comparing K-medoids algorithms 102
D.2 Real datasets used for comparing K-medoids algorithms 102

List of Algorithms

1 Cluster assigment, using bounds . . . 25
2 Initialising c, S and v . . . 26
3 Accumulation for sample i . . . 26
4 Mini-batch K-means . . . 26
5 Nested Mini-batch K-means . . . 28
6 The `trimed` algorithm for obtaining set medoid . . . 40
7 two-step iterative `medlloyd` algorithm (vector space, quadratic potential). 52
8 swap-based `clarans` algorithm (vector space, quadratic potential). 52
9 `medlloyd` algorithm . . . 79
10 `RAND` algorithm . . . 81
11 `TOPRANK` algorithm . . . 81
12 `TOPRANK2` algorithm . . . 82
13 `trikmeds` algorithm . . . 92
14 `trikmeds` initialization . . . 93
15 `update-medoids` . . . 94
16 `assign-to-clusters` . . . 95
17 `update-sum-bounds` . . . 96
18 `contiguate` . . . 96
19 The `pam` K-medoids algorithm . . . 104
20 One round of the `clarans` algorithm . . . 107
21 Unoptimized (level 0) algorithm for computing a proposal energy . . . 107
22 Unoptimized (level 0) algorithm for computing `clarans` assignments . . . 108
23 `clarans` level 1 optimization . . . 111
24 `clarans` levels 1 and 2 evaluation of energy of changed cluster . . . 112
25 `clarans` levels 1 and 2 energy evaluation of all other clusters . . . 112
26 `clarans` level 1 optimization for updating sample labels and distances . . 114
27 `clarans` level 2 energy evaluation for non-swapping clusters . . . 115
28 `clarans` levels 2 energy change evaluation . . . 115
29 `clarans` level 2 energy update . . . 116
30 `clarans` level 3 accelerated approximate energy estimation . . . 118

Introduction

The goal of machine learning is for computers to learn without being explicitly programmed. The field is commonly divided into two sub-fields, supervised and unsupervised learning. In the supervised setting, the data which the computer learns from consists of pairs, where each pair is an input and desired output. An example of supervised learning is regression, where model parameters are learned to minimize prediction error. The topic of this thesis, clustering, lies in the field of unsupervised learning, where data consists of only inputs; there are no specified ‘correct’ outputs, or labels.

The goal of clustering is to separate data into meaningful groups. It is often used as a first step in exploring new datasets; for example biologists may cluster gene expressions to discover biological pathways, and retailers may cluster customers for targeted marketing. It is also a commonly used technique in signal processing, where values are quantized to satisfy memory/bandwidth constraints.

Clustering techniques can be divided into two categories, hierarchical and flat clustering techniques. An example of hierarchical clustering is the construction of phylogenetic trees, where different tree depths represent different degrees of genetic relatedness; species, genus, family, order, etc. Flat clustering on the other hand assumes no relationship between clusters. This dissertation deals exclusively with the flat variant, commonly referred to as *partitional* clustering. This nomenclature can be misleading, as it suggests that hierarchical clustering is non-partitional.

Flat clustering techniques can be further sub-divided into those which aim to minimize a energy function, and those which do not. A popular algorithm where there is no energy function minimization is DBSCAN of Ester et al. [1996], which iteratively agglomerates items into clusters based on estimated point density. Such heuristic algorithms will not be considered in this dissertation, which focuses instead on algorithms which attempt to minimize an energy function.

With the energy minimization approach, an energy function is defined over a parameterized set of possible clusterings, and the goal is to minimize this function. The canonical example here is K-means, where the energy function is defined (in a vector space) as the sum over data-points of the squared Euclidean distance to the nearest cluster center.

Another example is spectral clustering of graphs, which attempts to minimize the number of edges between nodes in different clusters. A third example is K-medoids, which generalizes K-means to any loss function but adds the constraint that centers be samples.

Energy minimization clustering problems

Many energy minimization clustering problems such as K-means are NP-hard in the number of samples, and in practice approximation algorithms are used. For K-means, the most widely used is Lloyd's algorithm, which relies on a two-step iterative process. In the *assignment* step, each sample is assigned to the cluster whose center is nearest. In the *update* step, cluster centers are updated to be the mean of their assigned samples. Lloyd's algorithm is generally considered to be fast. However, the linear dependence on the number of clusters, the number of samples and the dimension of the space, means that it requires upwards of a billion floating point operations per round on medium-sized datasets. This, coupled with slow convergence and the fact that several runs are often performed to find improved solutions, can make it slow in practice. Accelerating Lloyd's algorithm will be the focus of Chapter 1 of this dissertation, with special attention paid to the triangle inequality bounding techniques of Elkan [2003].

Lloyd's algorithm is often referred to as the *exact* algorithm, which can lead to confusion as it does not solve the K-means problem exactly. The reason for this name is that there are other algorithms which approximate Lloyd's algorithm. Certain of these rely on a relaxation of the assignment step, for example by only considering certain clusters for each sample, according to some hierarchical ordering [Nister and Stewenius, 2006], or by using an approximate nearest neighbor search, as in Philbin et al. [2007]. Others rely on a relaxation of the update step, for example by using only a subset of data to update centroids [Frahling and Sohler, 2006, Sculley, 2010]. Such relaxations can result in enormous speed-ups for large datasets. These approximate algorithms, in particular the mini-batch K-means algorithm of Sculley [2010] will be the focus of Chapter 2.

Lloyd's algorithm only works in vector spaces, where a 'mean' sample, or centroid, can be computed. To generalize the algorithm to any metric space, one can replace centroids with medoids, the cluster elements whose mean energy with other cluster elements is minimal. The resulting K-medoids algorithm can be applied to graph and sequence data. Computing the medoid of a set has applications beyond clustering. In network analysis, the medoid may represent an influential person in a social network, or the most central station in a rail network. In operations research, the facility location problem requires placing one or several facilities so as to minimize the cost of connecting to clients. A simple algorithm for obtaining the medoid of a set of N elements is to directly compute all inter-element distances, which costs $\Theta(N^2)$. This is in contrast to the computation of a set mean, which is $O(N)$. In Chapter 3 we consider approximate and exact alternatives for computing the medoid of a set, which reduce this quadratic dependency.

Alternatives to Lloyd's algorithm and its K-medoids equivalent alluded to in the previous paragraph have been proposed. The majority of these algorithms do not scale linearly in the number of samples. Examples are the algorithm of Kanungo et al. [2002a] and the `clarans` K-medoids algorithm of Ng and Han [1994]. Lloyd's algorithm is *local*, in that far removed centers and points do not directly influence each other. This property contributes to its tendency to terminate in poor minima if not well initialized. Good initialization is key to guaranteeing that the refinement performed by `lloyd` is done in the vicinity of a good solution, and a topic of active research [Celebi et al., 2013, Bachem et al., 2016]. The `clarans` algorithm is robust to certain local minima of Lloyd's algorithm, indeed its local minima are a subset of those of Lloyd's. Initialization and these alternative clustering algorithms will be the topic of Chapter 4, where we consider using `clarans` for initializing Lloyd's algorithm.

Dissertation outline and contributions

The four chapters in this dissertation are based on four conference proceedings articles [Newling and Fleuret, 2016a,b, 2017a,b]. While all four chapters are about clustering, they are independent of each other and can be read in any order. Each chapter very closely follows its original paper, with certain connections between the works highlighted where relevant. Notation has been changed from the original papers where appropriate.

In Chapter 1, which is based on Newling and Fleuret [2016a], we propose a novel accelerated exact k-means algorithm, which outperforms state-of-the-art low-dimensional algorithm in 18 of 22 experiments, running up to 3× faster. We also propose a general improvement of existing state-of-the-art accelerated exact k-means algorithms through better estimates of the distance bounds used to reduce the number of distance calculations, obtaining speedups in 36 of 44 experiments, of up to 1.8×. We have conducted experiments with our own implementations of existing methods to ensure homogeneous evaluation of performance, and we show that our implementations perform as well or better than existing available implementations. Finally, we propose simplified variants of standard approaches and show that they are faster than their fully-fledged counterparts in 59 of 62 experiments.

In Chapter 2, which is based on Newling and Fleuret [2016b], a new algorithm is proposed which accelerates the mini-batch K-means algorithm of Sculley [2010] by using the distance bounding approach of Elkan [2003]. We argue that, when incorporating distance bounds into a mini-batch algorithm, already used data should preferentially be reused. To this end we propose using *nested* mini-batches, whereby data in a mini-batch at iteration t is automatically reused at iteration $t + 1$. Using nested mini-batches presents two difficulties. The first is that unbalanced use of data can bias estimates, which we resolve by ensuring that each data sample contributes exactly once to centroids. The second is in choosing mini-batch sizes, which we address by balancing premature fine-tuning of centroids with redundancy induced slow-down. Experiments show that the resulting `nmbatch` algorithm is very effective, often arriving within 1% of the empirical minimum 100× earlier than the standard mini-batch algorithm.

In Chapter 3, which is based on Newling and Fleuret [2017a], we present a new algorithm `trimed` for obtaining the *medoid* of a set, that is the element of the set which minimizes the mean distance to all other elements. The algorithm is shown to have, under certain weak assumptions, expected run time $O(N^{\frac{3}{2}})$ in $\mathbb{R}^d$ where N is the set size, making it the first sub-quadratic exact medoid algorithm for $d > 1$. Experiments show that it performs very well on spatial network data, frequently requiring two orders of magnitude fewer distance calculations than state-of-the-art approximate algorithms. As an application, we show how `trimed` can be used as a component in an accelerated K-medoids algorithm, and then how it can be relaxed to obtain further computational gains with only a minor loss in cluster quality.

In Chapter 4, which is based on Newling and Fleuret [2017b], we show experimentally that the algorithm `clarans` of Ng and Han [1994] finds better K-medoids solutions than the Voronoi iteration algorithm of Hastie et al. [2001]. This finding, along with the similarity between the Voronoi iteration algorithm and Lloyd's K-means algorithm, motivates us to use `clarans` as a K-means initializer. We show that `clarans` outperforms other algorithms on 23/23 datasets with a mean decrease over `k-means++` [Arthur and Vassilvitskii, 2007] of 30% for initialization mean squared error (MSE) and 3% for final MSE. We introduce algorithmic improvements to `clarans` which improve its complexity and runtime, making it an extremely viable initialization scheme for large datasets.

Finally, in Chapter 5, we summarize our findings and suggest directions for future investigation.

1 Fast K-means with Accurate Bounds

Chapter introduction

The K-means problem is to compute a set of K centroids (centers) to minimize the sum over data-points of the squared distance to the nearest centroid. As mentioned in the thesis introduction, it is an NP-hard problem for which the most popular approximation algorithm is Lloyd's algorithm, often referred to as *the* K-means algorithm. It has applications in data compression, data classification, density estimation and many other areas, and was recognised in Wu et al. [2008] as one of the top-10 algorithms in data mining.

Recall that Lloyd's algorithm is also called the *exact* K-means algorithm, as there is no approximation in the assignment or update step. Note that Lloyd's algorithm does not state how these steps should be performed, and as such provides a scaffolding on which more elaborate algorithms can be constructed. These more elaborate algorithms, often called *accelerated* exact K-means algorithms, are the primary focus of this chapter. They can be dropped-in wherever Lloyd's algorithm is used.

Approximate K-means

Alternatives to exact K-means have been proposed. Certain of these rely on a relaxation of the assignment step [Nister and Stewenius, 2006, Philbin et al., 2007]. Others rely on a relaxation of the update step, for example by using only a subset of data to update centroids as in [Sculley, 2010], which will be the focus of Chapter 2.

When comparing approximate K-means clustering algorithms such as those just mentioned, the two criteria of interest are the quality of the final clustering, and the computational requirements. The two criteria are not independent, making comparison between algorithms more difficult and often preventing their adoption. When comparing accelerated exact K-means algorithms on the other hand, all algorithms produce the same final clustering, and so comparisons can be made based on speed alone. Once an accelerated exact K-means algorithm has been confirmed to provide a speed-up, it is rapidly adopted, automatically inheriting the trust which the exact algorithm has gained through its simplicity and extensive use over several decades.

Accelerated exact K-means

The first published accelerated K-means algorithms borrowed techniques used to accelerate the nearest neighbour search. Examples are the adaptation of the algorithm of Orchard [1991] in Phillips [2002], and the use of kd-trees [Bentley, 1975] in Kanungo et al. [2002b]. These algorithms relied on storing centroids in special data structures, enabling nearest neighbor queries to be processed without computing distances to all K centroids.

The next big acceleration [Elkan, 2003] came about by maintaining bounds on distances between samples and centroids, frequently resulting in more than 90% of distance calculations being avoided. It was later shown [Hamerly, 2010] that in low-dimensions, it is more effective to keep bounds on distances to only the two nearest centroids, and that in general bounding-based algorithms are significantly faster than tree-based ones. Further bounding-based algorithms were proposed by Drake [2013] and Ding et al. [2015], each providing accelerations over their predecessors in certain settings. In this chapter, we continue in the same vain.

Our contribution

Our first contribution (Section 1.3.1) is a new bounding-based accelerated exact K-means algorithm, the Exponion algorithm. Its closest relative is the Annular algorithm [Drake, 2013], a state-of-the-art accelerated exact K-means algorithm in low-dimensions. We show that the Exponion algorithm is significantly faster than the Annular algorithm on a majority of low-dimensional datasets.

Our second contribution (Section 1.3.2) is a technique for making bounds tighter, allowing further redundant distance calculations to be eliminated. The technique, illustrated in Figure 1.1, can be applied to all existing bounding-based K-means algorithms.

Finally, we show how certain of the current state-of-the-art algorithms can be accelerated through strict simplifications (Section 1.2.2 and Section 1.2.6). Fully parallelised implementations of all algorithms are provided under an open-source license at https://github.com/idiap/eakmeans

Notation and baselines

We describe four accelerated exact K-means algorithms in order of publication date. For two of these we propose simplified versions which offer natural stepping stones in understanding the full versions, as well as being faster (Section 1.4.1).

Our notation is based on that of Hamerly [2010], and only where necessary is new notation introduced. We use for example N for the number of samples and K for the number of clusters. Indices i and j always refer to data and cluster indices respectively, with a sample denoted by $x(i)$ and the index of the cluster to which it is assigned by $a(i)$. A cluster's centroid is denoted as $c(j)$. We introduce new notation by letting $n_1(i)$ and $n_2(i)$ denote the indices of the clusters whose centroids are the nearest and second nearest to sample i respectively.

Note that $a(i)$ and $n_1(i)$ are different, with the objective in a round of K-means being to set $a(i)$ to $n_1(i)$. $a(i)$ is a variable maintained by algorithms, changing within loops whenever a better candidate for the nearest centroid is found. On the other hand, $n_1(i)$ is introduced purely to aid in proofs, and is external to any algorithmic details. It can be considered to be the hidden variable which algorithms need to reveal.

All of the algorithms which we consider are elaborations of Lloyd's algorithm, and thus consist of repeating the assignment step and update step, given respectively as

$$a(i) \leftarrow n_1(i),\ i \in \{1, \dots, N\} \tag{1.1}$$

$$c(j) \leftarrow \frac{\sum_{i:a(i)=j} x(i)}{\|i : a(i) = j\|},\ j \in \{1, \dots, K\}. \tag{1.2}$$

These two steps are repeated until there is no change to any $a(i)$, or some other stopping criterion is met. We reiterate that all the algorithms discussed provide the same output at each iteration of the two steps, differing only in how $a(i)$ is computed in (1.1).

Standard algorithm (`sta`)

The Standard algorithm, in this chapter reffered to as `sta`, is the simplest implementation of Lloyd's algorithm. The only variables kept are $x(i)$ and $a(i)$ for $i \in \{1, \dots, N\}$ and $c(j)$ for $j \in \{1, \dots, K\}$. The assignment step consists of, for each i, calculating the distance from $x(i)$ to all centroids, thus revealing $n_1(i)$.

Simplified Elkan's algorithm (`selk`)

Simplified Elkan's algorithm, henceforth `selk`, uses a strict subset of the strategies described in Elkan [2003]. In addition to $x(i)$, $a(i)$ and $c(j)$, the variables kept are

$p(j)$, the distance moved by $c(j)$ in the last update step, and bounds $l(i,j)$ and $u(i)$, maintained to satisfy,

$$l(i,j) \leq \|x(i) - c(j)\|, \qquad u(i) \geq \|x(i) - c(a(i))\|.$$

These bounds are used to eliminate unnecessary centroid-data distance calculations using,

$$u(i) < l(i,j) \implies \|x(i) - c(a(i))\| < \|x(i) - c(j)\| \implies j \neq n_1(i). \tag{1.3}$$

We refer to (1.3) as an *inner* test, as it is performed within a loop over centroids for each sample. This as opposed to an *outer* test which is performed just once per sample, examples of which will be presented later.

To maintain the correctness of the bounds when centroids move, bounds are updated at the beginning of each assignment step with

$$l(i,j) \leftarrow l(i,j) - p(j), \qquad u(i) \leftarrow u(i) + p(a(i)). \tag{1.4}$$

The validity of these updates is a simple consequence of the triangle inequality, with a proof in A.1.1. We say that a bound is *tight* if it is known to be equal to the distance it is bounding, a *loose* bound is one which is not tight. For `selk`, bounds are initialized to be tight, and tightening a bound evidently costs one distance calculation.

When in a given round $u(i) \geq l(i,j)$, the test (1.3) fails. The first time this happens in a round for sample i, both $u(i)$ and $l(i,j)$ are loose due to preceding bound updates of the form (1.4). Tightening either bound may result in the test succeeding, but bound $u(i)$ should be tightened before $l(i,j)$, as it reappears in all tests for sample i and will thus be reused. In the case of a test failure with tight $u(i)$ and loose $l(i,j)$ bound $l(i,j)$ is tightened. A test failure with $u(i)$ and $l(i,j)$ both tight implies that centroid j is nearer to sample i than the currently assigned cluster centroid, and so $a(i) \leftarrow j$ and $u(i) \leftarrow l(i,j)$.

Elkan's algorithm (`elk`)

The fully-fledged algorithm of Elkan [2003], henceforth `elk`, adds to `selk` an additional strategy for eliminating distance calculations in the assignment step. Two further variables, $cc(j,j')$, the matrix of inter-centroid distances, and $s(j)$, the distance from centroid j to its nearest other centroid, are kept. A simple application of the triangle inequality, proved in A.1.2, provides the following test,

$$\frac{cc(a(i),j)}{2} > u(i) \implies j \neq n_1(i). \tag{1.5}$$

Algorithm `elk` uses (1.5) in unison with (1.3) to obtain an improvement on the test of `elk`, of the form,

$$\max\left(l(i,j), \frac{cc(a(i),j)}{2}\right) > u(i) \implies j \neq n_1(i). \tag{1.6}$$

In addition to the inner test (1.6), `elk` uses an outer test, whose validity follows from that of (1.5), given by,

$$\frac{s(a(i))}{2} > u(i) \implies n_1(i) = a(i). \tag{1.7}$$

If the outer test (1.7) is successful, one proceeds immediately to the next sample without changing $a(i)$, thus not only saving K distance calculations but also K floating-point comparisons.

Hamerly's algorithm (`ham`)

The algorithm of Hamerly [2010], henceforth `ham`, represents a shift of focus from inner to outer tests, completely foregoing the inner test of `elk`, and providing an improved outer test.

The K lower bounds per sample of `elk` are replaced by a single lower bound on all centroids other than the one assigned, defined to satisfy

$$l(i) \leq \min_{j \neq a(i)} \|x(i) - c(j)\|.$$

The variables $p(j)$ and $u(i)$ used in `elk` have the same definition for `ham`. The test for a sample i is

$$\max\left(l(i), \frac{s(a(i))}{2}\right) > u(i) \implies n_1(i) = a(i), \tag{1.8}$$

with the proof of correctness being essentially the same as that for the inner test of `elk`. If test (1.8) fails for sample i, then $u(i)$ is made tight, by computing $\|x(i) - c(a(i))\|$. If test (1.8) fails with $u(i)$ tight, then all the distances from sample i to centroids are computed, thus revealing $n_1(i)$ and $n_2(i)$ and allowing the updates $a(i) \leftarrow n_1(i)$, $u(i) \leftarrow \|x(i) - c(n_1(i))\|$ and $l(i) \leftarrow \|x(i) - c(n_2(i))\|$. As with `elk`, at the start of the assignment step, bounds need to be adjusted to ensure their correctness following the update step. This is done via,

$$l(i) \leftarrow l(i) - \arg\max_{j \neq a(i)} p(j), \qquad u(i) \leftarrow u(i) + p(a(i)).$$

Annular algorithm (`ann`)

The Annular algorithm of Drake [2013], henceforth `ann`, is a strict extension of `ham`, adding one novel test. In addition to the variables used in `ham`, one new variable $b(i)$ is required, which roughly speaking is to $n_2(i)$ what $a(i)$ is to $n_1(i)$. Also, the centroid norms $\|c(j)\|$ should be computed and sorted in each round.

Upon failure of test (1.8) with tight bound $u(i)$ in `ham`, $\|x(i) - c(j)\|$ is computed for all $j \in \{1, \ldots, K\}$ to reveal $n_1(i)$ and $n_2(i)$. With `ann`, certain of these K calculations can be eliminated. Define the radius, and corresponding set of cluster indices,

$$\begin{aligned} R(i) &= \max\left(u(i), \|x(i) - c(b(i))\|\right), \\ \mathcal{J}(i) &= \{j : |\|c(j)\| - \|x(i)\|| \leq R(i)\}. \end{aligned} \tag{1.9}$$

The following implication, proved in A.1.3, is used

$$j \notin \mathcal{J}(i) \implies j \notin \{n_1(i), n_2(i)\}.$$

Thus only distances from sample i to centroids of the clusters whose indices are in $\mathcal{J}(i)$ need to be calculated for $n_1(i)$ and $n_2(i)$ to be revealed. Once $n_1(i)$ and $n_2(i)$ revealed, $a(i)$, $u(i)$ and $l(i)$ are updated as per `ham`, and $b(i) \leftarrow n_2(i)$.

Note that by keeping an ordering of $\|c(j)\|$ the set $\mathcal{J}(i)$ can be determined in $\Theta(\log(K))$ operations with two binary searches, one for each of the inner and outer radii of $\mathcal{J}(i)$.

Simplified Yinyang (`syin`) and Yinyang (`yin`) algorithms

The basic idea with the Yinyang algorithm [Ding et al., 2015] and the Simplified Yinyang algorithm, henceforth `yin` and `syin` respectively, is to maintain consistent lower bounds for groups of clusters as a compromise between the $K - 1$ lower bounds of `elk` and the single lower bound of `ham`. In Ding et al. [2015] the number of groups is fixed at one tenth the number of centroids. The groupings are determined and fixed by an initial clustering of the centroids. The algorithm appearing in the literature most similar to `yin` is Drake's algorithm of [Drake and Hamerly, 2012], not to be confused with `ann`. According to Ding et al. [2015], Drake's algorithm does not perform as well as `yin`, and we thus choose not to consider it in this chapter.

Denote by G the number of groups of clusters. Variables required in addition to those used in `sta` are $p(j)$ and $u(i)$, as per `elk`, $\mathcal{G}(f)$, the set of indices of clusters belonging to the f'th group, $g(i)$, the group to which cluster $a(i)$ belongs, $q(f) = \max_{j \in \mathcal{G}(f)} p(j)$, and bound $l(i, f)$, maintained to satisfy,

$$l(i, f) \leq \operatorname*{arg\,min}_{j \in \mathcal{G}(f) \setminus \{a(i)\}} \|x(i) - c(j)\|.$$

For both `syin` and `yin`, both an outer test and group tests are used. To these, `yin` adds an inner test. The outer test is

$$\min_{f \in \{1,\ldots,G\}} l(i,f) > u(i) \implies a(i) = n_1(i). \tag{1.10}$$

If and when test (1.10) fails, group tests of the form

$$l(i,f) > u(i) \implies a(i) \notin \mathcal{G}(f), \tag{1.11}$$

are performed. As with `elk` and `ham`, if test (1.11) fails with $u(i)$ loose, $u(i)$ is made tight and the test reperformed.

The difference between `syin` and `yin` arises when (1.11) fails with $u(i)$ tight. With `syin`, the simple approach of computing distances from $x(i)$ to all centroids in $\mathcal{G}(f)$, then updating $l(i,f), l(i,g(i)), u(i), a(i)$ and $g(i)$ as necessary, is taken. With `yin` a final effort at eliminating distance calculations by the use of a local test is made, as described in A.2.1. As will be shown (Section 1.4.1), it is not clear that the local test of `yin` makes it any faster. Finally, we mention how $u(i)$ and $l(i,f)$ are updated at the beginning of the assignment step for `syin` and `yin`,

$$\begin{aligned} l(i,f) &\leftarrow l(i,f) - \arg\max_{j \in \mathcal{G}(f)} p(a(i)), \\ u(i) &\leftarrow u(i) + p(a(i)). \end{aligned}$$

Contributions and new algorithms

We first present (Section 1.3.1) an algorithm which we call Exponion, and then (Section 1.3.2) an improved bounding approach.

Exponion algorithm (`exp`)

Like `ann`, `exp` is an extension of `ham` which adds a test to filter out $j \notin \{n_1(i), n_2(i)\}$ when test (1.8) fails. Unlike `ann`, where the filter is an origin-centered annulus, `exp` has as filter a ball centred on centroid $a(i)$. This change is motivated by the ratio of volumes of an annulus of width r at radius w and a ball of radius r from the origin, which is $d\left(\frac{w}{r}\right)^{d-1}$ in $\mathbb{R}^d$. We expect r to be greater than w, whence the expected improvement. Define,

$$\begin{aligned} R(i) &= 2u(i) + s(a(i)), \\ \mathcal{J}(i) &= \{j : \|c(j) - c(a(i))\| \leq R(i)\}. \end{aligned} \tag{1.12}$$

The underlying test used, proved in A.1.4, is

$$j \notin \mathcal{J}(i) \implies j \notin \{n_1(i), n_2(i)\}.$$

In moving from `ann` to `exp`, the decentralization from the origin to the centroids incurs two costs, one which can be explained algorithmically, the other is related to effective computer memory use.

Recall that `ann` sorts $\|c(j)\|$ in each round, thus guaranteeing that the set of candidate centroids (1.9) can be obtained in $O(\log(k))$ operations. To guarantee that the set of candidate centroids (1.12) can be obtained with $O(\log(k))$ operations requires that $\|c(j) - c(a(i))\|$ be sorted. For this to be true for all samples requires sorting $\|c(j) - c(j')\|$ for all $j \in \{1, \ldots, K\}$, increasing the overhead of sorting from $O(k \log k)$ to $O(k^2 \log k)$.

The cache memory cost incurred is that, unless samples are ordered by $a(i)$, the bisection search performed to obtain $\mathcal{J}(i)$ is done with a different row of $c(j, j')$ for each sample, resulting in cache memory misses.

To offset these costs, we replace the exact sorting of cc with a partial sorting, paying for this approximation with additional distance calculations. We maintain, for each centroid, $\lceil \log_2 k \rceil$ concentric annuli, each succesive annulus containing twice as many centroids as the one interior to it. For cluster j, annulus $f \in \{1, \ldots, \lceil \log_2 k \rceil\}$ is defined by inner and outer radii $e(j, f-1)$ and $e(j, f)$, and a list of indices $w(j, f)$ with $|w(j, f)| = 2^f$, where

$$w(j, f) = \{j' : e(j, f-1) < \|c(j') - c(j)\| \leq e(j, f)\}.$$

Note that $w(j, f)$ is not an ordered set, but there is an ordering between sets,

$$j' \in w(j, f), j'' \in w(j, f+1) \implies \|c(j') - c(j)\| < \|c(j'') - c(j)\|.$$

Given a search radius $R(i)$, without a complete ordering of $c(j, j')$ we cannot obtain $\mathcal{J}(i)$ in $O(\log(k))$ operations, but we can obtain a slightly larger set $\mathcal{J}^*(i)$ defined by

$$\begin{aligned} f^*(i) &= \min\{f : e(a(i), f) \geq R(i)\}, \\ \mathcal{J}^*(i) &= \bigcup_{f \leq f^*(i)} w(j, f), \end{aligned}$$

in $\log\log(k)$ operations. It is easy to see that $|\mathcal{J}^*(i)| \leq 2|\mathcal{J}(i)|$, and so using the partial sorting cannot cost more than twice the number of distance calculations.

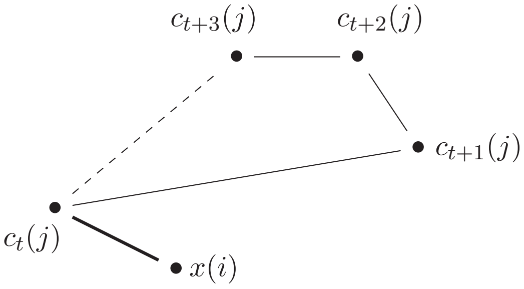

Figure 1.1 – The classical sn-bound is the sum of the last known distance between the sample to a previous position of the centroid (thick solid line), with all the distances between successive positions of the centroid since then (thin solid lines). The ns-bound we propose uses the actual distance between that previous location of the centroid and its current one (dashed line).

Improving bounds (sn to ns)

In all the algorithms presented so far, upper bounds (lower bounds) are updated in each round with increments (decrements) of norms of displacements. If tests are repeatedly successful, these increments (decrements) accumulate. Consider for example the upper bound update,

$$u_{t_0+1}(i) \leftarrow u_{t_0}(i) + p_{t_0}(a(i)),$$

where subscripts denote rounds. The upper bound after δt such updates without bound tightening is

$$u_{t_0+\delta t}(i) = u_{t_0}(i) + \sum_{t'=t_0}^{t+\delta t-1} p_{t'}(a(i)). \tag{1.13}$$

The summation term is a (s)um of (n)orms of displacement, thus we refer to it as an sn-bound and to an algorithm using only such an update scheme as an sn-algorithm. An alternative upper bound at round $t_0 + \delta t$ is,

$$\begin{aligned} u_{t_0+\delta t}(i) &= u_{t_0}(i) + \left\| \sum_{t'=t_0}^{t_0+\delta t-1} c_{t'+1}(i) - c_{t'}(i) \right\|, \\ &= u_{t_0}(i) + \|c_{t_0+\delta t}(i) - c_{t_0}(i)\|. \end{aligned} \tag{1.14}$$

Bound (1.14) derives from the (n)orm of a (s)um, and hence we refer to it as an ns-bound. An ns-bound is guaranteed to be tighter than its equivalent sn-bound by a simple application of the triangle inequality, shown in A.1.5. We have presented an upper ns-bound, but lower ns-bound formulations are similar. In fact, for cases where lower bounds apply to several distances simultaneously, due to the additional operation of taking a group maximum, there are three possible ways to compute a lower bound, as discussed in Appendix A.2.2.

Simplified Elkan's algorithm-ns (`selk-ns`)

In transforming an sn-algorithm into an ns-algorithm, additional variables need to be maintained. These include a record of previous centroids C, where $C(j,t) = c_t(j)$, and displacement of $c(j)$ with respect to previous centroids, $P(j,t) = \|c(j) - c_t(j)\|$. We no longer keep rolling bounds for each sample, instead we keep a record of when most recently bounds were made tight and the distances then calculated. For Simplified Elkan's Algorithm-ns, henceforth `selk-ns`, we define $T(i,j)$ to be the last time $\|x(i) - c(j)\|$ was calculated, with corresponding distance $l(i,j) = \|x(i) - c_{T(i,j)}(j)\|$. We emphasize that $l(i,j)$ is defined differently here to in `selk`, with $u(i)$ similarly redefined as $u(i) = \|x(i) - c_{T(i,a(i))}(a(i))\|$.

The underlying test is

$$u(i) + P(a(i), T(i,a(i))) < l(i,j) - P(j, T(i,j)) \implies j \neq n_1(i).$$

As with `selk`, the first bound failure for sample i results in $u(i)$ being updated, with subsequent failures resulting in $l(i,j)$ being updated to the current distance. In addition, when $u(i)$ ($l(i,j)$) is updated, $T(i,a(i))$ ($T(i,j)$) is set to the current round.

Due to the additional variables C, P and T, the memory requirement imposed is larger with `selk-ns` than with `selk-sn`. Ignoring constants, in round t the memory requirement assuming samples of size $O(d)$ is,

$$\texttt{mem}_{ns} = O(Nd + Nk + ktd),$$

where x, l and C are the principal contributors to the above three respective terms. `selk` consists of only the first two terms, and so when $t > N/\min(k,d)$, the dominant memory consumer in `selk-ns` is the new variable C. To guarantee that C does not dominate memory consumption, an sn-like reset is performed in rounds $\{t : t \equiv 0 \mod (N/\min(k,d))\}$, consisting of the following updates,

$$\begin{aligned}
u(i) &\leftarrow u(i) + P(a(i), T(i,a(i))), \\
l(i,j) &\leftarrow l(i,j) - P(j, T(i,j)), \\
T(i,j) &\leftarrow t,
\end{aligned}$$

and finally the clearing of C.

Changing bounds for other algorithms

All sn- to ns- coversions are much the same as that described in Section 1.3.3. We have implemented versions of `elk`, `syin` and `exp` using ns-bounds, which we refer to as `elk-ns`, `syin-ns` and `exp-ns` respectively.

Experiments and results

Our first set of experiments are conducted using a single core. We first establish that our implementations of baseline algorithms are as fast or faster than existing implementations. Having done this, we consider the effects of the novel algorithmic contributions presented; simplification, the Exponion algorithm, and ns-bounding. The final set of experiments are conducted on multiple cores, and illustrate how all algorithms presented parallelise gracefully.

We compare 23 K-means implementations, including our own implementations of all algorithms described, original implementations accompanying the papers [Hamerly, 2010, Drake, 2013, Ding et al., 2015], and implementations in two popular machine learning libraries, VLFeat and mlpack. We use the following notation to refer to implementations: {`codesource-algorithm`}, where `codesource` is one of `bay` [Hamerly, 2015], `mlp` [Curtin et al., 2013], `pow` [Low et al., 2010], `vlf` [Vedaldi and Fulkerson, 2008] and `own` (our own code), and `algorithm` is one of the algorithms described.

Unless otherwise stated, times are wall times excluding data loading. We impose a time limit of 40 minutes and a memory limit of 4 GB on all {dataset, implementation, K, seed} runs. If a run fails to complete in 40 minutes, the corresponding table entry is ‘t’. Similarly, failure to execute with 4GB of memory results in a table entry ‘m’. We confirm that for all {dataset, K, seed} triplets, all implementations which complete within the time and memory constraint take the same number of iterations to converge to a common local minimum, as expected.

The implementations are compared over the 22 datasets presented in Table 1.1, for $K \in \{100, 1000\}$, with 10 distinct centroid initializations (seeds). For all {dataset, K, seed} triplets, the 23 implementations are run serially on a machine with an Intel i7 processor and 8MB of cache memory. All experiments are performed using double precision floating point numbers.

Findings in Drake [2013] suggest that the best algorithm to use for a dataset depends primarily on dimension, where in low-dimensions, `ham` and `ann` are fastest, in high-dimensions `elk` is fastest, and in intermediate dimensions an approach maintaining a fractional number of bounds, Drake’s algorithm, is fastest. Our findings corroborate these on real datasets, although the lines separating the three groups are blurry. In presenting our results we prefer to consider a partitioning of the datasets into just two

	d	N
i	2	100k
ii	2	169k
iii	2	1m
iv	3	165k
v	3	164k

	d	N
vi	4	200k
vii	4	200k
viii	9	68k
ix	11	41k
x	15	166k
xi	17	23k

	d	N
xii	28	66k
xiii	30	1m
xiv	50	60k
xv	50	130k
xvi	55	581k

	d	N
xvii	68	2.6m
xviii	74	146k
xix	108	1m
xx	128	14k
xxi	310	95k
xxii	784	60k

Table 1.1 – The 22 datasets used in experiments, ranging in dimension from 2 to 784. The datasets come from: the UCI, KDD and KEEL repositories (11,2,2), MNIST and STL-10 image databases (2,1), random (2), European Bioinformatics Institute (1) and Joensuu University (1). Full names and further details in A.3.

groups about the dimension $d = 20$. `ham` and its derivatives are considered for $d < 20$, `elk` and its derivatives for $d \geq 20$, and `syin` and `yin` for all d.

Single core experiments

A complete presentation of wall times and number of iterations for all {dataset, implementation, K} triplets is presented over two pages in Tables A.2 and A.3 (Appendix A.3). Here we attempt to summarise our findings. We first compare implementations of published algorithms (Section 1.4.1), and then show how `selk` and `syin` often outperform their more complex counterparts (Section 1.4.1). We show that `exp` is in general much faster than `ann` (Section 1.4.1), and finally show how using ns-bounds can accelerate algorithms (Section 1.4.1) .

Comparing implementations of baselines

There are algorithmic techniques which can speedup all K-means algorithms discussed in this chapter, we mention a few which we use. One is pre-computing the squares of norms of all samples just once, and those of centroids once per round. Another, first suggested in Hamerly [2010], is to update the sum of samples by considering only those samples whose assignment changed in the previous round. A third optimization technique is to decompose while-loops which contain inner branchings dependant on the tightness of upper bounds into separate while-loops, eliminating unnecessary comparisons. Finally, while there are no large matrix operations with bounding-based algorithms, in high-dimensions distance calculations can be accelerated by the use of SSE, as in VLFeat, or by fast implementations of BLAS, such as OpenBLAS Xianyi [2016].

Our careful attention to optimization is reflected in Table 1.2 (Section 1.4.1), where implementations of `elk`, `ham`, `ann` and `yin` are compared. The values shown are ratios of mean runtimes using another implementation (column) and our own implementation of the same algorithm, on a given dataset (row). Our implementations are faster in all but 4 comparisons.

Benefits of simplification

We compare published algorithms `elk` and `yin` with their simplified counterparts `selk` and `syin`. The values in Table 1.3 are ratios of mean runtimes using simplified and original algorithms, values less than 1 mean that the simplified version is faster. We observe that `selk` is faster than `elk` in 16 of 18 experiments, and `syin` is faster than `yin` in 43 of 44 experiments, often dramatically so.

It is interesting to ask why the inventors of `elk` and `yin` did not instead settle on algorithms `selk` and `syin` respectively. A partial answer might relate to the use of BLAS, as the speedup obtained by simplifying `yin` to `syin` never exceeds more than 10% when BLAS is deactivated. `syin` is more responsive to BLAS than `yin` as it has larger matrix multiplications due to it not having a final filter.

From Annular to Exponion

We compare the Annular algorithm (`ann`) with the Exponion algorithm (`exp`). The values in Table 1.4 are ratios of mean runtimes (columns q_t) and of mean number of distance calculations (columns q_{au}). Values less than 1 denote better performance with `exp`. We observe that `exp` is markedly faster than `ann` on most low-dimensional datasets, reducing by more than 30% the mean runtime in 17 of 22 experiments. The primary reason for the speedup is the reduced number of distance calculations.

Table 1.5 summarises how many times each of the sn-algorithms is fastest on the 44 $\{$dataset, $K\}$ experiments, ns-algorithms excluded. The 13 experiments on which `exp` is fastest are all very low-dimensional ($d < 5$), the 24 on which `syin` is fastest are intermediate ($8 < d < 69$) and `selk` or `elk` are fastest in very high dimensions ($d > 73$). For a detailed comparison across all algorithms, consult Tables A.2 and A.3 (Appendix A.3).

From sn to ns bounding

For each of the 44 $\{$dataset, $K\}$ experiments, we compare the fastest sn-algorithm with its ns-variant. The results are presented in Table 1.6. Columns '`x`' denote the fastest sn-algorithm. Values are ratios of means over runs of some quantity using the ns- and sn- variants. The ratios are q_t (runtimes), q_a (number of distance calculations in the assignment step) and q_{au} (total number of distance calculations).

In all but 8 of 44 experiments (italicised), we observe a speedup using ns-bounding, by

		`bay-ham`	`mlp-ham`	`bay-ann`	`pow-yin`		`pow-yin`	`bay-elk`	`mlp-elk`	`vlf-elk`
100	i	1.23	1.04	*0.78*	7.52	xii	2.51	2.69	1.87	2.48
	ii	1.28	*0.99*	*0.86*	4.91	xiii	3.84	1.36	1.56	t
	iii	1.19	1.27	*0.88*	9.84	xiv	1.98	1.75	1.53	2.72
	iv	1.59	1.15	1.24	6.21	xv	2.21	1.48	1.48	2.01
	v	1.59	1.20	1.24	6.01	xvi	2.30	1.68	2.00	2.85
	vi	1.78	1.27	1.32	6.41	xvii	m	1.79	1.88	2.61
	vii	1.78	1.17	1.48	5.63	xviii	1.69	1.91	1.46	2.68
	viii	2.67	1.38	2.38	3.99	xix	1.49	1.64	1.74	2.44
	ix	2.93	1.51	2.90	3.65	xx	1.35	2.53	2.21	2.41
	x	3.59	1.75	2.67	3.28	xxi	1.24	2.35	1.57	1.81
	xi	3.89	2.04	3.18	2.17	xxii	1.16	2.86	1.43	1.35
1000	i	1.51	1.03	1.06	7.57	xii	3.37	6.21	3.20	2.44
	ii	1.52	1.04	1.17	8.03	xiii	t	m	m	m
	iii	1.47	1.05	1.04	8.57	xiv	2.09	1.89	1.86	2.07
	iv	1.77	1.09	1.59	6.98	xv	3.14	1.43	2.76	1.80
	v	1.77	1.09	1.59	7.01	xvi	3.98	m	m	m
	vi	2.07	1.17	1.79	7.23	xvii	m	m	m	m
	vii	1.99	1.17	1.73	6.57	xviii	1.82	1.78	1.40	1.92
	viii	3.01	1.38	2.97	4.63	xix	t	m	m	m
	ix	3.28	1.58	3.34	4.06	xx	2.06	6.17	2.60	1.72
	x	3.92	1.76	3.57	5.08	xxi	1.32	2.88	1.80	1.51
	xi	4.08	1.99	4.03	2.89	xxii	1.17	4.82	1.74	1.28

Table 1.2 – Comparing implementations. For 100 (above) and 1000 (below) clusters, and in low- (left) and high- (right) dimensions. Existing implementations (colums) of `ham`, `ann`, `yin` and `elk` are compared to our implementations as a ratio of mean runtimes, with the mean runtime of our implementation in the denominator. Values greater than 1 mean our implementation runs faster. ‘t’ and ‘m’ are described in paragraph 3 of Section 1.4.

`own-yin` → `own-syin`						`own-elk` → `own-selk`		
	100	1000		100	1000		100	1000
i	0.96	0.90	xii	0.58	0.76	xii	0.85	*1.05*
ii	*1.03*	0.86	xiii	0.66	0.61	xiii	0.97	m
iii	0.88	0.92	xiv	0.50	0.55	xiv	0.84	0.57
iv	0.94	0.87	xv	0.49	0.58	xv	0.54	0.49
v	0.93	0.88	xvi	0.49	0.66	xvi	0.92	m
vi	0.91	0.87	xvii	0.44	0.58	xvii	0.75	m
vii	0.96	0.90	xviii	0.42	0.47	xviii	0.86	0.66
viii	0.79	0.80	xix	0.36	0.42	xix	0.72	m
ix	0.77	0.80	xx	0.38	0.60	xx	*1.12*	0.74
x	0.72	0.73	xxi	0.32	0.36	xxi	0.89	0.73
xi	0.64	0.71	xxii	0.36	0.38	xxii	0.99	0.89

Table 1.3 – Comparing `yin` and `elk` to simplified versions `syin` and `selk`. Values are ratios of mean runtimes of simplified versions to their originals, for different low-dimensional datasets (rows) and K (columns). Values less than 1 mean that the simplified version is faster. In all but 3 of 62 cases (italicised), simplification results in speedup, by as much as 3×.

`own-ann` → `own-exp`									
	100		1000			100		1000	
	q_t	q_{au}	q_t	q_{au}		q_t	q_{au}	q_t	q_{au}
i	0.48	0.52	0.72	0.61	vii	0.71	0.80	0.36	0.32
ii	0.54	0.80	0.58	0.50	viii	*1.12*	*1.24*	*1.02*	*0.93*
iii	0.53	0.58	0.48	0.44	ix	0.96	0.99	0.73	0.64
iv	0.63	0.80	0.36	0.33	x	0.67	0.65	0.55	0.41
v	0.63	0.80	0.37	0.34	xi	*1.24*	*1.43*	*1.30*	*1.16*
vi	0.62	0.73	0.42	0.38					

Table 1.4 – Ratios of mean runtimes (‘q_t’) and mean number of distance calculations (‘q_{au}’) using the Exponion (`own-exp`) and Annular (`own-ann`) algorithms, on datasets with $d < 20$. Exponion is faster in all but the four italicised cases. The speedup is primarily due to the reduced number of distance calculations.

`ham`	`ann`	`exp`	`syin`	`yin`	`selk`	`elk`
0	0	13	24	0	6	1

Table 1.5 – Number of times each sn-algorithm is fastest, over the 44 {dataset, K} experiments, ns-algorithms not considered here.

`own-x` → `own-x-ns`								
	100				1000			
	`x`	q_t	q_a	q_{au}	`x`	q_t	q_a	q_{au}
i	`exp`	0.96	0.97	0.99	`exp`	0.99	0.98	1.00
ii	`exp`	0.94	0.97	0.97	`exp`	0.99	0.99	1.00
iii	`exp`	0.95	0.97	0.98	`exp`	0.98	0.96	0.99
iv	`exp`	0.97	0.97	0.97	`exp`	0.99	0.97	0.98
v	`exp`	0.96	0.97	0.97	`exp`	0.98	0.96	0.98
vi	`exp`	0.95	0.96	0.97	`exp`	0.97	0.96	0.98
vii	`syin`	0.98	0.82	0.86	`exp`	0.98	0.96	0.99
viii	`syin`	0.98	0.86	0.88	`syin`	0.87	0.44	0.65
ix	`syin`	0.98	0.83	0.86	`syin`	0.83	0.32	0.66
x	`syin`	*1.03*	*0.91*	*0.92*	`syin`	*1.11*	*0.72*	*0.80*
xi	`selk`	0.92	0.80	0.84	`syin`	0.81	0.56	0.69
xii	`syin`	*1.00*	*0.86*	*0.88*	`syin`	0.96	0.51	0.85
xiii	`syin`	0.96	0.84	0.84	`syin`	0.87	0.58	0.61
xiv	`syin`	0.99	0.86	0.87	`syin`	0.74	0.51	0.63
xv	`syin`	*1.06*	*0.93*	*0.94*	`syin`	0.94	0.58	0.69
xvi	`syin`	*1.04*	*0.91*	*0.93*	`syin`	0.98	0.61	0.79
xvii	`syin`	1.00	0.87	0.89	`syin`	m	m	m
xviii	`selk`	0.89	0.81	0.82	`syin`	0.75	0.64	0.68
xix	`selk`	0.88	0.84	0.85	`syin`	0.91	0.75	0.77
xx	`elk`	*1.02*	*0.96*	*1.02*	`selk`	*1.06*	*0.99*	*1.00*
xxi	`selk`	0.85	0.81	0.82	`selk`	0.72	0.72	0.73
xxii	`selk`	0.80	0.77	0.78	`selk`	0.55	0.68	0.69

Table 1.6 – The effect of using ns-bounds. Columns ‘`x`’ denotes the fastest sn-algorithm for a particular {dataset, K} experiment. Columns ‘q_t’ denote the ratio of mean runtimes of ns- and sn- variants of `x`. Italicised values are cases where using ns-bounding results in a slow down ($q_t > 1$), in the majority of cases there is a speedup. ‘q_a’ and ‘q_{au}’ denote ratios of ns- to sn- mean number of distance calculations in the assignment step (a) and in total (au). ‘m’ described in paragraph 3 of §1.4.

up to 45%. As expected, the number of distance calculations in the assignment step is never greater when using ns-bounds, however the total number of distance calculations is occasionally increased due to initial variables being maintained.

Multicore experiments

We have implemented parallelised versions of all algorithms described in this chapter using the C++11 thread support library. To measure the speedup using multiple cores, we compare the runtime using four threads to that using one thread on a non-hyperthreading four core machine. The results are summarised in Table 1.7, where near fourfold speedups are observed.

i-xi		
	100	1000
`own-exp-ns`	0.29	0.31
`own-syin-ns`	0.31	0.29

xii-xxii		
	100	1000
`own-selk-ns`	0.33	0.30
`own-elk-ns`	0.30	0.28
`own-syin-ns`	0.27	0.27

Table 1.7 – The median speedup using four cores. The median is over i-xi on the left and xii-xxii on the right.

Chapter conclusion and future work

The experimental results presented show that the ns-bounding scheme makes exact K-means algorithms faster, and that our Exponion algorithm is significantly faster than existing state-of-the-art algorithms in low-dimensions. Both can be seen as good default choices for K-means clustering on large data-sets.

The main practical weakness that remains is the necessary prior selection of which algorithm to use, depending on the dimensionality of the problem at hand. This should be addressed through an adaptive procedure able to select automatically the optimal algorithm through an efficient exploration/exploitation strategy. The second and more prospective direction of work will be to introduce a sharing of information between samples, instead of processing them independently. Finally we mention that for extremely large datasets, only algorithms which are sublinear in the number of datapoints are feasible. This is the topic of the following chapter.

2 Nested Mini-Batch K-means

Chapter introduction

Given N training samples $\mathcal{X} = \{x(1), \ldots, x(N)\}$ in vector space $\mathcal{V}$, the K-means task is to find $\mathcal{C} = \{c(1), \ldots, c(K)\}$ in $\mathcal{V}$ to minimize *energy* E defined by,

$$E(\mathcal{C}) = \frac{1}{N} \sum_{i=1}^{N} \|x(i) - c(a(i))\|^2, \tag{2.1}$$

where $a(i) = \arg\min_{j \in \{1,\ldots,K\}} \|x(i) - c(j)\|$. Recall that Lloyd's algorithm is a popular algorithm for approximately solving the K-means problem. In Chapter 1 we referred to the unoptimized version of Lloyd's algorithm as `sta`, in this chapter we will refer to it as `lloyd`. Please note the change in notation here: `lloyd` is no longer just the 'scaffolding' described in the previous chapter, but the full algorithm without any optimization.

In Chapter 1 we considered approaches for accelerating `lloyd`, where the required number of distance calculations and bound floating point value comparisons was reduced without changing the final clustering. Hamerly [2010] showed that previous approaches relying on triangle inequality based distance bounds [Phillips, 2002, Elkan, 2003, Hamerly, 2010] always provide greater speed-ups than those based on spatial data structures [Pelleg and Moore, 1999, Kanungo et al., 2002b]. We will rediscuss the bounding based approach in Section 2.2.1.

Previous approximate K-means algorithms

The assignment step of `lloyd` requires more computation than the update step. The majority of approximate algorithms thus focus on relaxing the assignment step, in one of two ways. The first is to assign all data approximately, so that centroids are updated using all data, but some samples may be incorrectly assigned. This is the approach used in Wang et al. [2012] with cluster closures. The second approach is to exactly assign a fraction of data at each iteration. This is the approach used in Agarwal et al. [2005], where a representative core-set is clustered, and in Bottou and Bengio [1995], and Sculley [2010], where random samples are drawn at each iteration. Using only a fraction of data is effective in reducing redundancy induced slow-downs.

The mini-batch K-means algorithm of Sculley [2010], henceforth `mbatch`, proceeds as follows. Centroids are initialized as a random selection of K samples. Then at every iteration, b of N samples are selected uniformly at random and assigned to clusters. Cluster centroids are updated as the mean of all samples ever assigned to them, and are therefore running averages of assignments. Samples randomly selected more often have more influence on centroids as they reappear more frequently in running averages, although the law of large numbers smooths out any discrepancies in the long run. `mbatch` is presented in greater detail in Section 2.2.2.

Contribution of this chapter

The underlying goal of this chapter is to accelerate `mbatch` by using triangle inequality based distance bounds. In so doing, we hope to merge the complementary strengths of two powerful and widely used approaches for accelerating `lloyd`: approximation as in Sculley [2010] and exact acceleration with the triangle inequality as in Elkan [2003].

The effective incorporation of bounds into `mbatch` requires a new sampling approach. To see this, first note that bounding can only accelerate the processing of samples which have already been visited, as the first visit is used to establish bounds. Next, note that the expected proportion of visits during the first epoch which are *re*visits is at most $1/e$, as shown in B.1. Thus the majority of visits are first time visits and hence cannot be accelerated by bounds. However, for highly redundant datasets, `mbatch` often obtains satisfactory clustering in a single epoch, and so bounds need to be effective during the first epoch if they are to contribute more than a minor speed-up.

To better harness bounds, one must preferentially reuse already visited samples. To this end, we propose nested mini-batches. Specifically, letting $\mathcal{M}_t \subseteq \{1, \ldots, N\}$ be the mini-batch indices used at iteration $t \geq 1$, we enforce that $\mathcal{M}_t \subseteq \mathcal{M}_{t+1}$. One concern with nesting is that samples entering in early iterations have more influence than samples entering at late iterations, thereby introducing bias. To resolve this problem, we enforce that samples appear at most once in running averages. Specifically, when a sample is revisited, its old assignment is first removed before it is reassigned. The idea of nested mini-batches is discussed in Section 2.3.1.

The second challenge introduced by using nested mini-batches is determining the size of $\mathcal{M}_t$. On the one hand, if $\mathcal{M}_t$ grows too slowly, then one may suffer from premature fine-tuning. Specifically, when updating centroids using $\mathcal{M}_t \subset \{1, \ldots, N\}$, one is using energy estimated on samples indexed by $\mathcal{M}_t$ as a proxy for energy over all N training samples. If $\mathcal{M}_t$ is small and the energy estimate is poor, then minimising the energy estimate exactly is a waste of computation, as as soon as the mini-batch is augmented the proxy energy loss function will change. On the other hand, if $\mathcal{M}_t$ grows too rapidly, the problem of redundancy arises. Specifically, if centroid updates obtained with a small fraction of $\mathcal{M}_t$ are similar to the updates obtained with $\mathcal{M}_t$, then it is waste of computation using $\mathcal{M}_t$ in its entirety. These ideas are pursued in Section 2.3.2.

Related works

Exact acceleration using the triangle inequality

We briefly rediscuss the use of the triangle inequality to accelerate the exact algorithm. The idea introduced in Elkan [2003] is to eliminate certain of the K distance calculations per sample by maintaining bounds on distances between samples and centroids. Several novel bounding based algorithms have since been proposed, such as the `yinyang` algorithm of Ding et al. [2015] and others described in Chapter 1. We illustrate the use of the triangle inequality in Alg. 1, where for sample i one maintains K lower bounds, $l(i,j)$ for $j \in \{1,\dots,K\}$, each bound satisfying $l(i,j) \le \|x(i) - c(j)\|$. Before computing $\|x(i) - c(j)\|$ on line 4 of Alg. 1, one checks that $l(i,j) < d(i)$, where $d(i)$ is the distance from sample i to the nearest currently found centroid. If $l(i,j) \ge d(i)$ then $\|x(i) - c(j)\| \ge d(i)$, and thus j can automatically be eliminated as a nearest centroid candidate.

Algorithm 1 `assignment-with-bounds`(i)

```
 1: $d(i) \leftarrow \|x(i) - c(a(i))\|$        ▷ where $d(i)$ is distance to nearest centroid found so far
 2: for all $j \in \{1,\dots,K\} \setminus \{a(i)\}$ do
 3:     if $l(i,j) < d(i)$ then
 4:         $l(i,j) \leftarrow \|x(i) - c(j)\|$        ▷ make lower bound on distance tight
 5:         if $l(i,j) < d(i)$ then
 6:             $a(i) = j$
 7:             $d(i) = l(i,j)$
 8:         end if
 9:     end if
10: end for
```

As detailed in Chapter 1, the fully-fledged algorithm of Elkan [2003] uses additional tests to the one shown in Alg. 1, and includes upper bounds and inter-centroid distances.

To maintain the validity of bounds, after each centroid update one performs $l(i,j) \leftarrow l(i,j) - p(j)$, where $p(j)$ is the distance moved by centroid j during the centroid update, the validity of this correction follows from the triangle inequality. Lower bounds are initialized as exact distances in the first iteration, and only in subsequent iterations can bounds help in eliminating distance calculations. Therefore, the triangle inequality base algorithms are all at least as slow as `lloyd` during the first iteration.

Mini-batch K-means

The work of Sculley [2010] introduces `mbatch`, presented in Alg. 4, as a scalable alternative to `lloyd`. Reusing notation from that paper, we let the mini-batch size be b, and the total number of assignments ever made to cluster j be $v(j)$. Let $S(j)$ be the cumulative sum of data samples assigned to cluster j, accumulated over all rounds. The centroid update, line 9 of Alg. 4, is then $c(j) \leftarrow S(j)/v(j)$. Sculley [2010] present `mbatch` in the context of sparse datasets, and at the end of each round an l_1-sparsification operation is performed to encourage sparsity. In this chapter we are interested in `mbatch` in a more general context and do not consider sparsification.

Algorithm 2 `initialize-c-S-v`

```
for $j \in \{1, \dots, K\}$ do
    $c(j) \leftarrow x(i)$ for some $i \in \{1, \dots, N\}$
    $S(j) \leftarrow x(i)$
    $v(j) \leftarrow 1$
end for
```

Algorithm 3 `accumulate`(i)

```
$S(a(i)) \leftarrow S(a(i)) + x(i)$
$v(a(i)) \leftarrow v(a(i)) + 1$
```

Algorithm 4 `mbatch`

```
1:  initialize-c-S-v()
2:  while convergence criterion not satisfied do
3:      $M \leftarrow$ uniform random sample of size $b$ from $\{1, \dots, N\}$
4:      for all $i \in M$ do
5:          $a(i) \leftarrow \arg\min_{j \in \{1,\dots,k\}} \|x(i) - c(j)\|$
6:          accumulate($i$)
7:      end for
8:      for all $j \in \{1, \dots, K\}$ do
9:          $c(j) \leftarrow S(j)/v(j)$
10:     end for
11: end while
```

Nested mini-batch K-means : `nmbatch`

The bottleneck of `mbatch` is the assignment step, on line 5 of Alg. 4, which requires K distance calculations per sample. The underlying motivation of this chapter is to reduce the number of distance calculations at assignment by using distance bounds. However, as already discussed in Section 2.1.2, simply wrapping line 5 in a bound test would not result in much gain, as only a minority of visited samples would benefit from bounds in the first epoch, as there is no acceleration for a first visit. For this reason, we will replace random mini-batches at line 3 of Alg. 4 by nested mini-batches. This modification motivates a change to the running average centroid updates, discussed in Section 2.3.1. It also introduces the need for a scheme to choose mini-batch sizes, discussed in Section 2.3.2. The resulting algorithm, which we refer to as `nmbatch`, is presented in Alg. 5.

There is no random sampling in `nmbatch`, although an initial random shuffling of samples can be performed to remove any ordering that may exist. Let b_t be the size of the mini-batch at iteration t, that is $b_t = |\mathcal{M}_t|$. We simply take $\mathcal{M}_t$ to be the first b_t indices, that is $\mathcal{M}_t = \{1, \ldots, b_t\}$. Thus $\mathcal{M}_t \subseteq \mathcal{M}_{t+1}$ corresponds to $b_t \leq b_{t+1}$. Let T be the number of iterations of `nmbatch` before terminating. We use as stopping criterion that no assignments change on the full training set, although this is not important and can be modified.

Modifying cumulative sums to prevent duplicity

In `mbatch`, a sample used n times makes n contributions to one or more centroids, through line 6 of Alg. 4. Due to the extreme and systematic difference in the number of times samples are used with nested mini-batches, it is necessary to curtail any potential bias that duplicated contribution may incur. To this end, we only allow a sample's most recent assignment to contribute to centroids. This is done by removing old assignments before samples are reused, shown on lines 15 and 16 of Alg. 5.

Balancing premature fine-tuning and redundancy

We now discuss how one may select mini-batch size b_t, where recall that the sample indices of the mini-batch at iteration t are $\mathcal{M}_t = \{1, \ldots, b_t\}$. The only constraint imposed so far is that $b_t \leq b_{t+1}$ for $t \in \{1, \ldots, T-1\}$, that is that b_t does not decrease. We consider two extreme schemes to illustrate the importance of finding a scheme where b_t grows neither too rapidly nor too slowly.

The first extreme scheme is $b_t = N$ for $t \in \{1, \ldots, T\}$. This is just a return to full batch K-means, and thus redundancy is a problem, particularly at early iterations. The second extreme scheme, where $\mathcal{M}_t$ grows very slowly, is the following: if any assignment changes at iteration t, then $b_{t+1} = b_t$, otherwise $b_{t+1} = b_t + 1$. The problem with this second scheme is that computation may be wasted in finding centroids which accurately minimize the energy estimated on unrepresentative subsets of the full training set. This is what we refer to as premature fine-tuning.

To develop a scheme which balances redundancy and premature fine-tuning, we need to find sensible definitions for these terms. A first attempt might be to define them in terms of energy (2.1), as this is ultimately what we wish to minimize. Redundancy would correspond to a slow decrease in energy caused by long iteration times, and premature fine-tuning would correspond to approaching a local minimum of a poor proxy for (2.1).

Algorithm 5 `nmbatch`

```
$t = 1$    ▷ Iteration number
$\mathcal{M}_0 \leftarrow \{\}$
$\mathcal{M}_1 \leftarrow \{1, \ldots, b_s\}$    ▷ Indices of samples in current mini-batch
initialize-c-S-v()
for $j \in \{1, \ldots, K\}$ do
    $sse(j) \leftarrow 0$    ▷ Initialize sum of squares of samples in cluster $j$
end for
while stop condition is false do
    for $i \in \mathcal{M}_{t-1}$ and $j \in \{1, \ldots, K\}$ do
        $l(i, j) \leftarrow l(i, j) - p(j)$    ▷ Update bounds of reused samples
    end for
    for $i \in \mathcal{M}_{t-1}$ do
        $a_{old}(i) \leftarrow a(i)$
        $sse(a_{old}(i)) \leftarrow sse(a_{old}(i)) - d(i)^2$    ▷ Remove old $sse, S$ and $v$ contributions
        $S(a_{old}(i)) \leftarrow S(a_{old}(i)) - x(i)$
        $v(a_{old}(i)) \leftarrow v(a_{old}(i)) - 1$
        assignment-with-bounds($i$)    ▷ Reset assignment $a(i)$
        accumulate($i$)
        $sse(a(i)) \leftarrow sse(a(i)) + d(i)^2$
    end for
    for $i \in \mathcal{M}_t \setminus \mathcal{M}_{t-1}$ and $j \in \{1, \ldots, K\}$ do
        $l(i, j) \leftarrow \|x(i) - c(j)\|$    ▷ Tight initialization for new samples
    end for
    for $i \in \mathcal{M}_t \setminus \mathcal{M}_{t-1}$ do
        $a(i) \leftarrow \arg\min_{j \in \{1, \ldots, K\}} l(i, j)$
        $d(i) \leftarrow l(i, a(i))$
        accumulate($i$)
        $sse(a(i)) \leftarrow sse(a(i)) + d(i)^2$
    end for
    for $j \in \{1, \ldots, K\}$ do
        $\hat{\sigma}_C(j) \leftarrow \sqrt{(sse(j))/\left(v(j)(v(j) - 1)\right)}$
        $c_{old}(j) \leftarrow c(j)$
        $c(j) \leftarrow S(j)/v(j)$
        $p(j) \leftarrow \|c(j) - c_{old}(j)\|$
    end for
    if $\min_{j \in \{1, \ldots, K\}} (\hat{\sigma}_c(j)/p(j)) > \rho$ then    ▷ Check doubling condition
        $\mathcal{M}_{t+1} \leftarrow \{1, \ldots, \min(2|\mathcal{M}_t|, N)\}$
    else
        $\mathcal{M}_{t+1} \leftarrow \mathcal{M}_t$
    end if
    $t \leftarrow t + 1$
end while
```

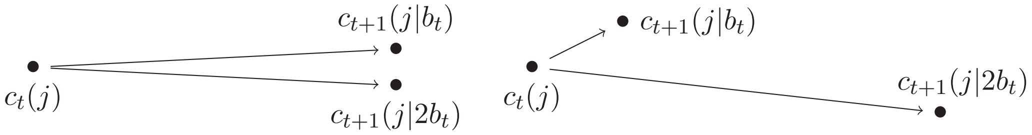

Figure 2.1 – Centroid based definitions of redundancy and premature fine-tuning. Starting from centroid $c_t(j)$, the update can be performed with a mini-batch of size b_t or $2b_t$. On the left, it makes little difference and so using all $2b_t$ points would be redundant. On the right, using $2b_t$ samples results in a much larger change to the centroid, suggesting that $c_t(j)$ is near to a local minimum of energy computed on b_t points, corresponding to premature fine-tuning.

A difficulty with an energy based approach is that we do not want to compute (2.1) at each iteration and there is no clear way to quantify the underestimation of (2.1) using a mini-batch. We instead consider definitions based on centroid statistics.

Balancing intra-cluster standard deviation with centroid displacement

Let $c_t(j)$ denote centroid j at iteration t, and let $c_{t+1}(j|b)$ be $c_{t+1}(j)$ when $\mathcal{M}_{t+1} = \{1, \ldots, b\}$, so that $c_{t+1}(j|b)$ is the update to $c_t(j)$ using samples $\{x(1), \ldots, x(b)\}$. Consider two options, $b_{t+1} = b_t$ with resulting update $c_{t+1}(j|b_t)$, and $b_{t+1} = 2b_t$ with update $c_{t+1}(j|2b_t)$. If,

$$\|c_{t+1}(j|2b_t) - c_{t+1}(j|b_t)\| \ll \|c_t(j) - c_{t+1}(j|b_t)\|, \tag{2.2}$$

then it makes little difference if centroid j is updated with $b_{t+1} = b_t$ or $b_{t+1} = 2b_t$, as illustrated in Figure 2.1, left. Using $b_{t+1} = 2b_t$ would therefore be redundant. If on the other hand,

$$\|c_{t+1}(j|2b_t) - c_{t+1}(j|b_t)\| \gg \|c_t(j) - c_{t+1}(j|b_t)\|, \tag{2.3}$$

this suggests premature fine-tuning, as illustrated in Figure 2.1, right. Balancing redundancy and premature fine-tuning thus equates to balancing the terms on the left and right hand sides of (2.2) and (2.3). Let us denote by $\mathcal{M}_t(j)$ the indices of samples in $\mathcal{M}_t$ assigned to cluster j. In B.2 we show that the term on the left hand side of (2.2) and (2.3) can be estimated by $\frac{1}{2}\hat{\sigma}_C(j)$, where

$$\hat{\sigma}_C^2(j) = \frac{1}{|\mathcal{M}_t(j)|^2} \sum_{i \in \mathcal{M}_t(j)} \|x(i) - c_t(j)\|^2. \tag{2.4}$$

$\hat{\sigma}_C(j)$ may underestimate $\|c_{t+1}(j|2b_t) - c_{t+1}(j|b_t)\|$ as samples $\{x(b_{t+1}), \ldots, x(2b_t)\}$ have not been used by centroids at iteration t, however our goal here is to establish dimensional homogeneity. The right hand sides of (2.2) and (2.3) can be estimated by the distance moved by centroid j in the preceding iteration, which we denote by $p(j)$. Balancing redundancy and premature fine-tuning thus equates to preventing $\hat{\sigma}_C(j)/p(j)$ from getting too large or too small.

It may be that $\hat{\sigma}_C(j)/p(j)$ differs significantly between clusters j. It is not possible to independently control the number of samples per cluster, and so a joint decision needs to be made by clusters as to whether or not to increase b_t. We choose to make the decision based on the minimum ratio, on line 37 of Alg. 5, as premature fine-tuning is less costly when performed on a small mini-batch, and so it makes sense to allow slowly converging centroids to catch-up with rapidly converging ones.

The decision to use a double-or-nothing scheme for growing the mini-batch is motivated by the fact that $\hat{\sigma}_C(j)$ drops by a constant factor when the mini-batch doubles in size. A linearly increasing mini-batch would be prone to premature fine-tuning as the mini-batch would not be able to grow rapidly enough.

Starting with an initial mini-batch size b_0, `nmbatch` iterates until $\min_j \hat{\sigma}_C(j)/p(j)$ is above some threshold ρ, at which point mini-batch size increases as $b_t \leftarrow \min(2b_t, N)$, shown on line 37 of Alg. 5. The mini-batch size is guaranteed to eventually reach N, as $p(j)$ eventually goes to zero. The doubling threshold ρ reflects the relative costs of premature fine-tuning and redundancy.

A note on parallelization

The parallelization of `nmbatch` can be done in the same way as in `mbatch`, whereby a mini-batch is simply split into sub-mini-batches to be distributed. For `mbatch`, the only constraint on sub-mini-batches is that they are of equal size to guarantee equal processing times. With `nmbatch` the constraint is slightly stricter, as the time required to process a sample depends on its time of entry into the mini-batch, due to bounds. Samples from all iterations should thus be balanced, the constraint becoming that each sub-mini-batch contains an equal number of samples from $\mathcal{M}_t \setminus \mathcal{M}_{t-1}$ for all t. This is easy to implement.

Results

We have performed experiments on 3 dense vector datasets and the sparse dataset used in Sculley [2010]. The INFMNIST dataset [Loosli et al., 2007] is an extension of MNIST, consisting of 28×28 hand-written digits ($d = 784$). We use 400,000 such digits for performing K-means and 40,000 for computing a validation energy E_V. STL10P [Coates et al., 2011] consists of 6×6×3 image patches ($d = 108$), we train with 960,000 patches and use 40,000 for validation. KDDC98 contains 75,000 training samples and 20,000 validation samples, in 310 dimensions. Finally, the sparse RCV1 dataset of Lewis et al. [2004] consists of data in 47,237 dimensions, with two partitions containing 781,265 and 23,149 samples respectively. As done in Sculley [2010], we use the larger partition to learn clusters.

The experimental setup used on each of the datasets is the following: for 20 random seeds, the training dataset is shuffled and the first K datapoints are taken as initialising centroids. Then, for each of the algorithms, K-means is run on the shuffled training set. At regular intervals, a validation energy E_V is computed on the validation set. The time taken to compute E_V is not included in run times. The batchsize for `mbatch` and initial batchsize for `nmbatch` are $5,000$, and $k = 50$ clusters. The mean and standard deviation of E_V over the 20 runs are computed, and this is what is plotted in Figure 2.2, relative to the lowest obtained validation energy over all runs on a dataset, E^*. Before comparing algorithms, we note that our implementation of the baseline `mbatch` is competitive with existing implementations, as shown in Section 2.6.

In Figure 2.2, we plot time-energy curves for `nmbatch` with three baselines. We use $\rho = 100$, as described in the next paragraph. On the 3 dense datasets, we see that `nmbatch` is much faster than `mbatch`, obtaining a solution within 2% of E^* between 10× and 100× earlier than `mbatch`. On the sparse dataset RCV1, the speed-up becomes noticeable within 0.5% of E^*. Note that in a single epoch `nmbatch` gets very near to E^*, whereas the full batch algorithms `lloyd` and `yinyang` only complete one iteration. The mean final energies of `nmbatch` and the exact algorithms are consistently within one initialization standard deviation. This means that the random initialization seed has a larger impact on final energy than the choose between `nmbatch` and an exact algorithm.

We now discuss the choice of ρ. Recall that the mini-batch size doubles whenever $\min_j \hat{\sigma}_C(j)/p(j) > \rho$. Thus a large ρ means smaller $p(j)$s are needed to invoke a doubling, which means less robustness against premature fine-tuning. The relative costs of premature fine-tuning and redundancy are influenced by the use of bounds. Consider the case of premature fine-tuning with bounds. $p(j)$ becomes small, and thus bound tests become more effective as they decrease more slowly (line 10 of Alg. 5). Thus, while premature fine-tuning does result in more samples being visited than necessary, each visit is processed rapidly and so is less costly. We have found that taking ρ to be large works well for `nmbatch`. Indeed, there is little difference in performance for $\rho \in \{10, 100, 1000\}$. To test that our formulation is sensible, we performed tests with the bound test (line 3 of Alg. 1) deactivated. When deactivated, $\rho = 10$ is in general better than larger values of ρ, as seen in Figure 2.3. Full time-energy curves for different ρ values are provided in Appendix B.3.

Chapter conclusion and future work

We have shown how triangle inequality based bounding can be used to accelerate mini-batch K-means. The key is the use of nested batches, which enables rapid processing of already used samples. The idea of replacing uniformly sampled mini-batches with

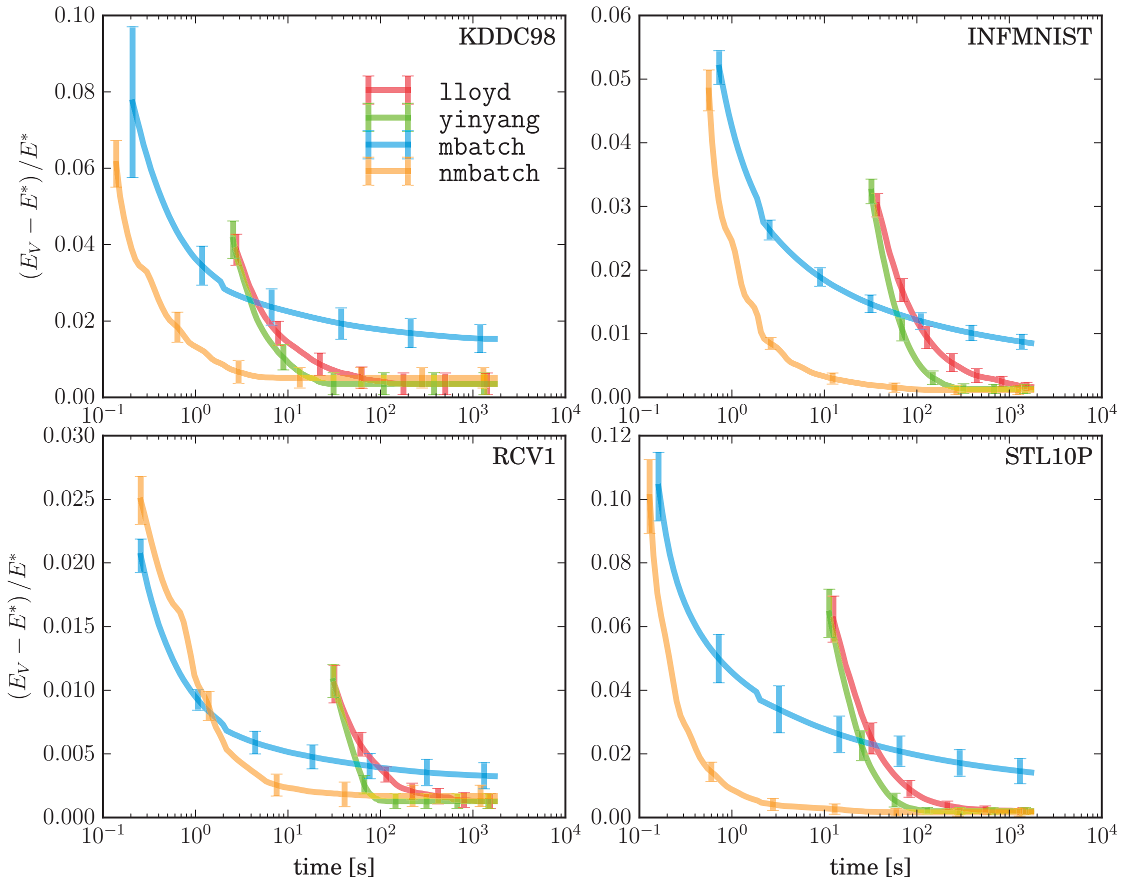

Figure 2.2 – The mean energy on validation data (E_V) relative to lowest energy (E^*) across 20 runs with standard deviations. Baselines are `lloyd`, `yinyang`, and `mbatch`, shown with the new algorithm `nmbatch` with $\rho = 100$. We see that `nmbatch` is consistently faster than all baselines, and obtains final minima very similar to those obtained by the exact algorithms. On the sparse dataset RCV1, the speed-up is noticeable within 0.5% of the empirical minimum E^*. On the three dense datasets, the speed-up over `mbatch` is between 10× and 100× at 2% of E^*, with even greater speed-ups below 2% where `nmbatch` converges very quickly to local minima.

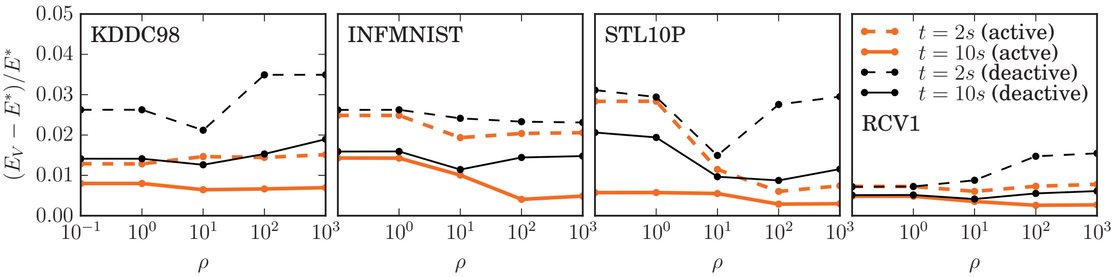

Figure 2.3 – Relative errors on validation data at $t \in \{2, 10\}$, for `nmbatch` with and with bound tests, for $\rho \in \{10^{-1}, 10^0, 10^1, 10^2, 10^3\}$. In the standard case of active bound testing, large values of ρ work well, as premature fine-tuning is less of a concern. However with the bound test deactivated, premature fine-tuning becomes costly for large ρ, and an optimal ρ value is one which trades off redundancy (ρ too small) and premature fine-tuning (ρ too large).

nested mini-batches is quite general, and applicable to other mini-batch algorithms. In particular, we believe that the sparse dictionary learning algorithm of Mairal et al. [2009] could benefit from nesting. One could also consider adapting nested mini-batches to stochastic gradient descent, although this is more speculative.

Celebi et al. [2013] show that specialised initialization schemes such as K-means++ can result in better clusterings. While this is not the case for the datasets we have used, it would be interesting to consider adapting such initialization schemes to the mini-batch context. It is not clear if using mini-batches, one can benefit as much from careful initialization.

Our nested mini-batch algorithm `nmbatch` uses a very simple bounding scheme. We believe that further improvements could be obtained through more advanced bounding as introduced in Chapter 1, and that the memory footprint of $O(KN)$ could be reduced by using a scheme where, as the mini-batch grows, the number of bounds maintained decreases, so that bounds on groups of clusters merge.

Comparing Baseline Implementations

We compare our implementation of `mbatch` with two publicly available implementations, that accompanying Sculley [2010] in C++, and that in scikit-learn [Pedregosa et al., 2011], written in Cython. Comparisons are presented in Table 2.1, where our implementations are seen to be competitive. Experiments were all single threaded. Our C++ and Python code is available at https://github.com/idiap/eakmeans.

INFMNIST (dense)		RCV1 (sparse)		
ours	sklearn	ours	sklearn	sofia
12.4	20.6	15.2	63.6	23.3

Table 2.1 – Comparing implementations of `mbatch` on INFMNIST (left) and RCV1 (right). Time in seconds to process N datapoints, where $N = 400,000$ for INFMNIST and $N = 781,265$ for RCV1. Implementations are our own (ours), that in scikit-learn (sklearn), and that of Sculley [2010] (sofia).

3 A Sub-Quadratic Exact Medoid Algorithm

Chapter introduction

A popular measure of the centrality of an element of a set is its mean distance to all other elements. In network analysis, this measure is referred to as *closeness centrality*, we will refer to it as *energy*. Given a set $\mathcal{S} = \{x(1), \dots, x(N)\}$ the energy of element $i \in \{1, \dots, N\}$ is thus given by,

$$E(i) = \frac{1}{N} \sum_{j \in \{1,\dots,N\}} \mathrm{dist}(x(i), x(j)).$$

Note the change in definition of energy from the K-means setting of the previous chapters where the distance was squared. An element in $\mathcal{S}$ with minimum energy is referred to as a *1-median* or a *medoid*. Without loss of generality, we will assume that $\mathcal{S}$ contains a unique medoid. The problem of determining the medoid of a set arises in the contexts of clustering, operations research, and network analysis. In clustering, the Voronoi iteration K-medoids algorithm [Hastie et al., 2001, Park and Jun, 2009] requires determining the medoid of each of K clusters at each iteration. In operations research, the facility location problem requires placing one or several facilities so as to minimize the cost of connecting to clients. In network analysis, the medoid may represent an influential person in a social network, or the most central station in a rail network.

Medoid algorithms and our contribution

A simple algorithm for obtaining the medoid of a set of N elements computes the energy of all elements and selects the one with minimum energy, requiring N^2 distance calculations. In certain settings $\Theta(N)$ algorithms exist, such as in 1-D where the problem is solved by Quickselect [Hoare, 1961], and more generally on trees. However, no general purpose $o(N^2)$ algorithm exists. An example illustrating the impossibility of such an algorithm is presented in Appendix C.1. Related to finding the medoid of a set is finding the *geometric median*, which in vector spaces is defined as the point in the vector space with minimum energy. The relationship between the two problems is discussed in Section 3.2.1.

Much work has been done to develop approximate algorithms in the context of network analysis. The `RAND` algorithm of Eppstein and Wang [2004] can be used to estimate the energy of all nodes in a graph. The accuracy of `RAND` depends on the diameter of the network, which motivated Cohen et al. [2014] to use pivoting to make `RAND` more effective for large diameter networks. The work most closely related to ours is that of Okamoto et al. [2008], where `RAND` is adapted to the task of finding the k lowest energy nodes, $k = 1$ corresponding to the medoid problem. The resulting `TOPRANK` algorithm of Okamoto et al. [2008] has complexity $\tilde{O}(N^{5/3})$ under certain assumptions, and returns the medoid with probability $1 - O(1/N)$, that is *with high probability* (w.h.p.). Note that only their run time result requires any assumption on the network data, obtaining the medoid w.h.p. is guaranteed. `TOPRANK` is discussed in Section 3.2.2.

In this chapter we present an algorithm which has expected run time $O(N^{3/2})$ under certain assumptions and always returns the medoid. In other words, we present an exact medoid algorithm with improved complexity over the state-of-the-art approximate algorithm, `TOPRANK`. We show through experiments that the new algorithm works well for low-dimensional data in $\mathbb{R}^d$ and for spatial network data. Our new medoid algorithm, which we call `trimed`, uses the triangle inequality to quickly eliminate elements which cannot be the medoid. The $O(N^{3/2})$ complexity follows from the surprising result that all but $O(N^{1/2})$ elements can be eliminated in this way.

The complexity bound on expected run time which we derive contains a term which grows exponentially in dimension d, and experiments show that in very high dimensions `trimed` often ends up computing $O(N^2)$ distances.

K-medoids algorithms and our contribution

The K-medoids problem is to partition a set into K clusters, so as to minimize the sum over elements of dissimilarites with their nearest medoids. That is, to choose $\mathcal{M} = \{m(1), \ldots, m(K)\} \subset \{1, \ldots, N\}$ to minimize,

$$\mathcal{L}(\mathcal{M}) = \sum_{i=1}^{N} \min_{k \in \{1,\ldots,K\}} \operatorname{diss}(x(i), x(m(k))).$$

In this chapter we focus on the special case where the dissimilarity is a distance (diss = dist), which is still more general than K-means which only applies to vector spaces. K-medoids is used in bioinformatics where elements are genetic sequences or gene expression levels [Chipman et al., 2003] and has been applied to clustering on graphs [Rattigan et al., 2007]. In machine vision, K-medoids is often preferred over K-means, as a medoid is more easily interpretable than a mean [Frahm et al., 2010].

The K-medoids problem is NP-hard, but there exist approximation algorithms. The Voronoi iteration algorithm, appearing in Hastie et al. [2001] and later in Park and Jun [2009], consists of alternating between updating medoids and assignments, much in the same way as Lloyd's algorithm works for the K-means problem. We will refer to it as `KMEDS`, and to Lloyd's K-means algorithm as `lloyd`.

One significant difference between `KMEDS` and `lloyd` is that the computation of a medoid is quadratic in the number of elements per cluster whereas the computation of a mean is linear. By incorporating our new medoid algorithm into `KMEDS`, we break the quadratic dependency of `KMEDS`, bringing it closer in performance to `lloyd`. We also show how ideas for accelerating `lloyd` presented in Elkan [2003] can be used in `KMEDS`.

It should be noted that algorithms other than `KMEDS` have been proposed for finding approximate solutions to the K-medoids problem, and we will show that they are very effective in Chapter 4. These include `PAM` and `CLARA` of Kaufman and Rousseeuw [1990], and `CLARANS` of Ng and Han [1994]. In this chapter we do not compare cluster qualities of these algorithms, but focus on accelerating the `lloyd` equivalent for K-medoids as a test setting for our medoid algorithm `trimed`.

Previous works

A related problem: the geometric median

A problem closely related to the medoid problem is the geometric median problem. In the vector space $\mathcal{V}$ the geometric median, assuming it is unique, is defined as,

$$g(\mathcal{S}) = \operatorname*{arg\,min}_{v \in \mathcal{V}} \left(\sum_{x \in \mathcal{S}} \|v - x\| \right). \tag{3.1}$$

While the medoid of a set is defined in any space with a distance measure, the geometric median is specific to vector spaces, where addition and scalar multiplication are defined. The convexity of the objective function being minimized in (3.1) has enabled the development of fast algorithms. In particular, Cohen et al. [2016] present an algorithm which obtains an estimate for the geometric median with relative error $1 + O(\epsilon)$ with complexity $O(Nd \log^3(\frac{N}{\epsilon}))$ in $\mathbb{R}^d$. In $\mathbb{R}^d$, one may hope that such an algorithm can be converted into an exact medoid algorithm, but it is not clear how to do this.

Thus, while it may be possible that fast geometric median algorithms can provide inspiration in the development of medoid algorithms, they do not work out of the box. Moreover, geometric median algorithms cannot be used for network data as they only work in vector spaces, thus they are not applicable for the spatial network datasets which we consider in Section 3.5.

Medoid algorithms : `TOPRANK` and `TOPRANK2`

In Eppstein and Wang [2004], the `RAND` algorithm for estimating the energy of all elements of a set $\mathcal{S} = \{x(1), \ldots, x(N)\}$ is presented. While `RAND` is presented in the context of graphs, where the N elements are nodes of an undirected graph and the metric is shortest path length, it can equally well be applied to any set endowed with a distance. The simple idea of `RAND` is to estimate the energy of each element from a sample of *anchor* nodes I, so that for $j \in \{1, \ldots, N\}$,

$$\hat{E}(j) = \frac{1}{|I|} \sum_{i \in I} \mathrm{dist}(x(j), x(i)).$$

An elegant feature of `RAND` in the context of sparse graphs is that Dijkstra's algorithm needs only be run from anchor nodes $i \in I$, and not from every node. The key result of Eppstein and Wang [2004] is the following. Suppose that $\mathcal{S}$ has diameter Δ, that is

$$\Delta = \max_{(i,j) \in \{1,\ldots,N\}^2} \mathrm{dist}(x(i), x(j)),$$

and let $\epsilon > 0$ be some error tolerance. If I is of size $\Omega(\log(N)/\epsilon)$, then $\mathbb{P}(|E(j) - \hat{E}(j)| > \epsilon\Delta)$ is $O\left(\frac{1}{N^2}\right)$ for all $j \in \{1, \ldots, N\}$. Using the union bound, this means there is a $O\left(\frac{1}{N}\right)$ probability that at least one energy estimate is off by more than $\epsilon\Delta$, and so we say that *with high probability* (w.h.p.) all errors are less than $\epsilon\Delta$.

`RAND` forms the basis of the `TOPRANK` algorithm of Okamoto et al. [2008]. Whereas `RAND` w.h.p. returns an element which has energy within ϵ of the minimum, `TOPRANK` is designed to w.h.p. return the true medoid. In motivating `TOPRANK`, Okamoto et al. [2008] observe that the expected difference between consecutively ranked energies is $O(\Delta/N)$, and so if one wishes to correctly rank all nodes, one needs to distinguish between energies at a scale $\epsilon = \Delta/N$, for which the result of Eppstein and Wang [2004] dictates that $\Theta(N \log N)$ anchor elements are required with `RAND`, which is more elements than $\mathcal{S}$ contains. However, to obtain just the highest ranked node should require less information than obtaining a full ranking of nodes, and it is to this task that `TOPRANK` is adapted.

The idea behind `TOPRANK` is to accurately estimate only the energies of promising elements. The algorithm proceeds in two passes, where in the first pass promising elements are earmarked. Specifically, the first pass runs `RAND` with $N^{2/3} \log^{1/3}(N)$ anchor elements to obtain $\hat{E}(i)$ for $i \in \{1, \ldots, N\}$, and then discards elements whose $\hat{E}(i)$ lie below threshold

τ given by,

$$\tau = \operatorname*{arg\,min}_{j \in \{1,\ldots,N\}} \hat{E}(j) + 2\hat{\Delta}\alpha' \left(\frac{\log n}{n}\right)^{\frac{1}{3}}, \tag{3.2}$$

where $\hat{\Delta}$ is an upper bound on Δ obtained from the anchor nodes, and α' is some constant satisfying $\alpha' > 1$. The second pass computes the true energy of the undiscarded elements, returning the one with lowest true energy. Note that a smaller α' value results in a lower (better) threshold, we discuss this point further in Appendix C.3.

To obtain run time guarantees, `TOPRANK` requires that the distribution of node energies is non-decreasing near to the minimum, denoted by E^*. More precisely, letting f_E be the probability distribution of energies, the algorithms require the existence of $\epsilon > 0$ such that,

$$E^* \leq \tilde{e} < e < E^* + \epsilon \implies f_E(\tilde{e}) \leq f_E(e). \tag{3.3}$$

If assumption 3.3 holds, then the run time is $\tilde{O}(N^{\frac{5}{3}})$. A second algorithm presented in Okamoto et al. [2008] is `TOPRANK2`, where the anchor set I is grown incrementally until some heuristic criterion is met. There is no runtime guarantee for `TOPRANK2`, although it has the potential to run much faster than `TOPRANK` under favourable conditions. Pseudocode for `RAND`, `TOPRANK` and `TOPRANK2` is presented in Appendix C.3.

K-medoids algorithm : `KMEDS`

The Voronoi iteration algorithm, which we refer to as `KMEDS`, is similar to `lloyd`, the main difference being that cluster medoids are computed instead of cluster means. It has been described in the literature at least twice, once in Hastie et al. [2001] and then in Park and Jun [2009], where a novel initialization scheme is developed. Pseudocode is presented in Appendix C.2.

All N^2 distances are computed and stored upfront with the `KMEDS` of Park and Jun [2009]. Then, at each iteration, KN comparisons are made during assignment and $\Omega(N^2/K)$ additions are made during medoid update. The initialization scheme of `KMEDS` requires all N^2 distances. Each iteration of `KMEDS` requires retrieving at least $\max\left(KN, N^2/K\right)$ distinct distances, as can be shown by assuming balanced clusters.

As an alternative to computing all distances upfront, one could store per-cluster distance matrices which get updated on-the-fly when assignments change. Using such an approach, the best one could hope for would be $\max\left(KN, N^2/K\right)$ distance calculations and $\Theta(N^2/K)$ memory. If one were to completely forego storing distances in memory and calculate distances only when needed, the number of distance calculations would be at least $r(KN + N^2/K)$, where r is the number of iterations.

The initialization scheme of Park and Jun [2009] selects K well centered elements as initial medoids. This scheme goes against the general wisdom for K-means initialization, where centroids are initialized to be well separated [Arthur and Vassilvitskii, 2007]. While the new scheme of Park and Jun [2009] performs well on a limited number of small 2-D datasets, we show in Apppendix C.1 that in general uniform initialization performs as well or better.

Our new medoid algorithm : `trimed`

We present our new algorithm, `trimed`, for determining the medoid of a given set $\mathcal{S} = \{x(1), \ldots, x(N)\}$. Whereas the approach with `TOPRANK` is to empirically *estimate* $E(i)$ for $i \in \{1, \ldots, N\}$, the approach with `trimed`, presented as Alg. 6, is to *bound* $E(i)$. When `trimed` terminates, an index $m^* \in \{1, \ldots, N\}$ has been determined, along with lower bounds $l(i)$ for all $i \in \{1, \ldots, N\}$, such that $E(m^*) \leq l(i) \leq E(i)$, and thus $x(m^*)$ is the medoid. The bounding approach uses the triangle inequality, as depicted in Figure 3.1.

Algorithm 6 The `trimed` algorithm for computing the medoid of $\{x(1), \ldots, x(N)\}$.

```
1:  $l \leftarrow \underline{0}_N$   // lower bounds on energies, maintained such that $l(i) \leq E(i)$ and initialized as $l(i) = 0$.
2:  $m^{cl}, E^{cl} \leftarrow -1, \infty$   // index of best medoid candidate found so far, and its energy.
3:  for $i \in$ shuffle $(\{1, \ldots, N\})$ do
4:      if $l(i) < E^{cl}$ then
5:          for $j \in \{1, \ldots, N\}$ do
6:              $d(j) \leftarrow \mathrm{dist}(x(i), x(j))$
7:          end for
8:          $l(i) \leftarrow \frac{1}{N}\sum_{j=1}^{N} d(j)$   // set $l(i)$ to be tight, that is $l(i) = E(i)$.
9:          if $l(i) < E^{cl}$ then
10:             $m^{cl}, E^{cl} \leftarrow i, l(i)$
11:         end if
12:         for $j \in \{1, \ldots, N\}$ do
13:             $l(j) \leftarrow \max(l(j), |l(i) - d(j)|)$   // using $E(i)$ and $\mathrm{dist}(x(i), x(j))$ to possibly improve bound on $E(j)$.
14:         end for
15:     end if
16: end for
17: $m^*, E^* \leftarrow m^{cl}, E^{cl}$ return $x(m^*)$
```

The algorithm `trimed` iterates through the N elements of $\mathcal{S}$ in a random order. Each time a new element with energy lower than the current lowest energy (E^{cl}) is found, the index of the current best medoid (m^{cl}) is updated (line 10). Lower bounds on energies are used to quickly eliminate poor medoid candidates (line 4). Specifically, if lower bound $l(i)$ on the energy of element i is greater than or equal to E^{cl}, then i is eliminated. If

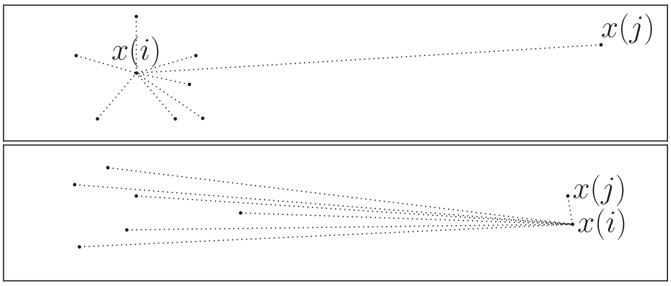

Figure 3.1 – Using the inequality $E(j) \geq |E(i) - \text{dist}(x(i), x(j))\,|$ to eliminate $x(j)$ as a medoid candidate. Computed element $x(i)$ with energy $E(i) \geq E^{cl}$ is used as a pivot to lower bound $E(j)$. The two cases where the inequality is effective are when (case 1, above) $\text{dist}(x(i), x(j)) - E(i) \geq E^{cl}$ and (case 2, below) $E(i) - \text{dist}(x(i), x(j)) \geq E^{cl}$, as both lead to $E(j) \geq E^{cl}$ which eliminates $x(j)$ as a medoid candidate.

the bound test fails to eliminate element i, then it is *computed*, that is, all distances to element i are computed (line 6). The computed distances are used to potentially improve lower bounds for all elements (line 13). Theorem 3.3.1 states that `trimed` finds the medoid. The proof relies on showing that lower bounds remain consistent when updated (line 13).

The algorithm is very straightforward to implement, and requires only two additional floating point values per datapoint: for sample i, one for $l(i)$ and one for $d(i)$. Computing either all or no distances from a sample makes particularly good sense for network data, where computing all distances to a single node is efficiently performed using Dijkstra's algorithm.

Theorem 3.3.1. *`trimed` returns the medoid of set $\mathcal{S}$.*

Proof. We need to prove that $l(j) \leq E(j)$ for all $j \in \{1, \dots, N\}$ at all iterations of the algorithm. Clearly, as $l(j) = 0$ at initialization, we have $l(j) \leq E(j)$ at initialization. $E(j)$ does not change, and the only time that $l(j)$ may change is on line 13, where we need to check that $|l(i) - d(j)| \leq E(j)$. At line 13, $l(i) = E(i)$ from line 8, and $d(j) = \text{dist}(x(i), x(j))$, so at line 13 we are effectively checking that $|E(i) - \text{dist}(x(i), x(j))| \leq E(j)$. But this is a simple consequence of the triangle inequality, as we now show. Using the definition, $E(j) = \frac{1}{N}\sum_{l=1}^{N} \text{dist}(x(l), x(j))$, we have on the one hand,

$$
\begin{aligned}
E(j) &\geq \frac{1}{N}\sum_{l=1}^{N} \text{dist}(x(l), x(i)) - \text{dist}(x(i), x(j)) \\
&\geq E(i) - \text{dist}(x(i), x(j)), \qquad (3.4)
\end{aligned}
$$

and on the other hand,

$$
\begin{aligned}
E(j) &\geq \frac{1}{N}\sum_{l=1}^{N} \mathrm{dist}(x(i), x(j)) - \mathrm{dist}(x(l), x(i)) \\
&\geq \mathrm{dist}(x(i), x(j)) - E(i). \qquad (3.5)
\end{aligned}
$$

Combining (3.4) and (3.5) we obtain the required inequality $|E(i) - \mathrm{dist}(x(i), x(j))| \leq E(j)$. ☐

The bound test (line 4) becomes more effective at later iterations, for two reasons. Firstly, whenever an element is computed, the lower bounds of other samples may increase. Secondly, E^{cl} will decrease whenever a better medoid candidate is found. The main result of this chapter, presented as Theorem 3.3.2, is that in $\mathbb{R}^d$ the expected number of computed elements is $O(N^{\frac{1}{2}})$ under some weak assumptions. We show in Section 3.5 that the $O(N^{\frac{1}{2}})$ result holds even in settings where the assumptions are not valid or relevent, such as for network data.

The shuffle on line 3 is performed to avoid w.h.p. pathological orderings, such as when elements are ordered in descending order of energy which would result in all N elements being computed.

Theorem 3.3.2. *Let $\mathcal{S} = \{x(1), \ldots, x(N)\}$ be a set of N elements in $\mathbb{R}^d$, drawn independently from probability distribution function f_X. Let the medoid of $\mathcal{S}$ be $x(m^*)$, and let $E(m^*) = E^*$. Suppose that there exist strictly positive constants ρ, δ_0 and δ_1 such that for any set size N with probability $1 - O(1/N)$*

$$
x \in \mathcal{B}_d(x(m^*), \rho) \implies \delta_0 \leq f_X(x) \leq \delta_1, \qquad (3.6)
$$

where $\mathcal{B}_d(x, r) = \{x' \in \mathbb{R}^d \ : \ \|x' - x\| \leq r\}$. Let $\alpha > 0$ be a constant (independent of N) such that with probability $1 - O(1/N)$ all $i \in \{1, \ldots, N\}$ satisfy,

$$
\begin{aligned}
&x(i) \in \mathcal{B}_d(x(m^*), \rho) \implies \qquad (3.7) \\
&\qquad E(i) - E^* \geq \alpha \|x(i) - x(m^*)\|^2.
\end{aligned}
$$

Then, the expected number of elements computed by `trimed` is $O\left(\left(V_d[1]\delta_1 + d\left(\frac{4}{\alpha}\right)^d\right) N^{\frac{1}{2}}\right)$, where $V_d[1] = \pi^{\frac{d}{2}} / (\Gamma(\frac{d}{2} + 1))$ is the volume of $\mathcal{B}_d(0, 1)$.

On the assumptions in theorem 3.3.2

The assumption of constants ρ, δ_0 and δ_1 made in Theorem 3.3.2 is weak, and only pathological distributions might fail it, as we now discuss. For the assumptions to fail requires that f_X vanishes or diverges at the distribution medoid. Any reasonably behaved

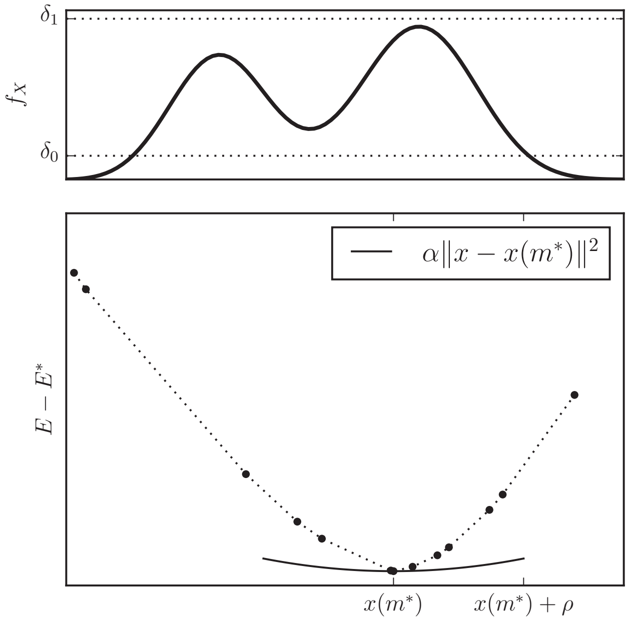

Figure 3.2 – Illustration in 1-D of the constants used in Theorem 3.3.2. Above, δ_0 and δ_1 bound the probability density function in a region containing the distribution medoid. Below, the energy of samples grows quadratically around the medoid $x(m^*)$. The energy E is a sum of cones centered on samples, which is approximately quadratic unless f_X vanishes or explodes, guaranteeing the existence of $\alpha > 0$ required in Theorem 3.3.2.

distribution does not have this behaviour, as illustrated in Figure 3.2. The constant α is a strong convexity constant. The existence of $\alpha > 0$ is guaranteed by the existence of ρ, δ_0 and δ_1, as the mean of a sum of uniformly spaced cones converges to a quadratic function. This is illustrated in 1-D in Figure C.2 in Appendix C.7, but holds true in any dimension.

Note that the assumptions made are on the distribution f_X, and not on the data itself. This allows us to prove the complexity result in N. We would like to formulate our result in terms of only the data, we are still investigating if this is possible.

Sketch of proof of theorem 3.3.2

We now sketch the proof of Theorem 3.3.2, showing how (3.6) and (3.7) are used. A full proof is presented in Appendix C.7. Firstly, let the index of the first element after the shuffle on line 3 be i'. Then, no elements beyond radius $2E(i')$ of $x(i')$ will subsequently be computed, due to type 1 eliminations (see Figure 3.1). Therefore, all computed elements are contained within $\mathcal{B}_d(x(i'), 2E(i'))$.

Next, notice that once an element $x(i)$ has been computed in `trimed`, no elements in the ball $\mathcal{B}_d(x(i), E(i) - E^{cl})$ will subsequently be computed, due to type 2 eliminations (see Figure 3.1). We refer to such a ball as an *exclusion ball.* By upper bounding the number of exclusion balls contained in $\mathcal{B}_d(x(i'), 2E(i'))$ using a volumetric argument, we can obtain a bound on the number of computed elements, but obtaining such an upper bound requires that the radii of exclusion ball $E(i) - E^{cl}$ be bounded below by a strictly positive value. However, by using a volumetric argument only beyond a certain positive radius of the medoid (a radius $N^{-1/2d}$), we have $\alpha > 0$ in (C.8) which provides a lower bound on exclusion ball radii, assuming $E^{cl} \approx E^*$. Using δ_0 we can show that E^{cl} approaches E^* sufficiently fast to validate the approximation $E^{cl} \approx E^*$.

It then remains to count the number of computed elements within radius $N^{-1/2d}$ of the medoid. One cannot find a strict upper bound here, but using the boundedness of f_X provided by δ_1, we have w.h.p. that the number of elements computed within $N^{-1/2d}$ is $O(\delta_1 N^{1/2})$, as the volume of a sphere scales as the d'th power of its radius.

Accelerated K-medoids algorithm : `trikmeds`

We adapt our new medoid algorithm `trimed` and borrow ideas from Elkan [2003] to show how `KMEDS` can be accelerated. We abandon the initial N^2 distance calculations, and only compute distances when necessary. The accelerated version of `lloyd` of Elkan [2003] maintains KN bounds on distances between points and centroids, allowing a large proportion of distance calculations to be eliminated. We use this approach to accelerate assignment in `trikmeds`, incurring a memory cost $O(KN)$. By adopting the exponion algorithm from Chapter 1, or that of Hamerly [2010], the memory overhead can be reduced to $O(N)$. We accelerate the medoid update step by adapting `trimed`, reusing lower bounds between iterations, so that `trimed` is only run from scratch once at the start. Details and pseudocode are presented in Appendix C.8.

One can relax the bound test in `trimed` so that for $\epsilon > 0$ element i is computed if $l(i)(1+\epsilon) < E^{cl}$, guaranteeing that an element with energy within a factor $1+\epsilon$ of E^* is found. It is also possible to relax the bound tests in the assignment step of `trikmeds`, such that the distance to an assigned cluster's medoid is always within a factor $1+\epsilon$ of the distance to the nearest medoid. We denote by `trikmeds`-ϵ the `trikmeds` algorithm where the update and assignment steps are relaxed as just discussed, with `trikmeds-0` being exactly `trikmeds`. The motivation behind such a relaxation is that, at all but the final few iterations, it is probably a waste of computation obtaining medoids and assignments at high resolution, as in subsequent iterations they may change.

Results

We first compare the performance of the medoid algorithms `TOPRANK`, `TOPRANK2` and `trimed`. We then compare the K-medoids algorithms, `KMEDS` and `trikmeds`.

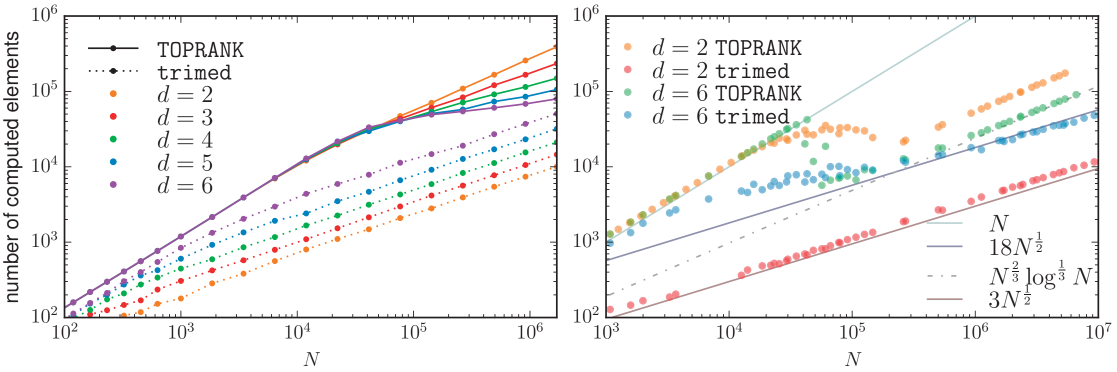

Figure 3.3 – Comparison of `TOPRANK` and our algorithm `trimed` on simulated data. On the left, points are drawn uniformly from $[0, 1]^d$ for $d \in \{2, \ldots, 6\}$, and on the right they are drawn from $\mathcal{B}_d(0, 1)$ for $d \in \{2, 6\}$, with an increased density near the edge of the ball. Fewer points (elements) are computed by `trimed` than by `TOPRANK` in all scenarios. For small N, `TOPRANK` computes $O(N)$ points, before transitioning to $\tilde{O}(N^{2/3})$ computed points for large N. `trimed` computes $O(N^{1/2})$ points. Note that `trimed` performs better in low-d than in high-d, with the reverse trend being true for `TOPRANK`. These observations are discussed in further detail in the text.

Medoid algorithm results

We compare our new exact medoid algorithm `trimed` with state-of-the-art approximate algorithms `TOPRANK` and `TOPRANK2`. Recall, Okamoto et al. [2008] prove that the approximate algorithms return w.h.p. the true medoid. We confirm that this is the case in all our experiments, where the approximate algorithms return the same element as `trimed`, which we know to be correct by Theorem 3.3.1. We now focus on comparing computational costs, which are proportional to the number of computed points.

Results on artificial datasets are presented in Figure 3.3, where our two main observations relate to scaling in N and dimension d. The artificial data are (left) uniformly drawn from $[0, 1]^d$ and (right) drawn from $\mathcal{B}_d(0, 1)$ with probability of lying within radius $1/2^{1/d}$ of $1/200$, as opposed to $1/2$ as would be the case under uniform density. Details about sampling from this distribution can be found in Appendix C.6. Results on a mix of publicly available real and artificial datasets are presented in Table 3.1 and discussed in Section 3.5.1.

Scaling with N and d on artificial datasets

In Figure 3.3 we observe that the number of points computed by `trimed` is $O(N^{1/2})$, as predicted by Theorem 3.3.2. This is illustrated (right) by the close fit of the number of computed points to exact square root curves at sufficiently large N for $d \in \{2, 6\}$.

Recall that `TOPRANK` consists of two passes, a first where $N^{2/3} \log^{1/3} N$ anchor points are computed, and a second where all sub-threshold points are computed. We observe that for small N `TOPRANK` computes all N points, which corresponds to all points lying below threshold. At sufficiently large N the threshold becomes low enough for all points to be eliminated after the first pass. The effect is particularly dramatic in high dimensions ($d = 6$ on right), where a phase transition is observed between all and no points being computed in the second pass.

Dimension d appears in Theorem 3.3.2 through a factor $d(4/\alpha)^d$, where α is the strong convexity of the energy at the medoid. In Figure 3.3, we observe that the number of computed points increases with d for fixed N, corresponding to a relatively small α. The effect of α on the number of computed elements is considered in greater detail in Appendix C.6.

In contrast to the above observation that the number of computed points increases as dimension increases for `trimed`, `TOPRANK` appears to scale favourably with dimension. This observation can be explained in terms of the distribution of energies, with energies close to E^* being less common in higher dimensions, as discussed in Appendix C.10.

Results on publicly available real and simulated datasets

We present the datasets used here in detail in Appendix C.9. For all datasets, algorithms `TOPRANK`, `TOPRANK2` and `trimed` were run 10 times with a distinct seed, and the mean number of iterations ($\hat{n}$) over the 10 runs was computed. We observe that our algorithm `trimed` is the best performing algorithm on all datasets, although in high-dimensions (MNIST-0) and on social network data (Gnutella) no algorithm computes significantly fewer than N elements. The failure in high-dimensions (MNIST-0) of `trimed` is in agreement with Theorem 3.3.2, where dimension appears as the exponent of a constant term. The small world network data, Gnutella, can be embedded in a high-dimensional Euclidean space, and thus the failure on this dataset can also be considered as being due to high-dimensions. For low-dimensional real and spatial network data, `trimed` consistently computes $O(N^{1/2})$ elements.

But who needs the exact medoid anyway?

A valid criticism that could be raised at this stage would be that for large datasets, finding the exact medoid is probably not necessary, as any point with energy reasonably close to E^* suffices for most applications. But consider, the `RAND` algorithm requires computing $\log N/\epsilon^2$ elements to confidently return an element with energy within ϵE^* of E^*. For $N = 10^5$ and $\epsilon = 0.05$, this is 4600, already more than `trimed` requires to obtain the exact medoid on low-d datasets of comparable size.

			`TOPRANK`	`TOPRANK2`	`trimed`
dataset	type	N	$\hat{n}$	$\hat{n}$	$\hat{n}$
Birch 1	2-d	1.0×10^5	57944	100180	**2180**
Birch 2	2-d	1.0×10^5	66062	100180	**2208**
Europe	2-d	1.6×10^5	176095	169535	**2862**
U-Sensor Net	u-graph	3.6×10^5	113838	327216	**1593**
D-Sensor Net	d-graph	3.6×10^5	99896	176967	**1372**
Pennsylvania road	u-graph	1.1×10^6	216390	time-out	**2633**
Europe rail	u-graph	4.6×10^4	35913	47041	**518**
Gnutella	d-graph	6.3×10^3	7043	6407	**6328**
MNIST	784-d	6.7×10^3	7472	6799	**6514**

Table 3.1 – Comparison of `TOPRANK`, `TOPRANK2` and our algorithm `trimed` on publicly available real and simulated datasets. Column 2 provides the type of the dataset, where 'x-d' denotes x-dimensional vector data, while 'd-graph' and 'u-graph' denote directed and undirected graphs respectively. Column $\hat{n}$ gives the mean number of elements computed over 10 runs. Our proposed `trimed` algorithm obtains the true medoid with far fewer computed points in low dimensions and on spatial network data. On the social network dataset (Gnutella) and the very high-d dataset (MNIST), all algorithms fail to provide speed-up, computing approximately N elements.

K-medoids algorithm results

With N elements to cluster, `KMEDS` of Park and Jun [2009] is $\Theta(N^2)$ in memory, rendering it unusable on even moderately large datasets. To compare the initialization scheme proposed in Park and Jun [2009] to random initialization, we have performed experiments on 14 small datasets, with $K \in \{10, \lceil N^{1/2} \rceil, \lceil N/10 \rceil\}$. For each of these 42 experimental set-ups, we run the deterministic `KMEDS` initialization once, and then uniform random initialization, 10 times. Comparing the mean final energy of the two initialization schemes, in only 9 of 42 cases does `KMEDS` initialization result in a lower mean final energy. A Table containing all results from these experiments in presented in Appendix C.5.

Having demonstrated that random uniform initialization performs at least as well as the initialization scheme of `KMEDS`, and noting that `trikmeds-0` returns exactly the same clustering as would `KMEDS` with uniform random initialization, we turn our attention to the computational performance of `trikmeds`. Table 3.2 presents results on 4 datasets, each described in Appendix C.9. The first numerical column is the relative number of distance calculations using `trikmeds-0` and `KMEDS`, where large savings in distance calculations, especially in low-dimensions, are observed. Columns ϕ_c and ϕ_E are the number of distance calculations and energies respectively, using $\epsilon \in \{0.01, 0.1\}$, relative to $\epsilon = 0$. We observe large reductions in the number of distance computations with only minor increases in energy.

			$K = 10$					$K = \lceil \sqrt{N} \rceil$				
			$\epsilon = 0$	$\epsilon = 0.01$		$\epsilon = 0.1$		$\epsilon = 0$	$\epsilon = 0.01$		$\epsilon = 0.1$	
Dataset	N	d	N_c/N^2	ϕ_c	ϕ_E	ϕ_c	ϕ_E	N_c/N^2	ϕ_c	ϕ_E	ϕ_c	ϕ_E
Europe	1.6×10^5	2	0.067	0.33	1.004	0.01	1.054	0.008	0.68	1.031	0.39	1.090
Conflong	1.6×10^5	3	0.042	0.67	1.001	0.08	1.014	0.006	0.92	1.003	0.61	1.026
Colormo	6.8×10^4	9	0.163	0.92	1.000	0.35	1.015	0.011	0.98	1.000	0.82	1.005
MNIST50	6.0×10^4	50	0.280	0.99	1.000	0.95	1.001	0.019	0.99	1.001	0.97	1.001

Table 3.2 – Relative numbers of distance calculations and final energies using `trikmeds-`ϵ for $\epsilon \in \{0, 0.01, 0.1\}$. The number of distance calculations with `trikmeds-0` is N_c, presented here relative to the number computed using `KMEDS` (N^2) in column N_c/N^2. The number of distance calculations with $\epsilon \in \{0.01, 0.1\}$ relative to `trikmeds-0` are given in columns ϕ_c, so $\phi_c = 0.33$ means 3× fewer calculations than with $\epsilon = 0$. The final energies with $\epsilon \in \{0.01, 0.1\}$ relative to `trikmeds-0` are given in columns ϕ_E. We see that `trikmeds-0` uses significantly fewer distance calculations than would `KMEDS`, especially in low-dimensions where a greater than $K\times$ reduction is observed ($N_C/N^2 < 1/K$). For low-d, additional relaxation further increases the saving in distance calculations with little cost to final energy.

Chapter conclusion and future work

We have presented our new `trimed` algorithm for computing the medoid of a set, and provided strong theoretical guarantees about its performance in $\mathbb{R}^d$. In low-dimensions, it outperforms the state-of-the-art approximate algorithm on a large selection of datasets. The algorithm is very simple to implement, and can easily be extended to the general ranking problem. In the future, we propose to explore the idea of using more complex triangle inequality bounds involving several points, with as goal to improve on the $O(N^{1/2})$ number of computed points. We would also like to derive an $O(N^{1/2})$ result which depends on the data alone, and not on the distribution it came from. This would be more elegant, and possibly more natural.

We have demonstrated how `trimed`, when combined with the approach of Elkan [2003], can greatly reduce the number of distance calculations required by the Voronoi iteration K-medoids algorithm of Park and Jun [2009]. In the future we would like to replace the strategy of Elkan [2003] with that of Hamerly [2010], which will be better adapted to graph clustering as either all or no distances are computed with it, making it more amenable to Dijkstra's algorithm.

4 K-medoids For K-means Seeding

Chapter introduction

Recall that the K-medoids problem differs from the K-means problem in that the center of a cluster is its medoid, not its mean, where the medoid is the cluster member which minimizes the sum of *dissimilarities* between itself and other cluster members. In this chapter, as our application is K-means seeding (initialization with samples), we focus on the case where dissimilarity is squared distance. Note that this is in contrast to the previous chapter where dissimilarity was exactly the distance. Note too that K-medoids generalises to non-metric spaces and arbitrary dissimilarity measures, as discussed in Appendix D.1.

Recall from previous chapters that the popular K-means Lloyd's algorithm, once again referred to as `lloyd` in this chapter, consists of an *assignment* step, where for each point the nearest (frozen) center is determined, and an *update* step, where each center is set to the mean of points assigned to it. By modifying the update step in `lloyd` to compute medoids instead of means, a viable K-medoids algorithm is obtained. This algorithm has been proposed at least twice [Hastie et al., 2001, Park and Jun, 2009] and is often referred to as the Voronoi iteration algorithm. In this chapter we continue to refer to it as `medlloyd`. The only difference between `medlloyd` and the `KMEDS` algorithm of the previous chapter is that `KMEDS` specifically minimizes the sum of distances, without any quadratic or other term. Indeed the bounding technique used in the `KMEDS` algorithm cannot be extended to the quadratic case, whereas the algorithms presented in this chapter are more widely applicable.

Another K-medoids algorithm is `clarans` of [Ng and Han, 1994], for which there is no direct K-means equivalent. It works by randomly proposing swaps between medoids and non-medoids, and accepting only those which decrease MSE. We will discuss how `clarans` works, what advantages it has over `medlloyd`, and our motivation for using it for K-means initialization in Section 4.2 and Appendix D.1.

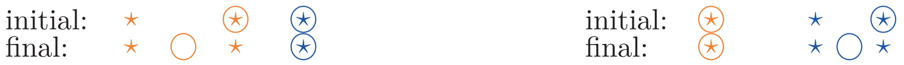

Figure 4.1 – $N = 3$ points, to be partitioned into $K = 2$ clusters with `lloyd`, with two possible initializations (top) and their solutions (bottom). Colors denote clusters, stars denote samples, rings denote means. Initialization with `clarans` enables jumping between the initializations on the left and right, ensuring that when `lloyd` eventually runs it avoids the local minimum on the left.

K-means initialization

`lloyd` is a *local* algorithm, in that far removed centers and points do not directly influence each other. This property contributes to `lloyd`'s tendency to terminate in poor minima if not well initialized. Good initialization is key to guaranteeing that the refinement performed by `lloyd` is done in the vicinity of a good solution, an example showing this is given in Figure 4.1.

In the comparative study of K-means initialization methods of Celebi et al. [2013], 8 schemes are tested across a wide range of datasets. Comparison is done in terms of speed (time to run initialization+`lloyd`) and energy (final MSE). They find that 3/8 schemes should be avoided, due to poor performance. One of these schemes is uniform initialization, henceforth `uni`, where K samples are randomly selected to initialize centers. Of the remaining 5/8 schemes, there is no clear best, with results varying across datasets, but the authors suggest that the algorithm of Bradley and Fayyad [1998], henceforth `bf`, is a good choice.

The `bf` scheme of Bradley and Fayyad [1998] works as follows. Samples are separated into J $(= 10)$ partitions. `lloyd` with `uni` initialization 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, initialized at each run with a distinct center set. The center set which obtains the lowest MSE on the superset is taken as the final initializer for the final run of `lloyd` on all N samples.

Probably the most widely implemented initialization scheme other than `uni` is K-means++ [Arthur and Vassilvitskii, 2007], henceforth `km++`. Its popularity stems from its simplicity, low computational complexity, 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 its distance to the nearest previously selected sample.

The work of Bachem et al. [2016] focused on developing sampling schemes to accelerate `km++`, while maintaining its theoretical guarantees. Their algorithm `afk-mc`^{2} results in as good initializations as `km++`, while using only a small fraction of the KN distance calculations required by `km++`. This reduction is important for massive datasets.

In none of the 4 schemes discussed is a center ever replaced once selected. Such refinement is only performed during the running of `lloyd`. In this chapter we show that performing refinement during initialization with `clarans`, before the final `lloyd` refinement, significantly lowers K-means MSEs.

Our contribution and chapter summary

We compare the K-medoids algorithms `clarans` and `medlloyd`, finding that `clarans` finds better local minima, in Section 4.3 and Appendix D.1. We offer an explanation for this, which motivates the use of `clarans` for initializing `lloyd` (Figure 4.2). We discuss the complexity of `clarans`, and briefly show how it can be optimized in Section 4.4, with a full presentation of acceleration techniques in Appendix D.4.

Most significantly, we compare `clarans` with methods `uni`, `bf`, `km++` and `afk-mc`^{2} for K-means initialization, and show that it provides significant reductions in initialization and final MSEs in Section 4.5. We thus provide a conceptually simple initialization scheme which is demonstrably better than `km++`, which has been the de facto initialization method for one decade now.

Our source code at https://github.com/idiap/zentas is available under an open source license. It consists of a C++ library with Python interface, with several examples for diverse data types (sequence data, sparse and dense vectors), metrics (Levenshtein, l_1, etc.) and potentials (quadratic as in K-means, logarithmic, etc.).

Other related works

Alternatives to `lloyd` have been considered which resemble the swapping approach of `clarans`. One is by Hartigan [1975], where points are randomly selected and reassigned. Telgarsky and Vattani [2010] show how this heuristic can result in better clustering when there are few points per cluster.

The work most similar to `clarans` in the K-means setting is that of Kanungo et al. [2002a], where it is indirectly shown that `clarans` finds a solution within a factor 25 of the optimal K-medoids clustering. The local search approximation algorithm they propose is a hybrid of `clarans` and `lloyd`, alternating between the two, with sampling from a kd-tree during the `clarans`-like step. Their source code includes an implementation of an algorithm they call ‘Swap’, which is exactly the `clarans` algorithm of Ng and Han [1994].

Two K-medoids algorithms

Like `km++` and `afk-mc`^{2}, K-medoids generalises beyond the standard K-means setting of Euclidean metric with quadratic potential, but we consider only the standard setting in the main body of this chapter, referring the reader to D.1 for a more general presentation. In Algorithm 7, `medlloyd` is presented. It is essentially `lloyd` with the update step modified for K-medoids. Alternatively, it is `KMEDS` of the previous chapter, with distance squared instead of just distance.

Algorithm 7 two-step iterative `medlloyd` algorithm (vector space, quadratic potential).

```
Initialize center indices $c(k)$, as distinct elements of $\{1, \ldots, N\}$, where index $k \in \{1, \ldots, K\}$.
do
    for $i = 1 : N$ do
        $a(i) \leftarrow \underset{k \in \{1,\ldots,K\}}{\arg\min} \|x(i) - x(c(k))\|^2$
    end for
    for $k = 1 : K$ do

        $c(k) \leftarrow \underset{i : a(i) = k}{\arg\min} \sum_{i' : a(i') = k} \|x(i) - x(i')\|^2$
    end for
while $c(k)$ changed for at least one $k$
```

Algorithm 8 swap-based `clarans` algorithm (vector space, quadratic potential).

```
1:  $n_r \leftarrow 0$
2:  Initialize center indices $\mathcal{C} \subset \{1, \ldots, N\}$
3:  $\psi_- \leftarrow \sum_{i=1}^{N} \min_{i' \in \mathcal{C}} \|x(i) - x(i')\|^2$
4:  while $n_r \leq N_r$ do
5:      sample $i_- \in \mathcal{C}$ and $i_+ \in \{1, \ldots, N\} \setminus \mathcal{C}$
6:
7:      $\psi_+ \leftarrow \sum_{i=1}^{N} \min_{i' \in \mathcal{C} \setminus \{i_-\} \cup \{i_+\}} \|x(i) - x(i')\|^2$
8:      if $\psi_+ < \psi_-$ then
9:          $\mathcal{C} \leftarrow \mathcal{C} \setminus \{i_-\} \cup \{i_+\}$
10:         $n_r \leftarrow 0, \quad \psi_- \leftarrow \psi_+$
11:     else
12:         $n_r \leftarrow n_r + 1$
13:     end if
14: end while
```

In Algorithm 8, `clarans` is presented. Following a random initialization of the K centers (line 2), it proceeds by repeatedly proposing a random swap (line 5) between a center

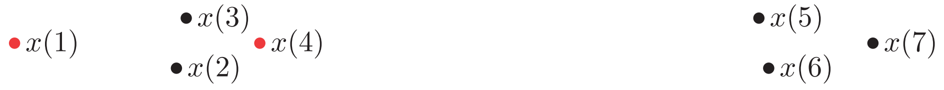

Figure 4.2 – Example with $N = 7$ samples, of which $K = 2$ are medoids. Current medoid indices are 1 and 4. Using `medlloyd`, this is a local minimum, with final clusters $\{x(1)\}$, and the rest. `clarans` may consider swap $(i_-, i_+) = (4, 7)$ and so escape to a lower MSE. The key to swap-based algorithms is that cluster assignments are never frozen. Specifically, when considering the swap of $x(4)$ and $x(7)$, `clarans` assigns $x(2), x(3)$ and $x(4)$ to the cluster of $x(1)$ *before* computing the new MSE.

(i_-) and a non-center (i_+). If a swap results in a reduction in energy (line 8), it is implemented (line 9). `clarans` terminates when N_r consecutive proposals have been rejected. Alternative stopping criteria could be number of accepted swaps, rate of energy decrease or time. We use $N_r = K^2$ throughout, as this makes proposals between all pairs of clusters probable, assuming balanced cluster sizes.

`clarans` was not the first swap-based K-medoids algorithm, being preceded by `pam` and `clara` of Kaufman and Rousseeuw [1990]. It can however provide better complexity than other swap-based algorithms if certain optimizations are used, as discussed in Section 4.4.

When updating centers in `lloyd` and `medlloyd`, assignments are frozen. In contrast, with swap-based algorithms such as `clarans`, assignments change along with the medoid index being changed (i_- to i_+). As a consequence, swap-based algorithms look one step further ahead when computing MSEs, which helps them escape from the minima of `medlloyd`. This is described in Figure 4.2.

A simple simulation study

We generate simple 2-D data, and compare `medlloyd`, `clarans`, and baseline K-means initializers `km++` and `uni`, in terms of MSEs. The data is described in Figure 4.3, where sample initializations are also presented. Results in Figure 4.4 show that `clarans` provides significantly lower MSEs than `medlloyd`, an observation which generalises across data types (genomic, sparse, etc), metrics (Levenshtein, l_∞, etc), and potentials (exponential, logarithmic, etc), as shown in Appendix D.1.

Complexity and accelerations

`lloyd` requires KN distance calculations to update K centers, assuming no acceleration technique such as that of Elkan [2003] is used. The cost of several iterations of `lloyd` outweighs initialization with any of `uni`, `km++` and `afk-mc`^{2}. We ask if the same is

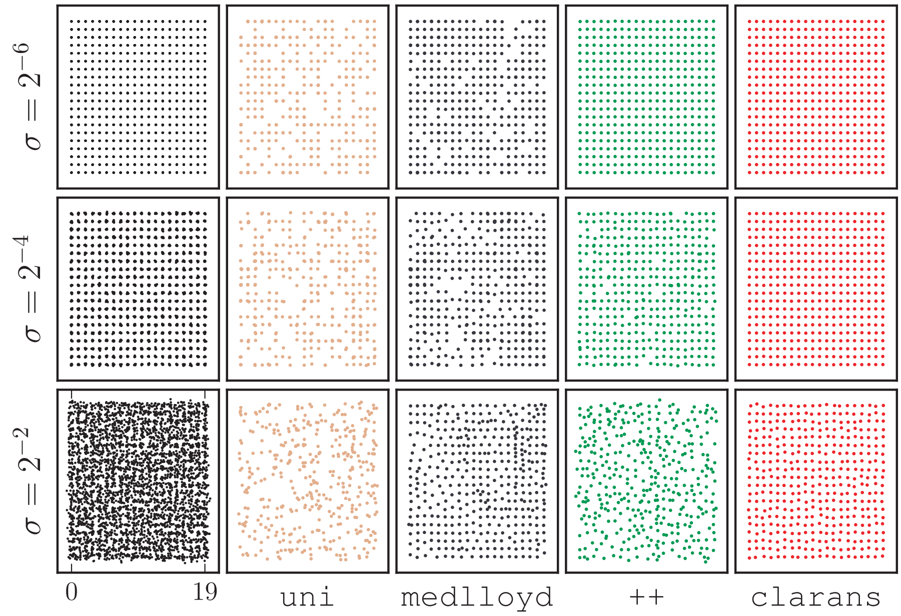

Figure 4.3 – (*Column 1*) Simulated data in $\mathbf{R}^2$. For each cluster center $g \in \{0, \ldots, 19\}^2$, 100 points are drawn from $\mathcal{N}(g, \sigma^2 I)$, illustrated here for $\sigma \in \{2^{-6}, 2^{-4}, 2^{-2}\}$. (*Columns 2,3,4,5*) Sample initializations. We observe 'holes' for methods `uni`, `medlloyd` and `km++`. `clarans` successfully fills holes by removing distant, under-utilised centers. The spatial correlation of `medlloyd`'s holes are due to its locality of updating.

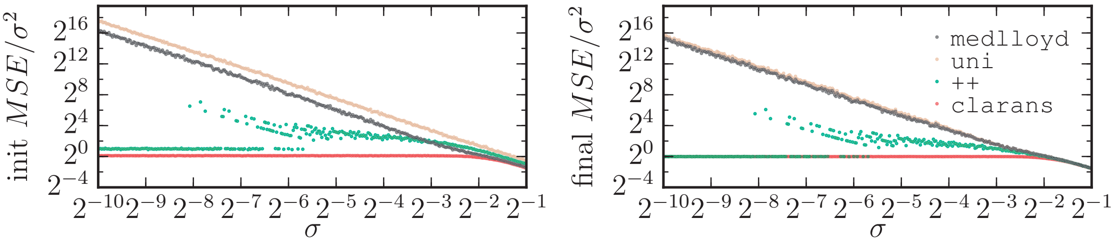

Figure 4.4 – Results on simulated data. For 400 values of $\sigma \in [2^{-10}, 2^{-1}]$, initialization (left) and final (right) MSEs relative to true cluster variances. For $\sigma \in [2^{-5}, 2^{-2}]$ `km++` never results in minimal MSE ($MSE/\sigma^2 = 1$), while `clarans` does for all σ. Initialization MSE with `medlloyd` is on average 4 times lower than with `uni`, but most of this improvement is regained when `lloyd` is subsequently run (final MSE/σ^2).

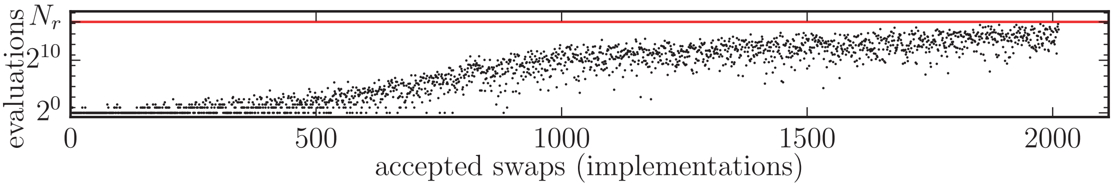

Figure 4.5 – The number of consecutive swap proposal rejections (evaluations) before one is accepted (implementations), for simulated data (Section 4.3) with $\sigma = 2^{-4}$.

true with `clarans` initialization, and find that the answer depends on how `clarans` is implemented. `clarans` as presented in Ng and Han [1994] is $O(N^2)$ in computation and memory, making it unusable for large datasets. To make `clarans` scalable, we have investigated ways of implementing it in $O(N)$ memory, and devised optimizations which make its complexity equivalent to that of `lloyd`.

`clarans` consists of two main steps. The first is swap *evaluation* (line 6) and the second is swap *implementation* (scope of if-statement at line 8). Proposing a good swap becomes less probable as MSE decreases, thus as the number of swap implementations increases the number of consecutive rejected proposals (n_r) is likely to grow large, illustrated in Figure 4.5. This results in a larger fraction of time being spent in the evaluation step.

We will now discuss optimizations in order of increasing algorithmic complexity, presenting their computational complexities in terms of evaluation and implementation steps. The explanations here are high level, with algorithmic details and pseudocode deferred to Appendix D.4.

Level -2 To evaluate swaps (line 6), simply compute all KN distances.

Level -1 Keep track of nearest centers. Now to evaluate a swap, samples whose nearest center is $x(i_-)$ need distances to all K samples indexed by $\mathcal{C} \setminus \{i_-\} \cup \{i_+\}$ computed in order to determine the new nearest. Samples whose nearest is not $x(i_-)$ only need the distance to $x(i_+)$ computed to determine their nearest, as either, (1) their nearest is unchanged, or (2) it is $x(i_+)$.

Level 0 Also keep track of second nearest centers, as in the implementation of Ng and Han [1994], which recall is $O(N^2)$ in memory and computes all distances upfront. Doing so, nearest centers can be determined for *all* samples by computing distances to $x(i_+)$. If swap (i_-, i_+) is accepted, samples whose new nearest is $x(i_+)$ require K distance calculations to recompute second nearests. Thus from level -1 to 0, computation is transferred from evaluation to implementation, which is good, as implementation is less frequently performed, as illustrated in Figure 4.5.

Level 1 Also keep track, for each cluster center, of the distance to the furthest cluster member as well as the maximum, over all cluster members, of the minimum distance

	-2	-1	0	1	2
1 evaluation	NK	N	N	$\frac{N}{K}+K$	$\frac{N}{K}$
1 implementation	1	1	N	N	N
K^2 evaluations, K implementations	K^3N	K^2N	K^2N	$NK+K^3$	KN
memory	N	N	N	N	$N+K^2$

Table 4.1 – The complexities at different levels of optimization of *evaluation* and *implementation*, in terms of required distance calculations, and overall memory. We see at level 2 that to perform K^2 evaluations and K implementations is $O(KN)$, equivalent to `lloyd`.

	-2	-1	0	1	2
$\log_2(\#\,\mathrm{dcs})$	44.1	36.5	35.5	29.4	26.7
time [s]	-	-	407	19.2	15.6

Table 4.2 – Total number of distance calculations (# dcs) and time required by `clarans` on simulation data of Section 4.3 with $\sigma = 2^{-4}$ at different optimization levels.

to another center. Using the triangle inequality, one can then frequently eliminate computation for clusters which are unchanged by proposed swaps with just a single center-to-center distance calculation. Note that using the triangle inequality requires that the K-medoids dissimilarity is metric based, as is the case in the K-means initialization setting.

Level 2 Also keep track of center-to-center distances. This allows whole clusters to be tagged as unchanged by a swap, without computing any distances in the evaluation step.

We have also considered optimizations which, unlike levels -2 to 2, do not result in the exact same clustering as `clarans`, but provide additional acceleration. One such optimization uses random sub-sampling to evaluate proposals, which helps significantly when N/K is large. Another optimization which is effective during initial rounds is to *not* implement the first MSE reducing swap found, but to rather continue searching for approximately as long as swap implementation takes, thus balancing time between searching (evaluation) and implementing swaps. Details can be found in Appendix D.4.3.

The computational complexities of these optimizations are in Table 4.1. Proofs of these complexities rely on there being $O(N/K)$ samples changing their nearest or second nearest center during a swap. In other words, for any two clusters of sizes n_1 and n_2, we assume $n_1 = \Omega(n_2)$. Using level 2 complexities, we see that if a fraction $p(\mathcal{C})$ of proposals reduce MSE, then the expected complexity is $O(N(1 + 1/(p(\mathcal{C})K)))$. One cannot marginalise $\mathcal{C}$ out of the expectation, as $\mathcal{C}$ may have no MSE reducing swaps, that is $p(\mathcal{C}) = 0$. If $p(\mathcal{C})$ is $O(K)$, we obtain complexity $O(N)$ per swap, which is equivalent to the $O(KN)$ for K center updates of `lloyd`. In Table 4.2, we consider run times and distance calculation counts on simulated data at the various levels of optimization.

dataset	#	N	dim	K	TL [s]
a1	1	3000	2	40	1.94
a2	2	5250	2	70	1.37
a3	3	7500	2	100	1.69
birch1	4	100000	2	200	21.13
birch2	5	100000	2	200	15.29
birch3	6	100000	2	200	16.38
ConfLong	7	164860	3	22	30.74
dim032	8	1024	32	32	1.13
dim064	9	1024	64	32	1.19
dim1024	10	1024	1024	32	7.68
europe	11	169308	2	1000	166.08

dataset	#	N	dim	K	TL [s]
housec8	12	34112	3	400	18.71
KDD*	13	145751	74	200	998.83
mnist	14	10000	784	300	233.48
Mopsi	15	13467	2	100	2.14
rna*	16	20000	8	200	6.84
s1	17	5000	2	30	1.20
s2	18	5000	2	30	1.50
s3	19	5000	2	30	1.39
s4	20	5000	2	30	1.44
song*	21	20000	90	200	71.10
susy*	22	20000	18	200	24.50
yeast	23	1484	8	40	1.23

Table 4.3 – The 23 datasets. Column 'TL' is time allocated to run with each initialization scheme, so that no new runs start after TL elapsed seconds. The starred datasets are those used in Bachem et al. [2016], the remainder are available at https://cs.joensuu.fi/sipu/datasets.

Results

We first compare `clarans` with `uni`, `km++`, `afk-mc`^{2} and `bf` on the first 23 publicly available datasets in Table 4.3 (datasets 1-23). As noted in Celebi et al. [2013], it is common practice to run initialization+`lloyd` several time and retain the solution with the lowest MSE. In Bachem et al. [2016] methods are run a fixed number of times, and *mean* MSEs are compared. However, when comparing *minimum* MSEs over several runs, one should take into account that methods vary in their time requirements.

Rather than run each method a fixed number of times, we therefore run each method as many times as possible in a given time limit, 'TL'. This dataset dependent time limit, given by columns TL in Table 4.3, is taken as 80× the time of a single run of `km++`+`lloyd`. The numbers of runs completed in time TL by each method are in columns 1-5 of Table 4.4. Recall that our stopping criterion for `clarans` is K^2 consecutively rejected swap proposals. We have also experimented with stopping criterion based on run time and number of swaps implemented, but find that stopping based on number of rejected swaps best guarantees convergence. We use K^2 rejections for simplicity, although have found that fewer than K^2 are in general needed to obtain minimal MSEs.

We use the fast `lloyd` implementation accompanying Newling and Fleuret [2016a] with the 'auto' flag set to select the best exact accelerated algorithm, and run until complete convergence. For initializations, we use our own C++/Cython implementation of level 2 optimized `clarans`, the implementation of `afk-mc`^{2} of Bachem et al. [2016], and `km++` and `bf` of Newling and Fleuret [2016a].

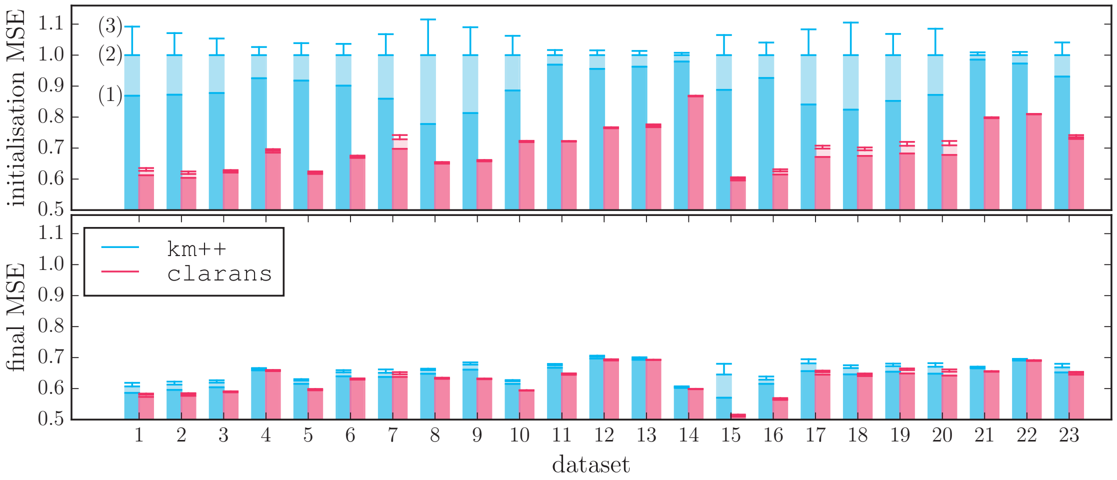

Figure 4.6 – Initialization (above) and final (below) MSEs for `km++` (left bars) and `clarans` (right bars), with minumum (1), mean (2) and mean + standard deviation (3) of MSE across all runs. For all initialization MSEs and most final MSEs, the lowest `km++` MSE is several standard deviations higher than the mean `clarans` MSE.

The objective of Bachem et al. [2016] was to prove and experimentally validate that `afk-mc`^{2} produces initialization MSEs equivalent to those of `km++`, and as such `lloyd` was not run during experiments. We consider both initialization MSE, as in Bachem et al. [2016], and final MSE after `lloyd` has run. The latter is particularly important, as it is the objective we wish to minimize in the *K*-means problem.

In addition to considering initialization and final MSEs, we also distinguish between mean and minimum MSEs. We believe the latter is important as it captures the varying time requirements, and as mentioned it is common to run `lloyd` several times and retain the lowest MSE clustering. In Table 4.4 we consider two MSEs, namely mean initialization MSE and minimum final MSE.

Baseline performance

We briefly discuss findings related to algorithms `uni`, `bf`, `afk-mc`^{2} and `km++`. Results in Table 4.4 corroborate the previously established finding that `uni` is vastly outperformed by `km++`, both in initialization and final MSEs. Table 4.4 results also agree with the finding of Bachem et al. [2016] that initialization MSEs with `afk-mc`^{2} are indistinguishable from those of `km++`, and moreover that final MSEs are indistinguishable. We observe in our experiments that runs with `km++` are faster than those with `afk-mc`^{2} (columns 1 and 2 of Table 4.4). We attribute this to the fast blas-based `km++` implementation of Newling and Fleuret [2016a].

Our final baseline finding is that MSEs obtained with `bf` are in general no better than

	runs completed					mean initial mse				minimum final mse				
	`km++`	`afkmc2`	`uni`	`bf`	`clarans`	`km++`	`afkmc2`	`uni`	`clarans`	`km++`	`afkmc2`	`uni`	`bf`	`clarans`
1	135	65	138	8	29	1	0.97	2	**0.63**	0.59	0.58	0.59	0.61	**0.57**
2	81	24	85	5	7	1	0.99	1.96	**0.62**	0.6	0.59	0.61	0.63	**0.58**
3	82	21	87	6	4	1	0.99	2.07	**0.63**	0.6	0.61	0.62	0.63	**0.59**
4	79	27	95	28	5	1	0.99	1.54	**0.69**	0.66	0.66	0.66	0.66	**0.66**
5	85	22	137	27	6	1	1	3.8	**0.62**	0.62	0.62	0.64	0.63	**0.59**
6	68	22	77	23	4	1	0.98	2.35	**0.67**	0.64	0.64	0.68	0.68	**0.63**
7	84	66	75	38	46	1	1	1.17	**0.73**	0.64	0.64	0.64	**0.64**	0.64
8	84	29	88	5	19	1	0.98	43.1	**0.65**	0.65	0.65	0.66	0.66	**0.63**
9	81	29	90	5	16	1	1.01	$>10^2$	**0.66**	0.66	0.66	0.66	0.69	**0.63**
10	144	52	311	24	18	1	0.99	$>10^2$	**0.72**	0.62	0.61	0.62	0.62	**0.59**
11	70	25	28	15	4	1	1	20.2	**0.72**	0.67	0.67	2.25	2.4	**0.64**
12	80	27	81	21	4	1	0.99	2.09	**0.77**	0.7	0.7	0.73	0.74	**0.69**
13	102	74	65	56	5	1	1	4	**0.77**	0.69	0.69	0.75	0.75	**0.69**
14	88	43	276	83	4	1	1	1	**0.87**	0.6	0.6	0.6	0.61	**0.6**
15	91	23	52	7	4	1	1	25	**0.6**	0.57	0.57	3.71	3.62	**0.51**
16	107	28	86	28	4	1	0.99	24.5	**0.62**	0.62	0.61	2.18	2.42	**0.56**
17	84	31	85	5	25	1	1.01	2.79	**0.7**	0.66	0.65	0.67	0.69	**0.65**
18	100	39	100	7	30	1	0.99	2.24	**0.69**	0.65	0.65	0.66	0.66	**0.64**
19	88	36	83	6	24	1	1.05	1.55	**0.71**	0.65	0.65	0.66	0.67	**0.65**
20	88	36	87	6	24	1	1.01	1.65	**0.71**	0.65	0.64	0.64	0.65	**0.64**
21	96	52	98	67	4	1	1	1.14	**0.8**	0.67	0.66	0.71	0.7	**0.65**
22	116	48	134	67	4	1	1	1.04	**0.81**	0.69	0.69	0.69	0.69	**0.69**
23	82	31	81	5	6	1	1	1.18	**0.74**	0.65	0.65	0.65	0.67	**0.64**
gm	90	34	93	14	8	1	1	4.71	**0.7**	0.64	0.64	0.79	0.8	**0.62**

Table 4.4 – Summary of results on the 23 datasets (rows). Columns 1 to 5 contain the number of initialization+`lloyd` runs completed in time limit TL. Columns 6 to 14 contain MSEs relative to the mean initialization MSE of `km++`. Columns 6 to 9 are mean MSEs after initialization but before `lloyd`, and columns 10 to 14 are minimum MSEs after `lloyd`. The final row (gm) contains geometric means of all columns. `clarans` consistently obtains the lowest across all MSE measurements, and has a 30% lower initialization MSE than `km++` and `afk-mc`^{2}, and a 3% lower final minimum MSE.

those with `uni`. This is not in strict agreement with the findings of Celebi et al. [2013]. We attribute this discrepancy to the fact that experiments in Celebi et al. [2013] are in the low K regime ($K < 50$, $N/K > 100$). Note that Table 4.4 does not contain initialization MSEs for `bf`, as `bf` does not initialize with data points but with means of sub-samples, and it would thus not make sense to compare `bf` initialization with the 4 seeding methods.

`clarans` performance

Having established that the best baselines are `km++` and `afk-mc`^{2}, and that they provide clusterings of indistinguishable quality, we now focus on the central comparison of this chapter, that between `km++` with `clarans`. In Figure 4.6 we present bar plots summarising all runs on all 23 datasets. We observe a very low variance in the initialization MSEs of `clarans`. We speculatively hypothesize that `clarans` often finds a globally minimal initialization. Figure 4.6 shows that `clarans` provides significantly lower initialization MSEs than `km++`.

The final MSEs are also significantly better when initialization is done with `clarans`, although the gap in MSE between `clarans` and `km++` is reduced when `lloyd` has run. Note, as seen in Table 4.4, that all 5 initializations for dataset 7 result in equally good clusterings.

As a final experiment, we have considered initialising with `km++` and `clarans` in series, thus using the three stage clustering `km++`+`clarans`+`lloyd`. We find that this can be slightly faster than just `clarans`+`lloyd` with identical MSEs. The slight speed-up is probably not significant enough to warrant the additional complexity. Results of this experiment are presented in Appendix D.9.

Chapter conclusion and future works

In this chapter, we have demonstrated the effectiveness of the algorithm `clarans` at solving the K-medoids problem. We have described techniques for accelerating `clarans`, and shown that `clarans` works very effectively as an initializer for `lloyd`, outperforming other initialization schemes, such as `km++`, on 23 datasets.

An interesting direction for future work might be to develop further optimizations for `clarans`. One idea could be to use importance sampling to rapidly obtain good estimates of post-swap energies. Another might be to propose two swaps simultaneously, as considered in Kanungo et al. [2002a], which could potentially lead to even better solutions, although we have hypothesized that `clarans` is already finding globally optimal initializations.

Connecting with Chapter 2, an important line of future research is incorporating this and other initialization schemes into mini-batch K-means algorithms. The main challenge will be to prevent good initializations from being washed out by noise inherent to the mini-batch approach.

All source code is made available under a public license. It consists of generic C++ code which can be extended to various data types and metrics, compiling to a shared library with extensions in Cython for a Python interface. It can be found in the git repository https://github.com/idiap/zentas.

5 Conclusions and Future Works

The common theme running through this dissertation has been the application of the triangle inequality to reduce the number of distance calculations in energy based clustering algorithms. The special property that these algorithms have which enables the elimination of certain center-sample distance calculations is that the energy depends only on the distance from a sample to its *nearest* center: a small fraction of all center-sample distances.

We ask, what other machine learning algorithms have this special property, and may benefit from similar techniques? Closely related to clustering is Gaussian Mixture Modeling, where the core algorithm iterates between expectation (assignment) and minimization (update) steps. However, in the expectation step, all center-sample Mahalanobis distances are required, suggesting that exact GMM cannot be accelerated through triangle inequality bounding.

Another related algorithm is dictionary learning with matching pursuit. This can be thought of as clustering, where each sample is assigned to a small number of clusters simultaneously, and so it seems likely that triangle inequality bounding can be used in this setting. Other situations where only partial distance information is required come from neural networks; for example a max-pooling layer only propagates the largest inner product, and the rectified linear unit (ReLU) layer sets negative values to zero.

Another idea we have investigated in this dissertation which may have applications beyond clustering is nesting mini-batches, the topic of Chapter 2. In any setting where computation with recently seen samples is cheap, it might be possible to adapt the idea of nesting batches. Besides reducing computation, preferentially using recently used data can result in reduced memory I/O and improved cache memory use.

In summary, the number of innovative ways in which the triangle inequality can be used to reduce computation for clustering and other tasks in machine learning appears vast, and much remains to be explored. Finally, we list the main findings of this dissertation.

1. The triangle inequality can be used in a large number of complex and complementary ways to accelerate simple algorithms such K-means. We showed in the first chapter that by storing historical centroid positions, one can greatly reduce the number of distance calculations. We also showed that in low-dimensions, by grouping centroids in concentric shells one can perform exact K-means with a greatly reduced computational cost.

2. Nested mini-batch sampling, an alternative to random mini-batch sampling where samples are preferentially reused, is very effective when the reuse of recently used samples is cheap. We presented this idea in the nested mini-batch K-means algorithm, where reuse of 'hot' samples is made cheap by the use of triangle inequality bounding.

3. Computing the exact medoid of a set of N points has complexity $O(N^{3/2})$, under certain assumptions. We presented a simple algorithm which uses the triangle inequality to do this. The complexity result is proved in $\mathbb{R}^d$. Complexity grows exponentially in d, and so the algorithm is most applicable in low dimensions. We showed how the algorithm performs well for spatial network data, scaling as $O(N^{3/2})$.

4. We considered a family of clustering algorithms which proceed by swapping a center with a non-center, and showed that this approach results in better minima than the Voronoi clustering algorithms. We considered using the swap-based `clarans` algorithm to initialize K-means, and showed that doing so results in lower energies than when other schemes such as K-means++ are used. The triangle inequality was used to accelerate the algorithm, with provable reductions in complexity and a clear empirical speed-up.

A Appendix for Chapter 1

Proofs

We will use subscripts to denote rounds of k-means, and $B(x, r)$ to denote the closed ball centered on x of radius r.

Proof of correctness of Elkan's algorithm update

By the definition of the lower bound update,

$$l_{t_0+1}(i, j) = l_{t_0}(i, j) - p_{t_0}(j).$$

Using that l_{t_0} is a valid bound, the definition of p_{t_0}, and the triangle inequality,

$$\begin{aligned}
&\leq \|x(i) - c_{t_0}(j)\| - p_{t_0}(j), \\
&\leq \|x(i) - c_{t_0}(j)\| - \|c_{t_0}(j) - c_{t_0+1}(j)\| \\
&\leq \|x(i) - c_{t_0+1}(j)\|.
\end{aligned}$$

Thus the lower bound update is valid. Similarly for the upper bound,

$$\begin{aligned}
u_{t_0+1}(i, j) &= u_{t_0}(i) + p_{t_0}(a(i)), \\
&\geq \|x(i) - c_{t_0}(a(i))\| + p_{t_0}(a(i)), \\
&\geq \|x(i) - c_{t_0}(a(i))\| + \|c_{t_0}(a(i)) - c_{t_0+1}(a(i))\|, \\
&\geq \|x(i) - c_{t_0+1}(a(i))\|.
\end{aligned}$$

This proves that the upper bound update is valid.

Proof of correctness of Elkan's algorithm inter-centroid test

Suppose that,

$$\frac{cc(a(i), j)}{2} > u(i).$$

Then, by the triangle inequality and previous definitions,

$$\begin{aligned}\|c(j) - x(i)\| &\geq \|c(j) - c(a(i))\| - \|c(a(i)) - x(i)\|,\\ &\geq cc(a(i), j) - u(i),\\ &\geq 2u(i) - u(i),\\ &\geq u(i).\end{aligned}$$

Thus $c(a(i))$ is nearer to $x(i)$ than $c(j)$ is, and so $j \neq n_1(i)$.

Proof of correctness of Annular algorithm test

Recall the definition of $R(i)$,

$$R(i) = \max\left(u(i), \|x(i) - c(b(i))\|\right).$$

Following directly from this definition and the definition of $u(i)$, we have $c(a(i)), c(b(i)) \in B(x(i), R(i))$. Therefore by the definitions of $n_1(i)$ and $n_2(i)$, we have that $c(n_1(i)), c(n_2(i)) \in B(x(i), R(i))$. The triangle inequality now provides

$$|\|c(j)\| - \|x(i)\|| > R(i) \implies \|c(j) - x(i)\| > R(i), \tag{A.1}$$

Thus by the definition of $\mathcal{J}(i)$,

$$\mathcal{J}(i) = \{j : |\|c(j)\| - \|x(i)\|| \leq R(i)\},$$

we can say,

$$j \notin \mathcal{J}(i) \implies j \notin \{n_1(i), n_2(i)\}.$$

Proof of correctness of Exponion algorithm test

Let $nn(j) \in \{1, \ldots, k\} \setminus \{j\}$ denote the index the cluster whose centroid is nearest to the centroid of cluster j other than j, that is the centroid at distance $s(j)$ from centroid j.

By definitions we have

$$\begin{aligned} c(a(i)) &\in B(x(i), u(i)), \\ c(nn(a(i))) &\in B(c(a(i)), s(a(i))). \end{aligned}$$

Combining these we have

$$c(a(i)), c(nn(a(i))) \in B(x(i), u(i) + s(a(i))), \tag{A.2}$$

Basic geometric arguments provide

$$B\left(x(i), u(i) + s(a(i))\right) \subseteq B(c(a(i)), 2u(i) + s(a(i))). \tag{A.3}$$

From (A.2) we deduce that

$$c(n_1(i)), c(n_2(i)) \in B(x(i), u(i) + s(a(i))),$$

and hence by (A.3) we have

$$c(n_1(i)), c(n_2(i)) \in B(c(a(i)), 2u(i) + s(a(i))),$$

completing the proof.

Proof that ns upper-bound is tighter than sn upper-bound

$$\begin{aligned} u^{ns}_{t_0+\delta t}(i) &= u_{t_0}(i) + \left\| \sum_{t'=t_0}^{t_0+\delta t-1} c_{t'+1}(i) - c_{t'}(i) \right\|, \\ &\leq u_{t_0}(i) + \sum_{t'=t_0}^{t_0+\delta t-1} \| c_{t'+1}(i) - c_{t'}(i) \|, \\ &\leq u^{sn}_{t_0+\delta t}(i). \end{aligned}$$

Detailed descriptions

The inner Yinyang test

We need some temporary notation to present the test which the Yinyang algorithm employs,

$$\begin{aligned}
j_1(i,f) &= \operatorname*{arg\,min}_{j \in \mathcal{G}(f)} \|x(i) - c(j)\|, \\
j_2(i,f) &= \operatorname*{arg\,min}_{j \in \mathcal{G}(f) \setminus \{j_1(f)\}} \|x(i) - c(j)\|, \\
r_2(i,f) &= \|x(i) - c(j_2(f))\|.
\end{aligned}$$

The Yinyang test hinges on the fact that centroids in $\mathcal{G}(f)$ which lie beyond radius $r_2(i,f)$ of $x(i)$ do not affect the variable updates and can thus be ignored. Extending this, suppose we have bounds $\tilde{r}_2(i,f)$ and $\tilde{l}(i,j)$ for $j \in \mathcal{G}$ such that $\tilde{r}_2(i,f) > r_2(f)$ and $\tilde{l}(i,j) < \|x(i) - c(j)\|$. Then $\tilde{r}_2(i,f) < \tilde{l}(i,j)$ means that centroid j can be ignored. It remains to define relevant bounds $\tilde{r}_2(i,f)$ and $\tilde{l}(i,j)$.

For $\tilde{r}_2(i,f)$, one keeps track of the second nearest centroid found thus far while looping over the centroids in $\mathcal{G}(f)$. Then for $\tilde{l}(i,j)$ we could take $l(i,f)$, but a better choice is $\tilde{l}(i,j) - q(f) + p(j)$, which replaces the maximum group displacement in the last round with the exact displacement of centroid j.

The Yinyang test to determine whether centroid j needs be considered is thus finally,

$$\begin{aligned}
&l(i,f) - q(f) + p(j) > \tilde{r}_2(i,f) \implies \\
&\qquad \text{centroid } j \text{ lies beyond radius } r_2 \text{, can be ignored.}
\end{aligned} \tag{A.4}$$

SMN, MSN, MNS

A lower bound to at time $t_0 + \delta_t$ on the distance from $x(i)$ to a group of centroids with group index f can be computed in three different ways. Letting Δ_{t_0,δ_t} denote the update term in

$$l_{t_0+\delta_t}(i,f) = \min_{j \in \mathcal{G}(i)} \left(\|x(i) - c_{t_0}(j)\|\right) - \Delta_{t_0,\delta_t},$$

	Data set	d	N
i	birch	2	100,000
ii	europe	2	169,300
iii	urand2	2	1,000,000
iv	ldfpads	3	164,850
v	conflongdemo	3	164,860
vi	skinseg	4	200,000
vii	tsn	4	200,000
viii	colormoments	9	68,040
ix	mv	11	40,760
x	wcomp	15	165,630
xi	house16h	17	22,780
xii	keggnet	28	65,550
xiii	urand30	30	1,000,000
xiv	mnist50	50	60,000
xv	miniboone	50	130,060
xvi	covtype	55	581,012
xvii	uscensus	68	2,458,285
xviii	kddcup04	74	145,750
xix	stl10	108	1,000,000
xx	gassensor	128	13,910
xxi	kddcup98	310	95,000
xxii	mnist784	784	60,000

Table A.1 – Full names of the 22 datasets used. All datasets are preprocessed such that features have mean zero and variance 1.

the three possibilities are

$$\begin{aligned}
\Delta^{SMN}_{t_0,\delta_t} &= \sum_{t'=t_0}^{t_0+\delta t-1} \max_{j\in\mathcal{G}(i)} \left(\|c_{t'+1}(j) - c_{t'}(j)\|\right), \\
\Delta^{MSN}_{t_0,\delta_t} &= \max_{j\in\mathcal{G}(i)} \left(\sum_{t'=t_0}^{t_0+\delta t-1} \|c_{t'+1}(j) - c_{t'}(j)\| \right), \\
\Delta^{MNS}_{t_0,\delta_t} &= \max_{j\in\mathcal{G}(i)} \left(\|c_{t_0+\delta_t}(j) - c_{t_0}(j)\|\right).
\end{aligned}$$

The term $\Delta^{SMN}_{t_0,\delta_t}$ corresponds to the classic approach used in all previous works. The term $\Delta^{MSN}_{t_0,\delta_t}$ corresponds to an intermendiate where improved bounds can be obtained without storing centroids. The term $\Delta^{MNS}_{t_0,\delta_t}$ corresponds to the approach providing the tightest bounds, and is the one we use throughout.

Results tables

	mean iterations	SD iterations	mean fastest [s]	SD fastest	`bay-sta`	`mlp-sta`	`pow-sta`	`vlf-sta`	`own-sta`	`bay-ham`	`mlp-ham`	`own-ham`	`bay-ann`	`own-ann`	`own-exp`	`own-exp-ns`	`own-syin`	`own-syin-ns`	`pow-yin`	`own-yin`	`own-selk`	`own-selk-ns`	`bay-elk`	`mlp-elk`	`vlf-elk`	`own-elk`	`own-elk-ns`
i	139	35.4	0.26	0.04	62.2	44.7	105	33.7	19.2	5.75	4.87	4.67	1.72	2.19	1.05	**1.00**	1.91	2.01	14.9	1.98	9.37	8.42	14.8	11.1	21.3	7.53	5.55
ii	335	132	1.69	0.25	38.2	28.1	60.3	21.5	10.3	6.44	5.00	5.05	1.71	1.99	1.07	**1.00**	3.05	3.13	14.6	2.96	6.08	5.60	10.8	8.70	15.9	6.94	6.70
iii	455	187	5.23	1.47	100	76.4	153	56.4	29.9	4.00	4.25	3.35	1.75	1.99	1.05	**1.00**	2.04	1.95	23.0	2.33	15.1	13.6	26.6	19.1	34.1	12.3	8.67
iv	270	96.3	2.17	0.48	29.0	17.9	37.7	15.2	6.79	3.99	2.90	2.51	2.03	1.64	1.04	**1.00**	1.02	1.06	6.78	1.09	3.66	3.30	7.09	6.68	11.0	4.54	3.80
v	277	73.6	2.35	0.48	27.5	18.1	35.7	14.4	6.80	4.07	3.06	2.55	2.05	1.66	1.04	**1.00**	1.01	1.05	6.48	1.08	3.47	3.14	6.72	6.57	10.5	4.31	3.63
vi	235	107	1.76	0.61	44.3	25.7	55.6	17.7	9.96	4.68	3.34	2.63	2.23	1.69	1.05	**1.00**	1.20	1.21	8.45	1.32	4.76	4.29	8.79	8.22	13.4	5.16	4.20
vii	117	25.2	1.32	0.18	28.9	15.9	34.2	11.7	5.85	4.29	2.83	2.42	2.61	1.76	1.25	1.19	1.02	**1.00**	5.95	1.06	3.24	2.89	5.72	4.90	8.82	3.19	2.60
viii	203	48.4	1.19	0.20	31.4	14.1	26.3	10.3	5.61	6.47	3.33	2.42	5.08	2.14	2.39	2.24	1.02	**1.00**	5.11	1.28	2.25	2.02	6.22	6.38	10.0	4.10	3.09
ix	127	35.4	0.49	0.07	32.6	14.7	25.5	12.0	5.02	7.80	4.02	2.66	7.99	2.75	2.63	2.51	1.02	**1.00**	4.83	1.32	2.12	1.89	6.12	6.12	9.77	3.43	2.37
x	130	27.2	2.16	0.46	38.7	17.2	29.2	12.0	5.11	10.7	5.21	2.97	4.90	1.83	1.23	1.21	**1.00**	1.03	4.56	1.39	1.89	1.67	3.29	3.28	5.46	2.00	1.67
xi	111	16.6	0.52	0.07	20.8	9.62	14.0	5.36	2.82	9.34	4.89	2.40	5.96	1.88	2.33	2.22	1.11	1.07	3.75	1.73	1.08	**1.00**	2.90	3.30	4.34	2.07	1.90
xii	113	33.4	0.92	0.14	53.3	23.3	30.2	10.3	4.86	12.0	4.21	2.11	5.58	1.47	1.82	1.75	**1.00**	1.00	4.36	1.74	1.66	1.46	5.26	3.64	4.84	1.95	1.60
xiii	2576	110	156	1.86	t	t	t	t	t	9.15	4.61	1.85	9.26	1.97	1.82	1.70	1.04	**1.00**	6.06	1.58	3.37	3.04	4.70	5.40	t	3.46	3.16
xiv	152	58.3	1.72	0.33	67.4	21.6	29.3	10.7	5.60	17.8	5.76	2.47	18.2	2.61	2.44	2.34	1.01	**1.00**	4.00	2.02	1.47	1.25	3.04	2.66	4.72	1.74	1.39
xv	376	90.1	7.18	1.13	87.5	27.2	37.9	13.7	6.97	25.4	7.84	3.40	15.3	2.44	3.27	3.09	**1.00**	1.06	4.53	2.05	1.54	1.38	4.20	4.22	5.72	2.84	2.76
xvi	127	35.0	8.45	0.94	120	38.9	51.4	20.3	10.8	18.1	6.34	2.67	15.2	2.51	1.92	1.85	**1.00**	1.04	4.72	2.05	1.85	1.60	3.38	4.01	5.73	2.01	1.58
xvii	102	38.6	43.9	6.73	t	28.3	m	14.0	6.09	19.5	6.11	2.55	17.5	2.47	2.34	2.26	1.00	**1.00**	m	2.29	1.37	m	3.27	3.44	4.77	1.83	m
xviii	334	118	13.5	3.46	63.9	18.7	26.0	9.58	4.85	23.8	6.90	2.89	22.3	2.90	2.86	2.81	1.14	1.12	4.63	2.75	1.13	**1.00**	2.51	1.91	3.51	1.31	1.16
xix	408	145	92.5	27.0	t	t	t	t	7.18	t	5.77	2.29	t	2.38	2.16	2.10	1.05	1.10	4.35	2.92	1.14	**1.00**	2.59	2.74	3.85	1.58	1.60
xx	57.3	15.9	0.23	0.05	101	28.2	40.0	14.6	6.73	33.7	9.58	3.65	15.8	2.27	3.37	3.25	1.72	1.78	6.14	4.56	1.12	1.13	2.53	2.21	2.41	**1.00**	1.02
xxi	178	58.8	9.96	1.97	121	30.1	44.0	16.7	7.98	41.4	10.5	3.72	42.1	3.94	3.66	3.57	1.47	1.47	5.76	4.64	1.18	**1.00**	3.10	2.07	2.39	1.32	1.14
xxii	131	25.9	9.89	1.23	143	34.8	50.1	20.0	10.0	40.1	10.2	3.97	40.6	4.10	3.87	3.75	1.29	1.34	4.23	3.64	1.25	**1.00**	3.63	1.82	1.71	1.27	1.03

Table A.2 – Results with $k = 100$ by dataset (rows). Columns 6 to the end contain mean times over the 10 initializations, relative to the fastest algorithm, that is the algorithm with the lowest mean time, corresponding to the entry **1.00**. The mean and standard deviation of the number of iterations to convergence are given in columns 2 and 3. The mean and standard deviation of the time of the fastest algorithm are given in columns 4 and 5. ‘t’ and ‘m’ correspond to timeout (40 minutes) and memory (4 GB) failures respectively. The fastest implementation for all data sets is always `own`. The fastest non-`own` implementation for each data set is underlined, where non-`own` implementations correspond to white columns.

	mean iterations	SD iterations	mean fastest [s]	SD fastest	`bay-sta`	`mlp-sta`	`pow-sta`	`vlf-sta`	`own-sta`	`bay-ham`	`mlp-ham`	`own-ham`	`bay-ann`	`own-ann`	`own-exp`	`own-exp-ns`	`own-syin`	`own-syin-ns`	`pow-yin`	`own-yin`	`own-selk`	`own-selk-ns`	`bay-elk`	`mlp-elk`	`vlf-elk`	`own-elk`	`own-elk-ns`
i	120	20.5	2.80	0.32	48.0	32.0	104	25.4	12.0	12.5	8.50	8.29	1.49	1.41	1.01	**1.00**	1.01	1.27	8.53	1.13	6.83	6.32	11.5	17.5	16.3	5.54	4.32
ii	533	66.7	12.1	1.22	82.6	56.0	t	43.7	21.6	19.0	13.0	12.5	2.05	1.76	1.01	**1.00**	1.62	2.10	15.2	1.90	11.6	11.1	19.9	36.2	26.6	13.0	11.9
iii	406	86.5	19.9	2.10	t	t	t	t	56.5	15.6	11.1	10.6	2.22	2.13	1.02	**1.00**	3.05	2.48	28.5	3.33	m	m	m	m	m	m	m
iv	193	46.5	7.22	1.07	60.4	34.2	95.4	30.2	13.5	17.8	11.0	10.0	4.49	2.83	1.01	**1.00**	1.15	1.34	9.19	1.32	6.93	6.43	12.0	24.9	18.6	7.51	6.32
v	197	25.2	7.16	0.55	62.1	35.1	98.1	31.1	13.7	17.3	10.7	9.79	4.41	2.77	1.02	**1.00**	1.15	1.34	9.17	1.31	7.10	6.61	12.4	25.3	19.3	7.65	6.39
vi	287	87.0	10.6	2.04	87.0	43.8	132	33.4	15.1	21.7	12.3	10.5	4.39	2.46	1.03	**1.00**	1.24	1.39	10.4	1.43	8.42	7.78	14.7	30.7	23.2	9.13	7.45
vii	94.0	21.8	4.17	0.41	71.7	36.2	105	28.0	12.6	15.6	9.18	7.83	4.85	2.80	1.02	**1.00**	1.25	1.37	9.10	1.39	7.23	6.71	11.9	19.2	18.3	6.08	4.86
viii	87.9	17.1	3.10	0.33	50.8	20.9	50.3	16.3	6.43	25.0	11.5	8.33	15.1	5.10	5.18	4.90	1.16	**1.00**	6.69	1.45	3.24	3.09	7.17	16.5	11.8	5.40	4.23
ix	51.1	5.74	1.28	0.07	48.7	20.8	44.6	17.6	6.10	24.3	11.7	7.40	23.0	6.90	5.01	4.80	1.21	**1.00**	6.13	1.51	2.92	2.73	6.39	15.2	9.73	4.30	3.51
x	201	41.6	12.5	1.30	102	42.0	90.3	30.7	11.3	50.6	22.7	12.9	12.6	3.54	1.96	1.91	**1.00**	1.11	6.93	1.36	4.28	3.95	7.80	16.9	13.1	5.41	4.64
xi	46.6	8.49	1.17	0.07	37.7	15.7	29.1	9.82	3.79	29.2	14.2	7.16	17.8	4.42	5.75	5.61	1.23	**1.00**	4.97	1.72	1.73	1.66	5.48	9.12	6.47	3.32	2.77
xii	32.8	3.81	1.87	0.07	73.4	31.5	47.5	15.0	6.86	28.9	8.55	4.36	9.20	2.09	1.99	1.95	1.04	**1.00**	4.60	1.37	2.17	2.03	12.8	6.60	5.04	2.07	1.57
xiii	738	108	382	31.1	t	t	t	t	t	t	t	t	t	t	t	t	1.14	**1.00**	t	1.87	m	m	m	m	m	m	m
xiv	58.9	7.76	5.05	0.21	88.2	26.8	43.5	14.0	5.87	55.6	16.9	8.14	57.4	8.18	7.97	7.67	1.36	**1.00**	5.15	2.47	1.75	1.46	5.77	5.68	6.33	3.05	1.96
xv	181	41.5	15.8	1.93	t	58.0	92.7	29.6	14.4	109	33.1	15.5	59.1	8.08	11.0	10.6	1.07	**1.00**	5.77	1.84	2.60	2.49	7.56	14.6	9.56	5.30	4.14
xvi	224	55.8	46.6	4.86	t	t	t	t	t	t	29.0	13.9	t	9.61	2.98	2.88	1.03	**1.00**	6.18	1.55	m	m	m	m	m	m	m
xvii	145	32.9	249	11.2	t	t	m	t	t	t	t	t	t	t	t	7.87	**1.00**	m	m	1.73	m	m	m	m	m	m	m
xviii	114	11.9	33.7	1.18	t	24.7	38.8	13.1	6.47	59.0	16.6	8.16	58.6	7.89	8.20	8.00	1.33	**1.00**	5.11	2.80	1.47	1.27	3.99	3.13	4.30	2.24	1.71
xix	612	160	587	76.3	t	t	t	t	t	t	t	t	t	t	t	t	1.09	**1.00**	t	2.63	m	m	m	m	m	m	m
xx	18.0	1.00	0.71	0.03	98.1	27.1	42.4	15.6	10.2	61.6	16.7	7.37	20.8	2.65	2.13	2.17	1.29	1.24	4.47	2.17	**1.00**	1.06	8.31	3.50	2.32	1.35	1.38
xxi	76.1	14.4	31.7	2.73	t	39.9	t	23.1	7.92	t	26.0	10.0	t	10.1	9.98	9.63	1.75	1.38	6.41	4.85	1.39	**1.00**	5.51	3.44	2.90	1.92	1.37
xxii	54.8	11.0	23.0	1.90	t	64.5	t	38.1	13.8	t	42.4	15.3	t	14.8	14.6	14.2	2.10	1.52	6.49	5.53	1.82	**1.00**	9.85	3.56	2.61	2.04	1.22

Table A.3 – As per Table A.2, but with $k = 1000$

B Appendix for Chapter 2

There are more first time visits than revisits in the first epoch

Let the probability that a sample is not visited in an epoch be p, where recall that an epoch consists of drawing N/b mini-batches, where we assume $N \bmod b = 0$. Denote by q the probability that the visit of a sample is a revisit. We argue that $q = p:$ the number of samples not visited exactly corresponds to the number of revisits, as the number of visits is the number of samples, by definition of an epoch. Clearly, $p = (1 - b/N)^{N/b}$, from which it can be shown that $1/4 \leq p < 1/e$. Thus $q \leq 1/e$ as we want. In other words, there are at least 1.718 first time visits for 1 revisit.

Showing that two expectations are approximately the same

We wish to show that $\|c_{t+1}(j|2b_t) - c_{t+1}(j|b_t)\|^2$ and $\frac{1}{2}\hat{\sigma}_C^2$ are approximately the same

Recall that $\mathcal{M}_t(j)$ are the samples used to obtain $c_j(t)$, that is

$$c_t(j) = \frac{1}{|\mathcal{M}_t(j)|} \sum_{i \in \mathcal{M}_t(j)} x(i).$$

The mean squared distance of samples in $\mathcal{M}_t(j)$ to $c_j(t)$ we denote by $\hat{\sigma}_S^2(j)$,

$$\hat{\sigma}_S^2(j) = \frac{1}{|\mathcal{M}_t(j)|} \sum_{i \in \mathcal{M}_t(j)} \|x(i) - c_t(t)\|^2.$$

We compute the expectation of $\|c_{t+1}(j|2b_t) - c_{t+1}(j|b_t)\|^2$, where the expectation is over all possible shufflings of the data. Recall that $c_{t+1}(j|2b_t)$ is centroid j at iteration $t+1$ if the mini-batch at size $t+1$ is $2b_t$, where b_t is the mini-batch size at iteration t. Recall that we use $\mathcal{M}_t(j)$ to denote samples assigned to $c_t(j)$. We will now denote by $\mathcal{M}_{t+1}^{2b_t}(j)$ the sample indices assigned to $c_{t+1}(j|2b_t)$ and $\mathcal{M}_{t+1}^{b_t}(j)$ the sample indices assigned to $c_{t+1}(j|b_t)$. Thus,

$$
\begin{aligned}
\mathbb{E}\Big(\ \|c_{t+1}(j|2b_t) - c_{t+1}(j|b_t)\|^2\Big) &= \\
&= \mathbb{E}\left(\Big\|\frac{1}{|\mathcal{M}^{2b_t}_{t+1}(j)|}\sum_{i\in\mathcal{M}^{2b_t}_{t+1}(j)} x(i) - \frac{1}{|\mathcal{M}^{b_t}_{t+1}(j)|}\sum_{i\in\mathcal{M}^{b_t}_{t+1}(j)} x(i)\Big\|^2\right) \\
&= \mathbb{E}\left(\Big\|\frac{1}{|\mathcal{M}^{2b_t}_{t+1}(j)|}\sum_{i\in\mathcal{M}^{2b_t}_{t+1}(j)\setminus\mathcal{M}^{b_t}_{t+1}(j)} x(i) - \right. \\
&\qquad \left.\left(\frac{1}{|\mathcal{M}^{b_t}_{t+1}(j)|} - \frac{1}{|\mathcal{M}^{2b_t}_{t+1}(j)|}\right)\sum_{i\in\mathcal{M}^{b_t}_{t+1}(j)} x(i)\Big\|^2\right)
\end{aligned}
$$

We now assume that the number of samples per centroid does not change significantly between iterations t and $t+1$ for a fixed batch size, so that $|\mathcal{M}^{b_t}_{t+1}(j)| \approx |M_t(j)|$ and $|\mathcal{M}^{2b_t}_{t+1}(j)| \approx 2|M_t(j)|$. Continuing we have,

$$
\begin{aligned}
&\approx \frac{1}{4|\mathcal{M}_t(j)|^2}\mathbb{E}\left(\Big\|\sum_{i\in\mathcal{M}^{2b_t}_{t+1}(j)\setminus\mathcal{M}^{b_t}_{t+1}(j)} x(i) - \sum_{i\in\mathcal{M}^{b_t}_{t+1}(j)} x(i)\Big\|^2\right) \\
&\approx \frac{1}{4|\mathcal{M}_t(j)|^2}\mathbb{E}\left(\Big\|\sum_{i\in\mathcal{M}^{2b_t}_{t+1}(j)\setminus\mathcal{M}^{b_t}_{t+1}(j)} (x(i) - c_t(j)) - \right. \\
&\qquad \left.\sum_{i\in\mathcal{M}^{b_t}_{t+1}(j)} (x(i) - c_t(j))\ \Big\|^2\right)
\end{aligned}
$$

The two summation terms are independant and the second has expectation approximately zero assuming the centroids do not move too much between rounds, so

$$
\begin{aligned}
&\approx \frac{1}{4|\mathcal{M}_t(j)|}\left(\mathbb{E}\left(\frac{1}{|\mathcal{M}_t(j)|}\Big\|\sum_{i\in\mathcal{M}^{2b_t}_{t+1}(j)\setminus\mathcal{M}^{b_t}_{t+1}(j)} (x(i) - c_t(j))\ \Big\|^2\right) + \right. \\
&\qquad \left.\mathbb{E}\left(\frac{1}{|\mathcal{M}_t(j)|}\Big\|\sum_{i\in\mathcal{M}^{b_t}_{t+1}(j)} (x(i) - c_t(j))\ \Big\|^2\right)\right)
\end{aligned}
$$

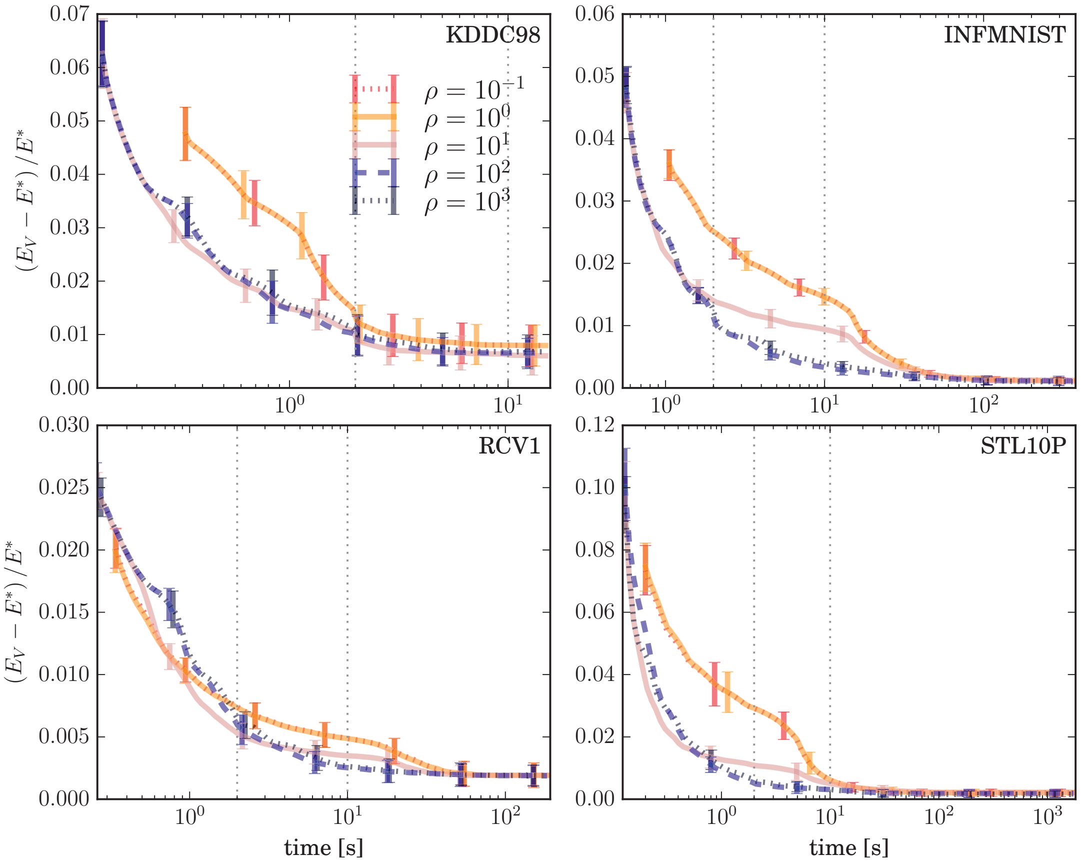

Figure B.1 – Time-energy curves for `nmbatch` with various ρ. The dotted vertical lines correspond to the time slices presented in Figure 2. We see that large ρ works best, with very little difference between $\rho = 10^2$ and $\rho = 10^3$.

Finally, each of the two expectation terms can be approximated by $\hat{\sigma}_S^2(j)$. Approximating the first term by $\hat{\sigma}_S^2(j)$, may be an underestimation as the summation is over data which was not used to obtain $c_t(j)$, whereas $\hat{\sigma}_S^2(j)$ is obtained using data used by $c_t(j)$. Using this estimation we get,

$$
\begin{aligned}
&\approx \frac{1}{2|\mathcal{M}_t(j)|}\hat{\sigma}_S^2(j),\\
&= \frac{1}{2|M_t(j)|^2}\sum_{i \in \mathcal{M}_t(j)} \|x(i) - c_t(t)\|^2,\\
&= \frac{1}{2}\hat{\sigma}_C^2(j).
\end{aligned}
$$

The final equality following from the definition of $\hat{\sigma}_C^2(j)$.

Time-energy curves with various doubling thresholds

Figures B.1 and B.2 show the full time-energy curves for various values of the doubling threshold ρ, for the cases where bounds are used and deactivated respectively.

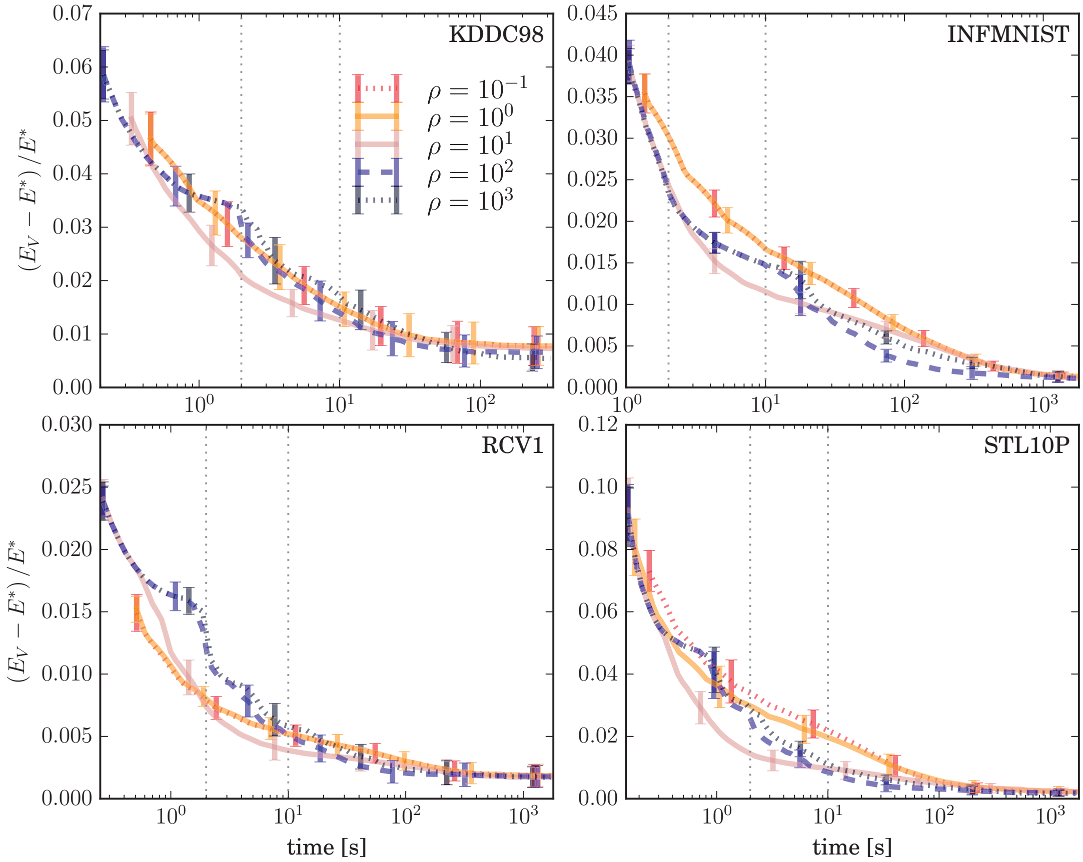

Figure B.2 – Time-energy curves for `nmbatch` with bounds disabled. The dotted vertical lines correspond to the time slices presented in Figure 2, that is $t = 2s$ and $t = 10s$. We see that with bounds disabled, $\rho = 10^1$ in general outperforms $\rho \in \{10^2, 10^3\}$, providing empirical support for the proposed doubling scheme.

On algorithms intermediate to `mbatch` and `nmbatch`

The primary argument presented in this paper for removing old assignments is to prevent a biased use of samples in `nmbatch`. However, a second reason for removing old assignments is that they can contaminate centroids if left unremoved. This second reason in favour of removing old assignments is also applicable to `mbatch`, and so it is interesting to see if `mbatch` can be improved by removing old assignments, without the inclusion of triangle inequality based bounds. We call this algorithm `mbatch.remove`. In addition, it is interesting to consider the performance of `nmbatch` without bound testing. We here call `nmbatch` without bound testing `nmbatch.deact`.

In Figure B.3 we see that `mbatch` is indeed improved by removing old assignments: `mbatch.remove` outperforms `mbatch`, especially at later iterations. We see that the algorithm `nmbatch.deact` does not perform as well as `nmbatch`, as expected, however it is comparable to `mbatch.remove`, if not slightly better. There is no algorithmic reason why `nmbatch.deact` should be better than `mbatch.remove`, as nesting was proposed purely as way to better harness bounds. One possible explanation for the good performance of `nmbatch.deact` is better memory usage: when samples are reused there are fewer cache memory misses.

Premature fine-tuning

The loss function being minimized changes when the mini-batch grows. With b_t samples, it is

$$E(\mathcal{C}) = \frac{1}{b_t}\sum_{i=1}^{b_t} \operatorname*{arg\,min}_{j\in\{1,\ldots,k\}} \|x(i) - c(j)\|^2,$$

and then with $2b_t$ it is

$$E(\mathcal{C})\frac{1}{2b_t}\sum_{i=1}^{2b_t} \operatorname*{arg\,min}_{j\in\{1,\ldots,k\}} \|x(i) - c(j)\|^2.$$

Minima of these two loss functions are different, although as b_t gets large they approach each each. Premature fine-tuning refers to putting a large amount of effort into getting very close to a minimum with b_t samples, when we know that as soon as we switch to $2b_t$ samples the minimum will move, undoing our effort to get very close to a mimumum.

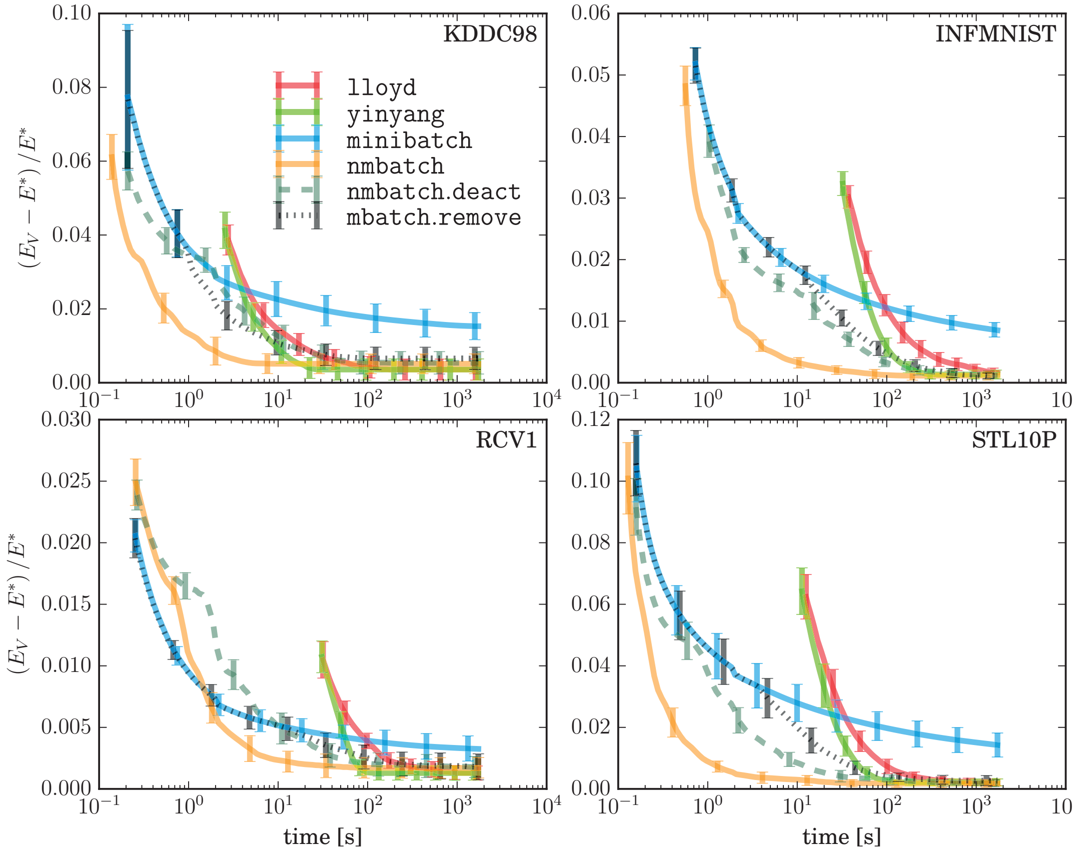

Figure B.3 – Performance of algorithms intermediate to `nmbatch` and `mbatch`. The intermediate algorithms are : `nmbatch.deact`, which is `nmbatch` with the bound test deactivated, and `mbatch.remove`, which is `mbatch` with the removal of old assignments. `nmbatch` and `nmbatch.deact` are with $\rho = 100$ as usual. We observe that, as expected, deactivation of the bound test results in a significant slow-down of `nmbatch`. We also observe that the removal of old assignments significantly improves `mbatch`, especially at later iterations.

C Appendix for Chapter 3

On the difficulty of the medoid problem

We construct an example showing that no general purpose algorithm exists to solve the medoid problem in $o(N^2)$. Consider an almost fully connected graph containing $N = 2m + 1$ nodes, where the graph is exactly m edges short of being fully connected: one node has $2m$ edges and the others have $2m - 1$ edges. The graph has $2m^2$ edges. With the shortest path metric, it is easy to see that the node with $2m$ edges is the medoid, hence the medoid problem is as difficult as finding the node with $2m$ edges. But, supposing that the edges are provided as an unsorted adjacency list, it is clearly an $O(m^2)$ task to determine which node has $2m$ edges as one must look at all edges until a node with $2m$ edges is found. Thus determining the medoid is $O(m^2)$ which is $O(N^2)$.

`medlloyd` pseudocode

Alg. 9 presents the `medlloyd` algorithm of Park and Jun [2009], with the novel initialization of `medlloyd` on line 1. `medlloyd` is essentially `lloyd`, with medoids instead of means.

Algorithm 9 `medlloyd` for clustering data $\{x(1), \ldots, x(N)\}$ around K medoids

```
1: Set all distances $D(i, j) \leftarrow \|x(i) - x(j)\|$ and sums $S(i) \leftarrow \sum_{j \in \{1,\ldots,N\}} D(i, j)$
2: Initialize medoid indices as $K$ indices minimising $f(i) = \sum_{j \in \{1,\ldots,N\}} D(i, j)/S(j)$
3: while Some convergence criterion has not been met do
4:     Assign each element to the cluster whose medoid is nearest to the element
5:     Update cluster medoids according to assignments made above
6: end while
```

`RAND`, `TOPRANK` and `TOPRANK2` pseudocode

We present pseudocode for the `RAND`, `TOPRANK` and `TOPRANK2` algorithms of Okamoto et al. [2008], and discuss the explicit and implicit constants.

On the number of anchor elements in `TOPRANK`

We consider the constant in $\Theta(N^{\frac{2}{3}}(\log N)^{\frac{1}{3}})$. Note that the number of anchor points used in `TOPRANK` does not affect the result that the medoid is w.h.p. returned. However, Okamoto et al. [2008] show that by choosing the size of the anchor set to be $q\,(\log N)^{\frac{1}{3}}$ for any q, the run time is guaranteed to be $\tilde{O}(N^{5/3})$. They do not suggest a specific q, the optimal q being dataset dependant. We choose $q = 1$.

Consider Figure 3.3 in Appendix 3.5.1 for example, where $q = 1$. Had q be chosen to be less than 1, the line $\texttt{ncomputed} = N^{2/3}\log^{1/3} N$ to which `TOPRANK` runs parallel for large N would be shifted up or down by $\log q$, however the N at which the transition from $\texttt{ncomputed} = N^{2/3}\log^{1/3} N$ to $\texttt{ncomputed} = N^{2/3}\log^{1/3} N$ takes place would also change.

On the parameter α' in `TOPRANK` and `TOPRANK2`

The threshold τ in (3.2) is proportional to the parameter α'. In Okamoto et al. [2008], it is stated that α' should be some value greater than 1. Note that the smaller α' is, the lower the threshold is, and hence fewer the number of computed points is, thus $\alpha' = 1.00001$ would be a fair choice. We use $\alpha' = 1$ in our experiments, and observe that the correct medoid is returned in all experiments.

Personal correspondence with the authors of Okamoto et al. [2008] has brought into doubt the proof of the result that the medoid is w.h.p. returned for any α' where $\alpha' > 1$. In our most recent correspondence, the authors suggest that the w.h.p. result can be proven with the more conservative bound of $\alpha' > \sqrt{1.5}$. Moreover, we show in Appendix C.4 that $\alpha' > 1$ is good enough to return the medoid with probability $N^{-(\alpha'-1)}$, a probability which still tends to 0 as N grows large, but not a w.h.p. result. Please refer to Appendix C.4 for further details on our correspondence with the authors.

On the parameters specific to `TOPRANK2`

In addition to α', `TOPRANK2` requires two parameters to be set. The first is l_0, the starting anchor set size, and the second is q, the amount by which l should be incremented at each iteration. Okamoto et al. [2008] suggest taking l_0 to be the number of top ranked nodes required, which in our case would be $l_0 = k = 1$. However, in our experience this is too small as all nodes lie well within the threshold and thus when l increases there is no change to number below threshold, which makes the algorithm break out of the search for the optimal l too early. Indeed, l_0 needs to be chosen so that at least some points have energies greater than the threshold, which in our experiments is already quite large. We choose $l_0 = \sqrt{N}$, as any value larger than $N^{2/3}$ would make `TOPRANK2` redundant to `TOPRANK`. The parameter q we take to be $\log N$ as suggested by Okamoto et al. [2008].

Algorithm 10 `RAND` for estimating energies of elements of set S [Eppstein and Wang, 2004].

```
$I \leftarrow$ random uniform sample from $\{1, \ldots, N\}$
// Compute all distances from anchor elements ($I$), using Dijkstra's algorithm on graphs
for $i \in I$ do
    for $j \in \{1, \ldots, N\}$ do
        $d(i, j) \leftarrow \|x(i) - x(j)\|,$
    end for
end for
// Estimate energies as mean distances to anchor elements
for $j \in \{1, \ldots, N\}$ do
    $\hat{E}(j) \leftarrow \frac{1}{|I|} \sum_{i \in I} d(i, j)$
end forreturn $\hat{E}$
```

Algorithm 11 `TOPRANK` for obtaining top k ranked elements of S [Okamoto et al., 2008].

```
$l \leftarrow N^{\frac{2}{3}} (\log N)^{\frac{1}{3}}$    // Okamoto et al. [2008] state that $l$ should be $\Theta((\log N)^{\frac{1}{3}})$, the choice of 1 as the constant is arbitrary (see comments in the text of Appendix C.3.1).
Run RAND with uniform random $I$ of size $l$ to get $\hat{E}(i)$ for $i \in \{1, \ldots, N\}$.
Sort $\hat{E}$ so that $\hat{E}[1] \leq \hat{E}[2] \leq \ldots \leq \hat{E}[N]$
$\hat{\Delta} \leftarrow 2 \min_{i \in I} \max_{j \in \{1, \ldots, N\}} \|x(i) - x(j)\|$    // where $\|x(i) - x(j)\|$ computed in RAND
$Q \leftarrow \left\{ i \in \{1, \ldots, N\} \mid \hat{E}(i) \leq \hat{E}[k] + 2\alpha' \Delta \sqrt{\frac{\log(n)}{l}} \right\}.$
Compute exact energies of all elements in $Q$ and return the element with the lowest energy.
```

Algorithm 12 `TOPRANK2` for obtaining top k ranked elements of S [Okamoto et al., 2008].

```
// In Okamoto et al. [2008], it is suggested that $l_0$ be taken as $k$, which in the case of the medoid problem is 1. We have experimented with several choices for $l_0$, as discussed in the text.
$l \leftarrow l_0$
Run RAND with uniform random $I$ of size $l$ to get $\hat{E}(i)$ for $i \in \{1, \ldots, N\}$.
$\hat{\Delta} \leftarrow 2 \min_{i \in I} \max_{j \in \{1,\ldots,N\}} \|x(i) - x(j)\|$    // where $\|x(i) - x(j)\|$ computed in RAND
Sort $\hat{E}$ so that $\hat{E}[1] \leq \hat{E}[2] \leq \ldots \leq \hat{E}[N]$
$Q \leftarrow \left\{ i \in \{1, \ldots, N\} \mid \hat{E}(i) \leq \hat{E}[k] + 2\alpha' \Delta \sqrt{\frac{\log(n)}{l}} \right\}$.
$g \leftarrow 1$
while $g$ is 1 do
    $p \leftarrow |Q|$
    // The recommendation for $q$ in Okamoto et al. [2008] is $\log(n)$, we follow the suggestion
    Increment $I$ with $q$ new anchor points
    Update $\hat{E}$ for all data according to new anchor points
    $l \leftarrow |I|$
    $\hat{\Delta} \leftarrow 2 \min_{i \in I} \max_{j \in \{1,\ldots,N\}} \|x(i) - x(j)\|$
    Sort $\hat{E}$ so that $\hat{E}[1] \leq \hat{E}[2] \leq \ldots \leq \hat{E}[N]$
    $Q \leftarrow \left\{ i \in \{1, \ldots, N\} \mid \hat{E}(i) \leq \hat{E}[k] + 2\alpha' \Delta \sqrt{\frac{\log(n)}{l}} \right\}$
    $p' \leftarrow |Q|$
    if $p - p' < \log(n)$ then
        $g \leftarrow 0$
    end if
end while
Compute exact energies of all elements in $Q$ and return the element with the lowest energy
```

On the proof that `TOPRANK` returns the medoid with high probability

Through correspondence with the authors of Okamoto et al. [2008], we have located a small problem in the proof that the medoid is returned w.h.p. for $\alpha' > 1$, the problem lying in the second inequality of Lemma 1. To arrive at this inequality, the authors have used the fact that for all i,

$$\mathbb{P}(E(i) \geq \hat{E}(i) + f(l) \cdot \Delta) \geq 1 - \frac{1}{2N^2}, \tag{C.1}$$

which is a simple consequence of the Hoeffding inequality as shown in Eppstein and Wang [2004]. Essentially (C.1) says that, for a fixed node i, from which the mean distance to other nodes is $E(i)$, if one uniformly samples l distances to i and computes the mean $\hat{E}(i)$, the probability that $\hat{E}(i)$ is less than $E(i) + f(l)$ is greater than $1 - \frac{1}{2N^2}$.

The inequality (C.1) is true for a fixed node i. However, it no longer holds if i is selected to be the node with the lowest $\hat{E}(i)$. To illustrate this, suppose that $E(i) = 1$ for all i, and compute $\hat{E}(i)$ for all i. Let $\hat{E}^* = \arg\min_i \hat{E}(i)$. Now, we have a strong prior on $\hat{E}^*$ being significantly less than 1, and (C.1) no longer holds as a statement made about $\hat{E}^*$.

In personal correspondence, the authors show that the problem can be fixed by the use of an additional layer of union bounding, with a correction to be published (if not already done so at time of writing). However, the additional layer of union bound requires a more conservative constraint on α', which is $\alpha' > 2$, although the authors propose that the w.h.p. result can be proven with $\alpha' > \sqrt{1.5}$ for N sufficiently large. We now present a small proof proving the w.h.p. result for $\alpha' > \sqrt{2}$ for N sufficiently large, with at the same time $\alpha' > 1$ guaranteeing that the medoid is returned with probability $O(N^{\alpha'-1})$.

Probability that the medoid is returned

We show that the medoid is returned *with high probability* holds for $\alpha' > \sqrt{2}$ and that *with vanishing probability* it is returned for $\alpha' > 1$ Recall that we have N nodes with *energies* $E(1), \ldots, E(n)$. We wish to find the k lowest energy nodes (the original setting of Okamoto et al. [2008]). From Hoeffding's inequality we have,

$$\mathbb{P}(|E(i) - \hat{E}(i)| \geq \epsilon\Delta) \leq 2\exp\left(-l\epsilon^2\right). \tag{C.2}$$

Set the probability on the right hand side of C.2 to be $2/N^{1+\beta}$, that is,

$$2\exp\left(-l\epsilon^2\right) = 2/N^{1+\beta},$$

which corresponds to

$$\epsilon = \sqrt{\left(\frac{1+\beta}{l}\right)\log(N)} \quad := \quad \tilde{f}(l).$$

Clearly $\sqrt{1+\beta}$ corresponds to α'. With this notation we have,

$$\mathbb{P}(|E(i) - \hat{E}(i)| \geq \tilde{f}(l)\Delta) \leq \frac{2}{N^{1+\beta}}. \tag{C.3}$$

Applying the union bound to (C.3) we have,

$$\mathbb{P}\left(\neg\left(\wedge_{i\in\{1,\dots,N\}}|E(i) - \hat{E}(i)| \leq \tilde{f}(l)\Delta\right)\right) \leq \frac{2}{N^{\beta}}. \tag{C.4}$$

Recall that we wish to obtain the k nodes with lowest energy. Denote by $r(j)$ the index of the node with the j'th lowest energy, so that

$$E(r(1)) \leq \ldots \leq E(r(j)) \leq \ldots \leq E(r(N)).$$

Denote by $\hat{r}(j)$ the index of the node with the j'th lowest estimated energy, so that

$$\hat{E}(\hat{r}(1)) \leq \ldots \leq \hat{E}(\hat{r}(j)) \leq \ldots \leq \hat{E}(\hat{r}(N)).$$

Now assume that for all i, it is true that $|E(i) - \hat{E}(i)| \leq \tilde{f}(l)$. Then consider, for $j \leq k$,

$$\begin{aligned}\hat{E}(\hat{r}(k)) - \hat{E}(r(j)) = &\\ &\underbrace{\left(\hat{E}(\hat{r}(k)) - E(r(k))\right)}_{\geq -\tilde{f}(l)\Delta} + \underbrace{\left(E(r(k)) - E(r(j))\right)}_{\geq 0} + \underbrace{\left(E(r(j)) - \hat{E}(r(j))\right)}_{\geq -\tilde{f}(l)\Delta},\\ &\geq -2\tilde{f}(l)\Delta.\end{aligned} \tag{C.5}$$

The first bound in (C.5) is obtained by considering the most extreme case possible under the assumption, which is $\hat{E}(i) = a(E) - \tilde{f}(l)$ for all i. The second bound follows from $j \leq k$, and the third bound follows directly from the assumption. We thus have that, under the assumption,

$$\hat{E}(r(j)) \leq \hat{E}(\hat{r}(k)) + 2\tilde{f}(l)\Delta,$$

which says that all nodes of rank less than or equal to k have approximate energy less than $\hat{E}(\hat{r}(k)) + 2\tilde{f}(l)\Delta$. As the assumption holds with probability greater than $1 - 2/N^{\beta}$ by (C.4), we are done. Take $\beta = 1$ if you want the statement *with high probability*, that is

$$\epsilon = \sqrt{\frac{2\log(n)}{l}},$$

but for any $\beta > 0$, which corresponds to $\alpha' > 1$, the probability of failing to return the k lowest energy nodes tends to 0 as N grows.

			$K = 10$		$K = \lceil\sqrt{N}\rceil$		$K = \lceil\frac{N}{10}\rceil$	
Dataset	N	d	$\mu_{\mathrm{u}}/\mu_{\texttt{park}}$	$\sigma_{\mathrm{u}}/\mu_{\texttt{park}}$	$\mu_{\mathrm{u}}/\mu_{\texttt{park}}$	$\sigma_{\mathrm{u}}/\mu_{\texttt{park}}$	$\mu_{\mathrm{u}}/\mu_{\texttt{park}}$	$\sigma_{\mathrm{u}}/\mu_{\texttt{park}}$
gassensor	256	128	1.09	0.08	**0.90**	0.03	**0.83**	0.01
house16H	1927	17	1.01	0.02	**0.97**	0.01	**0.93**	0.01
S1	5000	2	1.05	0.05	**0.75**	0.01	**0.32**	0.01
S2	5000	2	1.04	0.07	**0.68**	0.01	**0.34**	0.00
S3	5000	2	1.03	0.05	**0.76**	0.01	**0.35**	0.00
S4	5000	2	1.02	0.03	**0.75**	0.01	**0.41**	0.01
A1	3000	2	**0.82**	0.03	**0.43**	0.01	**0.19**	0.00
A2	5250	2	**0.98**	0.03	**0.47**	0.01	**0.25**	0.00
A3	7500	2	**0.96**	0.02	**0.42**	0.02	**0.22**	0.00
thyroid	215	5	**0.95**	0.08	**0.97**	0.04	**0.93**	0.04
yeast	1484	8	1.00	0.02	**0.96**	0.02	**0.91**	0.02
wine	178	14	1.01	0.02	1.02	0.01	**0.98**	0.02
breast	699	9	**0.79**	0.03	**0.77**	0.02	**0.68**	0.02
spiral	312	3	1.03	0.03	**0.99**	0.02	**0.82**	0.03

Table C.1 – Comparing the initialization scheme proposed in Park and Jun [2009] with random uniform initialization for the `medlloyd` algorithm. The final energy using the deterministic scheme proposed in Park and Jun [2009] is $\mu_{\texttt{park}}$. The mean over 10 random uniform initializations is μ_u, and the corresponding standard deviation is σ_u. For small K ($K = 10$), the performances using the two schemes are comparable, while for larger K, it is clear that uniform initialization performs much better on the majority of datasets.

On the initialization of Park and Jun [2009]

In Table C.1 we present the full results of the 48 experiments comparing the initialization proposed in Park and Jun [2009] with simple uniform initialization. The 14 datasets are all available from https://cs.joensuu.fi/sipu/datasets/.

Scaling with α, N, and dimension d

We perform more experiments to provide further validation of Theorem 3.3.2. In particular, we check how the number of computed elements scales with N, d and α. We generate data from a unit ball in various dimensions, according to two density functions with different strong convexity constants α. The first density function is uniform, so that the density everywhere in the ball is uniform. To sample from this distribution, we generate two random variables, $X_1 \sim \mathcal{N}_d(0,1)$ and $X_2 \sim U(0,1)$ and use

$$X_3 = X_1/\|X_1\| \cdot X_2^{\frac{1}{d}}, \tag{C.6}$$

as a sample from the unit ball $\mathcal{B}_d(0,1)$ with uniform distribution. The second distribution we consider has a higher density beyond radius $(1/2)^{1/d}$. Specifically, within this radius the density is 19× lower than beyond this radius. To sample from this distribution, we sample X_3 according to (C.6), and then points lying within radius $(1/2)^{1/d}$ are with probability 1/10 re-sampled uniformly beyond this radius.

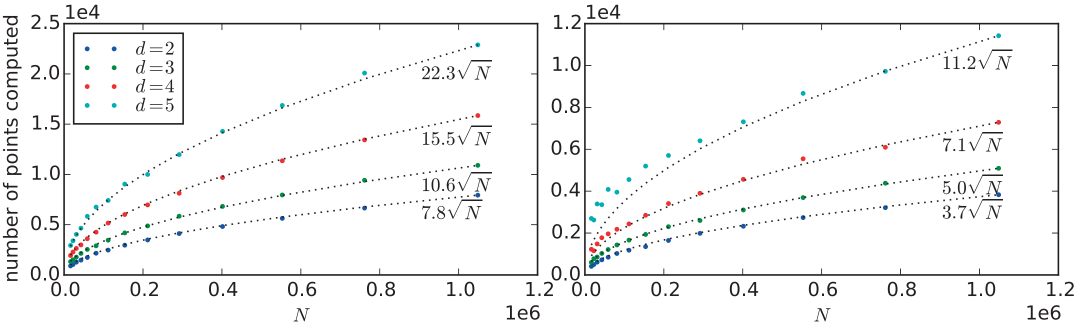

Figure C.1 – Number of points computed on simulated data. Points are drawn from $\mathcal{B}_d(0, 1)$, for $d \in \{2, 3, 4, 5\}$. On the left, points are drawn uniformly, while on the right, the density in $\mathcal{B}_d(0, (1/2)^{1/d})$ is 19× lower that in $\mathcal{A}_d(0, (1/2)^{1/2}, 1)$, where recall that $\mathcal{A}_d(x, r_1, r_2)$ denotes an annulus centred at x of inner radius r_1 and outer radius r_2. We observe a near perfect fit of the number of computed points to $\xi\sqrt{N}$ where the constant ξ depends on the dimension and the distribution (left and right). The number of computed points increases with dimension. The strong convexity constant of the distribution on the right is larger, corresponding to fewer distance calculations as predicted by Theorem 3.3.2.

The second distribution has a larger strong convexity constant α. To see this, note that the strong convexity constant at the center of the ball depends only on the density of the ball on its surface, that is at radius 1, as can be shown using an argument based on cancelling energies of internal points. As the density at the surface under distribution 2 is approximately twice that of under distribution 1, the change in energy caused by a small shift in the medoid is twice as large under distribution 2. Thus, according to Theorem 3.3.2, we expect the number of computed points to be larger under distribution 1 than under distribution 2. This is what we observe, as shown in Figure C.1, where distribution 1 is on the left and distribution 2 is on the right.

In Figure C.1 we observe a near perfect $N^{1/2}$ scaling of number of computed points. Dashed curves are exact $N^{1/2}$ relationships, while the coloured points are the observed number of computed points.

Proof of Theorem 3.3.2 (See page 42)

Theorem 3.3.2. *Let $\mathcal{S} = \{x(1), \ldots, x(N)\}$ be a set of N elements in $\mathbb{R}^d$, drawn independently from probability distribution function f_X. Let the medoid of $\mathcal{S}$ be $x(m^*)$, and let $E(m^*) = E^*$. Suppose that there exist strictly positive constants ρ, δ_0 and δ_1 such that for any set size N with probability $1 - O(1/N)$*

$$x \in \mathcal{B}_d(x(m^*), \rho) \implies \delta_0 \le f_X(x) \le \delta_1, \tag{3.6}$$

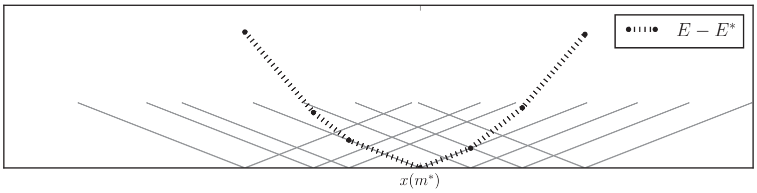

Figure C.2 – A sum of uniformly distributed cones is approximately quadratic.

where $\mathcal{B}_d(x,r)=\{x'\in\mathbb{R}^d\ :\ \|x'-x\|\le r\}$*. Let* $\alpha>0$ *be a constant (independent of* N*) such that with probability* $1-O(1/N)$ *all* $i\in\{1,\dots,N\}$ *satisfy,*

$$\begin{aligned} x(i)\in\mathcal{B}_d(x(m^*),\rho) &\implies \\ &E(i)-E^*\ge\alpha\|x(i)-x(m^*)\|^2. \end{aligned} \tag{3.7}$$

Then, the expected number of elements computed by `trimed` *is* $O\left(\left(V_d[1]\delta_1+d\left(\frac{4}{\alpha}\right)^d\right)N^{\frac{1}{2}}\right)$*, where* $V_d[1]=\pi^{\frac{d}{2}}/(\Gamma(\frac{d}{2}+1))$ *is the volume of* $\mathcal{B}_d(0,1)$*.*

Proof. We show that the assumptions made in Th. 3.3.2 validate the assumptions required in Thm C.7.1. Firstly, if $e(i)>\rho$ then $e(i)\ge\alpha\rho^2e(i)>\rho$, which follows from the convexity of the loss function and. Secondly, the existance of β follows from continuity of the gradient of the distance, combined with the existence of δ_1 (non-exploding). □

Theorem C.7.1 (Main Theorem Expanded)**.** *Let* $\mathcal{S}=\{x(1),\dots,x(N)\}\subset\mathbb{R}^d$ *have medoid* $x(m^*)$ *with minimum energy* $E(m^*)=E^*$*, where elements in* $\mathcal{S}$ *are drawn independently from probability distribution function* f_X*. Let* $e(i)=\|x(i)-x(m^*)\|$*. Suppose that for* f_X *there exist strictly positive constants* $\alpha,\beta,\rho,\delta_0$ *and* δ_1 *satisfying,*

$$x\in\mathcal{B}_d(x(m^*),\rho)\implies\delta_0\le f_X(x)\le\delta_1, \tag{C.7}$$

where $\mathcal{B}_d(x,r)=\{x'\in\mathbb{R}^d\ :\ \|x'-x\|\le r\}$*, and that for any set size* N*, w.h.p. all* $i\in\{1,\dots,N\}$ *satisfy,*

$$E(i)-E^*\ge\begin{cases}\alpha e(i)^2 & \textit{if}\quad e(i)\le\rho,\\ \alpha\rho^2 & \textit{if}\quad e(i)>\rho,\end{cases} \tag{C.8}$$

and,

$$E(i)-E^*\le\beta e(i)^2\qquad\textit{if}\quad e(i)\le\rho. \tag{C.9}$$

Then the expected number of elements computed, which is to say not eliminated on line 4 of `trimed`*, is* $O\left(\left(V_d[1]\delta_1+d\left(\frac{4}{\alpha}\right)^d\right)N^{\frac{1}{2}}\right)$*, where* $V_d[1]=\pi^{\frac{d}{2}}/(\Gamma(\frac{d}{2}+1))$ *is the volume of* $\mathcal{B}_d(0,1)$*.*

Proof. We first show that the expected number of computed elements in $\mathcal{B}_d(x(m^*),N^{-\frac{1}{2d}})$

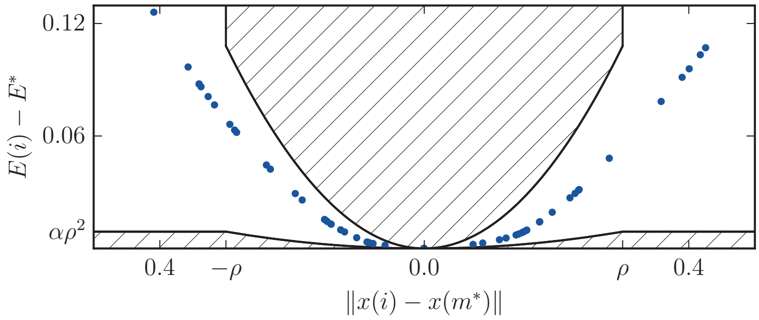

Figure C.3 – Illustrating the parameters α, β and ρ of Theorem 3.3.2. Here we draw $N = 101$ samples uniformly from $[-1, 1]$ and compute their energies, plotted here as the series of points. Theorem 3.3.2 states that their exists α, β and ρ such that irrespective of N, all energies (points) will lie in the envelope (non-hatched region).

is $O(V_d[1]\delta_1 N^{\frac{1}{2}})$. When N is sufficiently large, $f_X(x) \le \delta_1$ within $\mathcal{B}_d(x(m^*), N^{-\frac{1}{2d}})$. The expected number of samples in $\mathcal{B}_d(x(m^*), N^{-\frac{1}{2d}})$ is thus upper bounded by δ_1 multiplied by the volume of the ball. But the volume of a ball of radius $N^{-\frac{1}{2d}}$ in $\mathbb{R}^d$ is $V_d[1]N^{-\frac{1}{2}}$.

In Lemma C.7.2 we use a packing argument to show that the number of computed elements in the annulus $\mathcal{A}_d(x(m^*), N^{-\frac{1}{2d}}, \infty)$ is $O\left(d\left(\frac{4}{\alpha}\right)^d N^{\frac{1}{2}}\right)$, but we there assume that the medoid index m^* is the first element in `shuffle`$(\{1, \dots, N\})$ on line 3 of `trimed` and thus that the medoid energy is known from the first iteration ($E^{cl} = E^*$). We now extend Lemma C.7.2 to the case where the medoid is not the first element processed. We do this by showing that w.h.p. an element with energy very close to E^* has been computed after $N^{-\frac{1}{2}}$ iterations of `trimed`, and thus that the bounds on numbers of computed elements obtained using the packing arguments underlying Lemma C.7.2 are all correct to within some small factor after $N^{-\frac{1}{2}}$ iterations.

The probability of a sample lying within radius $N^{-\frac{2}{3d}}$ of $x(m^*)$ is $\Omega(\delta_0 N^{-\frac{2}{3}})$, and so the probability that none of the first $N^{\frac{1}{2}}$ samples lies within radius $N^{-\frac{2}{3d}}$ is $O((1-\delta_0 N^{-\frac{2}{3d}})^{N^{\frac{1}{2}}})$ which is $O(\frac{1}{N})$. Thus w.h.p. after $N^{\frac{1}{2}}$ iterations of `trimed`, E^{cl} is within $\beta N^{-\frac{4}{3d}}$ of E^*, which means that the radii of the balls used in the packing argument are overestimated by at most a factor $N^{-\frac{1}{3d}}$. Thus w.h.p. the upper bounds obtained with the packing argument are correct to within a factor $1 + N^{-\frac{1}{3}}$. The remaining $O(\frac{1}{N})$ cases do not affect the expectation, as we know that no more than N elements can be computed. □

Lemma C.7.2 (Packing beyond the vanishing radius)**.** *If we assume* (C.8) *from Theorem 3.3.2 and that the medoid index m^* is the first element processed by* `trimed`*, then the number of elements computed in $\mathcal{A}_d(x(m^*), N^{-\frac{1}{2d}}, \infty)$ is $O\left(d\left(\frac{4}{\alpha}\right)^d N^{\frac{1}{2}}\right)$.*

Proof. Follows from Lemmas C.7.3 and C.7.4. □

Lemma C.7.3 (Packing from the vanishing radius $N^{-\frac{1}{d}}$ to ρ)**.** *If we assume* (C.8) *from Theorem 3.3.2 and that the medoid index m^* is the first element processed in* `trimed`*, then the number of computed elements in $\mathcal{A}(x(m^*), N^{-\frac{1}{2d}}, \rho)$ is $O(d\left(\frac{4}{\alpha}\right)^d N^{\frac{1}{2}})$.*

Proof. According to Assumption C.8, an element at radius $r < \rho$ has surplus energy at least αr^2. This means that, assuming that the medoid has already been computed, an element computed at radius r will be surrounded by an exclusion zone of radius αr^2 in which no element will subsequently be computed. We will use this fact to upper bound the number of computed elements in $\mathcal{A}(x(m^*), N^{-\frac{1}{2d}}, \rho)$, firstly by bounding the number in an annulus of inner radius r and width αr^2, that is the annulus $\mathcal{A}_d(x(m^*), r, r + \alpha r^2)$, and then summing over concentric rings of this form which cover $\mathcal{A}(x(m^*), N^{-\frac{1}{2d}}, \rho)$. Recall that the number of computed elements in $\mathcal{A}_d(x(m^*), r, r + \alpha r^2)$ is denoted by $N_c(x(m^*), r, r + \alpha r^2)$.

We use Lemma C.7.5 to bound $N_c(x(m^*), r, r + \alpha r^2)$,

$$\begin{aligned}
N_c(x(m^*), r, r + \alpha r^2) &\leq (d+1)^2 \left(\frac{4}{\sqrt{3}}\right)^d \frac{\alpha r^2 (r + \alpha r^2)^{d-1}}{(\alpha r^2)^d} \\
&\leq (d+1)^2 \left(\frac{4}{\sqrt{3}}\right)^d \left(1 + \frac{1}{\alpha r}\right)^{d-1} \\
&\leq (d+1)^2 \left(\frac{4}{\sqrt{3}}\right)^d \left(\max\left(2, \frac{2}{\alpha r}\right)\right)^{d-1} \\
&\leq (d+1)^2 \left(\frac{4}{\sqrt{3}}\right)^d \left(\max\left(2^{d-1}, \left(\frac{2}{\alpha r}\right)^{d-1}\right)\right) \\
&\leq (d+1)^2 \left(\frac{4}{\sqrt{3}}\right)^d \left(2^{d-1} + \left(\frac{2}{\alpha r}\right)^{d-1}\right) \\
&\leq (d+1)^2 \left(\frac{8}{\sqrt{3}}\right)^d + (d+1)^2 \left(\frac{8}{\sqrt{3}}\right)^d \left(\frac{1}{\alpha r}\right)^{d-1}
\end{aligned}$$

Let $r_0 = N^{-\frac{1}{2d}}$ and $r_{i+1} = r_i + \alpha r_i^2$, and let T be the smallest index i such that $r_i \leq \rho$. With this notation in hand, we have

$$N_c(x(m^*), N^{-\frac{1}{2d}}, \rho) \leq \sum_{i=0}^{T} N_c(x(m^*), r_i, \alpha r_i + r_i^2).$$

The summation on the right-hand side can be upper-bounded by an integral. Using that the difference between r_i and r_{i+1} is αr_i^2, we need to divide terms in the sum by αr_i^2 when converting to an integral. Doing this, we obtain,

$$\begin{aligned}
N_c(x(m^*), N^{-\frac{1}{2d}}, \rho) &\leq \int_{N^{-\frac{1}{2d}}}^{\rho + \alpha\rho^2} N_c(x(m^*), r, \alpha r^2) dr \\
&\leq \text{const } + (d+1)^2 \left(\frac{8}{\sqrt{3}}\right)^d \left(\frac{1}{\alpha}\right)^d \int_{N^{-\frac{1}{2d}}}^{\infty} r^{-(1+d)} dr \\
&\leq \text{const } + (d+1) \left(\frac{4}{\alpha}\right)^d N^{\frac{1}{2}}.
\end{aligned}$$

This completes the proof, and provides the hidden constant of complexity as $(d+1)\left(\frac{4}{\alpha}\right)^d$. Thus larger values for α should result in fewer computed elements in the annulus $\mathcal{A}_d(x(m^*), r, r + \alpha r^2)$, which makes sense given that large values of α imply larger surplus energies and thus larger elimination zones. □

Lemma C.7.4 (Packing beyond ρ)**.** *If we assume* (C.8) *from Theorem 3.3.2 and that the medoid index* m^* *is the first element processed by* `trimed`*, then the number of computed elements in* $\mathcal{A}_d(x(m^*), \rho, \infty)$ *is less than* $(1 + 4E^*/(\alpha\rho^2))^d$*.*

Proof. Recall that we at assuming $m^* = 1$, that is that the medoid is the first element processed in `trimed`. All elements beyond radius $2E^*$ are eliminated by type 1 eliminations (Figure 3.1), which provides the first inequality below. Then, as the excess energy is at least $\epsilon = \alpha\rho^2$ for all elements beyond radius ρ of $x(m^*)$, we apply Lemma C.7.8 with $\epsilon = \alpha\rho^2/2$ to obtain the second inequality below,

$$
\begin{aligned}
N_c(m(x), \rho, \infty) &\leq N_c(m(x), \rho, 2E^*) \\
&\leq \frac{(2E^* + \frac{1}{2}\alpha\rho^2)^d}{(\frac{1}{2}\alpha\rho^2)^d} \\
&\leq \left(1 + \frac{4E^*}{\alpha\rho^2}\right)^d .
\end{aligned}
$$

□

Lemma C.7.5 (Annulus packing)**.** *For* $0 \leq r$ *and* $0 < \epsilon \leq w$*. If*

$$\mathcal{X} \subset \mathcal{A}_d(0, r, r + w),$$

where

$$\forall x \in \mathcal{X}, \mathcal{B}_d(x, \epsilon) \cup \mathcal{X} = \{x\}, \tag{C.10}$$

then,

$$|\mathcal{X}| \leq (d+1)^2 \left(\frac{4}{\sqrt{3}}\right)^d \frac{w\,(r+w)^{d-1}}{\epsilon^d}.$$

Proof. The condition (C.10) implies,

$$\forall x, x' \in \mathcal{X} \times \mathcal{X}, \mathcal{B}\left(x, \frac{\epsilon}{2}\right) \cup \mathcal{B}\left(x', \frac{\epsilon}{2}\right) = \emptyset. \tag{C.11}$$

Using that $\epsilon \in (0, w]$ and Lemma C.7.6, one can show that for all $x \in \mathcal{A}(0, r, r + w)$,

$$\text{volume}\left(\mathcal{B}\left(x, \frac{\epsilon}{2}\right) \cap \mathcal{A}(0, r, r+w)\right) > \frac{1}{d+1}\left(\frac{3}{4}\right)^{\frac{d}{2}} V_d\left[\frac{\epsilon}{2}\right] \tag{C.12}$$

Combining (C.11) with (C.12) we have,

$$\text{volume}\left(\bigcup_{x\in\mathcal{X}} \mathcal{B}\left(x, \frac{\epsilon}{2}\right) \cap \mathcal{A}(0, r, r+w)\right) > \frac{V_d\,[1]}{d+1}\left(\frac{\sqrt{3}}{4}\right)^d |\mathcal{X}|\epsilon^d. \tag{C.13}$$

Letting $S_d[\epsilon]$ denote the surface area of a $\mathcal{B}(0,\epsilon)$, it is easy to see that

$$\text{volume}\left(\mathcal{A}(0,r,r+w)\right) < S_d[1]\, w\,(r+w)^{d-1}. \tag{C.14}$$

Combining (C.13) with (C.14) we get,

$$\frac{V_d[1]}{d+1}\left(\frac{\sqrt{3}}{4}\right)^d |\mathcal{X}|\epsilon^d < S_d[1]\, w\,(r+w)^{d-1}.$$

which combined with the fact that

$$\begin{aligned}\frac{S_d[1]}{V_d[1]} &= \left(\frac{\frac{dV_d}{dr}}{V_d}\right)_{r=1}\\ &= d,\end{aligned}$$

provides us with,

$$|\mathcal{X}| \le (d+1)^2\left(\frac{4}{\sqrt{3}}\right)^d \frac{w\,(r+w)^{d-1}}{\epsilon^d}.$$

□

Lemma C.7.6 (Volume of ball intersection)**.** *For* $x_0, x_1 \in \mathbb{R}^d$ *with* $\|x_0 - x_1\| = 1$,

$$\frac{\text{volume}\left(\mathcal{B}_d(x_0,1)\cap\mathcal{B}_d(x_1,1)\right)}{\text{volume}\left(\mathcal{B}_d(x_0,1)\right)} \ge \frac{1}{d+1}\left(\frac{3}{4}\right)^{\frac{d}{2}}.$$

Proof. Let $V_d[r]$ denote the volume of $\mathcal{B}_d(0,r)$. It is easy to see that,

$$\begin{aligned}\text{volume}\left(\mathcal{B}_d(x_0,1)\cap\mathcal{B}_d(x_1,1)\right) &= 2\int_0^{\frac{1}{2}} V_{d-1}\left[\sqrt{x(2-x)}\right]dx\\ &\ge 2\int_0^{\frac{1}{2}} V_{d-1}\left[\sqrt{\frac{3}{2}x}\right]dx\\ &\ge 2V_{d-1}[1]\int_0^{\frac{1}{2}}\left(\frac{3}{2}x\right)^{\frac{d-1}{2}}dx\\ &\ge 2V_{d-1}[1]\left(\frac{3}{2}\right)^{\frac{d-1}{2}}\left(\frac{2}{d+1}\right)\left(\frac{1}{2}\right)^{\frac{d+1}{2}}\\ &\ge V_{d-1}[1]\left(\frac{3}{2}\right)^{\frac{d-1}{2}}\left(\frac{2}{d+1}\right)\left(\frac{1}{2}\right)^{\frac{d-1}{2}}\\ &\ge V_{d-1}[1]\left(\frac{3}{4}\right)^{\frac{d-1}{2}}\left(\frac{2}{d+1}\right).\end{aligned}$$

Using that $\frac{V_{d-1}[1]}{V_d[1]} > \frac{1}{\sqrt{\pi}}$, we divide the intersection volume through by $V_d[1]$ to obtain,

$$\begin{aligned}\frac{\text{volume}\left(\mathcal{B}_d\left(x_0,1\right)\cap\mathcal{B}_d\left(x_1,1\right)\right)}{\text{volume}\left(\mathcal{B}_d\left(x_0,1\right)\right)} &\geq \left(\frac{3}{4}\right)^{\frac{d-1}{2}}\left(\frac{2}{\sqrt{\pi}(d+1)}\right)\\ &\geq \frac{1}{d+1}\left(\frac{3}{4}\right)^{\frac{d}{2}}\end{aligned}$$

☐

Lemma C.7.7 (Packing balls in a ball)**.** *The number of non-intersecting balls of radius ϵ which can be packed into a ball of radius r in $\mathbb{R}^d$ is less than $\left(\frac{r}{\epsilon}\right)^d$*

Proof. The technique used here is a loose version of that used in proving Lemma C.7.5. The volume of $\mathcal{B}_d(0,\epsilon)$ is a factor $(r/\epsilon)^d$ smaller than that of $\mathcal{B}_d(0,r)$. As the balls of radius ϵ are non-overlapping, the volume of their union is simply the sum of their volumes. The result follow from the fact that the union of the balls of radius ϵ is contained within the ball of radius r. ☐

Lemma C.7.8 (Packing points in a ball)**.** *Given $\mathcal{X} \subset \mathcal{B}_d(0,r)$ such that no two elements of $\mathcal{X}$ lie within a distance of ϵ of each other, $|\mathcal{X}| < \left(\frac{2r+\epsilon}{\epsilon}\right)^d$.*

Proof. As no two elements lie within distance ϵ of each other, balls of radius $\epsilon/2$ centred at elements are non-intersecting. As each of the balls of radius $\epsilon/2$ centred at elements of $\mathcal{X}$ lies entirely within $\mathcal{B}_d(0,r+\epsilon/2)$, we can apply Lemma (C.7.7), arriving at the result. ☐

Pseudocode for `trikmeds`

In Alg. (13) we present `trikmeds`. It is decomposed into algorithms for initialization (14), updating medoids (15), assigning data to clusters (16) and updating bounds on the `trimed` derived bounds (17). Table C.2 summarised all of the variables used in `trikmeds`.

When there are no distance bounds, the location of the bottleneck in terms of distance calculations depends on N/K^2. If $N/K \gg K$, the bottleneck lies in updating medoids, which can be improved through the strategy used in `trimed`. If $N/K \ll K$, the bottleneck lies in assigning elements to clusters, which is effectively handled through the approach of Elkan [2003].

Algorithm 13 `trikmeds`

```
initialize()
while not converged do
    update-medoids()
    assign-to-clusters()
    update-sum-bounds()
end while
```

Algorithm 14 `initialize`

```
// Initialize medoid indices, uniform random sample without replacement (or otherwise)
$\{m(1), \ldots, m(K)\} \leftarrow$ uniform-no-replacement($\{1, \ldots, N\}$)
for $k = 1 : K$ do
    // Initialize medoid and set cluster count to zero
    $c(k) \leftarrow x(m(k))$
    $v(k) \leftarrow 0$
    // Set sum of in-cluster distances to medoid to zero
    $s(k) \leftarrow 0$
end for
for $i = 1 : N$ do
    for $k = 1 : K$ do
        // Tightly initialize lower bounds on data-to-medoid distances
        $l_c(i, k) \leftarrow \|x(i) - c(k)\|$
    end for
    // Set assignments and distances to nearest (assigned) medoid
    $a(i) \leftarrow \arg\min_{k \in \{1, \ldots, K\}} l_c(i, k)$
    $d(i) \leftarrow l_c(i, a(i))$
    // Update cluster count
    $v(a(i)) \leftarrow v(a(i)) + 1$
    // Update sum of distances to medoid
    $s(a(i)) \leftarrow s(a(i)) + d(i)$
    // Initialize lower bound on sum of in-cluster distances to $x(i)$ to zero
    $l_s(i) \leftarrow 0$
end for
$V(0) \leftarrow 0$
for $k = 1 : K$ do
    // Set cumulative cluster count
    $V(k) \leftarrow V(k - 1) + v(k)$
    // Initialize lower bound on in-cluster sum of distances to be tight for medoids
    $l_s(m(k)) \leftarrow s(k)$
end for
// Make clusters contiguous
contiguate()
```

Algorithm 15 `update-medoids`

```
for $k = 1 : K$ do
    for $i = V(k-1) : V(k) - 1$ do
        // If the bound test cannot exclude $i$ as $m(k)$
        if $l_s(i) < s(k)$ then
            // Make $l_s(i)$ tight by computing and cumulating all in-cluster distances to $x(i)$,
            $l_s(i) \leftarrow 0$
            for $i' = V(k-1) : V(k) - 1$ do
                $\tilde{d}(i') \leftarrow \|x(i) - x(i')\|$
                $l_s(i) \leftarrow l_s(i) + \tilde{d}(i')$
            end for
            // Re-perform the test for $i$ as candidate for $m(k)$, now with exact sums. If $i$ is the new best candidate, update some cluster information
            if $l_s(i) < s(k)$ then
                $s(k) \leftarrow l_s(i)$
                $m(k) \leftarrow i$
                for $i' = V(k-1) : V(k) - 1$ do
                    $d(i') \leftarrow \|x(i) - x(i')\|$
                end for
            end if
            // Use computed distances to $i$ to improve lower bounds on sums for all samples in cluster $k$ (see Figure X)
            for $i' = V(k-1) : V(k) - 1$ do
                $l_s(i') \leftarrow \max\left(l_s(i'), |\tilde{d}(i')v(k) - l_s(i)|\right)$
            end for
        end if
    end for
    // If the medoid of cluster $k$ has changed, update cluster information
    if $m(k) \neq V(k-1)$ then
        $p(k) \leftarrow \|c(k) - x(m(k))\|$
        $c(k) \leftarrow x(m(k))$
    end if
end for
```

Algorithm 16 `assign-to-clusters`

```
// Reset variables monitoring cluster fluxes,
for $k = 1 : K$ do
    // the number of arrivals to cluster $k$,
    $\Delta_{n-in}(k) \leftarrow 0$
    // the number of departures from cluster $k$,
    $\Delta_{n-out}(k) \leftarrow 0$
    // the sum of distances to medoid $k$ of samples which leave cluster $k$
    $\Delta_{s-out}(k) \leftarrow 0$
    // the sum of distances to medoid $k$ of samples which arrive in cluster $k$
    $\Delta_{s-in}(k) \leftarrow 0$
end for
for $i = 1 : N$ do
    // Update lower bounds on distances to medoids based on distances moved by medoids
    for $k = 1 : K$ do
        $l(i,k) = l(i,k) - p(k)$
    end for
    // Use the exact distance of current assignment to keep bound tight (might save future calcs)
    $l(i, a(i)) = d(i)$
    // Record current assignment and distance
    $a_{old} = a(i)$
    $d_{old} = d(i)$
    // Determine nearest medoid, using bounds to eliminate distance calculations
    for $k = 1 : K$ do
        if $l(i,k) < d(i)$ then
            $l(i,k) \leftarrow \|x(i) - c(k)\|$
            if $l(i,k) < d(i)$ then
                $a(i) = k$
                $d(i) = l(i,k)$
            end if
        end if
    end for
    // If the assignment has changed, update statistics
    if $a_{old} \neq a(i)$ then
        $v(a_{old}) = v(a_{old}) - 1$
        $v(a(i)) = v(a(i)) + 1$
        $l_s(i) = 0$
        $\Delta_{n-in}(a(i)) = \Delta_{n-in}(a(i)) + 1$
        $\Delta_{n-out}(a_{old}) = \Delta_{n-out}(a_{old}) + 1$
        $\Delta_{s-in}(a(i)) = \Delta_{s-in}(a(i)) + d(i)$
        $\Delta_{s-out}(a_{old}) = \Delta_{s-out}(a_{old}) + d_{old}$
    end if
end for
// Update cumulative cluster counts
for $k = 1 : K$ do
    $V(k) \leftarrow V(k-1) + v(k)$
end for
contiguate()
```

Table C.2 – Table Of Notation For `trikmeds`

N : number of training samples
i : index of a sample, $i \in \{1, \dots, N\}$
$x(i)$: sample i
K : number of clusters
k : index of a cluster, $k \in \{1, \dots, K\}$
$m(k)$: index of current medoid of cluster k, $m(k) \in \{1, \dots, N\}$
$c(k)$: current medoid of cluster k, that is $c(k) = x(m(k))$
$n_1(i)$: cluster index of centroid nearest to $x(i)$
$a(i)$: cluster to which $x(i)$ is currently assigned
$d(i)$: distance from $x(i)$ to $c(a(i))$
$v(k)$: number of samples assigned to cluster k
$V(k)$: number of samples assigned to a cluster of index less than $k+1$
$l_c(i,k)$: lowerbound on distance from $x(i)$ to $m(k)$
$l_s(i)$: lowerbound on $\sum_{i':a(i')=a(i)} \|x(i') - x(i)\|$
$p(k)$: distance moved (teleported) by $m(k)$ in last update
$s(k)$: sum of distances of samples in cluster k to medoid k

Algorithm 17 `update-sum-bounds`

```
for $k = 1 : K$ do
    // Obtain absolute and net fluxes of energy and count, for cluster $k$
    $\mathcal{J}_s^{abs}(k) = \Delta_{s-in}(k) + \Delta_{s-out}(k)$
    $\mathcal{J}_s^{net}(k) = \Delta_{s-in}(k) - \Delta_{s-out}(k)$
    $\mathcal{J}_n^{abs}(k) = \Delta_{n-in}(k) + \Delta_{n-out}(k)$
    $\mathcal{J}_n^{net}(k) = \Delta_{n-in}(k) - \Delta_{n-out}(k)$
    for $i = V(k-1) : V(k) - 1$ do
        // Update the lower bound on the sum of distances
        $l_s(i) \leftarrow l_s(i) - \min(\mathcal{J}_s^{abs}(k) - \mathcal{J}_n^{net}(k)d(i), \mathcal{J}_n^{abs}(k)d(i) - \mathcal{J}_s^{net}(k))$
    end for
end for
```

Algorithm 18 `contiguate`

```
// This function performs an in place rearrangement over of variables $a, d, l, x$ and $m$
// The permutation applied to $a, d, l$ and $x$ has as result a sorting by cluster,
// $a(i) = k$ if $i \in \{V(k-1), V(k)\}$ for $k \in \{1, \dots, K\}$
// and moreover that the first element of each cluster is the medoid,
// $m(k) = V(k-1)$ for $k \in \{1, \dots, K\}$
```

Datasets

- *Birch1*, *Birch2* : Synthetic 2-D datasets available from https://cs.joensuu.fi/sipu/datasets/
- *Europe* : Border map of Europe available from https://cs.joensuu.fi/sipu/datasets/
- *U-Sensor Net* : Undirected 2-D graph data. Points drawn uniformly from unit square, with an undirected edge connecting points when the distance between them is less than $1.25\sqrt{N}$
- *D-Sensor Net* : Directed 2-D graph data. Points drawn uniformly from unit square, with directed edge connecting points when the distance between them is less than $1.45\sqrt{N}$, direction chosen at random.
- *Europe rail* : The European rail network, the shapefile is available at http://www.mapcruzin.com/free-europe-arcgis-maps-shapefiles.htm. We extracted edges from the shapefile using `networkx` available at https://networkx.github.io/.
- *Pennsylvania road* The road network of Pennsylvania, the edge list is available directly from https://snap.stanford.edu/data/
- *Gnutella* Peer-to-peer network data, available from https://snap.stanford.edu/data/
- *MNIST (0)* The '0's in the MNIST training dataset.
- *Conflong* The conflongdemo data is available from https://cs.joensuu.fi/sipu/datasets/
- *Colormo* The colormoments data is available at http://archive.ics.uci.edu/ml/datasets/Corel+Image+Features
- *MNIST50* The MNIST dataset, projected into 50-dimensions using a random projection matrix where each of the 784×50 elements in the matrix is i.i.d. $\mathcal{N}(0,1)$.
- *S1, S2, S3, S4, A1, A2, A3* All of these synthetic datasets are available from https://cs.joensuu.fi/sipu/datasets/.
- *thyroid, yeast, wine, breast, spiral* All of these real world datasets are available from https://cs.joensuu.fi/sipu/datasets/.

Scaling with dimension of `TOPRANK` and `TOPRANK2`

Recall the assumption (3.3) made for the `TOPRANK` and `TOPRANK2` algorithms. The assumption states that as one approaches the minimum energy E^* from above, the density of elements decreases. In other words, the lowest energy elements stand out from the rest and are not bunched up with very similar energies.

Consider the case where elements are points in $\mathbb{R}^d$. Suppose that the density f_X of points around the medoid is bounded by $0 < \rho_0 \leq f_X \leq \rho_1$, and that the energy grows quadratically in radius about the medoid. Then, as the number of points at radius ϵ is

$O(\epsilon^{d-1})$, the density (by energy) of points at radius ϵ is $O(\epsilon^{d-2})$. Thus for $d = 1$ the assumption for `TOPRANK` and `TOPRANK` does not hold, which results in poor performance for $d = 1$. For $d = 2$, the assumption holds, as the density (by energy) of points is constant. For $d \geq 2$, as d increases the energy distribution becomes more and more favourable for `TOPRANK` and `TOPRANK2`, as the low ranking elements become more and more distinct with low energies becoming less probable. This explains the observation that `TOPRANK` scales well with dimension in Figure 3.3.

Example where geometric median is a poor approximation of medoid

There is no guarantee that the geometric median is close to the set medoid. Moreover, the element in $\mathcal{S}$ which is nearest to $g(\mathcal{S})$ is not necessarily the medoid, as illustrated in the following example. Suppose $S = \{x(1), \ldots, x(20)\} \subset \mathbb{R}^2$, with $x(i) = (0, 1)$ for $i \in \{1, \ldots, 9\}$, $x(i) = (0, -1)$ for $i \in \{10, \ldots, 18\}$, $x(19) = (1/2, 0)$ and $x(20) = (-1/2, 0)$. The geometric median is $(0, 0)$ and the nearest points to the geometric median, $x(19)$ and $x(20)$ have energy $1 + 18\sqrt{3}/2 \approx 16.6$. However, points $\{x(1), \ldots, x(18)\}$ have energy $2\sqrt{3}/2 + 9 = 10.7$. Thus by choosing a point in $\mathcal{S}$ which is nearest to the geometric median, one is choosing the element with the highest energy, the opposite of the medoid.

Note the above example appears to violate the assumptions required for $O(N^{3/2})$ convergence of `trimed`, as it requires that the probability density function vanishes at the distribution median. Indeed, in $\mathbb{R}^d$ it is the case that if the $O(N^{3/2})$ assumptions are satisfied, the set medoid converges to the geometric median, and so the geometric median is a good approximation. We stress however that the geometric median is only relevant in vector spaces.

Miscellaneous

Figure C.4 illustrates the idea behind algorithm `trimed`, comments in the caption.

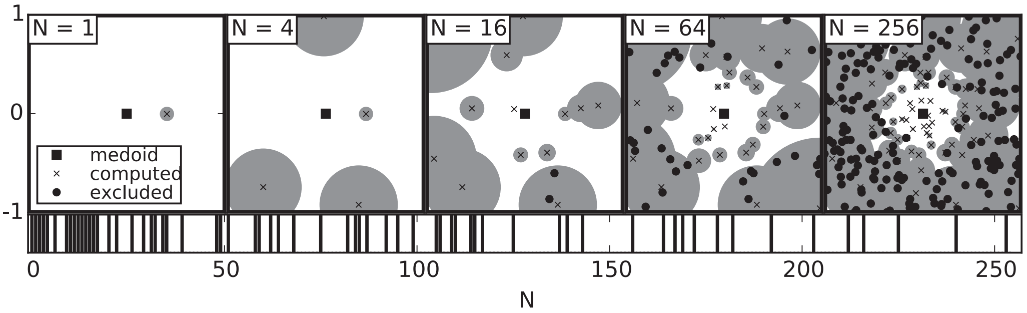

Figure C.4 – Eliminating samples as potential medoids using only type 1 elimination, where we assume that the medoid and its energy E^* are known, and so the radius of the exclusion ball of an element x is $E(x) - E^*$. Uniformly sampling from $[-1, 1] \times [-1, 1]$, energies are computed only if the sample drawn does not lie in the exclusion zone (union of balls). If the energy at x is computed, the exclusion zone is augmented by adding $\mathcal{B}_d(x, E(x) - E^*)$. Top left to right: the distribution of samples which are computed and excluded. Bottom: the times at which samples are computed. We prove that probability of computation at time n is $O(n^{-\frac{1}{2}})$.

D Appendix for Chapter 4

Generalised K-medoids results

The potential uses of `clarans` as a K-medoids algorithm go well beyond K-Means initialization. In this Appendix, we wish to demonstrate that `clarans` should be chosen as a default K-medoids algorithm, rather than `medlloyd`. In its most general form, the K-medoids problem is to minimize,

$$E(\mathcal{C}) = \frac{1}{N} \sum_{i=1}^{N} \arg\min_{i' \in \mathcal{C}} f(x(i), x(i')). \tag{D.1}$$

We assume that f is of the form,

$$f(x(i), x(i')) = \psi(\mathrm{dist}(x(i), x(i'))), \tag{D.2}$$

where ψ is non-decreasing, and samples belong to a metric space with metric $\mathrm{dist}(\cdot, \cdot)$. Constraint D.2 allows us to use the triangle inequality to eliminate certain distance calculations. We now present examples comparing `clarans` and `medlloyd` in various settings, showing the effectiveness of `clarans`. Table D.1 describes artificial problems, with results in Figure D.1. Table D.2 describes real-world problems, with results in Figure D.2.

The task

We state precisely 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,

$$\begin{aligned}
\mathrm{dist}(x(i), x(i')) &\geq 0, \\
\mathrm{dist}(x(i), x(i)) &= 0, \\
\mathrm{dist}(x(i), x(i')) &= \mathrm{dist}(x(i'), x(i)), \\
\mathrm{dist}(x(i), x(i'')) &\leq \mathrm{dist}(x(i), x(i')) + \mathrm{dist}(x(i'), x(i'')),
\end{aligned}$$

	N	K	type	metric	$\psi(d)$
syn-1	2000	40	sequence	Levenshtein	d
syn-2	20000	100	sparse-v	l_2	d^2
syn-3	28800	144	dense-v	l_1	e^d
syn-4	20000	100	dense-v	l_∞	$I_{d>0.05}$

Table D.1 – Synthetic datasets used for comparing K-medoids algorithms (Figure D.1). **syn-1**: Each of the cluster centers is a random binary sequence of 16 bits (0/1). 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 $\mathbb{R}^{10^6}$, non-zero at exactly 5 indices, with the 5 non-zero values drawn from $N(0,1)$. Each sample is a linear combination of two centers, with coefficients 1 and Q respectively, where $Q \sim U[-0.5, 0.5]$. **syn-3**: Centers are integer co-ordinates of an 12×12 grid. For each center, 50 samples are generated, each sample being the center plus Gaussian noise of identity covariance, as in the simulation data in the main text. **syn-4**: Data are points drawn uniformly from $[0,1]^2$. We attempt 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	$\psi(d)$
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

Table D.2 – Real datasets used for comparing K-medoids algorithms (Figure D.2), with data urls in D.5. **rcv1**: The Reuters Corpus Volume I training set of Lewis et al. [2004], a sparse datasets containing news article categorization annotation. **genome**: Nucleotide subsequences of lengths 10,11 or 12, randomly selected from chromosome 10 of a Homo Sapiens. Note that the normalised Levenshtein metric [Yujian and Bo, 2007] is used. **mnist**: The test images of the MNIST hand-written digit dataset. **words**: A comprehensive English language word list.

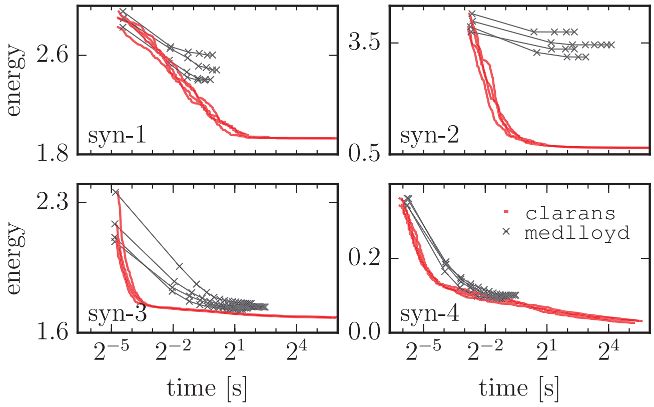

Figure D.1 – Results on synthetic datasets. Algorithms `clarans` and `medlloyd` are run four times with random seedings. Each experiment is run with a time limit of 64 seconds. The vertical axis is mean energy (dissimilarity) across samples. In all experiments, `medlloyd` gets trapped in local minima before 64 seconds have elapsed, and `clarans` always obtains significantly lower energies than `medlloyd`.

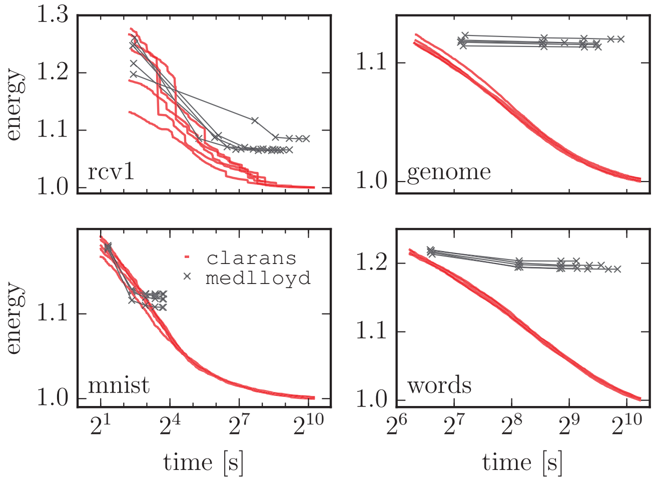

Figure D.2 – Results on real datasets. Vertical axes are energies relative to the lowest energy found. We observe that `medlloyd` performs very poorly on sequence datasets (right), failing to find clusterings significantly better than the random initializations. While an improvement over the initial seeding is obtained using `vik` on the vector datasets (left), the energies obtained using `clarans` are significantly lower. Runs with `clarans` appear to converge to a common energy solution, even though initial energies vary greatly, as is the case in on dataset rcv1. The majority of runs with `medlloyd` converge to a local minimum before the allotted time limit of 2^{10} seconds.

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

$$\begin{aligned}\psi(0) &= 0, \\ v_1 \leq v_2 &\iff \psi(v_1) \leq \psi(v_2),\end{aligned}$$

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

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

The `pam` algorithm

Algorithm 19 The `pam` algorithm of Kaufman and Rousseeuw [1990] is a computationally inefficient predecessor of `clarans`. 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 a decrease in energy, proceed, otherwise stop.

```
1:  $t \leftarrow 0$
2:  Initialize $\mathcal{C}_0 \subset \{1, \dots, N\}$.
3:  while true do
4:      for $i_p \in \{1, \dots, N\} \setminus \mathcal{C}_t$ do
5:          for $k_p \in \{1, \dots, K\}$ do
6:              $\psi^p_{t+1}(i_p, k_p) \leftarrow \sum_{i=1}^{N} \min_{i' \in \mathcal{C}_t \setminus \{c_t(k_p)\} \cup \{i_p\}} \psi(\text{dist}(x(i), x(i')))$
7:          end for
8:      end for
9:      $i^*_p, k^*_p \leftarrow \arg\min_{i_p, k_p} \psi^p_{t+1}(i_p, k_p)$
10:     if $\psi^p_{t+1}(i^*_p, k^*_p) < 0$ then
11:         $\mathcal{C}_{t+1} \leftarrow \mathcal{C}_t \setminus \{c_t(k^*_p)\} \cup \{i^*_p\}$
12:     else
13:         break
14:     end if
15:     $t \leftarrow t + 1$
16: end while
```

`clarans` In detail with accelerations

We start by presenting modified notation, required to describe our optimizations of `clarans` [Ng and Han, 1994] in full pseudocode. As before, we will let the N samples which we want to partition into K clusters be $x(1), \dots, x(N)$. Let $t \in \{1, \dots, \infty\}$ denote the current round of the algorithm. Let $c_t(k) \in \{1, \dots, N\}$ be the index of the sample chosen as the center of cluster $k \in \{1, \dots, K\}$ at iteration t, so that $x(c_t(k))$ is the center of cluster k at iteration t. Let $\mathcal{C}_t = \{c_t(k) \mid k \in \{1, \dots, K\}\} \subset \{1, \dots, N\}$ denote all such center indices. We let $a_t^1(i)$ be the cluster of sample i, that is

$$a_t^1(i) = \operatorname*{arg\,min}_{k \in \{1,\ldots,K\}} f(x(i), x(c_t(k))). \tag{D.3}$$

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

$$\psi_t(k) = \sum_{i : a_t^1(i) = k} f(x(i), x(c_t(k))).$$

Let $\psi_t = \sum_k \psi_t(k)$ be the total energy, the quantity which we ultimately wish to minimize.

We assume here that dissimilarity can be decomposed as in Eqn. (D.2), which will enable the use of the triangle inequality.

Let $d_t^1(i)$ be the distance at iteration t of sample i to its nearest center, that is

$$d_t^1(i) = \min_{i' \in \mathcal{C}_t} \operatorname{dist}(x(i), x(i')).$$

Under assumption (D.2), we now have (D.3) taking the form,

$$a_t^1(i) = \operatorname*{arg\,min}_{k \in \{1,\ldots,K\}} \operatorname{dist}(x(i), x(c_t(k))),$$

so that $d_t^1(i) = \operatorname{dist}(x(i), x(c_t(a_t^1(i))))$. In the same way as we use $a_t^1(i)$ and $d_t^1(i)$ for the nearest center, we will use $a_t^2(i)$ and $d_t^2(i)$ for the second nearest center, that is

$$\begin{aligned}
d_t^2(i) &= \min_{i' \in \mathcal{C}_t \setminus \{c_t(a_t^1(i))\}} \operatorname{dist}(x(i), x(i')), \\
a_t^2(i) &= \operatorname*{arg\,min}_{k \in \{1,\ldots,K\} \setminus \{a_t^1(i)\}} \operatorname{dist}(x(i), x(c_t(k))),
\end{aligned}$$

so that $d_t^2(i) = \operatorname{dist}(x(i), x(c_t(a_t^2(i))))$. 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 $\psi(d_t^1(i))$. Finally, let the *margin* of sample i be defined as $m_t(i) = \psi(d_t^2(i)) - \psi(d_t^1(i))$. Some cluster specific quantities which are required in the accelerated algorithm are,

$$\begin{aligned}
N_t(k) &= |\{i : a_t^1(i) = k\}|, \\
D_t^1(k) &= \max_{i : a_t^1(i) = k} d_t^1(i), \\
D_t^2(k) &= \max_{i : a_t^1(i) = k} d_t^2(i), \\
M_t^*(k) &= \frac{1}{N_t(k)} \sum_{i : a_t^1(i) = k} m_t(i).
\end{aligned} \tag{D.4}$$

The key triangle inequality results used to accelerate `clarans` evaluations are now presented, with proofs in Appendix refapp::accelerating. Firstly,

$$\begin{aligned}&\text{dist}(x(i_p), x(c_t(k_p))) \geq D_t^1(k_p) + D_t^2(k_p) \implies \\ &\qquad \text{change in energy of cluster } k_p \text{ is } N_t(k_p)M_t^*(k_p),\end{aligned}$$

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,

$$\begin{aligned}&k \neq k_p \ \wedge \text{dist}(x(c_t(k)), x(i_p)) \geq 2D_t^1(k) \implies \\ &\qquad\qquad\qquad\qquad \text{no change in energy of cluster } k,\end{aligned}$$

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 computation of this quantity can sometimes be avoided by using the inequality,

$$\text{dist}(x(c_t(k)), x(i_p)) \geq cc_t(a_t^1(i_p), k) - D_t^1(i_p),$$

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

$$\begin{aligned}&\min(\text{dist}(x(c_t(k_p)), x(c_t(k))), \text{dist}(x(i_p), x(c_t(k)))) \\ &\qquad > D_t^1(k) + D_t^2(k) \implies \text{no change in cluster } k.\end{aligned}$$

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 D.4.2.

Review of notation and ideas

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

$$\begin{aligned}\delta_t(i \mid k_p \lhd i_p) &= \text{ energy after swap } - \text{energy before swap} \\ &= \min_{i' \in \mathcal{C}_t \setminus \{c_t(k_p)\} \cup \{i_p\}} \psi(\text{dist}(x(i), x(i'))) - \psi(d_t^1(i)).\end{aligned}$$

We choose subscript 'p' for k_p and i_p, as together they define a *proposed* swap. We will write $a^{12}d_t^{12}(i) = \{a_t^1(i), a_t^2(i), d_t^1(i), d_t^2(i)\}$ throughout for brevity. Finally, let

$$
\begin{aligned}
D_t^1(k) &= \max_{i:a_t^1(i)=k} d_t^1(i), \\
D_t^2(k) &= \max_{i:a_t^1(i)=k} d_t^2(i).
\end{aligned}
$$

Algorithm 20 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 $\mathcal{C}_{t+1} \neq \mathcal{C}_t$, and all data whose nearest or second nearest center change needs updating.

```
 1: Make proposal $k_p \in \{1, \dots K\}$ and $i_p \in \{1, \dots, N\} \setminus \mathcal{C}_t$.
 2: $\Delta_t(k_p \lhd i_p) \leftarrow \frac{1}{N} \sum_{i=1}^{N} \delta_t(i \mid k_p \lhd i_p)$    ▷ The assignment evaluation step, see Alg. 21
 3: if $\Delta_t < 0$ then
 4:     $\mathcal{C}_{t+1} \leftarrow \mathcal{C}_t \setminus \{c_t(k_p)\} \cup \{i_p\}$
 5:     for $i \in \{1, \dots, N\}$ do
 6:         Set $a^{12}d_{t+1}^{12}(i)$    ▷ The update step, see Alg. 22
 7:     end for
 8: else
 9:     $\mathcal{C}_{t+1} \leftarrow \mathcal{C}_t$
10:     for $i \in \{1, \dots, N\}$ do
11:         $a^{12}d_{t+1}^{12}(i) \leftarrow a^{12}d_t^{12}(i)$
12:     end for
13: end if
```

Algorithm 21 Standard approach (level 0) with `clarans` for computing $\delta_t(i \mid k_p \lhd i_p)$ at iteration t, as described in Ng and Han [1994]. Note however that here we do not store all N^2 distances, as in Ng and Han [1994].

```
$d \leftarrow \text{dist}(x(i), x(i_p))$
if $a_t^1(i) = k_p$ then
    if $d \geq d_t^2(i)$ then
        $\delta_t(i \mid k_p \lhd i_p) \leftarrow \psi(d_t^2(i)) - \psi(d_t^1(i))$
    else
        $\delta_t(i \mid k_p \lhd i_p) \leftarrow \psi(d) - \psi(d_t^1(i))$
    end if
else
    if $d \geq d_t^1(i)$ then
        $\delta_t(i \mid k_p \lhd i_p) \leftarrow 0$
    else
        $\delta_t(i \mid k_p \lhd i_p) \leftarrow \psi(d) - \psi(d_t^1(i))$
    end if
end if
```

Algorithm 22 Simple approach (level 0) with `clarans` for computing $a^{12}d^{12}_{t+1}(i)$

```
// If the center which moves is nearest or second nearest, complete update required
if $a^1_t(i) = k_p$ or $a^2_t(i) = k_p$ then
    Get $\text{dist}(x(i), x(c_{t+1}(k)))$ for all $k \in \{1, \ldots, K\}$
    Use above $k$ distances to set $a^{12}d^{12}_{t+1}(i)$
else
    // $d^1_t(i)$ and $d^2_t(i)$ are still valid distances, so need only check new candidate center $k_p$
    $d \leftarrow \text{dist}(x(i), x(i_p))$
    Use the fact that $\{d^1_{t+1}(i), d^2_{t+1}(i)\} \subset \{d^1_t(i), d^2_t(i), d\}$ to set $a^{12}d^{12}_t(i)$
end if
```

Accelerating `clarans`

We now discuss in detail 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.

Basic triangle inequalities bounds

We show how $\delta_t(i \mid k_p \lhd i_p)$ can be bounded, with the final bounding illustrated in Figure D.3. There are four bounds to consider : upper and lower bounds for each of the two cases $k_p = a^1_t(i)$ (the center being replaced is the center of element i) and $k_p \neq a^1_t(i)$ (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 $k_p \neq a^1_t(i)$,

$$\begin{aligned}
\delta_t(i \mid k_p \lhd i_p) &= \min_{i' \in \mathcal{C}_t \setminus \{c_t(k_p)\} \cup \{i_p\}} \psi(\text{dist}(x(i), x(i'))) - \psi(d^1_t(i)), \\
&= \min_{i' \in \{c_t(a^1_t(i)), i_p\}} \psi(\text{dist}(x(i), x(i'))) - \psi(d^1_t(i)), \\
&\leq \psi(\text{dist}(x(i), x(c_t(a^1_t(i))))) - \psi(d^1_t(i)), \\
&= 0.
\end{aligned}$$

and thus we have

$$k_p \neq a^1_t(i) \implies \delta_t(i \mid k_p \lhd i_p) \leq 0. \tag{D.5}$$

Implication D.5 simply states the obvious fact that the energy of element i cannot increase when a center other than that of cluster $a^1_t(i)$ is replaced. The other upper bound case to consider is $k_p = a_t^1(i)$, which is similar,

$$\begin{aligned}
\delta_t(i \mid k_p \lhd i_p) &= \min_{i' \in \mathcal{C}_t \setminus \{c_t(k_p)\} \cup \{i_p\}} \psi(\mathrm{dist}(x(i), x(i'))) - \psi(d_t^1(i)), \\
&= \min_{i' \in \{c_t(a_t^2(i)), i_p\}} \psi(\mathrm{dist}(x(i), x(i'))) - \psi(d_t^1(i)), \\
&\leq \psi(\mathrm{dist}(x(i), x(c_t(a_t^2(i))))) - \psi(d_t^1(i)), \\
&= \psi(d_t^2(i)) - \psi(d_t^1(i)), \\
&= m_t(i), \\
&\leq M_t(k_p).
\end{aligned}$$

and thus we have

$$k_p = a_t^1(i) \implies \delta_t(i \mid k_p \lhd i_p) \leq M_t(i). \tag{D.6}$$

Implication D.6 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 $a_t^1(i)$ which is replaced. We now consider lower bounding $\delta_t(i \mid k_p \lhd i_p)$ for both the cases $a_t^1(i) = k_p$ and $a_t^1(i) \neq k_p$ simultaneously. We choose to bound them simultaneously as doing so separately arrives at the same bound.

$$\begin{aligned}
\delta_t(i \mid k_p \lhd i_p) &= \min_{i' \in \mathcal{C}_t \setminus \{c_t(k_p)\} \cup \{i_p\}} \psi(\mathrm{dist}(x(i), x(i'))) - \psi(d_t^1(i)), \\
&\geq \min_{i' \in \mathcal{C}_t \cup \{i_p\}} \psi(\mathrm{dist}(x(i), x(i'))) - \psi(d_t^1(i)), \\
&= \min_{i' \in \{c_t(a_t^1(i)), i_p\}} \psi(\mathrm{dist}(x(i), x(i'))) - \psi(d_t^1(i)), \\
&= \min\Big(0, \psi(\mathrm{dist}(x(i), x(i_p))) - \psi(d_t^1(i))\Big), \\
&\geq \min\Big(0, \psi(\mathrm{dist}(x(i), x(i_p))) - \psi(D_t^1(a_t^1(i)))\Big). \qquad (D.7)
\end{aligned}$$

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

$$d_p(k) = \mathrm{dist}(x(c_t(k)), x(i_p)).$$

The triangle inequality guarantees that,

$$\mathrm{dist}(x(i), x(i_p)) \geq \begin{cases} 0 & \text{if } d_p(a_t^1(i)) \leq D_t^1(a_t^1(i)), \\ d_p(a_t^1(i)) - D_t^1(a_t^1(i)) & \text{if } D_t^1(a_t^1(i)) < d_p(a_t^1(i)). \end{cases} \tag{D.8}$$

Using (D.8) in (D.7) we obtain,

$$\delta_t(i \mid k_p \lhd i_p) \geq \begin{cases} -\psi(D_t^1(a_t^1(i))) & \text{if } d_p(a_t^1(i)) \leq D_t^1(a_t^1(i)) \\ \psi(d_p(a_t^1(i)) - D_t^1(a_t^1(i))) - \psi(D_t^1(a_t^1(i))) & \text{if } D_t^1(a_t^1(i)) < d_p(a_t^1(i)) \leq 2D_t^1(a_t^1(i)) \\ 0 & \text{if } 2D_t^1(a_t^1(i)) < d_p(a_t^1(i)). \end{cases} \tag{D.9}$$

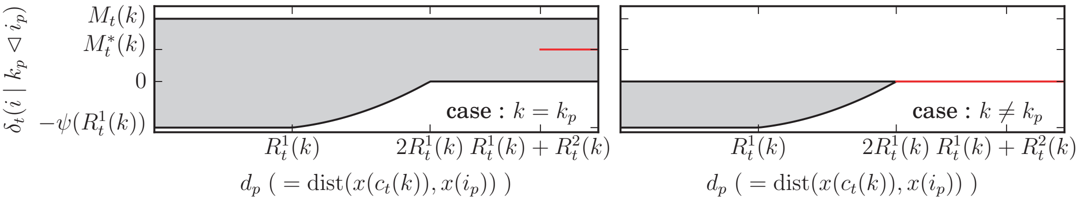

Figure D.3 – Illustrating the bounds. Dark gray regions denote possible changes in energy of elements. On the left, the case $k = k_p$, where the solid line segment is the average change in element energy in the case where d_p exceeds a certain radius. On the right, the case $k \neq k_p$, where sample energies can only decrease.

These are the lower bounds illustrated in Figure D.3. Define $\Delta(k \mid k_p \lhd i_p)$ to be the average change in energy for cluster k resulting from a proposed swap, that is,

$$\Delta_t(k \mid k_p \lhd i_p) = \frac{1}{N_t(k)} \sum_{i : a_t^1(i) = k_p} \delta_t(i \mid k_p \lhd i_p).$$

Let the average of the change in energy over all data resulting from a proposed swap be $\Delta_t(k_p \lhd i_p)$, that is

$$\Delta_t(k_p \lhd i_p) = \sum_k p_t(k) \Delta_t(k \mid k_p \lhd i_p).$$

One can show that for $k = k_p$,

$$d_p(k_p) \geq D_t^1(k_p) + D_t^2(k_p) \implies \Delta_t(k \mid k_p \lhd i_p) = M_t^*(k_p). \tag{D.10}$$

The equality (D.10) 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 D.3, left.

Level 1 proposal evaluation accelerations

What we wish to evaluate when considering a proposal is the mean change in energy, that is,

$$\frac{1}{N} \sum_{i=1}^{N} \delta_t(i \mid k_p \lhd i_p) = \frac{1}{N} \Bigg(\underbrace{\sum_{k : k \neq k_p} \sum_{i : a_t^1(i) = k} \delta_t(i \mid k_p \lhd i_p)}_{(N - N_t(k_p)) \Delta_t^-(k_p \lhd i_p)} + \underbrace{\sum_{i : a_t^1(i) = k_p} \delta_t(i \mid k_p \lhd i_p)}_{N_t(k_p) \Delta(k_p \mid k_p \lhd i_p)} \Bigg). \tag{D.11}$$

Where in (D.11) we define $\Delta_t^-(k_p \lhd i_p)$ as,

$$\Delta_t^-(k_p \lhd i_p) = \frac{1}{N - N_t(k_p)} \sum_{k : k \neq k_p} \sum_{i : a_t^1(i) = k} \delta_t(i \mid k_p \lhd i_p).$$

From D.4.2 we have the result, corresponding to the solid line in Figure D.3, that

$$a_t^1(i) = k \wedge k \neq k_p \wedge \operatorname{dist}(x(c_t(k)), x(i_p)) \geq 2D_t^1(k) \implies \delta_t(i \mid k_p \lhd i_p) = 0. \quad \text{(D.12)}$$

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 $k \neq k_p$,

$$\Delta_t^-(k_p \lhd i_p) = \frac{1}{N - N_t(k_p)} \sum_{\substack{k:k\neq k_p \wedge \\ \operatorname{dist}(x(i_p),x(c_t(k)))<2D_t^1(k)}} \sum_{i:a_t^1(i)=k} \delta_t(i \mid k_p \lhd i_p).$$

Implication D.10, corresponding to the solid line in Figure D.3, left, can be used in the case $k = k_p$ to rapidly obtain the second term in (D.11) if $\operatorname{dist}(x(c_t(k_p)), x(i_p)) \geq D_t^1(k) + D_t^2(k)$.

The level 1 techniques for obtaining whole cluster sums require the distances from $x(i_p)$ to all cluster centers, although in Appendix D.4.2 (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. 23.

Algorithm 23 `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 (D.10) is $O(1/K)$.

```
// Set distances from proposed center $x(i_p)$ to all current centers $\mathcal{C}_t$    ▷ $O(dK)$
for $k \in \{1, \ldots, K\}$ do
    $d_c(k) \leftarrow \operatorname{dist}(x(i_p), x(c_t(k)))$
end for
$d_{pp} \leftarrow d_c(k_p)$
// Process cluster $k_p$    ▷ $O(dN/K^2)$
CLARANS-12-EVAL-P()
// Process all other clusters    ▷ $O(dN/K)$
CLARANS-1-EVAL-N-P()
```

Level 1 cluster update accelerations

If a proposal is accepted, the standard `CLARANS` uses Alg. (22) to obtain $a^{12}d^{12}_{t+1}(i)$, 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 require K distance calculations. Here at level 1, we show how many samples requiring 1 distance calculation can be set without any distance calculations, and even better: how entire clusters can sometimes be processed in constant time.

Algorithm 24 `CLARANS-12-EVAL-P` : adding the contribution of cluster k_p to $\Delta_t(k_p \lhd i_p)$. The key inequality here is (D.10), 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.

```
// Try to use (D.10) to quickly process cluster $k_p$
if $d_{pp} \geq D_t^1(k_p) + D_t^2(k_p)$ then
    $\Delta_t(k_p \lhd i_p) = \Delta_t(k_p \lhd i_p) + p_t(k_p) M_t^*(k_p)$
else
    // Test (D.10) failed, enter element-wise loop for cluster $k_p$
    for $i \in \{i' : a_t^1(t') = k_p\}$ do
        // Try tighter element-wise version of (D.10) to prevent computing a distance
        if $d_{pp} \geq d_t^1(i) + d_t^2(i)$ then
            $\Delta_t(k_p \lhd i_p) = \Delta_t(k_p \lhd i_p) + m_t(i)/N$
        else
            // Test failed, need to compute distance
            $d \leftarrow \mathrm{dist}(x(i_p), x(i))$
            $\Delta_t(k_p \lhd i_p) = \Delta_t(k_p \lhd i_p) + \min(d, m_t(i))/N$
        end if
    end for
end if
```

Algorithm 25 `CLARANS-1-EVAL-N-P` : adding contributions of all clusters $k \neq k_p$ to $\Delta_t(k_p \lhd i_p)$. The key inequality used is (D.12), 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.

```
for $k \in \{1, \dots, K\} \setminus \{k_p\}$ do
    // Try to use (D.12) to quickly process cluster $k$
    if $d_c(k) < 2D_t^1(k)$ then
        // Test (D.12) failed, enter element-wise loop for cluster $k$
        for $i \in \{i' : a_t^1(t') = k\}$ do
            // Try tighter element-wise version of (D.12) to prevent computing a distance
            if $d_c(k) < 2d_t^1(i)$ then
                // Test failed, need to compute distance
                $d \leftarrow \mathrm{dist}(x(i_p), x(i))$
                if $d < d_t^1(i)$ then
                    $\Delta_t(k_p \lhd i_p) = \Delta_t(k_p \lhd i_p) + (d - d_t^1(i))/N$
                end if
            end if
        end for
    end if
end for
```

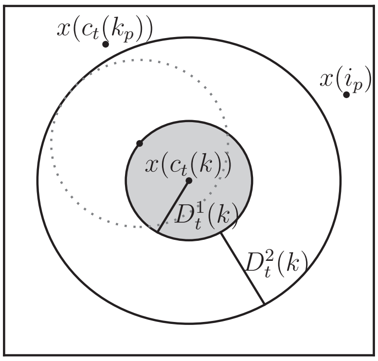

Figure D.4 – Illustrating test D.13. Consider an element $x(i)$ with $a_t^1(i) = k$, so that $x(i)$ lies in the inner gray circle, and $k \neq k_p$. Firstly, $\mathrm{dist}(x(c_t(k_p)), x(c_t(k))) > D_t^1(k) + D_t^2(k)$ means that $a_t^2(i) \neq k_p$, thus both $d_t^1(i)$ and $d_t^2(i)$ will be valid distances at iteration $t+1$. Then, as $\mathrm{dist}(x(i_p), x(c_t(k))) > D_t^1(k) + D_t^2(k)$, we have $\mathrm{dist}(x(i_p), x(i)) > D_t^2(k) \geq d_t^2(i)$, so $a_{t+1}^2(i) = a_t^2(i)$.

The inequality to eliminate an entire cluster is,

$$\min(\mathrm{dist}(x(c_t(k_p)), x(c_t(k))), \mathrm{dist}(x(i_p), x(c_t(k)))) > D_t^1(k) + D_t^2(k) \implies \text{no change in cluster } k. \tag{D.13}$$

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

$$\min(\mathrm{dist}(x(c_t(k_p)), x(c_t(k))), \mathrm{dist}(x(i_p), x(c_t(k)))) > d_t^1(i) + d_t^2(i) \implies \text{no change for sample } i. \tag{D.14}$$

Note that the inequalities need to be strict, '$\geq$' would not work. The test (D.13) is illustrated in Figure D.4, left. These bound tests are used in Alg. (26). 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 (D.13) potentially require updating.

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 $cc_t(k, k') = \mathrm{dist}(x(c_t(k)), x(c_t(k')))$ the distance between centers of clusters k and k'. At level 1, all distances $\mathrm{dist}(x(i_p), x(c_t(k)))$ for $k \in \{1, \dots, K\}$ are computed up-front for proposal evaluation, but here at level 2 we use,

$$\mathrm{dist}(x(i_p), x(c_t(k))) \geq cc_t(a_t^1(i_p), k) - d_t^1(i_p), \tag{D.15}$$

Algorithm 26 `CLARANS-1-UPDATE` : cluster update using level 1 accelerations. Inequalities (D.13) and (D.14) are used to accelerate the updating of $a^{12}d^{12}_{t+1}(i)$ for $i : a^1(i) \neq k_p \wedge a^2(i) \neq k_p$. Essentially these inequalities say that if neither the old center of cluster 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.

```
// Set distance from centers to the nearer of new and old cluster center $k_p$  ▷ $O(dK)$
for $k \in \{1, \ldots, K\}$ do
    $d_c(k) \leftarrow \min(\mathrm{dist}(x(c_t(k_p)), x(c_t(k))), \mathrm{dist}(x(i_p), x(c_t(k))))$
        $(= \min(\mathrm{dist}(x(c_t(k_p)), x(c_t(k))), \mathrm{dist}(x(c_{t+1}(k_p)), x(c_t(k)))))$
end for
// Process elements in cluster $k_p$ from scratch
for $i \in \{i' : a^1_t(t') = k_p\}$ do
    Obtain $a^{12}d^{12}_{t+1}(i)$ from scratch
end for
// Process all other clusters
for $k \in \{1, \ldots, K\} \setminus \{k_p\}$ do
    // Try to use (D.13) to quickly process cluster $k$
    if $d_c(k) \leq D^1_t(k) + D^2_t(k)$ then
        for $i \in \{i' : a^1_t(t') = k\}$ do
            // Try to use (D.14) to quickly process element $i$
            if $d_c(k) \leq d^1_t(k) + d^2_t(k)$ then
                if $a^2_t(i) = k_p$ then
                    Obtain $a^{12}d^{12}_{t+1}(i)$ from scratch
                else
                    $d \leftarrow \mathrm{dist}(x(i), x(i_p))$
                    Use $\{d^1_{t+1}(i), d^2_{t+1}(i)\} \subset \{d^1_t(i), d^2_t(i), d\}$ as in (22)
                end if
            else
                $a^{12}d^{12}_{t+1}(i) \leftarrow a^{12}d^{12}_t(i)$
            end if
        end for
    end if
end for
Update cluster statistics for $t + 1$ where necessary
```

to eliminate the need for certain of these distances. Combining (D.15) with (D.12) gives,

$$a_t^1(i) = k \wedge k \neq k_p \wedge cc_t(a_t^1(i_p), k) - d_t^1(i_p) \geq 2D_t^1(k) \implies \delta_t(i \mid k_p \lhd i_p) = 0. \quad \text{(D.16)}$$

Algorithm 27 `CLARANS-2-EVAL-N-P` : add contribution of all clusters $k \neq k_p$ to $\Delta_t(k_p \lhd i_p)$. In addition to the bound tests used at level 1, inequality (D.16) is used to test if a center-center distance needs to be calculated.

```
for $k \in \{1, \dots, K\} \setminus \{k_p\}$ do
    // Try to use (D.16) to quickly process cluster $k$
    if $cc_t(a_t^1(i_p), k) - 2D_t^1(k) < d_t^1(i_p)$ then
        // Test (D.16) failed, computing $d_c(k)$ and resorting to level 1 accelerations...
        $d_{pk} \leftarrow \text{dist}(x(i_p), x(c_t(k)))$
        if $d_{pk} < 2D_t^1(k)$ then
            // Test (D.10) failed, enter element-wise loop for cluster $k$
            for $i \in \{i' : a_t^1(t') = k\}$ do
                // Try tighter element-wise version of (D.12) to prevent computing a distance
                if $d_{pk} < 2d_t^1(i)$ then
                    // Test failed, need to compute distance
                    $d \leftarrow \text{dist}(x(i_p), x(i))$
                    if $d < d_t^1(i)$ then
                        $\Delta_t(k_p \lhd i_p) \leftarrow \Delta_t(k_p \lhd i_p) + (d - d_t^1(i))/N$
                    end if
                end if
            end for
        end if
    end if
end for
```

Algorithm 28 `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.

```
$d_{pp} \leftarrow \text{dist}(x(i_p), x(c_t(k_p)))$
// Process cluster $k_p$                                   ▷ $O(dN/K^2)$
CLARANS-12-EVAL-P()
// Process all other clusters                              ▷ $O(dN/K)$
CLARANS-2-EVAL-N-P()
```

Level 2 cluster update accelerations

The only acceleration added at level 2 for the cluster update is for the case $k_p \in \{a_t^1(i), a_t^2(i)\}$, where at level 1, $a^{12}d_{t+1}^{12}$ is set from scratch, requiring all K distances to centers to be computed. At level 2, we use cc_t to eliminate certain of these distances using `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.

Algorithm 29 `CLARANS-2-UPDATE` : update using level 2 accelerations. The only addition to level 1 accelerations is the use of `WARMSTART` to avoid computing all k sample-center distances for elements whose nearest or second nearest is k_p.

```
For $k \in \{1, \dots, K\} \backslash \{k_p\}$: compute dist$(x(i_p), x(c_t(k)))$ $(= \text{dist}(x(c_{t+1}(k_p)), x(c_{t+1}(k)))$ 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).
for $k \in \{1, \dots, K\}$ do
    $d_c(k) \leftarrow \min(cc_t(k_p, k), cc_{t+1}(k_p, k))$
end for
// Process elements in cluster $k_p$ from scratch
for $i \in \{i' : a_t^1(t') = k_p\}$ do
    $d \leftarrow \text{dist}(x(i), c_{t+1}(k_p))$
    Obtain $a^{12}d_{t+1}^{12}(i)$, using WARMSTART with $d$ and $d_t^2(i)$.
end for
// Process all other clusters
for $k \in \{1, \dots, K\} \setminus \{k_p\}$ do
    // Try to use (D.13) to quickly process cluster $k$
    if $d_c(k) \leq D_t^1(k) + D_t^2(k)$ then
        for $i \in \{i' : a_t^1(t') = k\}$ do
            // Try to use (D.14) to quickly process element $i$
            if $d_c(k) \leq d_t^1(k) + d_t^2(k)$ then
                $d \leftarrow \text{dist}(x(i), c_{t+1}(k_p))$
                if $a_t^2(i) = k_p$ then
                    Obtain $a^{12}d_{t+1}^{12}(i)$, using WARMSTART with $d$ and $d_t^1(i)$.
                else
                    Use $\{d_{t+1}^1(i), d_{t+1}^2(i)\} \subset \{d_t^1(i), d_t^2(i), d\}$ as in (22)
                end if
            else
                $a^{12}d_{t+1}^{12}(i) \leftarrow a^{12}d_t^{12}(i)$
            end if
        end for
    end if
end for
Update cluster statistics for $t + 1$ where necessary
```

Level 3

At levels 1 and 2, we showed how `clarans` can be accelerated using the triangle inequality. The accelerations were exact, in the sense that for a given initialization, the clustering obtained using `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×) 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 $\tilde{K}$ clusters which are not eliminated at level 2, we choose the number of elements chosen in the smallest sub sample to be $30\tilde{K}$. Thereafter the number of elements used to estimate the post-swap energy doubles.

Let the number of elements in the $\tilde{K}$ non-eliminated clusters by n_A and the number sampled be n_S, so that $n_S = 30\tilde{K}$. 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.

Links to datasets

The rcv1 dataset : 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

English word list : https://github.com/dwyl/english-words.git

Algorithm 30 Level 3: Schemata of using sub sampling to quickly eliminate unpromising proposals without computing an exact energy. This allows for more rapid proposal evaluation.

```
Determine which clusters are not eliminated at level 2, define to be $U$.
$\tilde{K} \leftarrow |U|$.
$N_T \leftarrow \sum_{k \in U} N_t(k)$
$N_S \leftarrow 30\tilde{K}$
$S \leftarrow$ uniform sample of indices of size $N_S$ from clusters $U$
$\hat{\Delta}_t(k_p \lhd i_p) \leftarrow \infty$
while $N_S < N_T$ and $\hat{\Delta}_t(k_p \lhd i_p) < 0$ do
    $\hat{\Delta}_t(k_p \lhd i_p) \leftarrow \frac{1}{N_S} \sum_{i \in S} \delta_t(i \mid k_p \lhd i_p)$
    $N_S \leftarrow \min(N_T, 2N_S)$
    $S \leftarrow S \cup$ uniform sample of indices so that $|S| = N_S$.
end while
if $N_S < N_T$ then return reject
else
    Compute $\Delta_t(k_p \lhd i_p)$
    if $\Delta_t(k_p \lhd i_p) < 0$ then return accept
    else return reject
    end if
end if
```

Local minima formalism

Theorem D.6.1. *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`. □

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

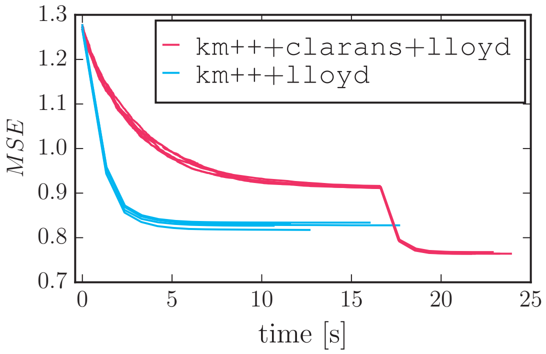

Figure D.5 – Comparing `km++`+`clarans`+`lloyd` and `km++`+`lloyd`, over ten runs, on the complete rna dataset at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#cod-rna with dimensions $N = 488,565$, $d = 8$, and $K = 2,000$. We ignore here the time to run `km++`, so that at $t = 0$ `km++` has finished. Running `clarans` before `lloyd` results in mean final MSE of 0.76, a significant improvement over 0.83 obtained without `clarans`. With `clarans`+`lloyd`, that is without pre-initializing with `km++`, the mean MSE is also 0.76, although `clarans` runs for 28 seconds, as opposed to 18 seconds with `km++`+`clarans`+`lloyd`.

while running the dynamic Needleman-Wunsch algorithm. We use this idea at all levels of acceleration.

A comment on similarities 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 $sim(a, b) = sim(a, c) \gg sim(b, c)$. 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.

Pre-initializing with `km++`

In Figure D.5, we compare `km++`+`clarans`+`lloyd` and `km++`+`lloyd`.

Comapring the different optimizations levels, and kmlocal

We briefly present results of the optimizations at each of the levels, as well as compare to the `clarans` implementation accompanying Kanungo et al. [2002a], an algorithm which they call ‘Swap’. The source code of Kanungo et al. [2002a] can be found at https://www.cs.umd.edu/~mount/Projects/KMeans/ and is called ‘kmlocal’, and our code is currently at https://github.com/anonymous1331/km4kminit. To the best of our knowledge, we compiled kmlocal correctly, and used the default -O3 flag in the Makefile.

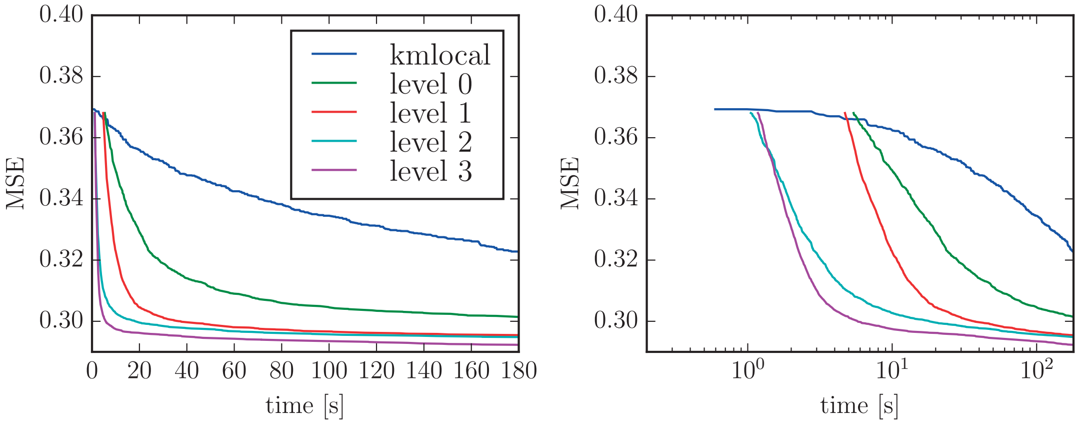

Figure D.6 – Comparing the different optimization levels and the implementation of `clarans` of Kanungo et al. [2002a], kmlocal at https://www.cs.umd.edu/~mount/Projects/KMeans/. Left and right are the same but for a logarithmic scale for the time-axis on the right. The data being clustered here is $N = 500,000$ elements in $d = 4$, drawn from a Gaussian distribution with identity covariance, and $K = 500$. We see that the various levels of optimization provide significant accelerations, and that the implementation in kmlocal is 2 orders of magnitude slower than our level 3 optimized implementation.

The only modification we made to it was to output the elapsed time after each iteration, which has negligible effect on performance.

The data consists in this experiment is $N = 500,000$ data points in $d = 4$, drawn i.i.d from a Gaussian with identity covariance, and $K = 500$. With all optimizations (level 3) convergence is obtained within 20 seconds. We notice that each optimization provides a significant boost to convergence speed. The faster initialization at levels 2 and 3 is due to the fact that using inter-center distances allows nearests and second nearests to be determined with fewer distances and comparisons.

Finally we note that the implementation of Kanungo et al. [2002a], kmlocal, is about 100× slower than our level 3 implememtation on this data. We have not run any other experiments comparing performance.

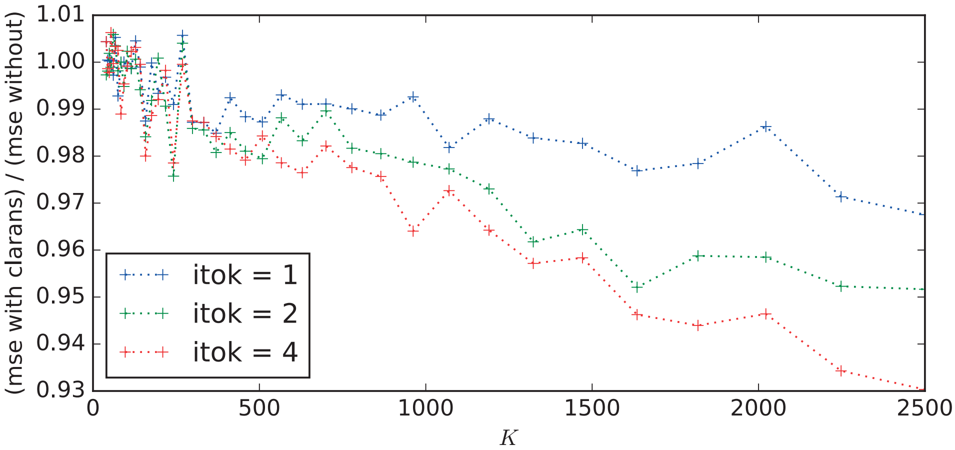

Figure D.7 – Improvement obtained using `clarans` for different values of K (horizontal axis). The experimental setup is as follows. $N = 20,000$ points are drawn from a 3-D Gaussian with identity covariance. Then for each of 40 values of K on the horizontal axis, (1) `km++` is run for fixed seed, and the time it takes to run is recorded (call it T_{++}). `clarans` is then run for a mulitple 'itok' of T_{++}, where 'itok' is one of $\{0, 1, 2, 4\}$. 'itok' of 0 corresponds to no `clarans`. After `clarans` has completed, `lloyd` is run. For 'itok' in $1, 2, 4$ the ratio of the final MSE with 'itok' 0 (no `clarans`) is plotted. This value is the fraction of the MSE without running `clarans`. We see that the dependence of the improvement on K is significant, with larger K values benefitting more from `clarans`. Also, as expected, larger 'itok' results in lower MSE.

Bibliography

Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. In *COMBINATORIAL AND COMPUTATIONAL GEOMETRY, MSRI*, pages 1–30. University Press, 2005.

David Arthur and Sergei Vassilvitskii. 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, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5. URL http://dl.acm.org/citation.cfm?id=1283383.1283494.

Olivier Bachem, Mario Lucic, S. Hamed Hassani, and Andreas Krause. Fast and provably good seedings for k-means. In *Neural Information Processing Systems (NIPS)*, December 2016.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. ISSN 0001-0782. doi: 10.1145/361002.361007. URL http://doi.acm.org/10.1145/361002.361007.

Léon Bottou and Yoshua Bengio. Convergence properties of the K-means algorithm. pages 585–592, 1995. URL http://www.iro.umontreal.ca/~lisa/pointeurs/kmeans-nips7.pdf.

Paul S. Bradley and Usama M. Fayyad. 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, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-556-8. URL http://dl.acm.org/citation.cfm?id=645527.657466.

M. Emre Celebi, Hassan A. Kingravi, and Patricio A. Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Syst. Appl.*, 40(1):200–210, 2013. doi: 10.1016/j.eswa.2012.07.021. URL http://dx.doi.org/10.1016/j.eswa.2012.07.021.

H. Chipman, T.J. Hastie, and R. Tibshirani. *Statistical Analysis of Gene Expression Microarray Data*. Chapman & Hall, 2003. URL http://folk.uib.no/nmaja/public/speed/c3278-CH04.pdf. Chapter 4.

A. Coates, H. Lee, and A.Y. Ng. An analysis of single-layer networks in unsupervised feature learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and*

Statistics, volume 15 of *JMLR Workshop and Conference Proceedings*, pages 215–223. JMLR W&CP, 2011. URL http://jmlr.csail.mit.edu/proceedings/papers/v15/coates11a.html.

Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F. Werneck. Computing classic closeness centrality, at scale. In *Proceedings of the Second ACM Conference on Online Social Networks*, COSN '14, pages 37–50, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3198-2. doi: 10.1145/2660460.2660465. URL http://doi.acm.org/10.1145/2660460.2660465.

Michael B. Cohen, Yin Tat Lee, Gary L. Miller, Jakub W. Pachocki, and Aaron Sidford. Geometric median in nearly linear time. In *STOC16*, 2016. submitted.

Ryan R. Curtin, James R. Cline, Neil P. Slagle, William B. March, P. Ram, Nishant A. Mehta, and Alexander G. Gray. MLPACK: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14:801–805, 2013.

Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 579–587, 2015. URL http://jmlr.org/proceedings/papers/v37/ding15.html.

Jonathan Drake. Faster k-means clustering, 2013. Accessed online 19 August 2015.

Jonathan Drake and Greg Hamerly. Accelerated k-means with adaptive distance bounds. In *5th NIPS Workshop on Optimization for Machine Learning, 2012*, pages 42–53, 2012. URL http://opt-ml.org/oldopt/papers/opt2012_paper_13.pdf.

Charles Elkan. Using the triangle inequality to accelerate k-means. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 147–153, 2003. URL http://www.aaai.org/Library/ICML/2003/icml03-022.php.

David Eppstein and Joseph Wang. Fast approximation of centrality. *J. Graph Algorithms Appl.*, 8(1):39–45, 2004.

Martin Ester, Hans peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

Gereon Frahling and Christian Sohler. A fast k-means implementation using coresets. In *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*, SCG '06, pages 135–143, New York, NY, USA, 2006. ACM. doi: 10.1145/1137856.1137879. URL http://doi.acm.org/10.1145/1137856.1137879.

Jan-Michael Frahm, Pierre Fite-Georgel, David Gallup, Tim Johnson, Rahul Raguram, Changchang Wu, Yi-Hung Jen, Enrique Dunn, Brian Clipp, Svetlana Lazebnik, and Marc Pollefeys. Building rome on a cloudless day. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 368–381, Berlin, Heidelberg,

2010. Springer-Verlag. ISBN 3-642-15560-X, 978-3-642-15560-4. URL http://dl.acm.org/citation.cfm?id=1888089.1888117.

Greg Hamerly. Making k-means even faster. In *SDM*, pages 130–140, 2010. doi: http://www.siam.org/proceedings/datamining/2010/dm10_012_hamerlyg.pdf.

Greg Hamerly. baylorml. https://github.com/BaylorCS/baylorml.git, 2015.

John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 99th edition, 1975. ISBN 047135645X.

Trevor J. Hastie, Robert John Tibshirani, and Jerome H. Friedman. *The elements of statistical learning : data mining, inference, and prediction*. Springer series in statistics. Springer, New York, 2001. ISBN 978-0-387-84857-0. URL http://opac.inria.fr/record=b1127878.

C. A. R. Hoare. Algorithm 65: Find. *Commun. ACM*, 4(7):321–322, July 1961. ISSN 0001-0782. doi: 10.1145/366622.366647. URL http://doi.acm.org/10.1145/366622.366647.

Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, SCG '02, pages 10–18, New York, NY, USA, 2002a. ACM. ISBN 1-58113-504-1. doi: 10.1145/513400.513402. URL http://doi.acm.org/10.1145/513400.513402.

Tapas Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):881–892, Jul 2002b. ISSN 0162-8828. doi: 10.1109/TPAMI.2002.1017616.

Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data : an introduction to cluster analysis*. Wiley series in probability and mathematical statistics. Wiley, New York, 1990. ISBN 0-471-87876-6. URL http://opac.inria.fr/record=b1087461. A Wiley-Interscience publication.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *JOURNAL OF MACHINE LEARNING RESEARCH*, 5:361–397, 2004.

Gaëlle Loosli, Stéphane Canu, and Léon Bottou. Training invariant support vector machines using selective sampling. In Léon Bottou, Olivier Chapelle, Dennis DeCoste, and Jason Weston, editors, *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA., 2007. URL http://leon.bottou.org/papers/loosli-canu-bottou-2006.

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, July 2010.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 689–696, New York, NY, USA, 2009. ACM.

ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553463. URL http://doi.acm.org/10.1145/1553374.1553463.

J. Newling and F. Fleuret. Fast k-means with accurate bounds. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 936–944, 2016a. URL http://fleuret.org/papers/newling-fleuret-icml2016.pdf.

J. Newling and F. Fleuret. Fast mini-batch k-means by nesting. In *Proceedings of the international conference on Neural Information Processing Systems (NIPS)*, pages 1352–1360, 2016b. URL http://fleuret.org/papers/newling-fleuret-nips2016.pdf.

J. Newling and F. Fleuret. A sub-quadratic exact medoid algorithm. In *Proceedings of the international conference on Artificial Intelligence and Statistics (AISTATS)*, pages 185–193, 2017a. URL http://fleuret.org/papers/newling-fleuret-aistats2017.pdf. (Best paper award).

J. Newling and F. Fleuret. K-medoids for k-means seeding. In *Proceedings of the international conference on Neural Information Processing Systems (NIPS)*, 2017b. (To appear).

Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 144–155, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN 1-55860-153-8. URL http://dl.acm.org/citation.cfm?id=645920.672827.

David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2597-0. doi: 10.1109/CVPR.2006.264. URL http://dx.doi.org/10.1109/CVPR.2006.264.

Kazuya Okamoto, Wei Chen, and Xiang-Yang Li. Ranking of closeness centrality for large-scale social networks. In *Proceedings of the 2Nd Annual International Workshop on Frontiers in Algorithmics*, FAW '08, pages 186–195, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-69310-9. doi: 10.1007/978-3-540-69311-6_21. URL http://dx.doi.org/10.1007/978-3-540-69311-6_21.

M.T. Orchard. A fast nearest-neighbor search algorithm. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pages 2297–2300 vol.4, Apr 1991. doi: 10.1109/ICASSP.1991.150755.

Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert Syst. Appl.*, 36(2):3336–3341, March 2009. ISSN 0957-4174. doi: 10.1016/j.eswa.2008.01.039. URL http://dx.doi.org/10.1016/j.eswa.2008.01.039.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 277–281, New York, NY, USA, 1999. ACM. ISBN 1-58113-143-7. doi: 10.1145/312129.312248. URL http://doi.acm.org/10.1145/312129.312248.

J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

S.J. Phillips. Acceleration of k-means and related clustering algorithms. volume 2409 of *Lecture Notes in Computer Science*. Springer, 2002.

Matthew J. Rattigan, Marc Maier, and David Jensen. Graph clustering with network structure indices. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 783–790, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273595. URL http://doi.acm.org/10.1145/1273496.1273595.

D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, 2010. ISBN 978-1-60558-799-8.

Matus Telgarsky and Andrea Vattani. Hartigan's method: k-means clustering without voronoi. In *AISTATS*, volume 9 of *JMLR Proceedings*, pages 820–827. JMLR.org, 2010.

A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

Jing Wang, Jingdong Wang, Qifa Ke, Gang Zeng, and Shipeng Li. Fast approximate k-means via cluster closures. In *CVPR*, pages 3037–3044. IEEE Computer Society, 2012.

Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey McLachlan, Angus Ng, Bing Liu, Philip Yu, Zhi-Hua Zhou, Michael Steinbach, David Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, January 2008. ISSN 0219-1377. URL http://dx.doi.org/10.1007/s10115-007-0114-2.

Zhang Xianyi. OpenBLAS. http://www.openblas.net, 2016.

Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1091–1095, June 2007. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.1078. URL http://dx.doi.org/10.1109/TPAMI.2007.1078.

James Newling

Address: Idiap Research Institute
Martigny, Valais
Switzerland, 1920
Citizenship : United Kingdom and
South Africa

Email: james.newling@idiap.ch
james.newling@gmail.com
Homepage: **https://www.idiap.ch/∼jnewling**
Github: **https://github.com/newling**

Research Interests

Statistical learning, deep learning, numerical algorithms, high performance computing

Education

Since September 2013, PhD candidate at the École Polytechnique Fédérale de Lausanne

June 2013, **MSc in Complexity Science** at École Polytechnique (Paris) and Warwick University

June 2011, **Masters in Applied Mathematics** at The University of Cape Town

December 2009, **Honours Degree in Mathematics and Statistics** at The University of Cape Town

Employment

Since September 2013, Research Assistant at the Idiap Research Institute

September 2016 - December 2016, Intern at Advanced Micro Devices (Austin, TX)

April 2013 - September 2013, Research Assistant in the Mukherjee Lab for Statistical Systems Biology, Netherlands Cancer Institute

February 2010 - June 2010, Maths Lecturer in Non-linear Optimization at the University of Cape Town

Software

MIOpenGEMM. OpenCL GEMM (matrix multiplication) kernels, auto-tuning, and API. I started this project while on internship at AMD in October 2016. MIOpenGEMM is currently used by AMD's machine learning library, MIOpen.

zentas and **eakmeans**. Partitional clustering software projects related to my PhD work.

Machine Learning Conference Proceedings

J. Newling and F. Fleuret. **K-Medoids For K-Means Seeding**. In Proceedings of the International Conference on Neural Information Processing Systems (NIPS), 2017. (To appear)

J. Newling and F. Fleuret. **A Sub-Quadratic Exact Medoid Algorithm**. In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), pages 185-193, 2017. (Best paper award)

J. Newling and F. Fleuret. **Nested Mini-Batch K-Means**. In Proceedings of the International Conference on Neural Information Processing Systems (NIPS), pages 1352-1360, 2016.

J. Newling and F. Fleuret. **Fast K-Means with Accurate Bounds**. In Proceedings of the International Conference on Machine Learning (ICML), pages 936-944, 2016

Computer Programming

Very familiar with C++11, Python, numerical algorithms. Familiar with OpenCL and deep learning software stacks.

Selected University Courses

École Polytechnique Fédérale de Lausanne : Advanced Algorithms, Topics in Theoretical Computer Science, Mathematics of Data, Statistical Physics for Computer Science, Topics on Datacenter Design

Warwick University : Algorithms, Mathematical Biology, Theoretical Neuroscience, Scientific Computing, Fundamentals of Modern Statistical Inference

École Polytechnique : Complex Systems, Dynamical Systems, Numerical ODEs and SDEs, Data Minimg, Statistical Learning, Signal Processing, Random Models in Evolution

University of Cape Town : Applied Mathematics (I, II, IV), Computer Science (Ia), Economics (I), Mathematics (I, II, III), Physics (I, II), Statistics (I, II, III)