AFFINITY PROPAGATION:
CLUSTERING DATA BY PASSING MESSAGES

by

Delbert Dueck

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Electrical & Computer Engineering
University of Toronto

Abstract

AFFINITY PROPAGATION: CLUSTERING DATA BY PASSING MESSAGES

Delbert Dueck

Doctor of Philosophy

Graduate Department of Electrical & Computer Engineering

University of Toronto

2009

Clustering data by identifying a subset of representative examples is important for detecting patterns in data and in processing sensory signals. Such "exemplars" can be found by randomly choosing an initial subset of data points as exemplars and then iteratively refining it, but this works well only if that initial choice is close to a good solution. This thesis describes a method called "affinity propagation" that simultaneously considers all data points as potential exemplars, exchanging real-valued messages between data points until a high-quality set of exemplars and corresponding clusters gradually emerges.

Affinity propagation takes as input a set of pairwise similarities between data points and finds clusters on the basis of maximizing the total similarity between data points and their exemplars. Similarity can be simply defined as negative squared Euclidean distance for compatibility with other algorithms, or it can incorporate richer domain-specific models (*e.g.*, translation-invariant distances for comparing images). Affinity propagation's computational and memory requirements scale linearly with the number of similarities input; for non-sparse problems where all possible similarities are computed, these requirements scale quadratically with the number of data points. Affinity propagation is demonstrated on several applications from areas such as computer vision and bioinformatics, and it typically finds better clustering solutions than other methods in less time.

Acknowledgements

I would first like to acknowledge and thank my wife, Candice, for her support and encouragement through my educational journey. I'd also like to thank my family for their backing, and more practically, my father for spending many hours carefully proofreading this thesis.

I have had many valuable experiences at the University of Toronto during my graduate studies, so I would like to acknowledge some of the people who helped me along. I would like to thank my Ph.D. supervisor, Brendan Frey, for guiding me through graduate studies and helpful discussions that spawned much of my research — most notably, affinity propagation. I would also like to thank the members of my Doctoral Supervisory Committee from both the Electrical/Computer Engineering and Computer Science departments, namely Frank Kschischang, Sam Roweis, Geoff Hinton, and Wei Yu. I've appreciated the insights and ideas they shared with me which have no doubt led to a better finished product. I would also like to thank John Winn of Microsoft Research (Cambridge, UK) who agreed to be my external examiner in spite of facing many other commitments at the time.

I would like to thank my fellow members of Brendan Frey's Probabilistic and Statistical Inference group for being a valuable sounding board for ideas, specifically Ofer Shai for reading and marking up early drafts of this thesis. Finally, I would like to thank the funding agencies that made this research possible: the Natural Sciences and Engineering Council of Canada and the Government of Ontario for their Ontario Graduate Scholarships in Science and Technology.

Contents

List of Figures

Caravaggio's *Vocazione di san Matteo* (The Calling of St. Matthew, [20]) is an artistic depiction of identifying exemplars based on the direction of gestures, gazes, and even lighting in the painting. This interpretation was suggested in [77].

Chapter 1

Introduction

Clustering or discovering meaningful partitions of data based on a measure of similarity is a critical step in scientific data analysis and a fundamental problem in computer science. A common approach within the machine learning community involves unsupervised learning of parameters that describe clusters (*e.g.* the location and scale/shape of the cluster) and partitioning the data by associating every point or region with one or more clusters. In many situations, data is better and more easily characterized by a measure of pairwise similarities rather than defaulting to negative squared Euclidean distance, and in this case, clusters can instead be represented by an "exemplar" data point rather than domain-specific parameters. This thesis introduces a novel algorithm, affinity propagation, that uses belief propagation methods to achieve outstanding results for exemplar-based clustering.

Identifying exemplars is advantageous because user-specified similarities offer a large amount of flexibility and allow the clustering algorithm to be decoupled from the details of how similarities between data points are computed. Unlike many algorithms that operate in vector space, there is no need for similarity to be based on squared Euclidean distance, or for the data space to be metric or continuous, or even ordinal; see Figure 1.1 for examples. Additionally, there is potential for significant improvement on existing algorithms, both in terms of solution time and solution quality.

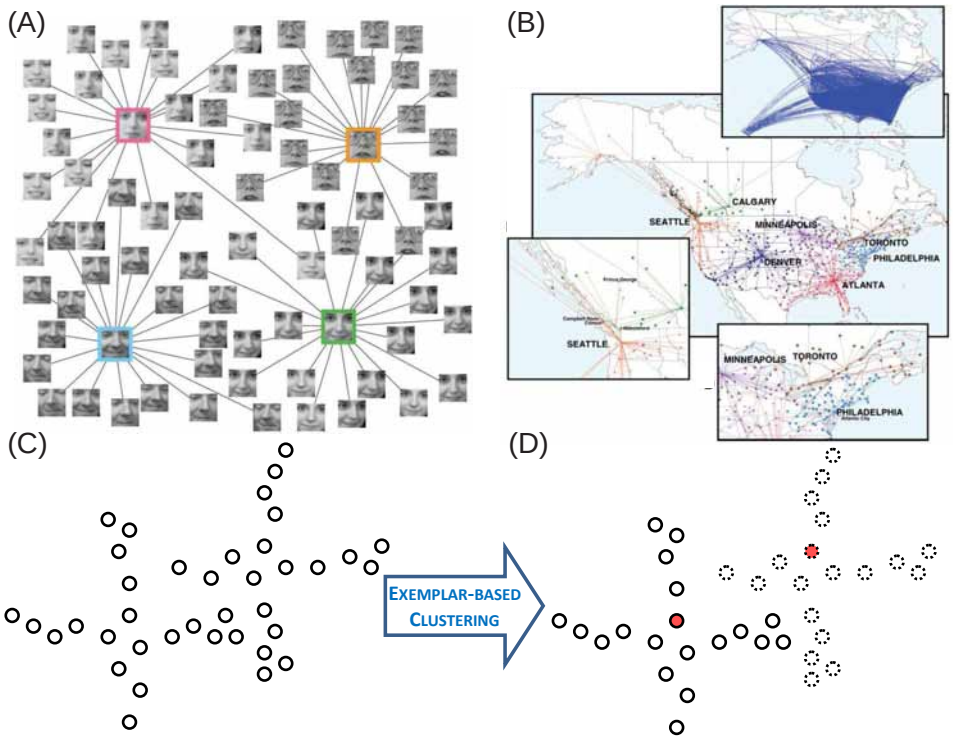


Figure 1.1: Several datasets are shown for which negative squared Euclidean distance would be an inappropriate measure of similarity. In (A), faces are clustered using translation-invariant comparisons (see Section 5.1.3 for details). In (B), North American cities are clustered with similarity defined as flight time, which depends on airline schedules, headwinds, earth curvature, *etc.* The dataset in (C) appears to contain two unconventional clusters that are shaped like two-dimensional "plus-signs". There are many realistic physical situations from which data such as this could have arisen, *e.g.* where a subset of sensors (in this case, one of two) are unreliable for each measurement. Conventional clustering algorithms would need special tuning or re-deriving to accommodate such a model; exemplar-based clustering algorithms that rely on pairwise similarities could just use a slightly different definition of similarity such as a Gaussian likelihood with two possible variances switched in for each dimension. The result of such clustering is shown in (D).

The task of exemplar-based clustering is to identify a subset of the N data points as exemplars and assign every other data point to one of those exemplars. The only inputs are a set of real-valued pairwise similarities between data points, $\{s(i,k)\}$, and the number of exemplars to find (K) or a real-valued exemplar cost to balance against similarities. A simple and fast algorithm for finding clustering solutions is the k-medoids algorithm [70], which begins by randomly selecting a set of K data points as initial exemplars and then refines these in alternating steps as shown in Figure 1.2. The algorithm monotonically maximizes the sum of similarities between data points and exemplars but considers only a fixed set of exemplars, and thus is quite sensitive to the initial selection of exemplars. For this reason, k-medoids clustering needs to be run with many different random initializations—it works well only when the number of clusters is small and chances are good that at least one restart lies close to a good clustering solution.

In contrast to k-medoids, affinity propagation simultaneously considers all data points as potential exemplars. By viewing each data point in a network, it recursively transmits real-valued messages along edges of the network until a good set of exemplars and corresponding clusters emerge; see Figure 1.3 for an illustration of these dynamics. Affinity propagation sends two types of message between data points: responsibilities are sent from data points to candidate exemplars and reflect the evidence of how well-suited the message-receiving point is to serve as an exemplar for the sending point. Availabilities are sent from candidate exemplars to data points and reflect the evidence for how appropriate it would be for the message-sending point to be the exemplar for the message-receiving point (see Figure 1.3). All data points can be considered to be either cluster members or candidate exemplars, depending on whether they are sending or receiving availability or responsibility messages.

Affinity propagation is outlined in the box below, with scalar responsibility and availability message updates shown in equation (1.1). At any time, a current estimate of cluster assignments can be obtained by adding responsibility and availability messages together.

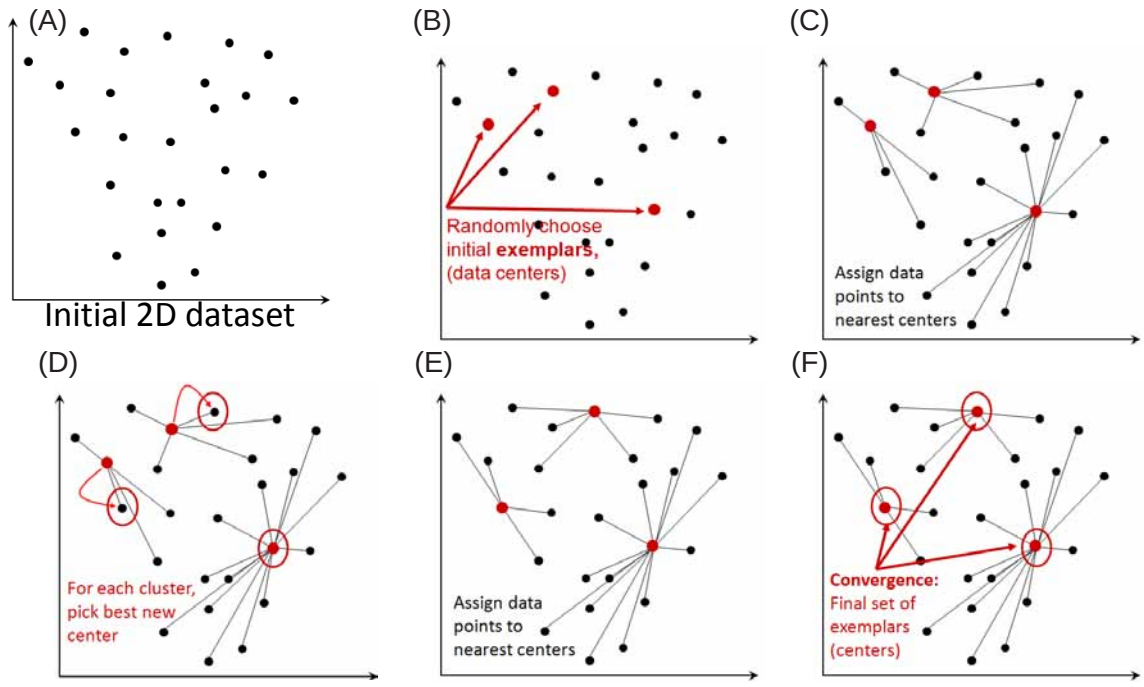


Figure 1.2: The k-medoids clustering algorithm is a simple algorithm that finds a greedy solution. Given the initial toy dataset in (A), the algorithm randomly chooses an initial set of exemplars (B), and assigns the remaining non-exemplar data points to the "closest" exemplar based on similarity (C). New exemplars are found for each cluster (D) to minimize the total sum of intra-cluster similarities, and the process is repeated (E) until convergence to the solution in (F).

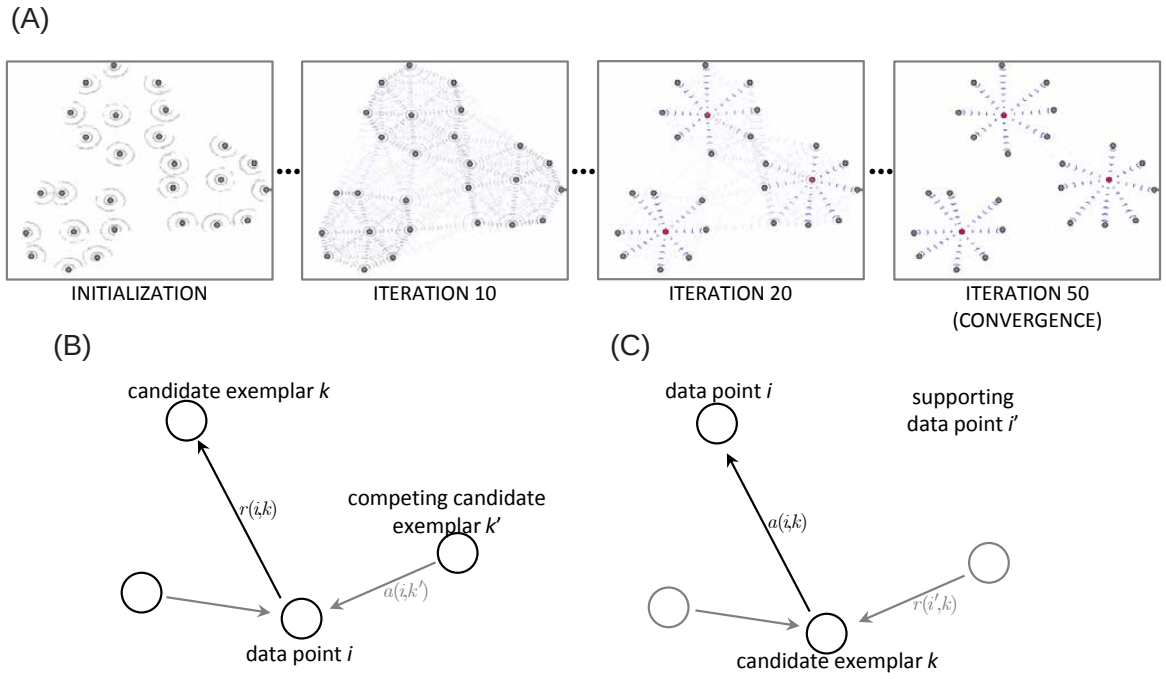


Figure 1.3: The affinity propagation clustering algorithm defines messages that are exchanged between data points indicating the 'affinity' each point has for another to act as its exemplar. The toy exemplar above (A) shows a solution gradually emerging, with uncertainty in the tenth iteration (shown as faded blue messages) being resolved to a good clustering solution shown at the final iteration. Two messages are passed between data points: (B) "responsibilities" $r(i, k)$ are sent from data point i to candidate exemplar k, and (C) "availabilities" $a(i, k)$ are sent from candidate exemplar k to data point i.

AFFINITY PROPAGATION

INPUT: a set of pairwise similarities, $\{s(i,k)\}_{(i,k)\in\{1,\ldots,N\}^2, i\neq k}$ where $s(i,k)\in\mathbb{R}$ indicates how well-suited data point k is as an exemplar for data point i.

e.g., $s(i,k) = -\left\|\mathbf{x}_i - \mathbf{x}_k\right\|^2,\ i\neq k$ (squared Euclidean distance)

For each data point k, a real number $s(k,k)$ indicating the *a priori* preference (negative cost of adding a cluster) that it be chosen as an exemplar.

e.g. $s(k,k) = p \quad \forall k \in \{1,\ldots,N\}$ (global preference)

INITIALIZATION: set availabilities to zero, $\forall i,k\colon a(i,k)=0$.

REPEAT: responsibility and availability updates until convergence

$$\forall i,k\colon\ r(i,k) = s(i,k) - \max_{k':k'\neq k}\left[s(i,k') + a(i,k')\right]$$

$$\forall i,k\colon\ a(i,k) = \begin{cases} \sum_{i':i'\neq i}\max[0,\ r(i',k)], & \text{for } k=i \\ \min\Big[0,\ r(k,k)+\sum_{i':i'\notin\{i,k\}}\max[0,\ r(i',k)]\Big], & \text{for } k\neq i \end{cases} \tag{1.1}$$

OUTPUT: assignments $\hat{\mathbf{c}} = (\hat{c}_1,\ldots,\hat{c}_N)$, where $\hat{c}_i = \operatorname{argmax}_k\left[a(i,k)+r(i,k)\right]$ and $\hat{c}_i$ indexes the cluster's exemplar to which point i is assigned. Specifically, if point i is in a cluster with point k serving as the exemplar, then $\hat{c}_i = k$ and $\hat{c}_k = k$. Note: one run of k-medoids may be needed to resolve contradictory solutions.

Affinity propagation achieves outstanding results by employing "loopy belief propagation" techniques (see Section 2.6) that have previously been used to approach Shannon's limit in error-correcting decoding [5] and solve random satisfiability problems with an order-of-magnitude increase in size [76]. The objective function it maximizes is the net similarity, $\mathcal{S}$, which is the sum of the similarities of non-exemplar data points to their exemplars plus the sum of exemplar preferences (negative costs of adding exemplars).

The affinity propagation algorithm is simple to implement and customize; it is also computationally efficient, scaling linearly in the number of similarities or quadratically in the number of data points if all possible pairwise similarities are used. Computing pairwise similarities typically takes more computation than does clustering them; the example described in Section 5.2 involves clustering 75,066 data points with roughly 15,000,000 similarities—this requires several minutes of computation on a typical notebook computer (as of 2008).

A background to parametric approaches to clustering, the facility location problem, and

belief propagation algorithms is given in Chapter 2. This leads into a derivation and discussion of the affinity propagation algorithm in Chapter 3 followed by thorough benchmarking of the methods in Chapter 4. Affinity propagation is applicable to a wide variety of applications spanning most areas of science and engineering. This thesis explores several application areas within computer vision and bioinformatics in Chapter 5. For interested readers, all software and data is available at http://www.psi.toronto.edu/affinitypropagation.

Chapter 2

Background

Clustering is the unsupervised learning task of organizing or partitioning data into meaningful groupings. For data embedded in a vector space, a common way to accomplish this is to view clusters as 'clumps' of data that are a certain [Euclidean] distance away from a center—in two dimensions, these 'clumps' would be circular or elliptical. Though not necessarily the most appropriate way to cluster (see Figure 1.1 for a counter-example), clustering such data based on squared Euclidean distance is widespread in the machine learning literature and provides an easy path to introducing affinity propagation.

2.1 k-means clustering

Given N column-vector data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ where each $\mathbf{x}_i \in \mathbb{R}^D$, the clustering task is to assign each of them to one of K classes labeled $1, 2, \ldots, K$. These assignments are denoted by latent class variables $z_1, z_2, \ldots, z_N$ where each $z_i \in \{1, 2, \ldots, K\}$. With the k-means clustering algorithm [70], each class is characterized by a cluster center, $\boldsymbol{\mu}_k$, which can be interpreted as the mean vector for a unit-covariance spherical Gaussian (*i.e.*, the covariance matrix is given by the identity matrix, $\mathbf{I}_D$). Given that data point $\mathbf{x}_i$ belongs to class k, its distribution is given

by:

$$P(\mathbf{x}_i|z_i=k,\boldsymbol{\mu}_k)=\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_k,\mathbf{I}_D)=\frac{1}{\sqrt{(2\pi)^D}}\exp\left(-\tfrac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_k)^\top(\mathbf{x}_i-\boldsymbol{\mu}_k)\right)$$

Class labels $\{z_i\}_{i=1}^N$ are hidden (latent) variables so overall the data is distributed according to a mixture of spherical Gaussians distribution,

$$P(\mathbf{x}_i)=\sum_{k=1}^{K}P(\mathbf{x}_i|z_i=k)\cdot P(z_i=k)=\sum_{k=1}^{K}\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_k,\mathbf{I}_D)\cdot\frac{1}{K}$$

as illustrated in Figure 2.1(A). Note that the *a priori* probabilities of data point class assignments are assumed to be uniform (for now), *i.e.* $\forall k\colon P(z_i=k)=\frac{1}{K}$.

An appropriate class assignment for the i^{th} data point involves maximizing the posterior probability $P(z_i=k|\mathbf{x}_i,\boldsymbol{\mu}_k)$ which by Bayes' rule is equivalent to maximizing $P(\mathbf{x}_i|z_i=k,\boldsymbol{\mu}_k)P(z_i=k)/P(\mathbf{x}_i)$ with respect to k. As shown above, $P(\mathbf{x}_i)$ does not depend on the class assignments $\{z_i\}$, so it is appropriate to write in this case:

$$\operatorname*{argmax}_{k\in\{1,2,\ldots,K\}}P(z_i=k|\mathbf{x}_i,\boldsymbol{\mu}_k)=\operatorname*{argmax}_{k\in\{1,2,\ldots,K\}}\frac{1}{K}\cdot P(\mathbf{x}_i|z_i=k,\boldsymbol{\mu}_k)$$

and thus each class label is assigned as follows:

$$z_i\leftarrow\operatorname*{argmax}_{k\in\{1,\ldots,K\}}\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_k,\mathbf{I}_D)=\operatorname*{argmin}_{k\in\{1,\ldots,K\}}\|\mathbf{x}_i-\boldsymbol{\mu}_k\| \tag{2.1}$$

This assignment, however, depends on the choices for $\{\boldsymbol{\mu}_k\}_{k=1}^K$, and for computational reasons it is typically optimized separately while holding values of $\{\boldsymbol{\mu}_k\}$ constant. The likelihood of the entire dataset given all class assignments is $P(\mathbf{x}|\mathbf{z},\boldsymbol{\mu})=\prod_{i=1}^{N}\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_{z_i},\mathbf{I}_D)$ so the Gaussian parameters are optimized by maximizing this likelihood (or rather, the log-likelihood, which is equivalent). Setting partial derivatives of $\log P(\mathbf{x}|\mathbf{z},\boldsymbol{\mu})$ with respect to each $\boldsymbol{\mu}_k$ to

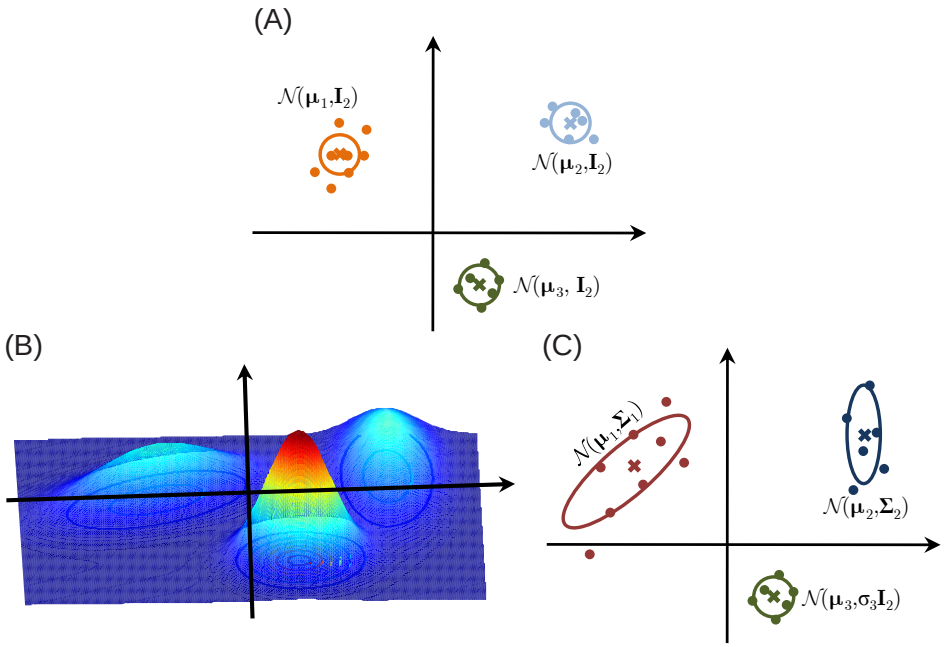


Figure 2.1: Datasets fit to k-means (A) and EM for mixture of Gaussians (C) are shown in two dimensions. A mixture of three spherical Gaussians are shown in (A) with means $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, and $\boldsymbol{\mu}_3$; these could have been fit by k-means (§2.1). A different mixture of three Gaussians distribution more suited to the EM algorithm (§2.3) is shown in (B); the contour plot is for the distribution: $P(\mathbf{x}) = \sum_{k=1}^{3} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. The plot displays the orientation of the Gaussians in (C), where the first Gaussian is shown in red and parameterized by $\{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\}$. The second Gaussian is shown in blue and parameterized by $\{\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2\}$—this covariance is diagonal as the Gaussian is axis-aligned. The third Gaussian is shown in green and parameterized by $\{\boldsymbol{\mu}_3, \sigma_3\}$, where the Gaussian is isotropic/spherical (same variance in all dimensions) and the covariance matrix is thus a scalar multiple of the identity matrix.

zero leads to the following update equation:

$$\begin{aligned} 0 &= \tfrac{\partial}{\partial \boldsymbol{\mu}_k} \textstyle\sum_{i=1}^{N} \log \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_{z_i}, \mathbf{I}_D) \propto \sum_{i=1}^{N} [z_i = k](\mathbf{x}_i - \boldsymbol{\mu}_k) \\ &\text{so } \boldsymbol{\mu}_k \leftarrow \textstyle\sum_{i=1}^{N} [z_i = k]\mathbf{x}_i \Big/ \sum_{i=1}^{N} [z_i = k] \end{aligned} \tag{2.2}$$

where $[\,\cdot\,]$ denotes Iverson notation with $[\text{true}] = 1$ and $[\text{false}] = 0$. Essentially, the update equation sets each $\boldsymbol{\mu}_k$ to be the vector mean of all data points in the k^{th} class, hence the name k-means.

K-MEANS CLUSTERING ALGORITHM
INPUT: $\{\mathbf{x}_i\}_{i=1}^N$ (data), K (number of clusters)
INITIALIZE: set each $\boldsymbol{\mu}_k$ to a random data point
REPEAT UNTIL CONVERGENCE:

$$\begin{aligned} &\forall i \colon z_i \leftarrow \operatorname*{argmin}_{k \in \{1,\ldots,K\}} \|\mathbf{x}_i - \boldsymbol{\mu}_k\| = \operatorname*{argmax}_{k \in \{1,\ldots,K\}} \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \mathbf{I}_D) \\ &\forall k \colon \boldsymbol{\mu}_k \leftarrow \text{mean}\,\{\mathbf{x}_i\}_{i:z_i=k} = \textstyle\sum_{i=1}^{N} [z_i = k]\mathbf{x}_i / \sum_{i=1}^{N} [z_i = k] \end{aligned} \tag{2.3}$$

OUTPUT: $\{z_i\}_{i=1}^N$ (cluster assignments), $\{\boldsymbol{\mu}_k\}_{k=1}^K$ (cluster centers)

2.2 k-medians clustering

A variant of k-means in occasional use is k-medians clustering, wherein the median is used instead of the mean when updating the cluster center parameters. This algorithm is summarized in equation (2.4).

K-MEDIANS CLUSTERING ALGORITHM
INPUT: $\{\mathbf{x}_i\}_{i=1}^N$ (data), K (number of clusters)
INITIALIZE: set each $\mathbf{m}_k$ to a random data point
REPEAT UNTIL CONVERGENCE:

$$\begin{aligned} z_i &\leftarrow \operatorname*{argmin}_{k \in \{1,\ldots,K\}} \|\mathbf{x}_i - \mathbf{m}_k\| \\ \mathbf{m}_k &\leftarrow \text{median}\,\{\mathbf{x}_i\}_{i:z_i=k} \end{aligned} \tag{2.4}$$

OUTPUT: $\{z_i\}_{i=1}^N$ (cluster assignments), $\{\mathbf{m}_k\}_{k=1}^K$ (cluster centers)

2.3 EM algorithm for Mixtures of Gaussians

The k-means algorithm makes all-or-nothing assignments of data points to clusters, and these hard decisions can often lead to poor solutions corresponding to local minima in $P(\mathbf{x}|\mathbf{z},\boldsymbol{\mu})$. A common refinement involves learning the covariances (instead of fixing them *i.e.* $\forall k : \boldsymbol{\Sigma}_k = \mathbf{I}_D$; see Figure 2.1(B–C)), learning mixing weights on class priors (instead of assuming $\forall k : \pi_k = \frac{1}{K}$), and to account for cluster assignment uncertainty by using the Expectation-Maximization (EM) algorithm [23] and representing it with a simple distribution $Q(\mathbf{z}) = \prod_{i=1}^{N}\prod_{k=1}^{K} q_{ik}^{[z_i=k]}$. Cluster assignments can be determined by minimizing the Kullback-Leibler divergence [18] between $Q(\mathbf{z})$ and $P(\mathbf{z}|\mathbf{x})$, $D\left(Q(\mathbf{z}) \,\|\, P(\mathbf{z}|\mathbf{x})\right) = \int_{\mathbf{z}} Q(\mathbf{z})\cdot\log\frac{Q(\mathbf{z})}{P(\mathbf{z}|\mathbf{x})}$. Finding a workable expression for the denominator, $P(\mathbf{z}|\mathbf{x}) = P(\mathbf{x}|\mathbf{z})P(\mathbf{z})/P(\mathbf{x})$, is not usually possible so the following is minimized instead:

$$\underset{\{q\}}{\operatorname{argmin}}\left(\int_{\mathbf{z}} Q(\mathbf{z})\cdot\log\frac{Q(\mathbf{z})}{P(\mathbf{z}|\mathbf{x})} - \overbrace{\log P(\mathbf{x})}^{\text{constant w.r.t. Q}(\mathbf{z})}\right) = \underset{\{q\}}{\operatorname{argmin}}\left(\int_{\mathbf{z}} Q(\mathbf{z})\cdot\log\frac{Q(\mathbf{z})}{P(\mathbf{x},\mathbf{z})}\right) \tag{2.5}$$

The joint distribution becomes $P(\mathbf{x},\mathbf{z}) = P(\mathbf{x}|\mathbf{z})P(\mathbf{z}) = \prod_{i=1}^{N}\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_{z_i},\boldsymbol{\Sigma}_{z_i})\cdot\pi_{z_i}$, and the expression to minimize—referred to as the free energy, $\mathcal{F}$—becomes:

$$\begin{aligned}\int_{\mathbf{z}} Q(\mathbf{z})\cdot\log\tfrac{Q(\mathbf{z})}{P(\mathbf{x},\mathbf{z})} &= \int_{\mathbf{z}}\prod_{i=1}^{N}\prod_{k=1}^{K} q_{ik}^{[z_i=k]}\left(\sum_{i'=1}^{N}\sum_{k'=1}^{K}[z_{i'}{=}k']\cdot\log q_{i'k'} - \sum_{i'=1}^{N}\log\pi_{z_{i'}}\mathcal{N}(\mathbf{x}_{i'};\boldsymbol{\mu}_{z_{i'}},\boldsymbol{\Sigma}_{z_{i'}})\right)\\ &= \sum_{i=1}^{N}\sum_{k=1}^{K} q_{ik}\cdot\log q_{ik} - \sum_{i=1}^{N}\sum_{k=1}^{K} q_{ik}\cdot\log\pi_k\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\end{aligned}$$

After adding the constraint $\forall i$: $\sum_{k=1}^{K} q_{ik} = 1$ to ensure $Q(\mathbf{z})$ is a valid distribution, we can optimize $Q(\mathbf{z})$ by setting its partial derivative (plus the Lagrange constraint) to zero:

$$0 = \frac{\partial\left[\mathcal{F}+\lambda_i\left(1-\sum_{k=1}^{K} q_{ik}\right)\right]}{\partial q_{ik}} = 1+\log q_{ik}-\log\pi_k\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)-\lambda_i \Rightarrow q_{ik} = \pi_k\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\cdot e^{\lambda_i-1}$$

$$\text{so } q_{ik} \leftarrow \frac{\pi_k\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\sum_{k'=1}^{K}\pi_{k'}\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_{k'},\boldsymbol{\Sigma}_{k'})}$$

Mean and covariance parameters for the Gaussians can be found similarly:

$$0 = \frac{\partial\left[\mathcal{F}+\lambda\left(1-\sum_{k=1}^{K}\pi_k\right)\right]}{\partial\pi_k} = -\frac{1}{\pi_k}\sum_{k=1}^{K} q_{ik} - \lambda \Rightarrow \pi_k = -\frac{1}{\lambda}\sum_{k=1}^{K} q_{ik}$$
$$0 = \frac{\partial}{\partial\boldsymbol{\mu}_k}\sum_{i=1}^{N}\sum_{k=1}^{K} q_{ik}\log\pi_k\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) \propto \sum_{i=1}^{N} q_{ik}(\mathbf{x}_i-\boldsymbol{\mu}_k)$$
$$0 = \frac{\partial}{\partial\boldsymbol{\Sigma}_k}\sum_{i=1}^{N}\sum_{k=1}^{K} q_{ik}\log\pi_k\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) \propto \sum_{i=1}^{N} q_{ik}\boldsymbol{\Sigma}_k^{\top} + \sum_{i=1}^{N} q_{ik}(\mathbf{x}_i-\boldsymbol{\mu}_k)(\mathbf{x}_i-\boldsymbol{\mu}_k)^{\top}$$
$$\text{so } \pi_k \leftarrow \frac{\sum_{i=1}^{N} q_{ik}}{N},\ \boldsymbol{\mu}_k \leftarrow \frac{\sum_{i=1}^{N} q_{ik}\mathbf{x}_i}{\sum_{i=1}^{N} q_{ik}},\ \text{and } \boldsymbol{\Sigma}_k^{\top} \leftarrow \frac{\sum_{i=1}^{N} q_{ik}(\mathbf{x}_i-\boldsymbol{\mu}_k)(\mathbf{x}_i-\boldsymbol{\mu}_k)^{\top}}{\sum_{i=1}^{N} q_{ik}}$$

where $\{\pi_k\}_{k=1}^{K}$ is constrained to be a valid probability distribution through Lagrange multiplier λ, which enforces $\sum_{k=1}^{K}\pi_k = 1$.

For high-dimensional input data, learning the full covariance matrix $\boldsymbol{\Sigma}_k$ involves $\frac{D(D-1)}{2}$ scalar parameters, which can be cumbersome and potentially be a cause of overfitting. For this reason the Gaussians are often assumed to have diagonal covariance matrices (in which case the off-diagonal elements of $\boldsymbol{\Sigma}_k$ are zeroed during updates) or even isotropic covariances.[1]. Class assignments can be easily read from the Q-distribution *i.e.* $\forall i\colon z_i \leftarrow \operatorname{argmax}_k q_{ik}$.

EM ALGORITHM FOR A MIXTURE OF GAUSSIANS

INPUT: $\{\mathbf{x}_i\}_{i=1}^{N}$ (data), K (number of clusters)

INITIALIZE: set $\{\boldsymbol{\mu}_k\}$ to random data points, $\forall k\colon \mu_k \leftarrow \frac{1}{K}$ and $\boldsymbol{\Sigma}_k \leftarrow \mathrm{var}(\{\mathbf{x}_i\})$

REPEAT UNTIL CONVERGENCE:

$$\forall i,k\colon q_{ik} \leftarrow \frac{\pi_k\cdot\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\sum_{k'=1}^{K}\pi_{k'}\cdot\mathcal{N}(\mathbf{x}_i;\boldsymbol{\mu}_{k'},\boldsymbol{\Sigma}_{k'})}$$
$$\forall k\colon \pi_k \leftarrow \frac{\sum_{i=1}^{N} q_{ik}}{N},\ \boldsymbol{\mu}_k \leftarrow \frac{\sum_{i=1}^{N} q_{ik}\mathbf{x}_i}{\sum_{i=1}^{N} q_{ik}},\ \boldsymbol{\Sigma}_k^{\top} \leftarrow \frac{\sum_{i=1}^{N} q_{ik}(\mathbf{x}_i-\boldsymbol{\mu}_k)(\mathbf{x}_i-\boldsymbol{\mu}_k)^{\top}}{\sum_{i=1}^{N} q_{ik}} \tag{2.6}$$

OUTPUT: $\{z_i \leftarrow \operatorname{argmax}_{k\in\{1,\ldots,K\}} q_{ik}\}_{i=1}^{N}$ (assignments), $\{\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\}_{k=1}^{K}$ (Gaussians)

Another item of interest is that setting the covariance matrices to $\boldsymbol{\Sigma}_k = \epsilon\cdot\mathbf{I}_D$, where $\epsilon \to 0$, polarizes the Q-distribution ($\max Q(\mathbf{z}) \to 1$) to reflect hard decisions and reduces the EM update equations to k-means.

[1] in which case $\boldsymbol{\Sigma}_k \leftarrow \mathbf{I}_D \cdot \frac{\sum_{i=1}^{N} q_{ik}(\mathbf{x}_i-\boldsymbol{\mu}_k)^{\top}(\mathbf{x}_i-\boldsymbol{\mu}_k)}{D\sum_{i=1}^{N} q_{ik}}$ where D is the dimension of each $\mathbf{x}_i$.

2.3.1 Heuristics for clustering

The clustering algorithms described previously monotonically increase objective functions (via coordinate ascent) and are thus prone to land in local minima. Various heuristics have been devised that assist in overcoming this.

Furthest-first traversal

Parametric clustering algorithms are sensitive to the initial set of cluster centers, $\boldsymbol{\mu}^{(0)}$, so a common initialization that often lies near a good solution (see [49] for theory) is to construct an initial set of centers with a furthest-first traversal. Specifically, the center $\boldsymbol{\mu}_1^{(0)}$ is a random data point $\mathbf{x}_{i_1}$, and subsequent centers, $\boldsymbol{\mu}_k^{(0)}$, are set to the "furthest" data point from $\{\boldsymbol{\mu}_1^{(0)}, \boldsymbol{\mu}_2^{(0)}, \ldots, \boldsymbol{\mu}_{k-1}^{(0)}\}$ where distance is between a point and set of centers is defined as:

$$\text{distance}\left[\mathbf{x}_i, \left\{\boldsymbol{\mu}_1^{(0)}, \boldsymbol{\mu}_2^{(0)}, \ldots, \boldsymbol{\mu}_{k-1}^{(0)}\right\}\right] = \min_{k' \in \{1,2,\ldots,k-1\}} \left\|\mathbf{x}_i - \boldsymbol{\mu}_{k'}^{(0)}\right\|$$

Random restarts

Another effective and commonly-used tactic to counteract sensitivity to the initialization of an exemplar set is to re-run clustering with many different initializations or random restarts. The final result can then be chosen as the restart achieving the best optimization.

Split-and-Merge

During clustering, centers can occasionally become poorly-dispersed in comparison to the data, with many centers describing a few tightly-bunched data points and relatively few centers describing more-dispersed data. In order for centers to migrate evenly to the proper regions, it often entails them traveling through low-likelihood intermediate solutions that will not occur due to the update equations monotonically optimizing their objective. This can be addressed by introducing a heuristic that merges cluster pairs (*e.g.*, where centers occupy roughly the same space, and combining their data into one cluster does not dramatically decrease the likelihood)

or splits clusters (*e.g.*, the center describing data with the lowest probability). The specifics of split-and-merge criteria are described in [98].

$k \cdot \log(k)$ heuristic

Dasgupta *et al.* show in [21, 22] for high-dimensional Gaussians (where dimension $D \gg \ln K$) that the EM algorithm for a mixture of Gaussians can avoid many poor local minima by initializing the algorithm with $\mathcal{O}(K \ln K)$ Gaussians and then pruning this back to K using heuristics. They describe a two-round variant of the EM algorithm which is summarized here:

- Pick L data points (where $L = \mathcal{O}(K \ln K)$) from $\{\mathbf{x}_i\}$ and use them as the initial centers, $\{\boldsymbol{\mu}_1^{(0)}, \boldsymbol{\mu}_2^{(0)}, \ldots, \boldsymbol{\mu}_L^{(0)}\}$. Initialize covariance matrices to $\forall k\colon \boldsymbol{\Sigma}_k^{(0)} = \frac{\mathbf{I}_D}{2D} \min_{i \neq j} \left\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\right\|^2$ for isotropic/spherical Gaussians.
- Run one iteration of the EM algorithm to estimate $\{\boldsymbol{\mu}_k^{(1)}, \boldsymbol{\Sigma}_k^{(1)}\}_{k=1}^L$ and $\forall i\colon \{q_{ik}^{(1)}\}_{k=1}^L$.
- Automatically prune away clusters whose mixing weights, $\sum_{i=1}^N q_{ik}$, fall below $\frac{1}{2L} + \frac{2}{N}$
- Prune away any further surplus clusters by selecting K means from the remaining means via a furthest-first traversal.
- Run an additional EM update to obtain final estimates for $\{\boldsymbol{\mu}_k^{(2)}\}$, $\{\boldsymbol{\Sigma}_k^{(2)}\}$ and $\{q_{ik}^{(2)}\}$.

A more thorough treatment of this heuristic can be found in [21].

2.4 Exemplar-based clustering and the *k*-medoids algorithm

The clustering methods in the previous section were based on assigning data to clusters characterized by location and shape parameters such as means, variances, and even medians. The k-medians clustering algorithm shown in equation (2.7) provides a natural segue to an alternative cluster representation—by actual data points called *exemplars*. For high-dimensional data, it is slightly more efficient to store a pointer to a data point, m_k, rather than the full cluster data

median, $\mathbf{m}_k$. This changes the k-medians update equations to $z_n \leftarrow \underset{k\in\{1,\ldots,K\}}{\operatorname{argmin}} \|\mathbf{x}_i - \mathbf{x}_{m_k}\|$ and $m_k \leftarrow \underset{n:z_n=k}{\operatorname{argmin}} \sum_{n':z_{n'}=k} \|\mathbf{x}_{n'} - \mathbf{x}_n\|$.

If the z_n and m_k updates are iterated, there are numerous needlessly repeated distance computations in both steps—these could be efficiently pre-computed as $\forall i,j \in \{1,\ldots,N\}$: $d(i,j) = \|\mathbf{x}_i - \mathbf{x}_j\|$ or equivalently, similarities[2] $s(i,j) = -d(i,j)$. To complete the transition, a final notation switch is storing the index of each data point's exemplar in an N-ary variable c_i: $c_i \in \mathcal{K} \subseteq \{1,\ldots,N\}$, instead of storing a K-ary index to a cluster number, $z_i \in \{1,\ldots,K\}$. Here, the set of exemplars (*i.e.*, valid assignments for c_i) is $\mathcal{K} \subseteq \{1,\ldots,N\}$, indicated by $\forall k \in \mathcal{K},\ c_k = k$) and the set of non-exemplars is $\overline{\mathcal{K}} = \{1,\ldots,N\}\backslash\mathcal{K}$. The k-medoids algorithm is summarized below in equation (2.7).

K-MEDOIDS CLUSTERING ALGORITHM

INPUT: $\{s(i,j)\}_{i,j\in\{1,\ldots,N\}}$ (data similarities), K (number of clusters)

INITIALIZE: set $\mathcal{K}$ to a random subset of $\{1,\ldots,N\}$ where $|\mathcal{K}| = K$.

REPEAT UNTIL CONVERGENCE:

$$\begin{aligned} &\forall i \notin \mathcal{K}\colon c_i \leftarrow \underset{k\in\mathcal{K}}{\operatorname{argmax}}\, s(i,k) \text{ and } \forall k \in \mathcal{K}\colon c_k \leftarrow k \\ &\forall k \in \mathcal{K}\colon k \leftarrow \underset{\substack{j:c_j=k \\ c_i=k \text{ but } i\neq j}}{\operatorname{argmax}} \sum_{i=1}^{N} s(i,j) \end{aligned} \tag{2.7}$$

OUTPUT: $\{c_i\}_{i=1}^N$ (cluster assignments), $\mathcal{K}$ (exemplars)

The k-medoids algorithm [8] greedily maximizes a quantity referred to as the data similarity, $\mathcal{S}_{\text{data}} = \sum_{i\in\overline{\mathcal{K}}} s(i,c_i)$. There is no longer any reason for enforcing similarity to be defined as $s(i,j) = -\|\mathbf{x}_i - \mathbf{x}_j\|$; for example, with expression data in bioinformatics it is often more convenient to use $s(i,j) = \mathbf{x}_i^\top \mathbf{x}_j$. There is no need for similarities to form a valid metric: symmetry is optional ($s(i,j) \neq s(j,i)$), as is the triangle inequality ($s(i,k) \not\leq s(i,j) + s(j,k)$).

A more general objective function, henceforth referred to as the net similarity, is obtained

[2] To be comparable with k-means and EM for mixture of Gaussians and optimize the same objective, similarities should be re-defined as negative *squared* L^2 distance, unlike k-medoids which uses negative L^2 distance.

by including a model complexity term:

$$\mathcal{S} = \sum\nolimits_{i \in \overline{\mathcal{K}}} s(i, c_i) - \lambda \left|\mathcal{K}\right| \tag{2.8}$$

where λ is a user-specified regularization parameter. If the number of clusters, $K = |\mathcal{K}|$ is not specified in advance, it may seem at first glance that the net similarity is maximized by making all data points exemplars, but this is not the case because of the $\lambda |\mathcal{K}|$ penalty term. For example, if the similarity of one data point to another were greater than $-\lambda$, the net similarity would be higher if the first point were not an exemplar but instead assigned to the second point. Some data points could be known *a priori* to be more or less suitable as exemplars, in which case the model complexity term can depend on which data points are exemplars, $\sum_{k \in \mathcal{K}} \lambda(k)$. We incorporate this into the framework by denoting these as self-similarities $s(k, k) = -\lambda(k)$ or, for the constant-preference case, $\forall k : \; s(k, k) = p = -\lambda$. This simplifies the net similarity objective to:

$$\mathcal{S} = \sum\nolimits_{i=1}^{N} s(i, c_i) = \sum\nolimits_{i \in \overline{\mathcal{K}}} s(i, c_i) + \sum\nolimits_{k \in \mathcal{K}} s(k, k) \tag{2.9}$$

In addition to similarities, the k-medoids algorithm takes as input the number of exemplars, $K = |\mathcal{K}|$, and monotonically optimizes the data similarity, $\mathcal{S}_{\text{data}}$. The algorithm is quite sensitive to its initial exemplar set, and is thus typically re-run with many (*e.g.*, hundreds of) random initializations in order to find a solution with high net similarity and thus avoid more unfortunate restarts that find poor local maxima of $\mathcal{S}$. This is typically not computationally burdensome in the larger context—k-medoids clustering requires $\mathcal{O}(N^2)$ binary operations whereas pre-computing a similarity matrix from data can require $\mathcal{O}(N^2 D)$ operations (or worse), depending on the similarity definition in use.

If the preference regularization parameter, $\forall k : \; p = s(k, k)$ is specified with no value of K, the net similarity in equation (2.9) can be maximized by intelligently[3] searching over net similarities resulting from multiple runs of k-medoids clustering initialized with different values of

[3] *e.g.*, binary search, interpolation search

K.

2.4.1 Linear Programming Relaxation

Maximizing the net similarity objective function in equation (2.9)—or even $\mathcal{S}_{\text{data}}$ for that matter—has been shown to be $\mathcal{NP}$-hard in general [56]. Linear programming relaxations can, however, be employed to find optimal solutions in small problems where $N < 1000$; this is outlined in the 0–1 integer program of equations (2.10–2.11).

0–1 INTEGER PROGRAM FOR K-MEDIANS PROBLEM

INPUT: $\{s(i,j)\}_{i,j\in\{1,\ldots,N\}}$ (data similarities), K (optional number of clusters)

VARIABLES: $b_{ij} \in \{0,1\}$ where $i,j = 1,\ldots,N$

MAXIMIZE:

$$\mathcal{S} = \sum_{i=1}^{N}\sum_{k=1}^{N} b_{ik}\cdot s(i,k) \tag{2.10}$$

SUBJECT TO:

$$\begin{aligned} &\forall i\colon \textstyle\sum_{k=1}^{N} b_{ik} = 1 \text{ (always in exactly one cluster)} \\ &\forall i,k\neq i\colon b_{kk} \geq b_{ik} \text{ (each cluster has an exemplar)} \\ &\textstyle\sum_{k=1}^{N} b_{kk} = K \text{ (optional total number of clusters)} \end{aligned} \tag{2.11}$$

OUTPUT: $\{c_i\}_{i=1}^{N}$ (cluster assignments),
where $b_{ik}=1 \Rightarrow c_i = k$ and $\forall j\neq k\colon b_{ij}=0$

The 0–1 integer program rephrases the previous setup of N integer-valued variables $\{c_i\}_{i=1}^{N}$ as N^2 binary-valued variables, $\{b_{ik}\}$ where $c_i = k$ implies $b_{ik} = 1$. The constraints in equation (2.11) ensure the consistent mapping so $b_{ik} = 1$ for only one k-value and that if $\exists i \neq k\colon b_{ik} = 1$ then point k must be an exemplar ($b_{kk} = 1$). Finally, a constraint on the number of clusters

can be included ($\sum_{k=1}^{N} b_{kk} = K$) if the net similarity, $\mathcal{S}$, is being minimized and not just the data-point similarity, $\mathcal{S}_{\text{data}}$.

A common approach is to solve a linear program relaxation [55, 57] where $\forall i, j : \hat{b}_{ij} \in \mathbb{R} \text{ and } 0 \leq \hat{b}_{ij} \leq 1$ or implemented in optimization software packages such as CPLEX [19]. If the resulting solution is non-integer, stochastic rounding techniques or heuristics [69] have been shown to produce satisfactory results. With current computing technology, such approaches are feasible for problems up to about 1000 data points containing millions of constraints. For the exact solutions shown in Section 4.1, CPLEX 7.1 software was utilized which takes advantage of branch-and-bound techniques and Gomory's cutting-plane method [42].

Other possible approaches to exemplar-based clustering borrow from techniques employed for minimizing the sum of cut weights while partitioning graphs (graph cuts) or its dual formulation, maximizing network flow [31, 32]. The optimal two-way (binary) graph-cut can be found in polynomial time [43], which corresponds to finding a $K = 2$ clustering solution whose search space is only $\mathcal{O}(N^2)$. There are many approximate techniques for finding general K-way graph cuts, such as simulated annealing [13, 62, 75], Gibbs sampling [41], and iterated conditional modes [6], but more recent techniques such as using expansion moves and swap moves [9] have shown greatly improved performance. For example, α-expansion moves involve iteratively solving binary subproblems constructed by choosing one class label and lumping the remainder in the other class; α-β-swap moves involve finding swaps of two class labels that improve the objective similar to the vertex substitution heuristic described in Section 2.5. A useful overview of these formulations can be found in [64].

2.5 The Facility Location Problem

Facility location is an important area of operations research. Simply stated, it is concerned with finding facility locations to be matched with subsets of customers so as to minimize a delivery cost. If the task involves selecting a subset of possible facility locations to be utilized

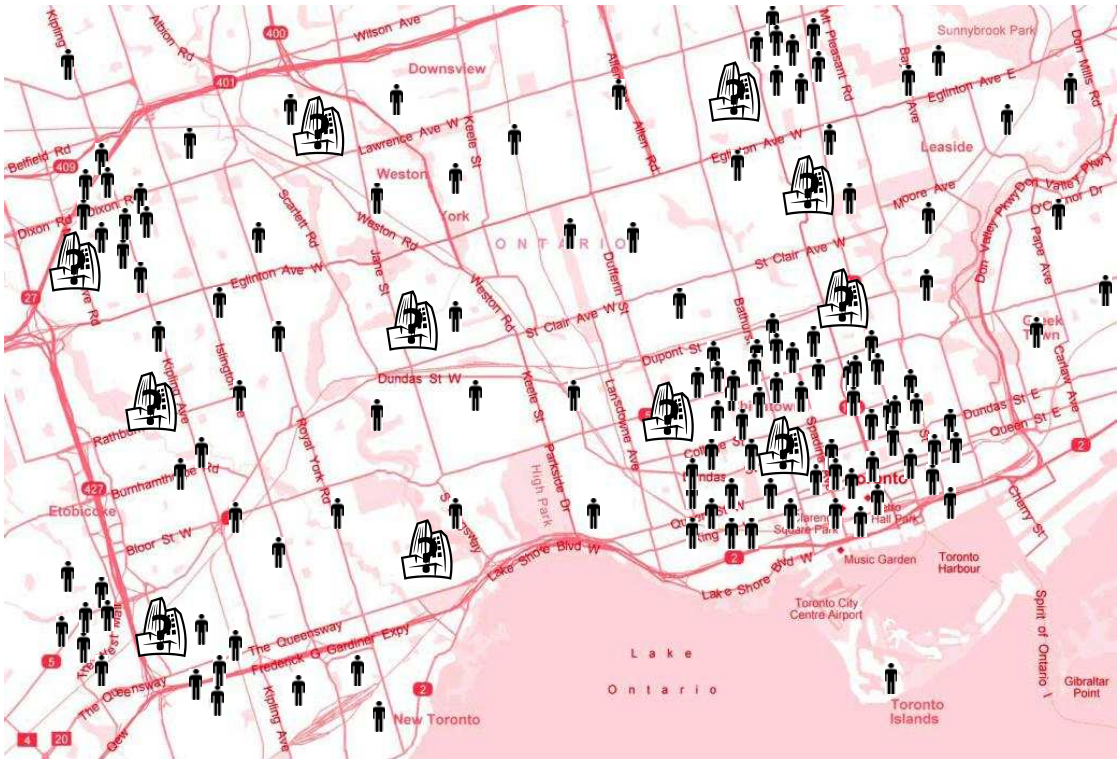


Figure 2.2: The discrete facility location problem is concerned with finding a subset of potential facility locations (shown as buildings with '?') to open in order to best serve a population of customers (stick-figures above). Note that the background image is a map of Toronto, suggesting that 'distance' need not be defined as Euclidean (L^2) distance. In fact, it could be Manhattan (L^1)—or perhaps more aptly, Toronto (L^1)—distance to reflect road travel distance, average driving time (to account for traffic congestion and expressway speeds), or even travel time via public transit.

(*i.e.* discrete facility location) and the cost is the sum of customer distances from said facilities, this is known as the p-median problem (PMP). Alternatively, the cost could be the maximum distance between customers and facilities—known as the p-center problem—in which case the objective function is of the minimax variety instead of minisum. The clustering framework described in Section 2 is closely related to the p-median problem.

The p-median problem was formally defined and investigated in literature from the early 1960s with notable contributions from Cooper ([15],[17],[16]) and Hakimi ([45],[46]); for a more recent survey of approaches to the problem see [78]. It is defined as follows: given a set, $\mathcal{N}$, of possible facility locations and a set, $\mathcal{M}$, of customers to be serviced, select a subset

$\mathcal{L} \subseteq \mathcal{M}$ of those facilities to open (where $p = |\mathcal{L}|$) such that the sum of distances from customers to their closest open facility is minimized. Matrix $D \in \mathbb{R}^{M \times N}$, where $M = |\mathcal{M}|$ is the number of customers and $N = |\mathcal{N}|$ is the number of facility locations, contains distances such that element $d_{mn} \geq 0$ is the distance from customer m to potential facility n.

In purely mathematical terms, the task is to select p columns of matrix D such that the sum of the smallest element of each row is minimized. The cost function is

$$\mathcal{D}(\mathcal{L}) = \sum\nolimits_{m \in \mathcal{M}} \min_{n \in \mathcal{L}} d_{mn} \; . \tag{2.12}$$

The search space for this problem is of size $\binom{M}{p}$ and finding the optimal subset, $\mathcal{L}^*$, has been shown to be $\mathcal{NP}$-hard in general [56]. An exact solution is possible for many problems with hundreds of facilities based on linear programming relaxations of the integer programming problem [11, 86]. MN binary variables $\{b_{mn}\}$ are introduced to indicate which facilities serve each customer (*i.e.*, $b_{mn} = 1$ if customer m is served by facility n, and $b_{mn} = 0$ otherwise), and N binary variables $\{a_n\}$ indicate which facilities are opened (*i.e.*, $a_n = 1$ facility n is open, and $a_n = 0$ otherwise).

0–1 INTEGER PROGRAM FOR P-MEDIAN PROBLEM

INPUT: distances $\{d_{mn}\}$ where $m \in \{1,\ldots,M\}$ and $n \in \{1,\ldots,N\}$
number of open facilities, p

VARIABLES: $b_{mn} \in \{0,1\}$ and $a_n \in \{0,1\}$ where $m = 1,\ldots,M$ and $n = 1,\ldots,N$

MINIMIZE:

$$\mathcal{D} = \sum_{n=1}^{N} \sum_{m=1}^{M} b_{mn} d_{mn} \tag{2.13}$$

SUBJECT TO:

$$\begin{aligned} &\forall m\colon \textstyle\sum_{n=1}^{N} b_{mn} = 1 && \text{(demand of each customer must be met)} \\ &\forall m,n\colon b_{mn} \leq a_n && \text{(unopened facilities cannot service customers)} \\ &\textstyle\sum_{n=1}^{N} a_n = p && \text{(number of opened facilities)} \end{aligned} \tag{2.14}$$

The p-median formulation in equations (2.13–2.14) is the same as k-medians from equations (2.10–2.11) if the customer and location sets are identical, *i.e.*, $\mathcal{M} = \mathcal{N}$.

For problems containing larger number of facilities ($N > 1000$), exact solutions via linear programming relaxations are usually unavailable with current computing technology so the task is left to heuristics. Standard facility-location heuristics include:

Greedy Heuristic [66]: Initialize the set of open facilities, $\mathcal{L}^{(0)}$, to be the empty set. Perform p rounds during which an unopened facility $n_t \in \mathcal{M} \backslash \mathcal{L}$ is opened during the t^{th} round ($\mathcal{L}^{(t)} = \mathcal{L}^{(t-1)} \cup n_t$) so that the cost decrease between rounds, $\left|\mathcal{D}(\mathcal{L}^{(t)}) - \mathcal{D}(\mathcal{L}^{(t-1)})\right|$, is maximized.

Stingy Heuristic [29]: Initialize the set of open facilities, $\mathcal{L}^{(0)}$ to be $\mathcal{N}$. Perform $M-p$ rounds during which one open facility $n_t \in \mathcal{L}^{(t)}$ is closed so that the cost increase between rounds, $\left|\mathcal{D}(\mathcal{L}^{(t)}) - \mathcal{D}(\mathcal{L}^{(t-1)})\right|$, is minimized.

Alternating Heuristic [72]: The alternating heuristic is identical to k-medoids clustering in equation (2.7), whereby there are alternating phases of assigning users to the closest

opened facility and open facilities are replaced by new facilities nearest to median of their customers' location.

Vertex Substitution Heuristic (VSH) [96]: Randomly initialize $\mathcal{L}$ to contain p facilities. For each unopened facility $n \in \mathcal{M} \setminus \mathcal{L}$, find the open facility, $\ell \in \mathcal{L}$ to substitute with it so as to most-improve the cost function if possible, *i.e.* $\max_\ell [\mathcal{D}(\mathcal{L}) - \mathcal{D}(\mathcal{L} \cup n \setminus \ell)]$. This process is repeated until convergence, when no cost-reducing substitutions are possible.

Some algorithms have provable worst-case guarantees (*e.g.* [14]), whereby their solution's cost $\mathcal{D}(\mathcal{L})$ is related to the optimal cost $\mathcal{D}(\mathcal{L}^*)$ by a constant factor as follows: $\left| \frac{\mathcal{D}(\mathcal{L}) - \mathcal{D}(\mathcal{L}*)}{\mathcal{D}(\mathcal{L})} \right| \leq \varepsilon$. Values of ε are rarely small and often much larger than the typical error, so this may be a poor guide to selecting an algorithm [78]. The vertex substitution heuristic [96] has been the standard for comparison for four decades and provides the basis for the variable neighborhood search meta-heuristic [47] that was compared with affinity propagation in [10, 35].

Variable-neighborhood search utilizes speedups to the original vertex substitution heuristic by storing all nearest and second-nearest open facilities for each customer and only recomputing certain elements in these lists when necessary [106] (*i.e.*, a pertinent substitution is made). It also restructures the $p(N-p)$ possible interchanges to involve fewer comparisons with early exit conditions, and randomly chooses higher-order π-ary interchanges[4] to escape local minima.

2.6 Factor Graphs and the Sum-Product Algorithm

Many physical systems involve complex interactions among large numbers of variables, which can be realistically approximated by relationships between small subsets of variables. For example, an image's pixels may all be interrelated, however, for some applications this is ap-

[4]The π-ary interchange search space for each iteration has size $\binom{p}{\pi}\binom{N-p}{\pi}$, which grows impractically large for interesting problems where $N > 1000$ and p is non-trivial ($5 < p < N-5$). Experiments for $\pi = 2$ have been conducted [28] but only for $N \leq 30$.

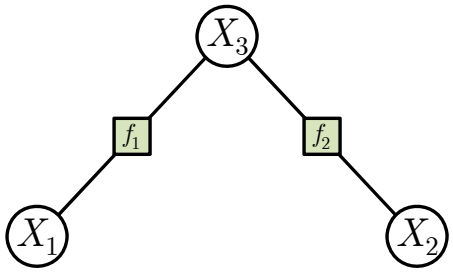


Figure 2.3: A sample factor graph showing a relationship between three variables, X_1, X_2, and X_3, and two connecting function nodes, f_1 and f_2.

proximated as a regular network of correlations between neighboring pixels. Graphical models are a useful device for succinctly expressing and visualizing the structure and dependencies present in networks of variables.

2.6.1 Factor Graphs

Standard graphical models such as Bayesian networks [83] and Markov random fields [60] have long been used for modeling hierarchical dependencies and energy-based models, respectively. A more recent innovation is the factor graph [65], a graphical model that provides a natural way of representing global functions or probability distributions that can be factored into simpler local functions. A factor graph is a bi-partite graph consisting of a set of N nodes representing random variables $\mathbf{X} = \{X_1, X_2, \ldots, X_N\}$ (from domain $\mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n$) and M nodes representing a set of functions $\mathbf{F} = \{f_1, f_2, \ldots, f_M\}$. Each function node, f_m, represents a function with codomain $[0, \infty)$ that depends only on the subset of variable nodes, neighbors $N(m) \subseteq \{1, \ldots, N\}$, directly connected to it. The factor graph represents a global function, customarily taken to be proportional to the joint probability of each configuration $\{X_1{=}x_1, X_2{=}x_2, \ldots, X_n{=}x_n\}$, that is the product of all its function nodes:

$$f(\mathbf{X}{=}\mathbf{x}) = \prod_{m=1}^{M} f_m(\mathbf{x}_{N(m)}) \ ,$$

where $\mathbf{x}_{N(m)}$ denotes the argument of function f_m, $\{x_n\}_{n \in N(m)}$.

For example, the factor graph in Figure 2.3 shows an interconnection between N=3 random variables, X_1, X_2, and X_3, each of which can take on values $x_1 \in \mathcal{X}_1$, $x_2 \in \mathcal{X}_2$, and $x_3 \in \mathcal{X}_3$, respectively. The factor graph shows a topology with $M=2$ function nodes where f_1 is connected to X_1 and X_3 (so $N(1)=\{1,3\}$), and f_2 is connected to X_2 and X_3 (so $N(2)=\{2,3\}$). This implies a global function that can be factorized in the following way:

$$f(x_1, x_2, x_3) = f_1(x_1, x_3) \cdot f_2(x_2, x_3) \, .$$

Interpreting the global function as proportional to the configuration probabilities, the marginal probability distributions, $\{p_1(x_1), p_2(x_2), p_3(x_3)\}$ can be found by summing over all configurations of the other variables:

$$\begin{aligned}
P(X_1=x_1) &\propto \sum_{x_2\in\mathcal{X}_2} \sum_{x_3\in\mathcal{X}_3} f(x_1, x_2, x_3) = \sum_{x_2\in\mathcal{X}_2} \sum_{x_3\in\mathcal{X}_3} f_1(x_1, x_3) \cdot f_2(x_2, x_3) \, , \\
P(X_2=x_2) &\propto \sum_{x_1\in\mathcal{X}_1} \sum_{x_3\in\mathcal{X}_3} f(x_1, x_2, x_3) = \sum_{x_1\in\mathcal{X}_1} \sum_{x_3\in\mathcal{X}_3} f_1(x_1, x_3) \cdot f_2(x_2, x_3) \, , \\
P(X_3=x_3) &\propto \sum_{x_1\in\mathcal{X}_1} \sum_{x_2\in\mathcal{X}_2} f(x_1, x_2, x_3) = \sum_{x_1\in\mathcal{X}_1} \sum_{x_2\in\mathcal{X}_2} f_1(x_1, x_3) \cdot f_2(x_2, x_3) \, .
\end{aligned} \tag{2.15}$$

Using the distributive law

$$\begin{aligned}
&\textstyle\sum_y [\text{factors independent of } y] \cdot [\text{factors dependent on } y] \\
&\qquad = [\text{factors independent of } y] \cdot \textstyle\sum_y [\text{factors dependent on } y] \, ,
\end{aligned}$$

the computation can be simplified to:

$$\begin{aligned}
P(X_1=x_1) &\propto \sum_{x_2\in\mathcal{X}_2} \sum_{x_3\in\mathcal{X}_3} \overbrace{f_1(x_1, x_3) \cdot f_2(x_2, x_3)}^{f(x_1,x_2,x_3)} = \sum_{x_3\in\mathcal{X}_3} f_1(x_1, x_3) \cdot \textstyle\sum_{x_2\in\mathcal{X}_2} f_2(x_2, x_3) \, , \\
P(X_2=x_2) &\propto \sum_{x_1\in\mathcal{X}_1} \sum_{x_3\in\mathcal{X}_3} \overbrace{f_1(x_1, x_3) \cdot f_2(x_2, x_3)}^{f(x_1,x_2,x_3)} = \sum_{x_3\in\mathcal{X}_3} f_2(x_2, x_3) \cdot \textstyle\sum_{x_1\in\mathcal{X}_1} f_1(x_1, x_3) \, , \\
P(X_3=x_3) &\propto \sum_{x_1\in\mathcal{X}_1} \sum_{x_2\in\mathcal{X}_2} \overbrace{f_1(x_1, x_3) \cdot f_2(x_2, x_3)}^{f(x_1,x_2,x_3)} = \left[\sum_{x_1\in\mathcal{X}_1} f_1(x_1, x_3)\right] \cdot \left[\sum_{x_2\in\mathcal{X}_2} f_2(x_1, x_3)\right] \, .
\end{aligned} \tag{2.16}$$

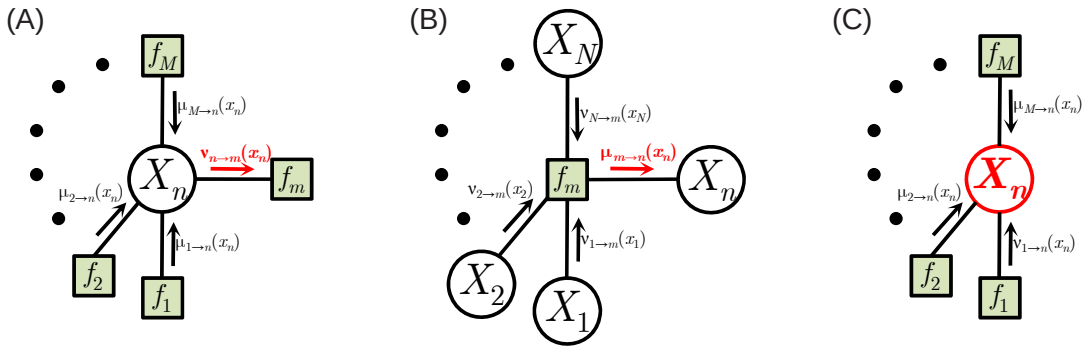


Figure 2.4: Belief propagation on a factor graph. The function-to-variable node message, shown in (A), is computed from incoming variable-to-function node messages along other edges. Likewise, the variable-to-function node message in (B) is computed from incoming function-to-variable node messages. The marginals for a variable (C) can be computed from all incoming function-to-variable messages.

In effect, the brute-force approach of summing together all possible products (equation (2.15)) can be simplified by intelligently organizing the factors into sum-product-sum-product form (equation (2.16)). The *sum-product algorithm*, described below, represents a computationally-efficient rearrangement of these sums and products.

2.6.2 Sum-Product Algorithm

A notable formulation of the sum-product algorithm is Judea Pearl's use of it as "belief propagation" in [84] for marginalizing variables in Bayesian networks. The algorithm is a series of rules—framed as passing messages between factor graph nodes—that organize and automate the application of the distributive property. For example, in equation (2.16) where marginal $P(X_1 = x_1)$ is computed, if the inner sum is considered a function of x_3 (*i.e.*, $\mu_2(x_3) = \sum_{x_2 \in \mathcal{X}_2} f_2(x_2, x_3)$), the expression becomes $P(X_1 = x_1) = \sum_{x_3 \in \mathcal{X}_3} f_1(x_1, x_3) \cdot \mu_2(x_3)$ where μ_2 is a "message" from the inner summation to the outer.

The sum-product algorithm involves the propagation of messages from variable nodes to function nodes (ν in Figure 2.4 (A)) and from function nodes to variable nodes (μ in Figure 2.4 (B)). The message sent by a variable node, X_n, to a function node f_m is a function of $x_n \in \mathcal{X}_n$ and reflects the current probability distribution ('beliefs') about X_n given evidence

from all its other neighbors, $N(n)\backslash m$. It is an element-wise product:

$$\nu_{n\to m}(x_n) = \prod\nolimits_{m'\in N(n)\backslash m} \mu_{m'\to n}(x_n)\ . \tag{2.17}$$

The message a function node, f_m, sends to variable node X_n is a function of $x_n \in \mathcal{X}_n$ and reflects the current beliefs about X_n given the function and its other neighbors, $N(m)\backslash n$:

$$\mu_{m\to n}(x_m) = \sum_{\mathbf{x}_{N(m)\backslash n}} f_m(\mathbf{x}_{N(m)}) \cdot \prod\nolimits_{n'\in N(m)\backslash n} \nu_{n'\to m}(x_n)\ . \tag{2.18}$$

Note the shorthand use of $\mathbf{x}_{N(m)}$ for $\{x_n\}_{n\in N(m)}$ and $\mathbf{x}_{N(m)\backslash n}$ for $\{x_{n'}\}_{n'\in N(m)\backslash n}$

Finally, the current beliefs about any variable, X_n, can be computed any time by fusing incoming function-to-variable messages:

$$q_n(X_n{=}x_n) = \prod\nolimits_{m\in N(n)} \mu_{m\to n}(x_n) \tag{2.19}$$

These are known as pseudo-marginals, though for singly-connected graphs (*i.e.*, tree-structured graphs that contain no loops), these converge to the true marginals of $f(\mathbf{x})$ in a finite number of message-update iterations [84].

2.6.3 Loopy Belief Propagation

Pearl [84] shows that belief propagation converges to the true marginals in a finite number of message-update iterations for singly-connected graphs. With respect to the general case, he states:

> When loops are present, the network is no longer singly-connected and local propagation schemes will invariably run into trouble ... If we ignore the existence of loops and permit the nodes to continue communicating with each other as if the network were singly-connected, messages may circulate indefinitely around the loops and the process may not converge to a stable equilibrium ... (even if it does) this asymptotic equilibrium is not coherent, in the sense that it does not represent the posterior probabilities of all the nodes of the network. (p. 195)

Previous impressive empirical results from the area of coding theory [5] were shown to be a special case of belief propagation in this loopy case [73]. Additional factor graphs (involving applications such as medical diagnostics [79] and phase-unwrapping [63]) were investigated at the time; positive results gave additional credence to the idea that pseudo-marginals provided useful enough approximations that $\operatorname{argmax}_{x_n} q(x_n) = \operatorname{argmax}_{x_n} P(x_n)$ for many nodes (values of n). Theoretical justification for loopy belief propagation was later shown [111], where update equations (2.17–2.19) were related to minimizing a KL-divergence [18] computed using Bethe's free energy approximation [7] from 1935.

Given a factor graph describing a probability density of $P(\mathbf{x}) \propto \prod_m f_m(\mathbf{x}_{N(m)})$, one can search for a simpler approximating distribution, $Q(\mathbf{x})$, such that the KL-divergence between them, $D\big(Q(\mathbf{x}) \,\|\, P(\mathbf{x})\big) = \sum_{\mathbf{x}} Q(\mathbf{x}) \cdot \log \frac{Q(\mathbf{x})}{P(\mathbf{x})}$, is minimized. Where $P(\mathbf{x}) \propto \prod_m f_m(\mathbf{x}_{N(m)})$, this expands to:

$$D\big(Q(\mathbf{x}) \,\|\, P(\mathbf{x})\big) + \text{constant} = \sum_{\mathbf{x}} Q(\mathbf{x}) \cdot \log Q(\mathbf{x}) - \sum_{\mathbf{x}} Q(\mathbf{x}) \sum_m \log f_m(\mathbf{x}_{N(m)})$$

If it is assumed that the approximating distribution $Q(\mathbf{x})$ can be factorized into single-node marginals $q_n(x_n)$ for each variable node, and function-node or clique marginals $q_m(\mathbf{x}_{N(m)})$ for each function node[5] like this:

$$Q(\mathbf{x}) = \frac{\prod_{m=1}^{M} q_m(\mathbf{x}_{N(m)})}{\prod_{n=1}^{N} q_n(x_n)^{|N(n)|-1}} \,, \tag{2.20}$$

then the Bethe approximation to the free energy is obtained:

$$\begin{aligned} \mathcal{F}_{\text{Bethe}} = \; & \sum_m \sum_{\mathbf{x}_{N(m)}} q_m(\mathbf{x}_{N(m)}) \cdot \log q_m(\mathbf{x}_{N(m)}) - \sum_m \sum_{\mathbf{x}_{N(m)}} q_m(\mathbf{x}_{N(m)}) \cdot \log f_m(\mathbf{x}_{N(m)}) \\ & - \sum_n \big(|N(n)|{-}1\big) \sum_{x_n} q_n(x_n) \cdot \log q_n(x_n) \,. \end{aligned} \tag{2.21}$$

[5] For example, in the factor graph from Figure 2.3, the single-node marginals are $q_{n=1}(x_1)$, $q_{n=2}(x_2)$, and $q_{n=3}(x_3)$; the clique marginals are $q_{m=1}(x_1, x_3)$ and $q_{m=2}(x_2, x_3)$.

It is useful to perform coordinate descent minimization on the Bethe free energy subject to the valid distribution constraints that $\forall n: \sum_{x_n} q_n(x_n) = 1$ and $\forall m: \sum_{\mathbf{x}_{N(m)}} q_m(\mathbf{x}_{N(m)}) = 1$, and the constraint that single-node marginals are consistent with clique marginals $\forall m, n \in N(m)$: $q_n(x_n) = \sum_{\mathbf{x}_{N(m)\setminus n}} q_m(\mathbf{x}_{N(m)})$. The complete derivation is shown in Appendix A, however, the end-result yields familiar-looking update equations as follows:

$$\begin{aligned} \nu_{n\to m}(x_n) &\propto \prod_{m'\in N(n)\setminus m} \mu_{m'\to n}(x_n) \quad \text{and} \quad \mu_{m\to n}(x_n) \propto \sum_{\mathbf{x}_{N(m)\setminus n}} f_m(\mathbf{x}_{N(m)}) \cdot \prod_{n'\in N(m)\setminus n} \nu_{n'\to m}(x_{n'}) \,; \\ q_n(x_n) &\propto \prod_{m\in N(n)} \mu_{m\to n}(x_n) \quad \text{and} \quad q_m(\mathbf{x}_{N(m)}) \propto f_m(\mathbf{x}_{N(m)}) \cdot \prod_{n\in N(m)} \nu_{n\to m}(x_n) \,, \end{aligned} \tag{2.22}$$

where the $q_n(x_n)$ are pseudo-marginals from earlier. It should be noted that for singly-connected graphs the factorization in equation (2.20) is exact and the algorithm converges to the exact marginals.

2.6.4 Max-Product Algorithm

As described in [1], the idea behind the sum-product algorithm and factor graphs can be applied to any commutative semiring[6]. In many cases it is more desirable or efficient to use the max-product algorithm, whose messages are updated as follows:

$$\nu_{n\to m}(x_n) \propto \prod_{m'\in N(n)\setminus m} \mu_{m'\to n}(x_n) \quad \text{and} \quad \mu_{m\to n}(x_n) \propto \max_{\mathbf{x}_{N(m)\setminus n}} f_m(\mathbf{x}_{N(m)}) \cdot \prod_{n'\in N(m)\setminus n} \nu_{n'\to m}(x_{n'}) \,. \tag{2.23}$$

For the max-product algorithm, the computation of $q_n(x_n)$ and $q_m(\mathbf{x}_{N(m)})$ pseudo-marginals is intended for estimating the best configuration, $\mathbf{x}^*$, of the variables, $\mathbf{x}$, which can be performed as follows:

[6] A semiring is an algebraic structure which generalizes the additive and multiplicative properties of the set of natural numbers $\mathbb{N}$ (including zero); unlike rings, there is no requirement that each element have an additive inverse.

$$x_n^* = \operatorname*{argmax}_{x_n} \prod_{m \in N(n)} \mu_{m \to n}(x_n) \tag{2.24}$$

These quantities are closely related to evaluations of best-path configurations in the Viterbi algorithm more so than the probabilistic interpretation in Section 2.6.2. Note that for reasons of numerical stability, much of the work in this thesis is performed in the max-sum semiring, which is isomorphic to max-product via the mapping $x \to \log(x)$, assuming $x > 0$.

Chapter 3

Affinity Propagation

The exemplar-based clustering algorithms described in Sections 2.4–2.5 operate by iteratively refining a randomly-chosen initial set of exemplars, $\mathcal{K} \subseteq \{1, 2, \ldots, N\}$, but this works well only if that initial subset of data points is close to a good solution. Affinity propagation simultaneously considers all data points as possible exemplars, exchanging real-valued messages between them until a high-quality set of exemplars (and corresponding clusters) emerges. Messages are updated on the basis of simple formulae that reflect sum-product or max-product update rules and, at any point in time, the magnitude in each message reflects the current affinity that one point has for choosing another data point as its exemplar, hence the name "affinity propagation".

Affinity propagation takes as input a collection of real-valued similarities between data points, $\{s(i, k)\}$, where each similarity $s(i, k)$ indicates how well the data point with index k is suited to be the exemplar for data point i. Each data point is paired with a variable node, c_i in a factor graph (Section 2.6) as shown in Figure 3.1. A value of $c_i = k$ for $i \neq k$ indicates that data point i is assigned to a cluster with point k as its exemplar; $c_k = k$ indicates that data point k serves as a cluster exemplar. The graph's function is a constrained net similarity (exponentiated, so the function is non-negative), defined as follows:

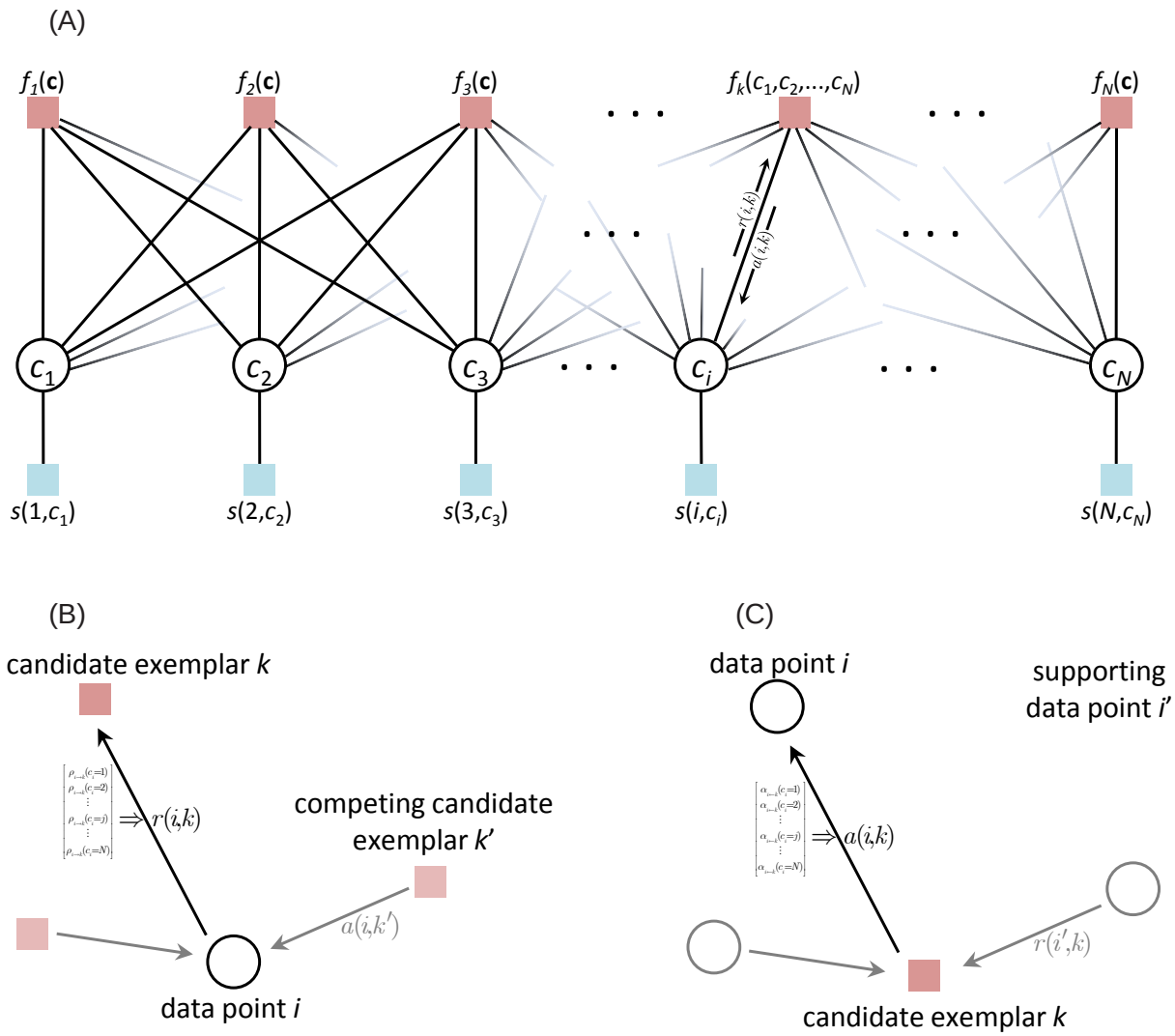


Figure 3.1: Affinity propagation is an exemplar-based clustering algorithm that performs belief propagation on the factor graph shown in (A). Two kinds of message are passed in the graph; *responsibilities* (B) are passed from variable nodes to function nodes (*i.e.*, data points to candidate exemplars). *Availabilities* are passed from function nodes to variable nodes (C), interpreted as candidate exemplars to data points..

$$F(\mathbf{c};\,\mathbf{s}) = e^{\overbrace{\sum_{i=1}^{N} s(i,c_i)}^{\mathcal{S}\text{ from (2.9)}} + \overbrace{\sum_{k=1}^{N} \log f_k(\mathbf{c})}^{\text{coherence constraint}}} = \prod_{i=1}^{N} e^{s(i,c_i)} \cdot \prod_{k=1}^{N} f_k(\overbrace{c_1, c_2, \ldots, c_N}^{\mathbf{c}}) \quad (3.1)$$

Note that the first term in the exponent involves the net similarity, $\mathcal{S}$, from the k-median problem, except that similarities are exponentiated to ensure $F(\mathbf{c};\mathbf{s})$ always evaluates to a positive function. The second term contains a coherence constraint defined as follows:

$$f_k(\mathbf{c}) = \begin{cases} 0, \text{ if } c_k \neq k \text{ but } \exists i\colon c_i = k \text{ (disallow clusters without an exemplar)} \\ 1, \text{ otherwise} \end{cases} \quad (3.2)$$

which causes the function to evaluate to zero for the incoherent configuration of a cluster without an exemplar, *i.e.*, a data point i has chosen k as its exemplar ($c_i = k$) with k having been incorrectly labeled as a non-exemplar ($c_k \neq k$).

Each component of $F(\mathbf{c};\,\mathbf{s})$ is represented by a function node and each label c_i is represented by a variable node. Each $f_k(\mathbf{c})$ term appearing in equation (3.1) has a corresponding function node that is connected to all variables $c_1, c_2, \ldots, c_N$. In addition, each $s(i, c_i)$ term has a corresponding function node that is connected to the single variable c_i. The log of the global function $F(\mathbf{c};\mathbf{s})$—in this case $\mathcal{S}(\mathbf{c})$ (previously referred to as net similarity, $\mathcal{S}$)—is given by the sum of all the log-functions represented by function nodes.

3.1 Sum-Product Affinity Propagation

The sum-product algorithm can be used to search over configurations of variables $\mathbf{c}$ in the factor graph to maximize $F(\mathbf{c};\,\mathbf{s})$, which also maximizes $e^{\mathcal{S}(\mathbf{c})}$ (and $\mathcal{S}(\mathbf{c})$) subject to the coherency constraint. The sum-product algorithm for this particular graph topology can be derived in a straightforward fashion and consists of sending messages from variables to functions and from functions to variables in a recursive fashion (see Section 2.6.2).

$\mathcal{O}(N^N)$ vector message updates

The message sent from variable node c_i to function node $f_k(\mathbf{c})$ consists of N non-negative real numbers—one for each possible value, j, of c_i—and can be denoted $\rho_{i\to k}(j)$ as shown in Figure 3.1(B). A later simplification reduces this N-vector to a scalar value, making affinity propagation scale linearly in time and memory with the number of similarities. The message sent from function node $f_k(\mathbf{c})$ to variable node c_i also consists of N real numbers and can be denoted $\alpha_{i\leftarrow k}(j)$ as shown in Figure 3.1(C). At any time, the value of c_i can be estimated by multiplying together all incoming messages.

Since the ρ-messages are outgoing from variables, they are computed as the element-wise multiplication of all incoming messages:

$$\rho_{i\to k}(c_i) = e^{s(i,c_i)} \cdot \prod_{\substack{k'=1,\\ k'\neq k}}^{N} \alpha_{i\leftarrow k'}(c_i) \quad . \tag{3.3}$$

Messages sent from functions to variables are computed by multiplying incoming messages and then summing over all variables except the variable the message is being sent to. Because all function nodes are connected to all N variable nodes, this nominally involves N–1 summations over N^N possible configurations—for each of N function nodes.

$\mathcal{O}(N^3)$ vector message updates

Fortunately, all functions $\{f_k(\mathbf{c})\}_{k=1}^{N}$ are binary-valued constraints that are completely factorizable given c_k as follows:

$$f_k(\mathbf{c}) = \begin{cases} \prod_{\substack{i=1,\\ i\neq k}}^{N} [c_i \neq k], \text{ for } c_k \neq k \ , \\ \qquad 1, \text{ for } c_k = k \ . \end{cases}$$

If the two cases, $c_k = k$ and $c_k \neq k$, are handled with separate expressions, the functions can be absorbed into the summations by changing limits, and incoming messages can be independently summed (sum and product operators change places). Accordingly, the message sent

from function node f_k to variable node c_i is:

$$\alpha_{i\leftarrow k}(c_i) = \overbrace{\sum_{j_1}\sum_{j_2}\cdots\sum_{j_{i-1}}\sum_{j_{i+1}}\cdots\sum_{j_N}\left[f_k(j_1,j_2,\ldots,j_{i-1},c_i,j_{i+1},\ldots,j_N)\cdot\prod_{i':i'\neq i}\rho_{i'\to k}(j_{i'})\right]}^{\text{sum over all configurations satisfying } f_k \text{ given } c_i}$$

$$= \begin{cases} \overbrace{\prod_{i':i'\neq i}\sum_j \rho_{i'\to k}(j)}^{\text{all configurations, with or without cluster } k}, & \text{for } i=k \text{ and } c_k=k\,; \\ \overbrace{\prod_{i':i'\neq i}\sum_{j:j\neq k} \rho_{i'\to k}(j)}^{\text{all configurations without cluster } k}, & \text{for } i=k \text{ and } c_k\neq k\,; \\ \overbrace{\rho_{k\to k}(k)}^{k \text{ is an exemplar}} \cdot \overbrace{\prod_{i':i'\notin\{i,k\}}\sum_j \rho_{i'\to k}(j)}^{\text{all configurations, with or without cluster } k}, & \text{for } i\neq k \text{ and } c_i=k\,; \\ \overbrace{\sum_{j:j\neq k}\rho_{k\to k}(j)\cdot\prod_{i':i'\notin\{i,k\}}\sum_{j:j\neq k}\rho_{i'\to k}(j)}^{\text{all configurations with no cluster } k} + \overbrace{\rho_{k\to k}(k)\cdot\prod_{i':i'\notin\{i,k\}}\sum_j \rho_{i'\to k}(j)}^{\text{all configurations with a cluster } k}, & \text{for } i\neq k \text{ and } c_i\neq k\,. \end{cases}$$

These vector messages are easier to interpret if we view them as the product of constant and variable (with respect to c_i) components as follows: $\rho_{i\to k}(c_i) = \bar{\rho}_{i\to k}\cdot\tilde{\rho}_{i\to k}(c_i)$ and $\alpha_{i\leftarrow k}(c_i) = \bar{\alpha}_{i\leftarrow k}\cdot\tilde{\alpha}_{i\leftarrow k}(c_i)$. This changes the messages to:

$$\rho_{i\to k}(c_i) = e^{s(i,c_i)}\cdot\textstyle\prod_{k':k'\neq k}\bar{\alpha}_{i\leftarrow k'}\cdot\prod_{k':k'\neq k}\tilde{\alpha}_{i\leftarrow k'}(c_i) \tag{3.4}$$

and

$$\alpha_{i\leftarrow k}(c_i) = \begin{cases} \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\prod_{i':i'\neq i}\sum_j\tilde{\rho}_{i'\to k}(j), & \text{for } i=k \text{ and } c_k=k\,; \\ \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\prod_{i':i'\neq i}\sum_{j:j\neq k}\tilde{\rho}_{i'\to k}(j), & \text{for } i=k \text{ and } c_k\neq k\,; \\ \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\tilde{\rho}_{k\to k}(k)\cdot\prod_{i':i'\notin\{i,k\}}\sum_j\tilde{\rho}_{i'\to k}(j), & \text{for } i\neq k \text{ and } c_i=k\,; \\ \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\prod_{i':i'\neq i}\sum_{j:j\neq k}\tilde{\rho}_{i'\to k}(j) + \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\tilde{\rho}_{k\to k}(k)\cdot\prod_{i':i'\notin\{i,k\}}\sum_j\tilde{\rho}_{i'\to k}(j), & \text{for } c_i\neq k\neq i\,. \end{cases} \tag{3.5}$$

For convenience, if we let $\bar{\rho}_{i\to k} = \sum_{j:j\neq k}\rho_{i\to k}(j)$ then $\sum_{j:j\neq k}\tilde{\rho}_{i\to k}(j) = 1$ and thus $\sum_j\tilde{\rho}_{i\to k}(j) = 1+\tilde{\rho}_{i\to k}(k)$. Also note that in the update for $\alpha_{i\leftarrow k}(c_i)$ (equation (3.5)), none of the expressions

explicitly contain c_i—only the choice of expression depends on c_i. Consequently, the N-vector of messages, $\alpha_{i\leftarrow k}(c_i)$ has only two unique values: one for $c_i = k$ and another for $c_i \neq k$. Setting $\bar{\alpha}_{i\leftarrow k} = \alpha_{i\leftarrow k}(c_i : c_i \neq k)$ makes $\tilde{\alpha}_{i\leftarrow k}(c_i) = 1$ for all $c_i \neq k$. This also means that $\prod_{k':k'\neq k} \tilde{\alpha}_{i\leftarrow k'}(c_i) = \tilde{\alpha}_{i\leftarrow c_i}(c_i)$ for all $c_i \neq k$ and $\prod_{k':k'\neq k} \tilde{\alpha}_{i\leftarrow k'}(c_i) = 1$ for $c_i = k$, leading to further simplification:

$$\rho_{i\to k}(c_i) = \begin{cases} e^{s(i,k)} \cdot \prod_{k':k'\neq k} \bar{\alpha}_{i\leftarrow k'}, \text{ for } c_i = k\,; \\ e^{s(i,c_i)} \cdot \tilde{\alpha}_{i\leftarrow c_i}(c_i) \cdot \prod_{k':k'\neq k} \bar{\alpha}_{i\leftarrow k'}, \text{ for } c_i \neq k\,, \end{cases} \tag{3.6}$$

and

$$\alpha_{i\leftarrow k}(c_i) = \begin{cases} \prod_{i':i'\neq i} \bar{\rho}_{i'\to k} \cdot \prod_{i':i'\neq i} [1 + \tilde{\rho}_{i'\to k}(k)], \text{ for } i = k \text{ and } c_k = k\,; \\ \prod_{i':i'\neq i} \bar{\rho}_{i'\to k} \cdot \prod_{i':i'\neq i} 1, \text{ for } i = k \text{ and } c_k \neq k\,; \\ \prod_{i':i'\neq i} \bar{\rho}_{i'\to k} \cdot \tilde{\rho}_{k\to k}(k) \cdot \prod_{i':i'\notin\{i,k\}} [1 + \tilde{\rho}_{i'\to k}(k)], \text{ for } i \neq k \text{ and } c_i = k\,; \\ \prod_{i':i'\neq i} \bar{\rho}_{i'\to k} \cdot \prod_{i':i'\neq i} 1 + \prod_{i':i'\neq i} \bar{\rho}_{i'\to k} \cdot \tilde{\rho}_{k\to k}(k) \cdot \prod_{i':i'\notin\{i,k\}} [1 + \tilde{\rho}_{i'\to k}(k)], \text{ for } i \neq k \text{ and } c_i \neq k\,. \end{cases} \tag{3.7}$$

Next, solve for $\tilde{\rho}_{i\to k}(c_i = k) = \rho_{i\to k}(c_i = k)/\bar{\rho}_{i\to k}$ and $\tilde{\alpha}_{i\leftarrow k}(c_i = k) = \alpha_{i\leftarrow k}(c_i = k)/\bar{\alpha}_{i\leftarrow k}$ to obtain simple update equations where the $\bar{\rho}$ and $\bar{\alpha}$ terms cancel:

$$\tilde{\rho}_{i\to k}(c_i{=}k) = \frac{\rho_{i\to k}(c_i{=}k)}{\bar{\rho}_{i\to k}} = \frac{\rho_{i\to k}(k)}{\sum_{j:j\neq k} \rho_{i\to k}(j)} = \frac{e^{s(i,k)} \cdot \cancel{\prod_{k':k'\neq k} \bar{\alpha}_{i\leftarrow k'}}}{\sum_{j:j\neq k} [e^{s(i,j)} \cdot \tilde{\alpha}_{i\leftarrow j}(j)] \cdot \cancel{\prod_{k':k'\neq k} \bar{\alpha}_{i\leftarrow k'}}} \tag{3.8}$$

and

$$\begin{aligned} \tilde{\alpha}_{i\leftarrow k}(c_i{=}k) = \;& \alpha_{i\leftarrow k}(c_i{=}k)/\bar{\alpha}_{i\leftarrow k} = \alpha_{i\leftarrow k}(c_i{=}k)/\alpha_{i\leftarrow k}(j{:}\, j{\neq}k) \\ = \;& \begin{cases} \frac{\cancel{\prod_{i':i'\neq i} \bar{\rho}_{i'\to k}} \cdot \prod_{i':i'\neq i} [1+\tilde{\rho}_{i'\to k}(k)]}{\cancel{\prod_{i':i'\neq i} \bar{\rho}_{i'\to k}}}, \text{ for } k = i\,; \\ \frac{\cancel{\prod_{i':i'\neq i} \bar{\rho}_{i'\to k}} \cdot \tilde{\rho}_{k\to k}(k) \cdot \prod_{i':i'\notin\{i,k\}} [1+\tilde{\rho}_{i'\to k}(k)]}{\cancel{\prod_{i':i'\neq i} \bar{\rho}_{i'\to k}} + \cancel{\prod_{i':i'\neq i} \bar{\rho}_{i'\to k}} \cdot \tilde{\rho}_{k\to k}(k) \cdot \prod_{i':i'\notin\{i,k\}} [1+\tilde{\rho}_{i'\to k}(k)]}, \text{ for } k \neq i\,. \end{cases} \end{aligned} \tag{3.9}$$

$\mathcal{O}(N^3)$ scalar message updates

Noting that $\tilde{\rho}_{i\to k}(c_i)$ and $\tilde{\alpha}_{i\leftarrow k}(c_i)$ for $c_i \neq k$ are not used in the updates (specifically, because $\tilde{\alpha}_{i\leftarrow k}(c_i{\neq}k) = 1$), messages can be considered to be scalar instead of N-ary vectors. Working in the log-domain for numerical range reasons, scalar variable-to-function messages are defined as $e^{r(i,k)} = \tilde{\rho}_{i\to k}(k)$ and scalar function-to-variable messages as $e^{a(i,k)} = \tilde{\alpha}_{i\leftarrow k}(k)$

$$e^{r(i,k)} = \frac{e^{s(i,k)}}{\sum\limits_{k':k'\neq k}\left[e^{s(i,k')}e^{a(i,k')}\right]} \text{ and } e^{a(i,k)} = \begin{cases} \prod\limits_{i':i'\neq i}\left[1+e^{r(i',k)}\right], \text{ for } k=i\,; \\ \left(e^{-r(k,k)}\cdot\prod\limits_{i':i'\notin\{i,k\}}\left[1+e^{r(i',k)}\right]^{-1}+1\right)^{-1}, \text{ for } k\neq i\,. \end{cases} \tag{3.10}$$

Message $r(i,k)$ is referred to as the "responsibility" sent from data point i to candidate exemplar point k, reflecting the accumulated evidence for how well-suited point k is to serve as the exemplar for point i, taking into account other potential exemplars for point i. The "availability" $a(i,k)$, sent from candidate exemplar point k to data point i, reflects the accumulated evidence for how appropriate it would be for point i to choose point k as its exemplar, taking into account the support from others that point k should be an exemplar. All data points can be considered to be either cluster members or candidate exemplars, depending on whether they are sending or receiving availability or responsibility messages.

To estimate the value of a variable c_i after any iteration, multiply (fuse) together all incoming messages to variable node c_i and use the value $\hat{c}_i$ that maximizes the product:

$$\begin{aligned} \hat{c}_i &= \operatorname{argmax}_j\left[e^{s(i,j)}\cdot\textstyle\prod_{k=1}^{N}\alpha_{i\leftarrow k}(j)\right] = \operatorname{argmax}_j\left[e^{s(i,j)}\cdot\prod_{k=1}^{N}\bar{\alpha}_{i\leftarrow k}\cdot\prod_{k=1}^{N}\tilde{\alpha}_{i\leftarrow k}(j)\right] \\ &= \operatorname{argmax}_j\left[e^{s(i,j)}\cdot e^{a(i,j)}\right] = \operatorname{argmax}_j\left[s(i,j)+a(i,j)\right]\,. \end{aligned} \tag{3.11}$$

An alternative form that includes availabilities and responsibilities but not input similarities can be obtained by including an additional term inside the $\operatorname{argmax}[\,\cdot\,]$ that leaves the result unchanged as follows:

$$\hat{c}_i = \underset{k}{\operatorname{argmax}} \left[e^{a(i,k)} \cdot \overbrace{e^{s(i,k)} \Big/ \sum\nolimits_{k':k' \neq k} \left(e^{s(i,k')} e^{a(i,k')} \right)}^{e^{r(i,k)} \text{ from (3.10)}} \right] = \underset{k}{\operatorname{argmax}} \left[e^{a(i,k)} \cdot e^{r(i,k)} \right] \tag{3.12}$$
$$= \operatorname{argmax}_k \left[a(i,k) + r(i,k) \right] .$$

$\mathcal{O}(N^2)$ scalar message updates

Each iteration involves computing N^2 availability and responsibility messages (from all N points to all N points), and each message expression in equation (3.10) involves $\mathcal{O}(N)$ binary operations which yields an $\mathcal{O}(N^3)$ algorithm. If intermediate expressions $R(i) = \sum_{k'=1}^{N} e^{s(i,k')+a(i,k')}$ and $A(k) = \prod_{i'=1}^{N} \left[1 + e^{r(i',k)} \right]$ are defined—which can be calculated at the start of each iteration in $\mathcal{O}(N^2)$ time—each message can be computed in $\mathcal{O}(1)$ time as follows:

$$e^{r(i,k)} = \frac{e^{s(i,k)}}{R(i) - e^{s(i,k)+a(i,k)}} \text{ and } e^{a(i,k)} = \begin{cases} \frac{A(k)}{1+e^{r(i,k)}}, & \text{for } k = i \text{ ;} \\ \left(1 + e^{-r(k,k)} . \frac{\left[1+e^{r(i,k)}\right] \cdot \left[1+e^{r(k,k)}\right]}{A(k)} \right)^{-1}, & \text{for } k \neq i \text{ .} \end{cases}$$

which makes an entire iteration possible in $\mathcal{O}(N^2)$ space and time (using a parallel message-passing schedule).

In practice, this approach leads to numerical instability due to limited machine precision when correcting $R(i) = \sum_{k'} e^{s(i,k')+a(i,k')}$ into $\sum_{k':k' \neq k} e^{s(i,k')+a(i,k')}$ by subtracting $e^{s(i,k)+a(i,k)}$. To overcome this, it is necessary to store cumulative sums and manipulate expressions to compute $R(k)$ and $A(k)$ accordingly.

3.2 Max-Product Affinity Propagation

Applying the max-product algorithm to the factor graph in Figure 3.1 overcomes some of the numerical precision difficulties described previously and also produces clustering results that

are invariant not only to similarities with arbitrary additive constants[1], but also to multiplicative constants (the 'units' similarities are expressed in). The variable-to-function messages are unchanged from sum-product, *i.e.* $\rho_{i\to k}(c_i) = e^{s(i,c_i)} \cdot \prod_{k':k'\neq k} \alpha_{i\leftarrow k'}(c_i)$ but the sums in the function-to-variable messages are replaced by max operators:

$$\alpha_{i\leftarrow k}(c_i) = \overbrace{\max_{j_1}\max_{j_2}\cdots\max_{j_{i-1}}\max_{j_{i+1}}\cdots\max_{j_N}\left[f_k(j_1,j_2,\ldots,j_{i-1},c_i,j_{i+1},\ldots,j_N)\cdot\prod_{i':i'\neq i}\rho_{i'\to k}(j_{i'})\right]}^{\text{best possible configuration satisfying } f_k \text{ given } c_i}$$

$$= \begin{cases} \overbrace{\prod_{i':i'\neq i}\max_j \rho_{i'\to k}(j)}^{\text{best configuration with or without cluster } k}, & \text{for } i=k \text{ and } c_k=k\,; \\ \overbrace{\prod_{i':i'\neq i}\max_{j:j\neq k} \rho_{i'\to k}(j)}^{\text{best configuration without cluster } k}, & \text{for } i=k \text{ and } c_k\neq k\,; \\ \overbrace{\rho_{k\to k}(k)}^{k \text{ is an exemplar}} \cdot \overbrace{\prod_{i':i'\notin\{i,k\}}\max_j \rho_{i'\to k}(j)}^{\text{best configuration, with or without cluster } k}, & \text{for } i\neq k \text{ and } c_i=k\,; \\ \max\left[\overbrace{\max_{j:j\neq k}\rho_{k\to k}(j)\cdot\prod_{i':i'\notin\{i,k\}}\max_{j:j\neq k}\rho_{i'\to k}(j)}^{\text{best configuration with no cluster } k}, \overbrace{\rho_{k\to k}(k)\cdot\prod_{i':i'\notin\{i,k\}}\max_j \rho_{i'\to k}(j)}^{\text{best configuration with a cluster } k}\right], & \text{for } i\neq k \text{ and } c_i\neq k\,. \end{cases}$$

As with sum-product, representing these N-vector messages as the product of constant and variable components changes the messages to:

$$\rho_{i\to k}(c_i) = e^{s(i,c_i)} \cdot \prod_{k':k'\neq k} \bar{\alpha}_{i\leftarrow k'} \cdot \prod_{k':k'\neq k} \tilde{\alpha}_{i\leftarrow k'}(c_i) \quad \cdot$$

[1]For sum-product affinity propagation, similarities appear only in responsibility equations and solutions are invariant to additive constants *i.e.*, $s'(i,k) = s(i,k) + \text{constant}$ because $e^{r(i,k)} = e^{s(i,k)} / \sum_{k':k'\neq k}\left[e^{s(i,k')}e^{a(i,k')}\right] = e^{s(i,k)+\text{constant}} / \sum_{k':k'\neq k}\left[e^{s(i,k')+\text{constant}}e^{a(i,k')}\right]$

and

$$\alpha_{i\leftarrow k}(c_i)=\begin{cases}\prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\prod_{i':i'\neq i}\max_j\tilde{\rho}_{i'\to k}(j),\ \text{for } i=k \text{ and } c_k=k\ ;\\ \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\prod_{i':i'\neq i}\max_{j:j\neq k}\tilde{\rho}_{i'\to k}(j),\ \text{for } i=k \text{ and } c_k\neq k\ ;\\ \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\tilde{\rho}_{k\to k}(k)\cdot\prod_{i':i'\notin\{i,k\}}\max_j\tilde{\rho}_{i'\to k}(j),\ \text{for } i\neq k \text{ and } c_i=k\ ;\\ \max\left[\begin{array}{c}\prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\prod_{i':i'\neq i}\max_{j:j\neq k}\tilde{\rho}_{i'\to k}(j),\\ \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\tilde{\rho}_{k\to k}(k)\cdot\prod_{i':i'\notin\{i,k\}}\max_j\tilde{\rho}_{i'\to k}(j)\end{array}\right],\ \text{for } c_i\neq k\neq i\ .\end{cases}$$

Let $\bar{\rho}_{i\to k}=\max_{j:j\neq k}\rho_{i\to k}(j)$ so $\max_{j:j\neq k}\tilde{\rho}_{i\to k}(j)=0$ and $\max_j\tilde{\rho}_{i\to k}(j)=\max[0,\tilde{\rho}_{i\to k}(k)]$. For availabilities, let $\bar{\alpha}_{i\leftarrow k}=\alpha_{i\leftarrow k}(c_i:\ c_i\neq k)$ which makes $\tilde{\alpha}_{i\leftarrow k}(c_i)=1$ for all $c_i\neq k$ and $\prod_{k':k'\neq k}\tilde{\alpha}_{i\leftarrow k'}(c_i)=\tilde{\alpha}_{i\leftarrow c_i}(c_i)$ for $c_i\neq k$ and $\prod_{k':k'\neq k}\tilde{\alpha}_{i\leftarrow k'}(k)=1$ otherwise. This leads to simplification:

$$\rho_{i\to k}(c_i)=\begin{cases}e^{s(i,k)}\cdot\prod_{k':k'\neq k}\bar{\alpha}_{i\leftarrow k'},\ \text{for } c_i=k\ ;\\ e^{s(i,c_i)}\cdot\tilde{\alpha}_{i\leftarrow c_i}(c_i)\cdot\prod_{k':k'\neq k}\bar{\alpha}_{i\leftarrow k'},\ \text{for } c_i\neq k\ ,\end{cases}$$

and

$$\alpha_{i\leftarrow k}(c_i)=\begin{cases}\prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\prod_{i':i'\neq i}\max\left[1,\tilde{\rho}_{i'\to k}(k)\right],\ \text{for } i=k \text{ and } c_k=k\ ;\\ \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\prod_{i':i'\neq i}1,\ \text{for } i=k \text{ and } c_k\neq k\ ;\\ \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\tilde{\rho}_{k\to k}(k)\cdot\prod_{i':i'\notin\{i,k\}}\max\left[1,\tilde{\rho}_{i'\to k}(k)\right],\ \text{for } i\neq k \text{ and } c_i=k\ ;\\ \max\left[\prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\prod_{i':i'\neq i}1,\ \prod_{i':i'\neq i}\bar{\rho}_{i'\to k}\cdot\tilde{\rho}_{k\to k}(k)\cdot\prod_{i':i'\notin\{i,k\}}\max\left[1,\tilde{\rho}_{i'\to k}(k)\right]\right],\ \text{for } c_i\neq k\neq i\ .\end{cases}$$

Solving for $\tilde{\rho}_{i\to k}(c_i=k)=\rho_{i\to k}(c_i=k)/\bar{\rho}_{i\to k}$ and $\tilde{\alpha}_{i\leftarrow k}(c_i=k)=\alpha_{i\leftarrow k}(c_i=k)/\bar{\alpha}_{i\leftarrow k}$ yields further cancelation:

$$\tilde{\rho}_{i\to k}(c_i=k)=\frac{\rho_{i\to k}(k)}{\bar{\rho}_{i\to k}}=\frac{\rho_{i\to k}(k)}{\sum_{j:j\neq k}\rho_{i\to k}(j)}=\frac{e^{s(i,k)}\cdot\cancel{\prod_{k':k'\neq k}\bar{\alpha}_{i\leftarrow k'}}}{\sum_{j:j\neq k}[e^{s(i,j)}\cdot\tilde{\alpha}_{i\leftarrow j}(j)]\cdot\cancel{\prod_{k':k'\neq k}\bar{\alpha}_{i\leftarrow k'}}}\quad(3.13)$$

and

$$\tilde{\alpha}_{i\leftarrow k}(c_i{=}k) = \alpha_{i\leftarrow k}(c_i{=}k)/\bar{\alpha}_{i\leftarrow k} = \alpha_{i\leftarrow k}(c_i{=}k)/\alpha_{i\leftarrow k}(j{:}\,j{\neq}k)$$
$$= \begin{cases} \dfrac{\cancel{\prod_{i',i'\neq i}\tilde{\rho}_{i'\rightarrow k}}\cdot\prod_{i':i'\neq i}\max[1,\,\tilde{\rho}_{i'\rightarrow k}(k)]}{\cancel{\prod_{i',i'\neq i}\tilde{\rho}_{i'\rightarrow k}}}, & \text{for } k{=}i\ ; \\ \dfrac{\cancel{\prod_{i',i'\neq i}\tilde{\rho}_{i'\rightarrow k}}\cdot\tilde{\rho}_{k\rightarrow k}(k)\cdot\prod_{i':i'\notin\{i,k\}}\max[1,\,\tilde{\rho}_{i'\rightarrow k}(k)]}{\max\left[\cancel{\prod_{i',i'\neq i}\tilde{\rho}_{i'\rightarrow k}},\ \cancel{\prod_{i',i'\neq i}\tilde{\rho}_{i'\rightarrow k}}\cdot\tilde{\rho}_{k\rightarrow k}(k)\cdot\prod_{i',i'\notin\{i,k\}}\max[1,\,\tilde{\rho}_{i'\rightarrow k}(k)]\right]}, & \text{for } k{\neq}i\ . \end{cases} \tag{3.14}$$

Finally, messages are defined as $r(i,k) = \log\tilde{\rho}_{i\rightarrow k}(k)$ and $a(i,k) = \log\tilde{\alpha}_{i\leftarrow k}(k)$ to obtain simple update equations shown in (3.15), where the final expression for $a(i,k)$ is due to $\log\frac{x}{\max(1,x)} = \min(0,\log x)$ and the estimate $\hat{c}_i$ is as before (3.12).

AFFINITY PROPAGATION

INPUT: $\{s(i,j)\}_{i,j\in\{1,\ldots,N\}}$ (data similarities and preferences)

INITIALIZE: set 'availabilities' to zero *i.e.* $\forall i,k\colon a(i,k){=}0$

REPEAT: responsibility and availability updates until convergence

$$\forall i,k\colon\ r(i,k) = s(i,k) - \max_{k':k'\neq k}\left[s(i,k') + a(i,k')\right]$$
$$\forall i,k\colon\ a(i,k) = \begin{cases} \sum_{i':i'\neq i}\max[0,\,r(i',k)], & \text{for } k{=}i \\ \min\Big[0,\,r(k,k){+}\sum_{i':i'\notin\{i,k\}}\max[0,\,r(i',k)]\Big], & \text{for } k{\neq}i \end{cases} \tag{3.15}$$

OUTPUT: cluster assignments $\hat{\mathbf{c}}{=}(\hat{c}_1,\ldots,\hat{c}_N)$, $\hat{c}_i{=}\operatorname{argmax}_k\left[a(i,k){+}r(i,k)\right]$

Note: $\hat{c}$ may violate $\{f_k\}$ constraints, so initialize k-medoids with $\hat{c}$ and run to convergence for a coherent solution.

The simplicity and effectiveness of these update equations have made it the standard incarnation of affinity propagation since its initial 2007 publication in *Science* [38]. All experiments in Chapters 4–5 use this form of the algorithm.

It is interesting to note that the greedy k-medoids clustering algorithm can be rewritten to use responsibilities and thus more closely resemble affinity propagation:

$$\forall i{\in}\{1,\ldots,N\}, k{\in}\mathcal{K}\colon r(i,k){=}s(i,k){-}\max_{k':k'\neq k}s(i,k')\ \text{and}\ c_i{=}\operatorname*{argmax}_{k\in\mathcal{K}}r(i,k)$$
$$\forall k{\in}\mathcal{K}\colon k{\leftarrow}\operatorname*{argmax}_{j:c_j=k}\ \sum_{i:c_i=k}s(i,j)$$

The major difference is that in computing responsibilities, there are no availabilities to modulate similarities and hard decisions are made. Instead, the refinements are performed on a fixed set of exemplars, $\mathcal{K}$.

3.2.1 Max-Product vs. Sum-Product Affinity Propagation

Replacing the summations with max operators eliminates the numerical precision issues from sum-product affinity propagation; finding $\max_{k':k'\neq k}$ can be done cleanly in linear time by holding the cumulative largest and next-largest array elements in memory. Another advantage is that additive constants in the similarities are canceled out in the responsibility update equation and multiplicative constants, *i.e.,* $s'(i,k) = s(i,k) \cdot \text{constant}$ scale both the responsibilities and availabilities but leaves cluster assignments $\hat{c}_i$ unchanged (as long as numerical issues are kept in check).

3.2.2 Dynamics of Affinity Propagation

Availabilities are initialized to zero for the first iteration, so $r(i,k)$ is set to the input similarity between point i and point k minus the largest competing similarity between point i and other potential exemplars, *i.e.* $r(i,k) = s(i,k) - \max_{k':k'\neq k} s(i,k')$. This competitive update does not take into account how many other points favor each candidate exemplar, though in later iterations when some points are effectively assigned to other exemplars, their availabilities will drop below zero. This decreases the effective value of the corresponding similarity to which it is added and gradually withdraws them from the competition to be an exemplar. For $i = k$, the "self-responsibility" $r(k,k)$ is set to the input preference, $s(k,k)$, minus the largest of the similarities between point i and all other candidate exemplars. This reflects accumulated evidence that point k is an exemplar, based on its input preference tempered by how ill-suited it is to be assigned to another cluster's exemplar.

The responsibility update lets all candidate exemplars compete for ownership of a data

point so accordingly, availability updates gather evidence from data points indicating whether each candidate exemplar would make a good exemplar. The availability of point i to serve as an exemplar for point k, $a(i,k)$, is set to the self-responsibility $r(k,k)$ plus the sum of positive responsibilities candidate exemplar k receives from other points (not including i, the message destination). Only the positive portions of incoming responsibilities are added ($\max[0,\cdot]$ term), because it is only necessary for a good exemplar to explain some data points well (those with positive responsibilities) regardless of how poorly it explains other extremely-dissimilar data points (those with negative responsibilities). If the self-responsibility $r(k,k)$ is negative—indicating that point k is currently better suited as belonging to a cluster rather than being an exemplar itself—the availability of point k to serve as an exemplar could be increased if some other points have positive responsibilities toward point k. To limit the influence of unjustifiably-strong incoming positive responsibilities—which could arise from a pair of extremely-similar 'twin' data points—the total sum is thresholded so that it cannot rise above zero due to the $\min[0,\cdot]$ operation. The "self-availability" $a(k,k)$ is updated by adding positive components of incoming responsibilities but without the final threshold.

The message-passing dynamics of affinity propagation applied to a toy dataset of 27 two-dimensional points are shown in Figure 3.2.

3.2.3 Preferences for Affinity Propagation

A global shared preference p, where $\forall i \in \{1,\ldots,N\}: s(i,i) = p$, is often used as a control knob to govern the number of clusters found by affinity propagation. As shown in Figure 3.3, lower values of p penalize the use of data points as exemplars more heavily and lead to fewer clusters, while the effect with higher preference values is the opposite[2]. This is an advantage in that the

[2]Further intuition is obtained by imagining an algorithm that has currently identified K exemplars and is considering labeling another data point as an additional exemplar. By switching away from their old exemplars and choosing the new exemplar, some data points will cause an increase in the net similarity. However, creating the additional exemplar will cause its preference to be added to the net similarity and, since it will no longer be assigned to one of the old exemplars, the net similarity will decrease by that similarity value. Since the preference is usually low compared to similarities, the difference is usually large and the additional exemplar will be created only if the total gain in similarity exceeds this negative value. This intuition shows that the preferences have

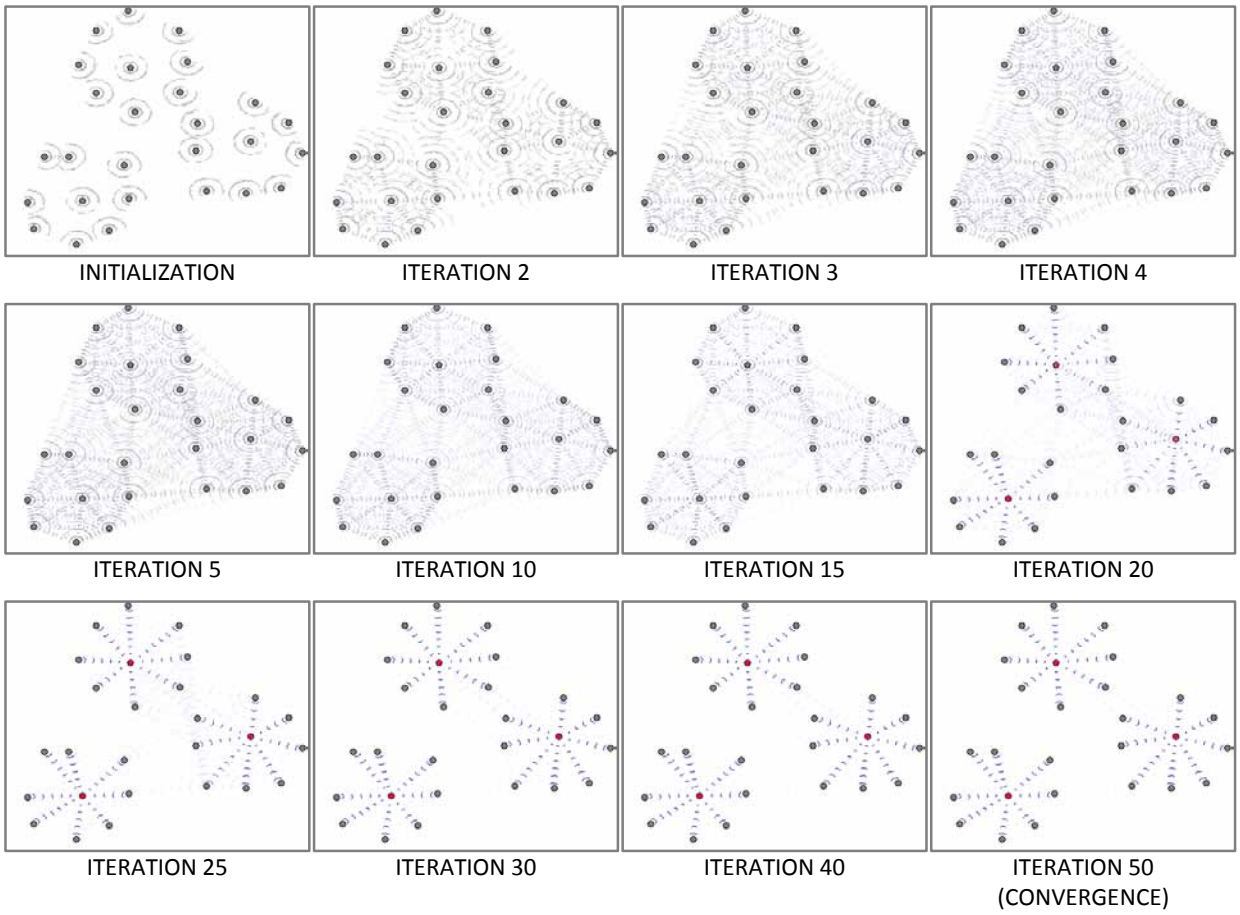


Figure 3.2: The dynamics of affinity propagation are shown for a hand-crafted dataset of $N = 27$ two-dimensional points. Negative squared Euclidean distance is used as a measure of similarity and a global preference p is set to the minimum similarity which yields the natural-looking three clusters (see Figure 3.3 for further analysis). Messages are depicted as chevrons pointed towards candidate exemplars; the blue intensity is in proportion to the responsibility plus availability, $r(i,k)+a(i,k)$, which is used in clustering decisions (see equation (3.15)). Note that messages are weak and uniform in all directions for early iterations, but clusters begin to emerge by iteration 15, with corner points far more certain of their exemplar than central points. The algorithm is fully-converged by iteration 50, though faint probabilities of alternate configurations remain visible near the center of the plot.

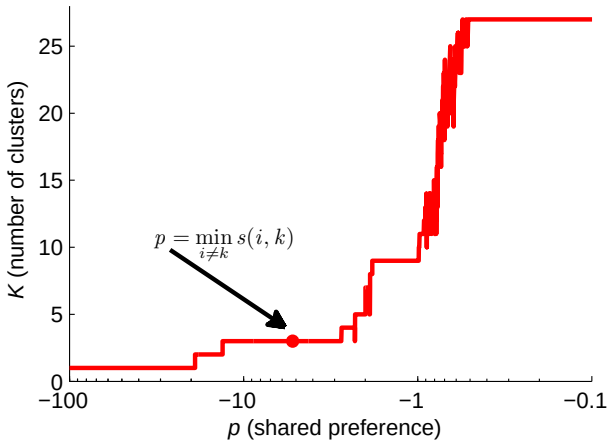


Figure 3.3: A shared preference value, p, can be used as a control knob to vary the number of clusters, K, found by affinity propagation in the toy example from Figure 3.2. Notice that a relatively wide range of preference values, between roughly $p = -10$ and $p = -3$, lead to the natural-looking three clusters ($p = \min_{i \neq k} s(i,k) \approx -5.2$ from Figure 3.2 is marked) compared to a narrow range leading to an apparently unnatural two clusters. The dynamics tend to become quite unstable and non-monotonic past $K = 10$ clusters, which is sensible given that there are only $N = 27$ data points.

number of exemplars need not be specified beforehand, enabling automatic model selection based on a prior specification of how preferable (*a priori* log-probability) each point is as an exemplar. Note that a relatively wide range of preference values lead to the natural-looking configuration with three clusters as opposed to a narrow range leading to two clusters.

3.2.4 Implementation Details

A MATLAB implementation, `apcluster.m`, is available for download at http://www.psi.toronto.edu., and has been downloaded several hundred times to date. Part of affinity propagation's appeal is the fact that the simple update equations can be easily implemented as shown in the following ∼20 lines of MATLAB source code:

the same 'units' as the similarities, since similarities and preferences are traded when deciding whether or not to create exemplars.

```
01  N=size(S,1); A=zeros(N,N); R=zeros(N,N); % initialize messages
02  S=S+1e-12*randn(N,N)*(max(S(:))-min(S(:))); % remove degeneracies
03  lambda=0.9; % set dampening factor
04  for iter=1:100,
05    Rold=R; % NOW COMPUTE RESPONSIBILITIES
06    AS=A+S; [Y,I]=max(AS,[],2);
07    for i=1:N, AS(i,I(i))=-inf; end; [Y2,I2]=max(AS,[],2);
08    R=S-repmat(Y,[1,N]);
09    for i=1:N,R(i,I(i))=S(i,I(i))-Y2(i); end;
10    R=(1-lambda)*R+lambda*Rold; % dampening responsibilities
11    Aold=A; % NOW COMPUTE AVAILABILITIES
12    Rp=max(R,0); for k=1:N, Rp(k,k)=R(k,k); end;
13    A=repmat(sum(Rp,1),[N,1])-Rp;
14    dA=diag(A); A=min(A,0); for k=1:N, A(k,k)=dA(k); end;
15    A=(1-lambda)*A+lambda*Aold; % dampening availabilities
16  end;
17  E=R+A; % pseudomarginals
18  I=find(diag(E)>0); K=length(I); % indices of exemplars
19  [tmp c]=max(S(:,I),[],2); c(I)=1:K; idx=I(c); % assignments
```

Several implementation details should be noted. First, random noise on the order of machine precision[3] should be added to input similarities (line `02`) in order to remove possible degeneracies. For example, if similarities are symmetric and two data points are isolated from the rest, there may be indecision as to which one of them should be the exemplar and that can lead to oscillations. Another example is if multiple clustering solutions have the same optimal net similarity $\mathcal{S}$—especially common when similarities belong to a finite set *e.g.* range of integers—the algorithm may oscillate between optima. For most datasets, however, this is not necessary and the solution is invariant to the added noise[4].

As described in Section 2.6.1, belief propagation methods converge to exact solutions in a finite number of iterations when the factor graph topology is singly-linked (no cycles). Behavior is generally reported as less-stable and more prone to oscillation as the number of

[3]This can also be seen as randomly flipping the least significant bits of the floating-point mantissa for each input similarity value.

[4]For example, affinity propagation was input the 400-point Olivetti faces dataset (see Section 4.1) with a typical preference of $p = -60$ and over the course of 1000 trials with different random noise additions, there was no consequential variation in the algorithms's behavior (exemplar set at each iteration was identical). On the other hand, randomly-initializing availabilities to non-zero values can lead to slightly better solutions.

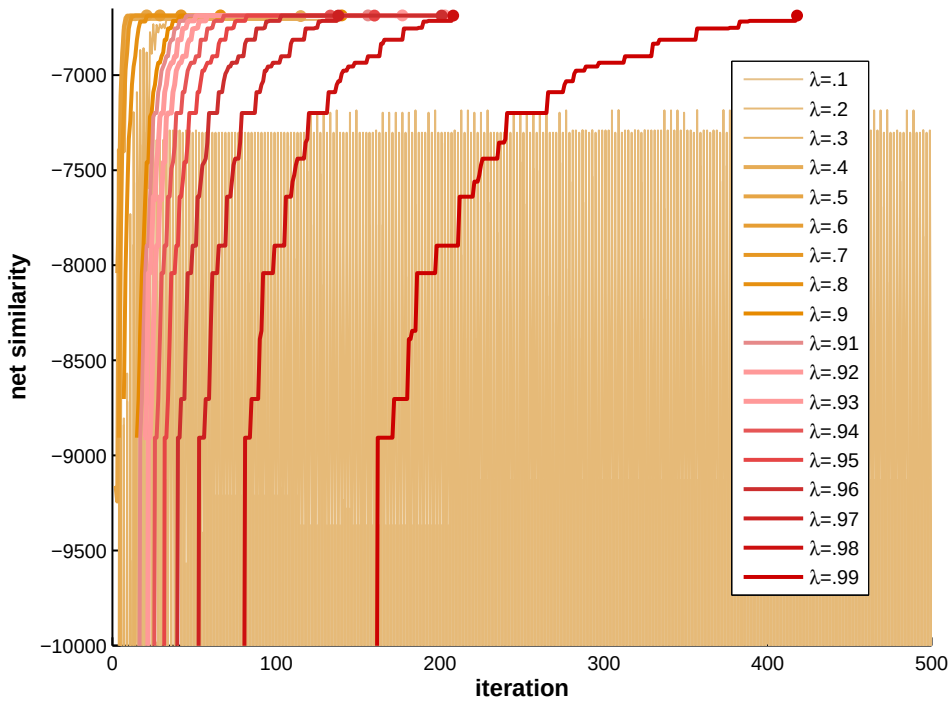


Figure 3.4: The net similarity, $\mathcal{S}$, is shown as a function of iteration for runs of affinity propagation with different dampening factors, λ. The higher values of λ unsurprisingly lead to slower convergence rates but often lead to more stable maximization (see $\lambda = 0.3$ for an example of non-monotonic oscillatory behavior that ultimately converges).

tight cycles increase. Affinity propagation's [potentially] completely-connected factor graph is an extreme case, and thus the implementation containing parallel message updates—where $\forall i,k\colon\ r(i,k)$ is computed and then $\forall i,k\colon\ a(i,k)$—necessitates that messages be dampened, which is done as follows:

$$r(\cdot,\cdot) = \lambda \cdot r^{\text{old}}(\cdot,\cdot) + (1-\lambda) \cdot r^{\text{new}}(\cdot,\cdot) \text{ and } a(\cdot,\cdot) = \lambda \cdot a^{\text{old}}(\cdot,\cdot) + (1-\lambda) \cdot a^{\text{new}}(\cdot,\cdot)$$

setting the dampening factor λ to 0.9 has been sufficient in almost all cases to ensure convergence. For the 400 Olivetti faces dataset examined in Section 4.1, a dampening factor of $\lambda = 0.4$ is sufficient for convergence with no oscillation; this is shown with other values of λ in Figure 3.4.

At any stage in the message-passing procedure, intermediate clustering solutions can be estimated from $\hat{c}_i = \underset{k}{\operatorname{argmax}}\,[a(i,k)+r(i,k)]$. This often produces $\{f_k\}$ constraint-violating solutions which can be resolved by defining the set of exemplars as $\mathcal{K}$ and the set of non-exemplars $\overline{\mathcal{K}} \equiv \{1,2,\ldots,N\}\backslash\mathcal{K}$ such that $\forall k \in \mathcal{K} : \hat{c}_k = k$ and $\forall i \in \overline{\mathcal{K}} : \hat{c}_i \neq i$. Then run a half-iteration of k-medoids to properly assign the non-exemplars, *i.e.* $\forall i \in \overline{\mathcal{K}} : c_i \leftarrow \underset{k\in\mathcal{K}}{\operatorname{argmax}}\, s(i,k)$. Affinity propagation is considered converged if, for some constant `convits` iterations, exemplar set $\mathcal{K}$ remains unchanged. The number of iterations should also be bounded by constants `minits` (in case $\lambda \approx 1$ leading to no exemplars $|\mathcal{K}| = 0$ for many iterations initially while self-availabilities $a(k,k)$ slowly rise) and `maxits` (in case of non-convergence, perhaps due to degeneracies). For final results, performance can be slightly enhanced by running k-medoids to convergence and not just the half-iteration to satisfy constraints.

3.2.5 Sparse Similarities and Affinity Propagation

Affinity propagation is well-suited to take advantage of sparsity in data. When similarities are computed between each data point, the algorithm shown in equation (3.15) is $\mathcal{O}(N^2)$. Some problems are structured in such a way that many data points cannot possibly be represented by many others as exemplars; *i.e.* $\exists i,k : s(i,k) = -\infty$. In this case, $r(i,k)$ is automatically $-\infty$ and $a(i,k)$ is inconsequential because it is overpowered in $\max[\overbrace{s(i,k)}^{-\infty}+a(i,k)]$. For such a sparse[5] dataset with N data points but only $M < N^2$ values of $(i,k) \in \{1,2,\ldots,N\}^2$ where $s(i,k) > -\infty$, only M responsibility and availability messages need to be computed and exchanged. In terms of storage, the sparseness structure can be stored for quick traversal using $2M$ 2- or 4-byte integers (M for each i-value, M for each k-value), and M similarities, responsibilities, and availabilities need to be stored as 4-byte or 8-byte floating-point values. This results in memory requirements between $16 \cdot M$ and $32 \cdot M$ bytes.[6]

[5] The data is only sparse in the sense that a matrix of $\exp(\text{similarities})$ has zero entries; in the log-domain the missing similarities become $-\infty$.

[6] There is no need to store M values of $a^{(\text{old})}$ or $r^{(\text{old})}$, as there is no advantage to dampening messages *en masse*.

3.3 Alternate Factor Graph for Affinity Propagation

An equivalent formulation of affinity propagation can be derived by using N^2 binary variable nodes (or $M<N^2$ for sparse situations) and $2N$ constraint function nodes more in line with the 0–1 integer programming formulations from Section 2.4.1. As before, variables $b_{ik} \in \{0,1\}$ indicate cluster assignments where $b_{ik}=1$ for $i \neq k$ indicates that data point i is in a cluster characterized by exemplar data point k, and $b_{kk}=1$ indicates that point k is an exemplar. Function nodes $\{f_k(b_{1k}, b_{2k}, \ldots, b_{Nk})\}_{k=1}^{N}$ enforce a similar constraint as before (disallowing clusters without an exemplar), and additional function nodes $\{g_i(b_{i1}, b_{i2}, \ldots, b_{iN})\}_{i=1}^{N}$ enforce that all data points are in exactly one cluster (possibly as an exemplar):

$$f_k(b_{1k}, b_{2k}, \ldots, b_{Nk}) = \left[b_{kk} = \max_i b_{ik}\right] = \begin{cases} 0, \text{ if } b_{kk} \neq 1 \text{ but } \exists i\colon b_{ik}=1\,; \\ 1, \text{ otherwise}\,, \end{cases} \tag{3.16}$$

and

$$g_i(b_{i1}, b_{i2}, \ldots, b_{iN}) = \left[\sum\nolimits_{k=1}^{N} b_{ik} = 1\right] = \begin{cases} 0, \text{ if } \sum_{k=1}^{N} b_{ik} \neq 1\,; \\ 1, \text{ otherwise}\,. \end{cases} \tag{3.17}$$

The global function is:

$$\begin{aligned} F(\mathbf{b};\mathbf{s}) &= \prod_{i=1}^{N}\prod_{k=1}^{N} e^{b_{ik}\cdot s(i,k)} \cdot \prod_{k=1}^{N} f_k(b_{1k}, b_{2k}, \ldots, b_{Nk}) \cdot \prod_{i=1}^{N} g_i(b_{i1}, b_{i2}, \ldots, b_{iN})\,; \\ &= e^{\sum_{i=1}^{N}\sum_{k=1}^{N} b_{ik}\cdot s(i,k) + \sum_{k=1}^{N} \log f_k(b_{1k}, b_{2k}, \ldots, b_{Nk}) + \sum_{i=1}^{N} \log g_i(b_{i1}, b_{i2}, \ldots, b_{iN})}\,. \end{aligned}$$

Four types of messages are passed between variable nodes and function nodes as shown in Figure 3.5. Messages outgoing from variable nodes are the element-wise product of all other incoming messages as shown here:

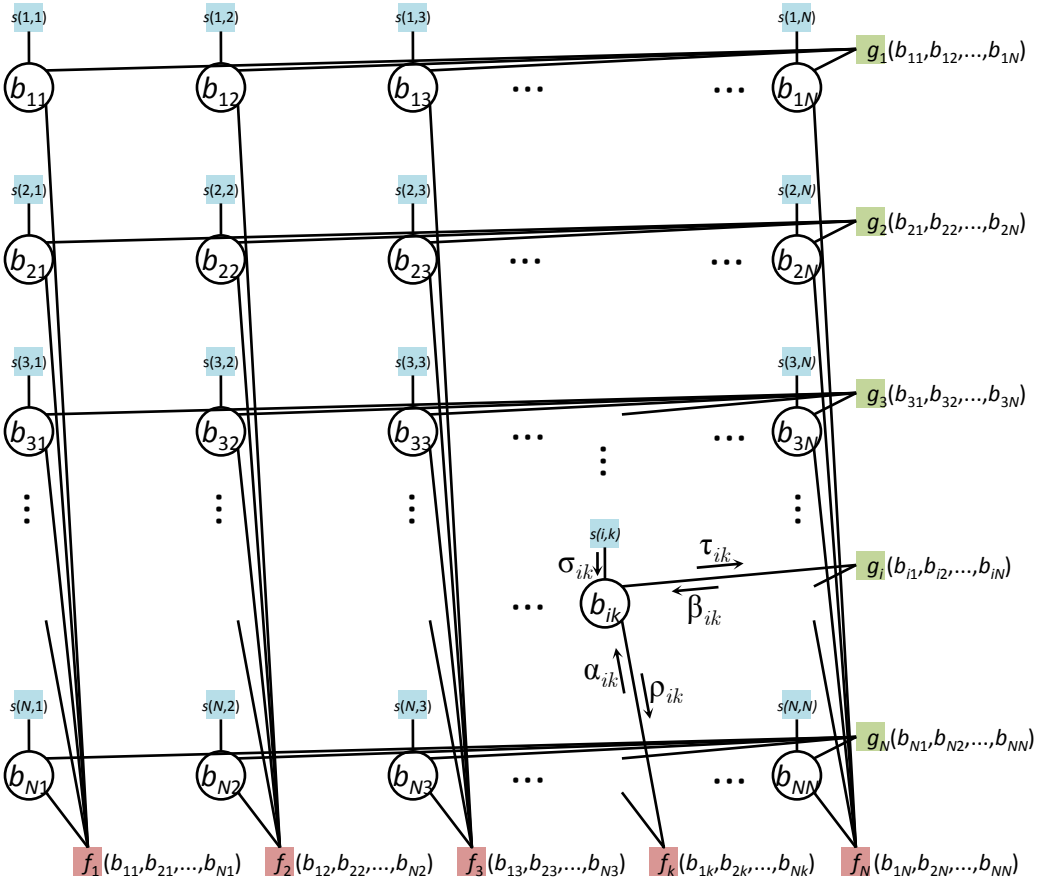


Figure 3.5: An alternate grid-topology factor graph can be used to derive affinity propagation. N-state variable c_i is divided into N binary variables $b_{i1}, b_{i2}, \ldots, b_{iN}$ under additional constraint g_i that $\sum_{k=1}^{N} b_{ik} = 1$.

$$\tau_{ik}(b_{ik}) = \begin{cases} \sigma_{ik}(0)\cdot\alpha_{ik}(0), \text{ for } b_{ik}=0\,; \\ \sigma_{ik}(1)\cdot\alpha_{ik}(1), \text{ for } b_{ik}=1\,; \end{cases} \quad \text{and } \rho_{ik}(b_{ik}) = \begin{cases} \sigma_{ik}(0)\cdot\beta_{ik}(0), \text{ for } b_{ik}=0\,; \\ \sigma_{ik}(1)\cdot\beta_{ik}(1), \text{ for } b_{ik}=1\,. \end{cases} \tag{3.18}$$

These binary messages can be 'normalized' and instead represented by a scalar ratio $\tau_{ik} = \frac{\tau_{ik}(1)}{\tau_{ik}(0)} = \frac{\sigma_{ik}(1)\cdot\alpha_{ik}(1)}{\sigma_{ik}(0)\cdot\alpha_{ik}(0)} = \sigma_{ik}\alpha_{ik}$ and $\rho_{ik} = \frac{\rho_{ik}(1)}{\rho_{ik}(0)} = \frac{\sigma_{ik}(1)\cdot\beta_{ik}(1)}{\sigma_{ik}(0)\cdot\beta_{ik}(0)} = \sigma_{ik}\beta_{ik}$. Likewise, messages outgoing from the similarity function nodes can be interpreted as $\sigma_{ik}(b_{ik}) = e^{b_{ik}s(i,k)}$ so the normalized ratio would be $\sigma_{ik} = \frac{\sigma_{ik}(1)}{\sigma_{ik}(0)} = e^{b_{ik}s(i,k)}$. Messages outgoing from the g-constraint nodes differ for sum-product and max-product message-passing; they are as follows:

$$\begin{aligned} \beta_{ik}^{\mathrm{SP}}(b_{ik}) &= \sum_{j_1}\sum_{j_2}\cdots\sum_{j_{k-1}}\sum_{j_{k+1}}\cdots\sum_{j_N}\left[g_i(j_1,j_2,\ldots,j_{k-1},b_{ik},j_{k+1},\ldots,j_N)\cdot\textstyle\prod_{k':k'\neq k}\tau_{ik'}(j_{k'})\right] \\ &= \begin{cases} \sum_{k':k'\neq k}\left[\tau_{ik'}(1)\cdot\prod_{k'':k''\notin\{k,k'\}}\tau_{ik''}(0)\right], \text{ for } b_{ik}=0\,; \\ \prod_{k':k'\neq k}\tau_{ik'}(0), \text{ for } b_{ik}=1\,; \end{cases} \\ \beta_{ik}^{\mathrm{MP}}(b_{ik}) &= \max_{j_1}\max_{j_2}\cdots\max_{j_{k-1}}\max_{j_{k+1}}\cdots\max_{j_N}\left[g_i(j_1,j_2,\ldots,j_{k-1},b_{ik},j_{k+1},\ldots,j_N)\cdot\prod_{k':k'\neq k}\tau_{ik'}(j_{k'})\right] \\ &= \begin{cases} \max_{k':k'\neq k}\left[\tau_{ik'}(1)\cdot\prod_{k'':k''\notin\{k,k'\}}\tau_{ik''}(0)\right], \text{ for } b_{ik}=0\,; \\ \prod_{k':k'\neq k}\tau_{ik'}(0), \text{ for } b_{ik}=1\,. \end{cases} \end{aligned}$$

These can be expressed in terms of the scalar ratios, so $\tau_{ik}(0)$ is replaced with 1 and $\tau_{ik}(1)$ with τ_{ik} to yield simple sum- and max-product update equations $\beta_{ik}^{\mathrm{SP}} = \frac{\beta_{ik}^{\mathrm{SP}}(1)}{\beta_{ik}^{\mathrm{SP}}(0)} = 1/\sum_{k':k'\neq k}\tau_{ik'}$ and $\beta_{ik}^{\mathrm{MP}} = \frac{\beta_{ik}^{\mathrm{MP}}(1)}{\beta_{ik}^{\mathrm{MP}}(0)} = 1/\max_{k':k'\neq k}\tau_{ik'}$.

Messages outgoing from the other f-constraint function nodes are somewhat more complicated:

$$\alpha_{ik}^{\mathrm{SP}}(b_{ik}) = \sum_{j_1}\sum_{j_2}\cdots\sum_{j_{i-1}}\sum_{j_{i+1}}\cdots\sum_{j_N}\left[f_i(j_1,j_2,\ldots,j_{i-1},b_{ik},j_{i+1},\ldots,j_N)\cdot\textstyle\prod_{i':i'\neq i}\rho_{i'k}(j_{i'})\right]$$

$$= \begin{cases} \prod_{i':i'\neq i}\rho_{i'k}(0) = 1, \text{ for } i=k \text{ and } b_{ik}=0\,; \\ \prod_{i':i'\neq i}[\rho_{i'k}(0)+\rho_{i'k}(1)] = \prod_{i':i'\neq i}[1+\rho_{i'k}], \text{ for } i=k \text{ and } b_{ik}=1\,; \\ \rho_{kk}(1)\prod_{i':i'\notin\{i,k\}}[\rho_{i'k}(0)+\rho_{i'k}(1)] + \rho_{kk}(0)\prod_{i':i'\notin\{i,k\}}\rho_{i'k}(0) = \rho_{kk}\cdot\prod_{i':i'\notin\{i,k\}}[1+\rho_{i'k}]+1, \text{ for } i\neq k,\ b_{ik}=0\,; \\ \rho_{kk}(1)\cdot\prod_{i':i'\notin\{i,k\}}[\rho_{i'k}(0)+\rho_{i'k}(1)] = \rho_{kk}\cdot\prod_{i':i'\notin\{i,k\}}[1+\rho_{i'k}], \text{ for } i\neq k \text{ and } b_{ik}=1\,, \end{cases}$$

$$\text{so } \alpha_{ik}^{\mathrm{SP}} = \frac{\alpha_{ik}^{\mathrm{SP}}(1)}{\alpha_{ik}^{\mathrm{SP}}(0)} = \begin{cases} \prod_{i':i'\neq i}[1+\rho_{i'k}], \text{ for } i=k\,; \\ \left(1+\rho_{kk}^{-1}\cdot\prod_{i':i'\notin\{i,k\}}[1+\rho_{i'k}]\right)^{-1}, \text{ for } i\neq k\,, \end{cases}$$

and for the max-product algorithm,

$$\alpha_{ik}^{\mathrm{MP}}(b_{ik}) = \max_{j_1}\max_{j_2}\cdots\max_{j_{i-1}}\max_{j_{i+1}}\cdots\max_{j_N}\left[f_i(j_1,j_2,\ldots,j_{i-1},b_{ik},j_{i+1},\ldots,j_N)\cdot\prod_{i':i'\neq i}\rho_{i'k}(j_{i'})\right]$$

$$= \begin{cases} \prod_{i':i'\neq i}\rho_{i'k}(0) = 1, \text{ for } i=k \text{ and } b_{ik}=0\,; \\ \prod_{i':i'\neq i}\max[\rho_{i'k}(0),\rho_{i'k}(1)] = \prod_{i':i'\neq i}\max[1,\rho_{i'k}], \text{ for } i=k \text{ and } b_{ik}=1\,; \\ \max\left(\rho_{kk}(1)\cdot\prod_{i':i'\notin\{i,k\}}\max[\rho_{i'k}(0),\rho_{i'k}(1)],\ \rho_{kk}(0)\cdot\prod_{i':i'\neq i}\rho_{i'k}(0)\right) \\ \qquad = \max\left(1,\ \rho_{kk}\cdot\prod_{i':i'\notin\{i,k\}}\max[1,\rho_{i'k}]\right), \text{ for } i\neq k \text{ and } b_{ik}=0\,; \\ \rho_{kk}(1)\cdot\prod_{i':i'\notin\{i,k\}}\max[\rho_{i'k}(0),\rho_{i'k}(1)] = \rho_{kk}\cdot\prod_{i':i'\notin\{i,k\}}\max[1,\rho_{i'k}], \text{ for } i\neq k \text{ and } b_{ik}=1\,, \end{cases}$$

$$\text{so } \alpha_{ik}^{\mathrm{MP}} = \frac{\alpha_{ik}^{\mathrm{MP}}(1)}{\alpha_{ik}^{\mathrm{MP}}(0)} = \begin{cases} \prod_{i':i'\neq i}\max[1,\rho_{i'k}], \text{ for } i=k\,; \\ \min\left(0,\ \rho_{kk}\cdot\prod_{i':i'\notin\{i,k\}}\max[1,\rho_{i'k}]\right), \text{ for } i\neq k\,. \end{cases}$$

Because $\tau_{ik}=\sigma_{ik}\alpha_{ik}$ and $\rho_{ik}=\sigma_{ik}\beta_{ik}$, we can absorb the β-updates into the ρ-updates by re-writing it as $\rho_{ik}^{\mathrm{SP}}=\sigma_{ik}\beta_{ik}^{\mathrm{SP}}=\sigma_{ik}/\sum_{k':k'\neq k}\tau_{ik'}=\sigma_{ik}/\sum_{k':k'\neq k}\sigma_{ik'}\alpha_{ik'}$ and $\rho_{ik}^{\mathrm{MP}}=\sigma_{ik}\beta_{ik}^{\mathrm{MP}}=\sigma_{ik}/\max_{k':k'\neq k}\tau_{ik'}=\sigma_{ik}/\max_{k':k'\neq k}\sigma_{ik'}\alpha_{ik'}$. Then, the standard affinity propagation formulation from Sections 3.1–3.2 can be recovered by labeling responsibilities $r(i,k)=\log\rho_{ik}^{\text{SP or MP}}$ and availabilities $a(i,k)=\log\alpha_{ik}^{\text{SP or MP}}$.

3.4 Other algorithms for clustering via belief propagation

Several additional algorithms that employ belief propagation for exemplar-based clustering are possible and have been explored. The following section describes several possibilities; they are more tightly-constrained (Section 3.4.1) or have higher computational complexity (3.4.2-3.4.4) than affinity propagation and are thus not developed further.

3.4.1 Affinity propagation with added non-empty cluster constraint

In its earliest formulation [34], affinity propagation clustering employed a factor graph with identical topology but more stringent constraint functions, $\{f'_k(\mathbf{c})\}_{k=1}^N$. Not only did this function disallow exemplar-less clusters (as is the case with $\{f_k(\mathbf{c})\}$), but also clusters without any non-exemplar members. This was coupled with a sequential message-passing schedule to prevent periodic swings where all data points were considered exemplars (heading empty clusters). The constraint is as follows:

$$f'_k(\mathbf{c}) = \begin{cases} 0, \text{ if } c_k \neq k \text{ but } \exists i\text{: } c_i = k \text{ (disallow clusters without an exemplar)} ; \\ 0, \text{ if } c_k = k \text{ and } \forall i \neq k\text{: } c_i \neq k \text{ (disallow clusters lacking non-exemplar members)} ; \\ 1, \text{ otherwise .} \end{cases}$$

The second line in $f'_k(\mathbf{c})$ does not appear in $f_k(\mathbf{c})$ and is shown in red. This leads to significantly-more complicated update equations (3.19) and (3.20), that are less intuitive:

Sum-product formulation:

$$\begin{aligned} e^{r(i,k)} &= e^{s(i,k)} \Big/ \textstyle\sum_{k':k'\neq k} \left[e^{s(i,k')} e^{a(i,k')}\right] \\ e^{a(i,k)} &= \begin{cases} \prod_{i':i'\neq i} \left[1 + e^{r(i',k)}\right] - 1, \text{ for } k = i ; \\ \left(\left[e^{-r(k,k)} - 1\right] \cdot \prod_{i':i'\notin\{i,k\}} \left[1 + e^{r(i',k)}\right]^{-1} + 1\right)^{-1}, \text{ for } k \neq i . \end{cases} \end{aligned} \tag{3.19}$$

Max-product formulation:

$$r(i,k) = s(i,k) - \max_{k':k'\neq k} \left[s(i,k') + a(i,k') \right]$$

$$a(i,k) = \begin{cases} \max\limits_{i':i'\neq i} \min\left[0, r(i',k)\right] + \sum_{i':i'\neq i} \max\left[0, r(i',k)\right], & \text{for } k = i \text{ ;} \\ \min\left(-\max\limits_{i':i'\notin\{i,k\}} \min\left[0, r(i',k)\right],\ r(k,k) + \sum\limits_{i':i'\notin\{i,k\}} \max\left[0, r(i',k)\right] \right), & \text{for } k \neq i \text{ .} \end{cases} \tag{3.20}$$

This algorithm, while still $\mathcal{O}(N^2)$, was shown in [25] to have quite inferior performance due to the extra constraint unnecessarily preventing the algorithm from moving through regions of the search space on the way to better solutions. To compare the two algorithms, the task of clustering image patches described in [34] was examined[7]. Patches were clustered using both versions of affinity propagation and the resulting likelihood was compared to 100,000 restarts of k-medoids clustering (requiring roughly 1000 times the computation as both affinity propagation algorithms), and the *Science* version of affinity propagation achieves better results for a wide variety of parameters, as shown in Figure 3.6.

3.4.2 Alternate factor graph: N binary nodes

Instead of having variables $c_1, c_2, \ldots, c_N$ that make N-ary assignments of each data point to clusters, it is possible to use N binary variables that only indicate whether or not each point is an exemplar. For each binary variable, $\{b_i\}_{i=1}^N$, $b_i = 1$ indicates that data point i serves as an exemplar and $b_i = 0$ that it is a non-exemplar—implicitly belonging in the cluster to which it has highest similarity, $\text{argmax}_{k:b_k=1} s(i,k)$. This setup is reflected in the factor graph from Figure 3.7, with constraint functions absorbing similarities as follows:

[7] Briefly (see [34] for details), a tiling of 24×24 non-overlapping patches was extracted from the image and translation-invariant similarities were computed by comparing smaller 16×16 windows within each patch. The lowest squared error between windows (over all possible translations) was chosen as a similarity measure.

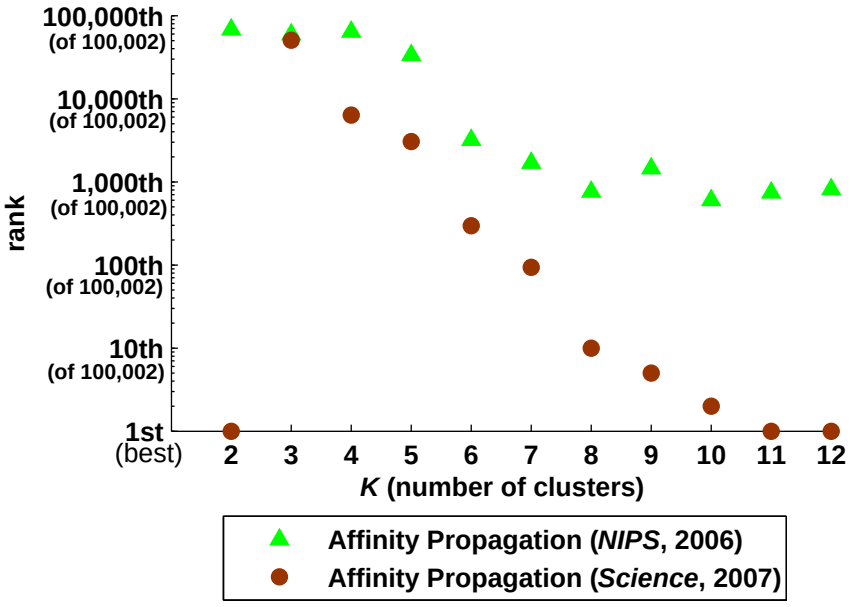


Figure 3.6: Performance (measured by ranking within 100,000 restarts of k-medoids clustering plus one run of each version of max-product affinity propagation) for various numbers of clusters on a patch clustering task. A common value for the exemplar preferences was tuned to produce $K = \{2, 3, \ldots, 12\}$ clusters and the result was compared to that obtained from 100 000 random initializations of k-medoids. In most cases, the *Science* version of affinity propagation [38] finds a configuration at or near the best of the many k-medoids initializations, while the older *NIPS* version [34] performs comparatively worse. For few clusters (*e.g.* $K = \{3, 4, 5\}$), the clustering task is trivial (search spaces of size $\binom{64}{3} = 41664$, $\binom{64}{4} = 635376$, and $\binom{64}{5} = 7624512$, respectively), so 100 000 restarts of k-medoids works well. Larger search spaces (e.g. $K = 9$ has $\binom{64}{9} \approx 3 \times 10^{10}$) show a clearer benefit for affinity propagation.

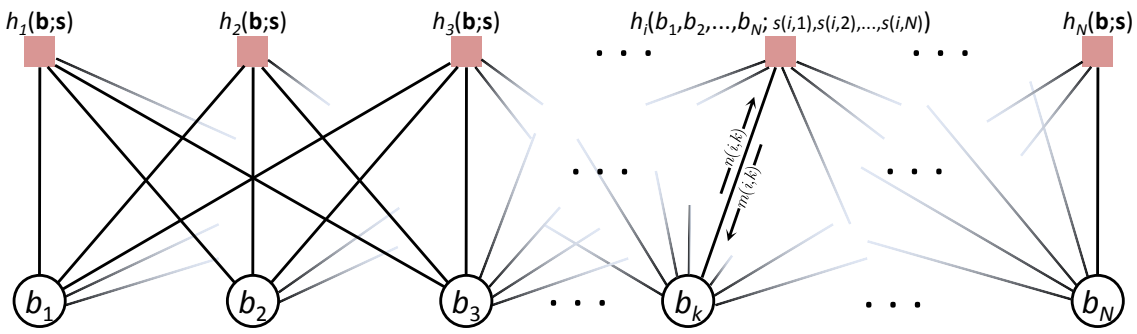


Figure 3.7: An alternate factor graph for exemplar-based clustering with N binary variables, $b_1, b_2, \ldots, b_N$, where $b_k = 1$ indicates that data point k is an exemplar. Non-exemplars (indicated by $b_k = 0$) are implicitly assigned to the exemplar most-similar to them; this is incorporated into constraints $h_1, h_2, \ldots, h_N$.

$$h_i(b_1, b_2, \ldots, b_N; \mathbf{s}) = \begin{cases} [\sum_{i'} b_{i'} > 0] \cdot \max_k e^{s(i,k)\cdot[b_k=1]}, \text{ for } b_i = 0 \;\; (i \text{ is a non-exemplar}) \text{ ;} \\ e^{s(i,i)}, \text{ for } b_i = 1 \;\; (i \text{ is an exemplar}) \text{ .} \end{cases}$$

Note that the initial $[\sum_{i'} b_{i'} > 0]$ multiplier ensures that the degenerate case where there are no exemplars, *i.e.* $\forall i \colon b_i = 0$, is not allowed. The global function is:

$$F(\mathbf{b};\mathbf{s}) = \prod_{i=1}^{N} h_i(\mathbf{b};\mathbf{s}) = \overbrace{\left[\sum_{i'} b_{i'}\right]^N}^{\substack{\text{no-exemplar} \\ \text{case not allowed}}} \cdot \overbrace{\prod_{i:b_i=1} e^{s(i,i)}}^{\substack{\text{exemplar} \\ \text{preferences}}} \cdot \overbrace{\prod_{i:b_i=0} \max_k \left[e^{s(i,k)\cdot[b_k=1]}\right]}^{\substack{\text{non-exemplars belong to cluster} \\ \text{with most-similar exemplar}}} \quad (3.21)$$

The algorithm is $\mathcal{O}(N^3)$ after some optimization, and preliminary experimentation with small problems has shown it to be even slower than exact (linear programming) methods with worse results than affinity propagation.

3.4.3 Alternate factor graph: K N-ary nodes

It is possible to re-write the exemplar-based clustering problem explicitly in terms of finding K clusters using K N-ary variables, each of which indicate an exemplar such that $c_k = i$ indicates

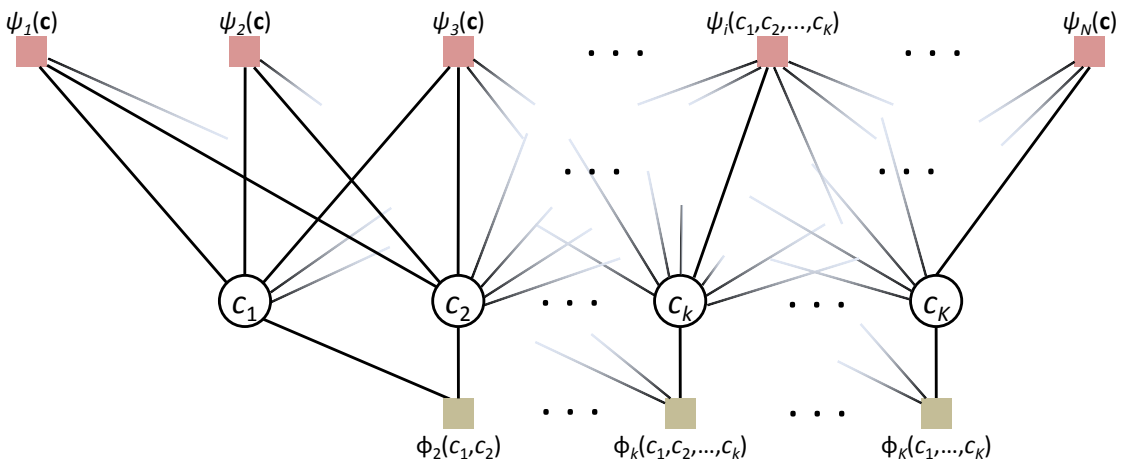


Figure 3.8: An alternate factor graph for exemplar-based clustering using K N-ary variables, $c_1, c_2, \ldots, c_K$, each of which index an exemplar. Non-exemplars are implicitly assigned to the exemplar most-similar to them; this is incorporated into constraints $\psi_1, \psi_2, \ldots, \psi_N$. Constraints $\phi_2, \phi_3, \ldots, \phi_K$ ensure that no points are duplicated in the set of exemplars.

that data point i is an exemplar. The factor graph topology is shown in Figure 3.8. There are N function nodes, $\{\psi_i\}_{i=1}^{N}$, each of which computes the i^{th} non-exemplar data point's contribution to the net similarity by choosing the most-similar cluster. In addition, there are K function nodes, $\{\phi_k\}_{k=2}^{K}$, that include exemplar preferences in the global function and also enforce the tedious constraint that there are no duplicate or repeated exemplars. These functions are defined as follows:

$$\psi_i(c_1, c_2, \ldots, c_K) = \prod_{i=1}^{N} \left(\max_{k \in \{1,2,\ldots,K\}} e^{s(i,c_k)} \cdot \prod_{k'=1}^{K} [c_{k'} \neq i] \right) ;$$

$$\psi_i(c_1, c_2, \ldots, c_K) = \prod_{i=1}^{N} \left(\max_{k \in \{1,2,\ldots,K\}} e^{s(i,c_k)} \right)^{\prod_{k'=1}^{K} [c_{k'} \neq i]} ;$$

$$\phi_k(c_1, c_2, \ldots, c_k) = \prod_{k=1}^{K} \left(e^{s(c_k,c_k)} \cdot \prod_{k'=1}^{k-1} [c_k \neq c_{k'}] \right), \; k > 1 \, .$$

They lead to the following global function:

$$F(c_1, c_2, \ldots, c_K; \mathbf{s}) = \prod_{k=1}^{K} \phi_k(c_1, c_2, \ldots, c_k) \cdot \prod_{i=1}^{N} \psi_i(c_1, c_2, \ldots, c_K)$$
$$= \prod_{k=1}^{K} \left(\overbrace{e^{s(c_k,c_k)}}^{\text{preferences}} \cdot \prod_{k'=1}^{k-1} \overbrace{[c_k \neq c_{k'}]}^{\text{no duplicates!}} \right) \cdot \prod_{i=1}^{N} \left(\overbrace{\max_{k \in \{1,2,\ldots,K\}} e^{s(i,c_k)}}^{\text{in most-similar cluster}} \cdot \prod_{k'=1}^{K} \overbrace{[c_{k'} \neq i]}^{\text{only non-exemplars}} \right).$$

Computation of sum-product and max-product messages can be done in polynomial time (in particular, the ϕ-constraints can be efficiently absorbed into the ψ-constraints for max-product), but this is not explored further in this thesis.

3.4.4 Alternate factor graph: ternary nodes

A final factor graph worth brief mention is shown in Figure 3.9. It involves ternary variables $\{t_{ik}\}_{(i,k) \in \{1,2,\ldots,N\}^2 \text{ and } i<k}$ for indicating cluster membership, where $t_{ik} = 1$ indicates that data point i is in a cluster whose exemplar is data point k, $t_{ik} = -1$ that data point k is in a cluster whose exemplar is data point i, and $t_{ik} = 0$ otherwise. There are N binary variables $\{b_i\}_{i=1}^{N}$ indicating whether each data point, i, is an exemplar ($b_i = 1$) or non-exemplar ($b_i = 0$). Function nodes enforce the usual constraints with $\zeta_i(t_{1i}, \ldots, b_i, \ldots, t_{iN}) = \left[b_i \geq \max_{k \in \{1,2,\ldots,i-1\}} t_{ki} \right] \cdot \left[b_i \geq \max_{k \in \{i+1,\ldots,N\}} -t_{ik} \right] \cdot \left[1 = \sum_{k=1}^{i-1} [t_{ki}{=}{-1}] + b_i + \sum_{k=i+1}^{N} [t_{ik}{=}1] \right]$ acting similar to equations (3.16) and (3.17) in Section 3.3.

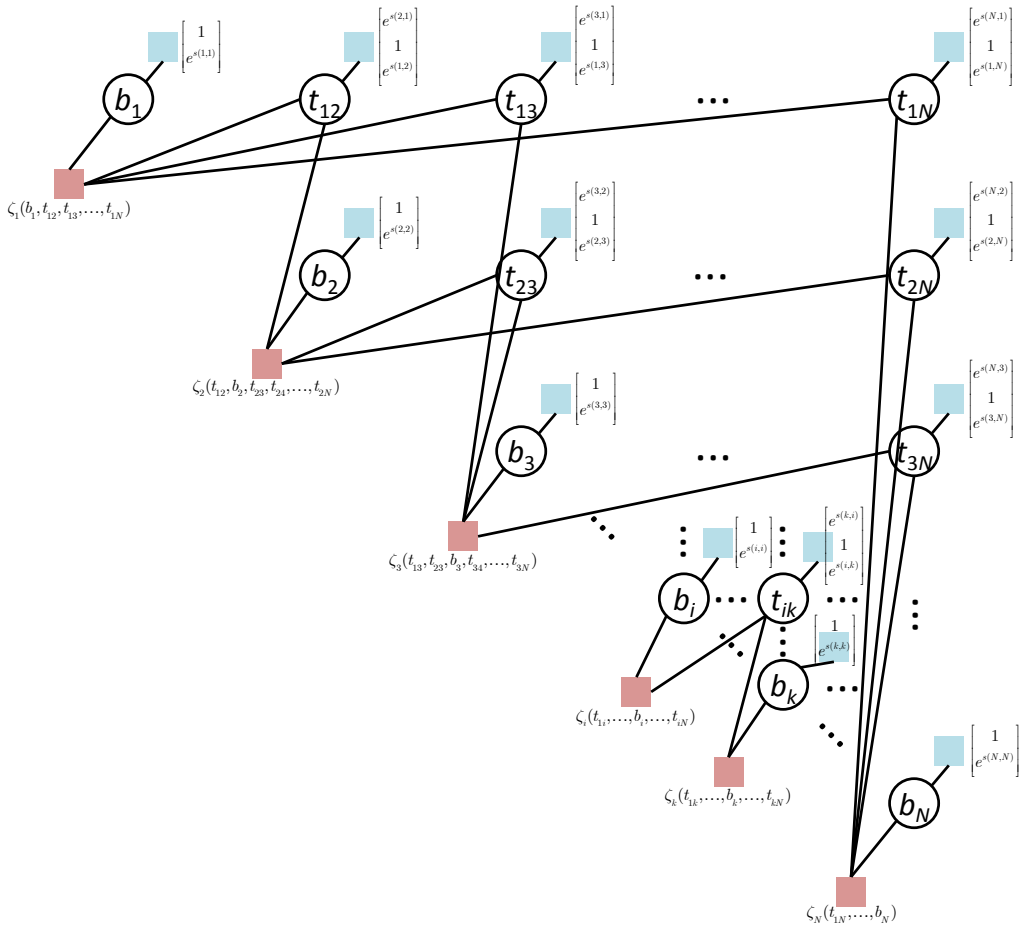


Figure 3.9: This factor graph for exemplar-based clustering involves ternary variables $\{t_{ik}\}_{(i,k)\in\{1,2,\ldots,N\}^2 \text{ and } i<k}$ for indicating cluster membership where $t_{ik} = 1$ indicates that data point i is in a cluster whose exemplar is data point k, $t_{ik} = -1$ that data point k is in a cluster whose exemplar is data point i, and $t_{ik} = 0$ otherwise.

Chapter 4

Benchmarking Affinity Propagation

The affinity propagation algorithm for exemplar-based clustering performs well at finding clustering solutions that optimize net similarity. This chapter benchmarks affinity propagation alongside 15 other clustering methods for a range of small ($N < 1000$) and large ($N > 5000$) datasets.

4.1 Olivetti faces: Clustering a small dataset

The Olivetti faces dataset [87] is a collection of 400 $64{\times}64$ greyscale images of human faces (10 from each of 40 people) with varying facial expressions and lighting conditions. The complete dataset is available at http://www.cs.toronto.edu/∼roweis/data.html and shown in Figure 4.1.

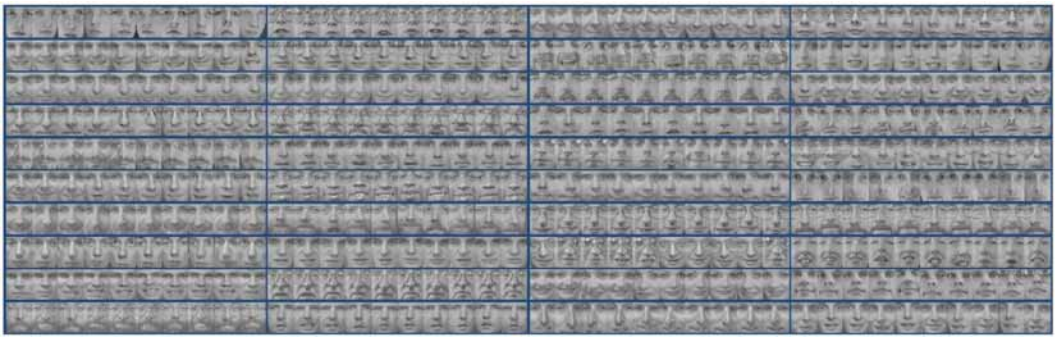


Figure 4.1: The Olivetti dataset [87] consists of 400 64×64 greyscale images of human faces.

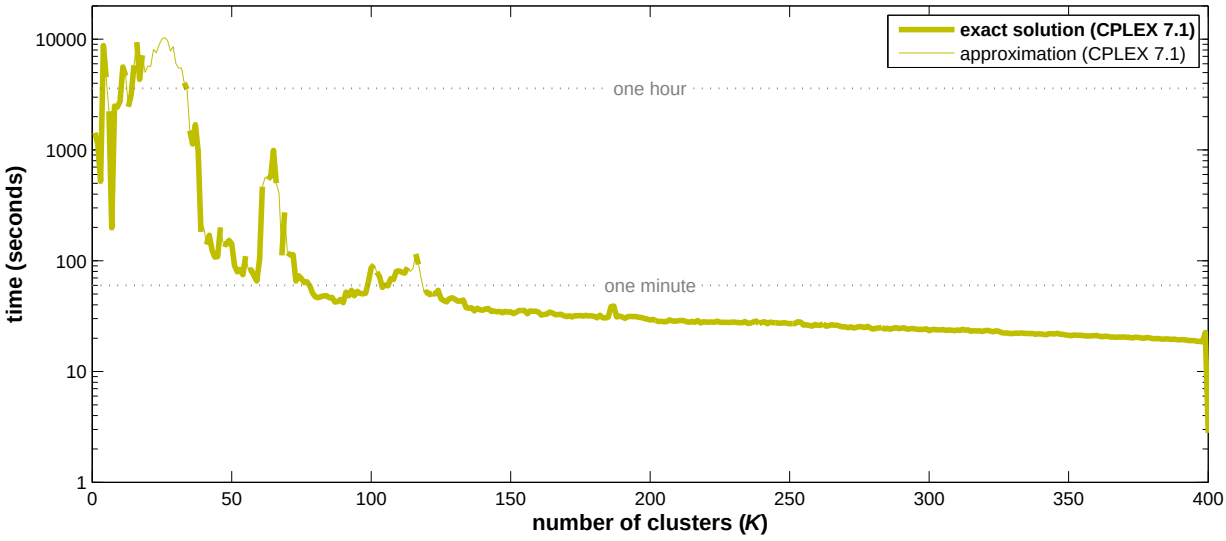


Figure 4.2: The exact (CPLEX 7.1) solution time is plotted as a function of K (number of clusters). The curve is thin in cases where only an approximate solution was found (*i.e.*, a bound to several decimal places); for most K-values the optimal solution was found along with the set of exemplars that yields it. Most interesting solutions take minutes to find, with some in the $K < 50$ range requiring several hours.

To lessen the effects of the background, only the central 50×50 pixel window (normalized to have mean intensity zero and standard deviation 0.1) is used to compute pairwise similarities. The similarity $s(i, k)$ of face image i to face image k was set to the negative sum of squared pixel differences between these two images.

4.1.1 Exact clustering solutions

This dataset was of particular interest because it has a large enough search space to challenge approximate algorithms (*e.g.*, for $N = 400$ points and $K = 40$ clusters, $\binom{N}{K} \approx 10^{55}$ possible exemplar sets) but is small enough for the linear programming relaxation (§2.4.1) to feasibly find exact solutions for most parameter settings.

CPLEX 7.1 optimization software was used to compute optimal clusterings of the Olivetti data for all possible numbers of clusters, *i.e.*, $K = 1, 2, \ldots, N$ using the linear programming formulation from Section 2.4.1. In most interesting cases, *i.e.*, $K < 100$, several minutes of

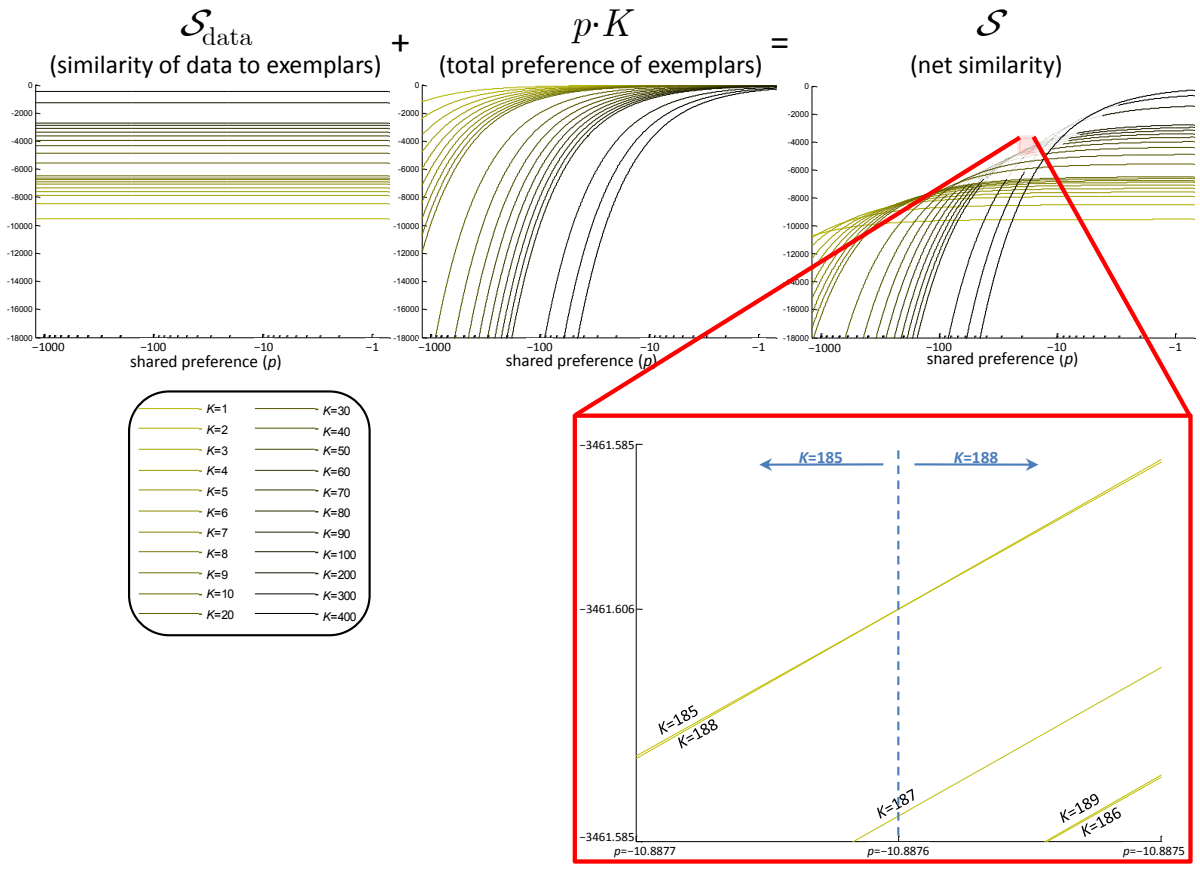


Figure 4.3: The contributions (and tradeoffs) of data similarities ($\mathcal{S}_{\text{data}}$) and exemplar preferences to net similarity ($\mathcal{S}$) are shown for exact CPLEX solutions. Increasing the exemplar preference (p) monotonically increases the optimal number of clusters (K), but the zoomed-in region illustrates that not all K-values (*e.g.* $K \in \{186, 187\}$) have associated preference ranges. Note that $p \cdot K$ is linear but appears logarithmic due to the horizontal axis using a log scale.

computation time were required for CPLEX to find a solution as shown in Figure 4.2. Exact solutions were found for 366 K-values; bounds (though not solutions with exemplar sets) that were tight to several digits of precision were found for the remaining 34 K-values and are shown as thin lines in the figure.

Closer examination of the exact linear programming results provide several valuable insights into the exemplar-based clustering problem setup. Figure 4.3 shows the similarity of data to exemplars ($\mathcal{S}_{\text{data}}$), the total preference of exemplars ($p \cdot K$), and their sum, the net similarity ($\mathcal{S}$). Separate curves are shown for different numbers of clusters, and each quantity is shown as a function of the globally-shared preference, p. $\mathcal{S}_{\text{data}}$ is constant with respect to p

so it is shown as an ensemble of horizontal lines; each $p \cdot K$ is linear with slope K. Noting that all similarities and preferences are negative, the curves are ordered differently in both of the component plots: for $\mathcal{S}_{\text{data}}$, the curves are parallel and sorted according to ascending K because $\mathcal{S}_{\text{data}}$ is highest where there are the fewest non-exemplar data points (high K). The total preference of exemplars has the reverse situation, where curves with increasing slope are sorted in order of descending K because solutions with few clusters contain the smallest preference contributions/penalties. The sum, net similarity $\mathcal{S} = \mathcal{S}_{\text{data}} + p \cdot K$, is also plotted and illustrates the tradeoff between the two terms and how the optimal number of clusters increases as the preference is increased.

The optimal number of clusters increases monotonically with the preference, however, not every K-value has an associated interval of preference values for which it is optimal—for this problem the set of $K \in \{20, 21, 23, 28, 34, 46, 62, 100, 101, 118, 129, 186, 187\}$ falls into this category. Apparently, CPLEX has difficulty optimizing in regions specifically around these missing K-values and with the only exceptions being near $K \in \{129, 186, 187\}$. The situation around $K \in [185, 189]$ is illustrated in the highly zoomed-in Section around $p \in [-10.8877, -10.8875]$. At the point where the $K = 185$ and $K = 188$ curves intersect, the $K = 186$ and the $K = 187$ curves are both below; because their slopes are higher than the $K = 185$ curve they are never greater for lower preferences nor are they ever higher for greater preferences than the more-sloped $K = 188$. The intuition is that the preference cost of adding an extra exemplar in the area around $p = -10.8876$ (where $K = 185$ is optimal) never outweighs the savings of a better $\mathcal{S}_{\text{data}}$ solution, and it only becomes worthwhile (the net similarity improves) when three additional exemplars are added.

4.1.2 Performance of Affinity Propagation

Next, affinity propagation was run using 1000 different global preference values logarithmically sampled between -1200 and -0.7 (randomly sampled from the horizontal axis of Figure 4.4), each requiring less than one minute of computation time. An important consideration

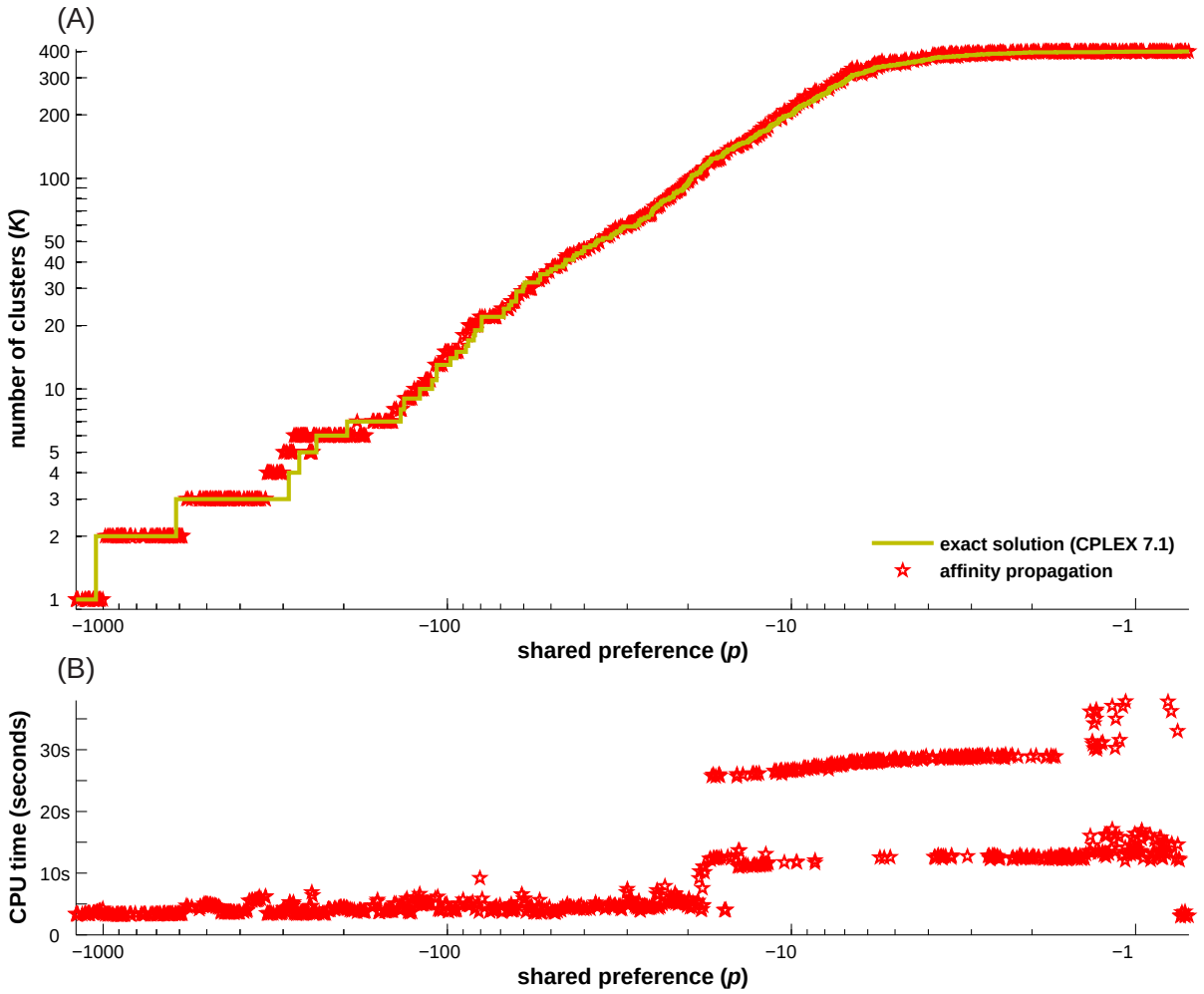


Figure 4.4: Affinity propagation was run on the Olivetti faces dataset with 1000 global preferences (p) sampled randomly from the horizontal axis shown above. The number of clusters, K, found is plotted as a function of p in (A) and superimposed over the exact solution found by CPLEX. There is little divergence between the two curves, and it mostly takes place for $K \leq 6$. The CPU time required for affinity propagation is shown in (B); it typically requires seconds whereas CPLEX requires minutes or even hours. With $p > -10$, the solutions split into two regions because many runs oscillate; this represents less-interesting solutions where $K > 100$ for $N = 400$.

is whether affinity propagation finds the 'correct' number of clusters; the resulting number found are shown in Figure 4.4 alongside optimal values found by CPLEX. For a large range of preferences (until $p > -10$ *i.e.* $K > 300$), affinity propagation finds solutions quite close to the optimal K. The net similarities, $\mathcal{S}$, achieved by affinity propagation are shown in Figure 4.5 as well as optimal values derived from CPLEX (computed using $\mathcal{S} = \mathcal{S}_{\text{data}} + K \cdot p$). In addition, Figure 4.5 shows the total similarity of non-exemplar points to their exemplars, $\mathcal{S}_{\text{data}}$, as a function of K for both algorithms.

Finding a suitable preference using cross-validation

For datasets such as the 400 Olivetti faces where negative squared Euclidean distance is used as the similarity measure, a suitable choice of preference can be made by employing ten-fold cross-validation. This was attempted by running affinity propagation for the set of 1000 global preference values described in Section 4.1.2 on ten different $N = 360$-sized training sets with 10% of the data held out for validation. Using the clustering solutions found by affinity propagation, mixtures of isotropic Gaussians were found (one M-step from EM) and the likelihood of the validation sets under the mixture was computed. This is shown in the plot of total validation log-likelihood as a function of preference (p) in Figure 4.6. The preference value with the highest log-likelihood, $p^* \approx -31.5082$, corresponds to affinity propagation finding $K = 58$ exemplars as shown in Figure 4.4(A).

4.1.3 Performance of other clustering techniques

Results were computed for 15 additional clustering algorithms and are shown in Figures 4.7 to 4.22. They were computed using a 64-node computing cluster[1] and involved roughly a year of total single-node processing time in MATLAB 7.5 (R2007b). Random restarts (where applicable)

[1]The information processing lab at the University of Toronto, a joint effort between Profs. B. Frey, F. Kschischang, and W. Yu, consists of a Linux-based computing cluster of 64 computing nodes, each with 12 GB of RAM and two dual-core AMD Opteron 2220 CPUs running at 2.8 GHz.

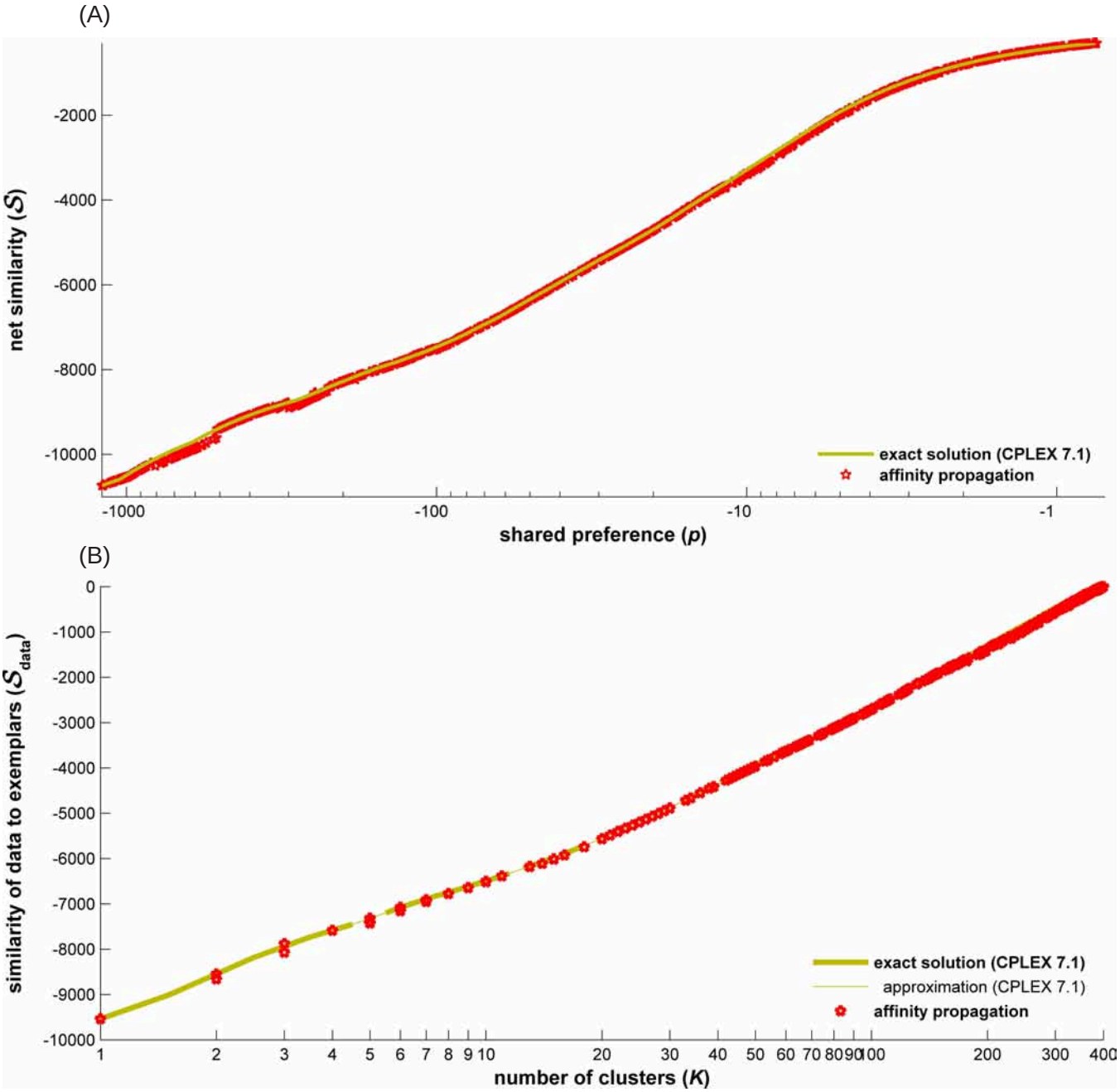


Figure 4.5: The quality of affinity propagation's solutions for the Olivetti faces data are shown as a function of the preference (A) and number of clusters (B). These two different setups lead to different cost functions as shown in the vertical axes; when the number of clusters is controlled by a global preference, p, as in (A), the net similarity, $\mathcal{S}$, is maximized reflecting the tradeoff between exemplar preferences and data point similarities. When the number of clusters is explicitly defined as in (B), the similarity between data points and their exemplars, $\mathcal{S}_{\mathrm{data}}$, is maximized.

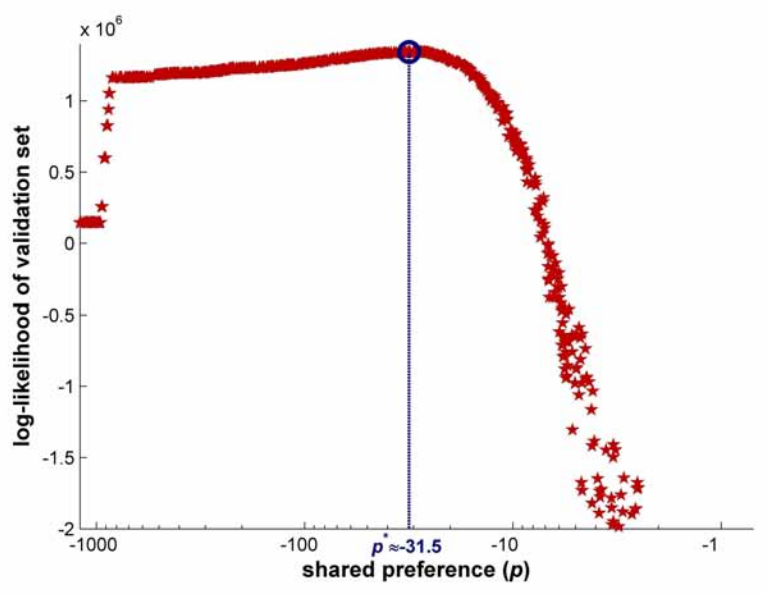


Figure 4.6: A suitable choice of preference for the 400 Olivetti faces dataset using ten-fold cross-validation is shown. Affinity propagation was run on training sets for 1000 shared preference values, and Gaussian mixture distributions were learned for each clustering solution found. The validation log-likelihoods are shown, with the best preference, $p^* \approx -31.5082$, corresponding to a solution with $K = 58$ exemplars.

are shown (shaded) as described in the legend, with median best and worst of 10, 100, 1000, *etc.* runs demarcating boundaries where shading is lightened; specifically, the median best and worst runs of t restarts are defined as the median result of as many sample t-run trials as available[2]. Performance after one second, one minute, and one hour of CPU time are also shown to better illustrate the tradeoff between running time and quality of solution. To facilitate easier comparison, all algorithms are shown on plots with the same scales on both axes and, in contrast with Figure 4.5(B), the horizontal axis (number of clusters, K) is shown with a linear scale, and the vertical axis is shown relative to CPLEX's optimal $\mathcal{S}_{\mathrm{data}}$.

The algorithms examined include k-means and the EM algorithm for a mixture of Gaussians, which is suitable because similarity is defined as the negative sum-of-squared-differences

[2]The sample median best and worst runs of t restarts are the $(0.5)^{1/t}$ and $1-(0.5)^{1/t}$ quantile results of $T \gg t$ restarts. This can be seen by considering the median maximum of t random variables, $U_1, U_2, \ldots, U_t$, distributed uniformly on $[0, 1]$. Defining $M^{(t)} = \mathrm{median}[\max(U_1, U_2, \ldots, U_t)]$, then $P[U_1 < M^{(t)}, U_2 < M^{(t)}, \ldots, U_t < M^{(t)}] = \left(P[U < M^{(t)}]\right)^t = 0.5$ so $M^{(t)} = (0.5)^{1/t}$. For example, if a total of $T = 10^6$ random restarts are computed, the median best of ten restarts is the $(0.5)^{1/10} \approx 0.933^{\mathrm{rd}}$ quantile *i.e.*, the 66700^{th}-best restart.

between 50×50 pixel windows, a 2500-dimensional data space. These algorithms were all run until convergence or 100 iterations, whichever came first. For algorithms that find only a partitioning of data points but no exemplars, the resulting partition is 'quantized' by initializing a single restart of k-medoids to find the locally-optimal set of exemplars for the partition (or if means characterize cluster centers, quantized to exemplars). In more detail, the algorithms examined are as follows:

Affinity Propagation

Affinity propagation was run with a dampening factor of $\lambda = 0.9$ for a maximum of 1000 iterations (where 100 consecutive iterations of no change in exemplar set qualifies as 'convergence') and is shown on all plots for easier comparison. Running times are shown in Figure 4.4; it is typically several seconds. Performance noticeably degrades between $K = 150$ and $K = 300$, where the solution search space is largest.

Vertex substitution heuristic (VSH) with variable neighbor search

The vertex substitution heuristic is shown alongside CPLEX and affinity propagation in Figure 4.7. This is by far the most competitive algorithm, achieving optimal results for this problem in a matter of minutes for all K-values. It is implemented with variable neighbor search as described in Section 2.5 and [47]. For problems of this size, VSH is competitive with affinity propagation in terms of computation time and Figure 4.7 shows that initializing VSH with the output of affinity propagation usually achieves solutions far superior to the median VSH run.

k-medoids clustering (with or without $k \cdot \log(k)$ heuristic)

k-medoids clustering is shown alongside CPLEX and affinity propagation in Figure 4.8 with several million random initializations computed per K-value. It is the simplest and fastest algorithm available, and results from many restarts show wide variation. Affinity propagation outperforms millions of k-medoids runs beyond $K = 10$; interestingly, the number of restarts

typically required for an optimal solution roughly increases by a factor of ten for every additional cluster beyond $K = 3$ until 10 million restarts no longer finds an optimal solution for $K = 11$'s search space (which is roughly 10^{20} in size). k-medoids with the $k \cdot \log(k)$ heuristic is shown in Figure 4.9. This is implemented as described in Section 2.3.1 with the caveat for extreme cases that $\min(K \cdot \ln(K), N)$ is used as the initial number of clusters before pruning. This algorithm is somewhat slower per restart than regular k-medoids but shows modest improvements given the same amount of time (apparent by comparing the one hour performance curves).

k-means (with or without $k \cdot \log(k)$ heuristic; exemplars from partitions or means)

k-means clustering is shown alongside CPLEX and affinity propagation in Figures 4.10–4.13. This is implemented as described in Section 2.1 and is run on 2500-dimensional data points to convergence; exemplars are identified either by feeding the resulting cluster partition into k-medoids or by initializing k-medoids with an exemplar set consisting of points closest to each mean[3]. Due to the high dimensionality of the input data, only hundreds or thousands of runs are typically feasible within a few hours. The k-means algorithm with the $k \cdot \log(K)$ heuristic is also shown; it substantially improves the result when compared to k-means without the heuristic.

EM for mixture of diagonal or isotropic Gaussians (exemplars from partitions or means)

The EM algorithm for mixtures of Gaussians is shown alongside CPLEX and affinity propagation in Figures 4.14–4.18. Exemplars can be identified by partitions, in which case cluster assignment is determined to be the Gaussian with the maximum responsibility[4], or they can be identified by data points closest to the means (as with k-means). Runs with diagonal and

[3] If fewer than K exemplars are found due to two means being closest to the same exemplar, the run is discarded.

[4] In cases where a Gaussian has no member data points after thresholding (leading to fewer than K partitions), the run is discarded.

runs with isotropic/spherical Gaussians are shown; the high dimensionality of the data makes computation time per run quite high, negatively impacting results. Finally, to more-closely approximate the k-means results—which may be better because they optimize the net similarity unadulterated by different covariances for each cluster—runs of the EM algorithm with spherical Gaussians using a shared global variance annealed to near-zero are shown in Fig. 4.18[5].

Hierarchical Agglomerative Clustering

Hierarchical agglomerative clustering is shown alongside CPLEX and affinity propagation in Figure 4.19. Hierarchical agglomerative clustering ([61, 93, 103]) involves creating a linkage structure (easily visualized as a dendrogram) for a dataset containing a series of nested clusters, beginning with N clusters and ending with one large cluster. At each of $N-1$ steps, two subclusters are selected to be merged together using one of several possible criteria; a K-clusters partitioning of the dataset can be realized by halting (or revisiting the result of the $(N-K)^{\text{th}}$ agglomeration), akin to 'chopping' off the dendrogram tree at the height where it has K branches. The MATLAB statistics toolbox implementation of this algorithm was used, with all implemented linkage methods attempted: single linkage (*a.k.a.* nearest neighbor), complete linkage (*a.k.a.* furthest neighbor), average linkage (*a.k.a.* unweighted pair group method with arithmetic mean, UPGMA), weighted average distance, centroid linkage, median linkage (*a.k.a.* weighted center of mass distance, WPGMC), and Ward's linkage (inner squared distance). The resulting partitions for each K-value were then fed into k-medoids clustering for refinement and exemplar discovery. For this Olivetti dataset, Ward's linkage method—which combines clusters so as to minimize the increase in the total within-cluster sum of squares—is the only method with competitive results; not surprising considering its cost function is

[5]For these experiments, EM for a mixture of isotropic/spherical Guassians was employed with means initialized from data via a furthest-first traversal and a shared (global) variance $\forall k : \boldsymbol{\Sigma}_k = \sigma^2 \mathbf{I}_D$ where $\sigma^2 = \frac{1}{ND}\sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu}_{c_i})^\top(\mathbf{x}_i - \boldsymbol{\mu}_{c_i})$. Variances (or standard deviations) were gradually annealed to near-zero via the $\sigma_{\text{new}} = 0.99 \cdot \sigma_{\text{old}}$ update at each EM iteration. A total of 1000 iterations were run for each EM optimization, meaning $\sigma_{\text{final}} = 0.99^{1000} \cdot \sigma_{\text{initial}} \approx .00004 \cdot \sigma_{\text{initial}}$. Also, all $N = 400$ possible furthest-first traversal initializations were attempted for each K-value, resulting in the range of results shown in Figure 4.18.

equivalent to the definition of similarity.

Convex Clustering (Lashkari-Golland, 2007)

The convex clustering method (Lashkari-Golland, 2007) [67] is shown alongside CPLEX and affinity propagation in Figure 4.20. This algorithm is based on the idea that instead of maximizing a typical mixture-model log-likelihood $\frac{1}{N}\sum_{i=1}^{N}\log\left[\sum_{k=1}^{K}\pi_k f(\mathbf{x}_i;\mu_k)\right]$ (where f is an exponential family distribution function), the problem can be reformulated to maximizing $\frac{1}{N}\sum_{i=1}^{N}\log\left[\sum_{j=1}^{N}\pi_j e^{-\beta d_\varphi(x_i,x_j)}\right]$ where mixture component densities are located at each data point. Here $d_\varphi(x_i,x_j)$ must be a Bregman divergence [3] (*e.g.* $d_\varphi(x_i,x_j)=-s(i,j)$), and so the latter likelihood is convex whose global optimum, subject to $\sum_{j=1}^{N}\pi_j = 1$, can be found in polynomial time. The β parameter is used to control the sharpness of the mixture components, which turns out to be a multiplicative adjustment of the negative similarity for this example; it controls the number of exemplars found by the algorithm. Consistent with other experiments [82], the convex clustering algorithm seems to have poor performance in practice[6].

Markov Clustering Algorithm (van Dongen, 2000)

The Markov clustering algorithm [99] is a graph-based clustering algorithm based on simulation of stochastic flow in graphs; results of the algorithm applied to the Olivetti faces are shown alongside CPLEX and affinity propagation in Figure 4.21. The algorithm performs competently for very large numbers of clusters, but is difficult to configure for finding small K.

Spectral Clustering

Spectral Clustering is shown alongside CPLEX and affinity propagation in Figure 4.22. Spectral clustering methods use the top eigenvectors of a matrix derived from the distance (nega-

[6]Perhaps the dimensionality of the Olivetti dataset plays a part in convex clustering's poor performance; results on two-dimensional toy datasets of various sizes yield more satisfactory results.

tive similarity) between points to partition the dataset into clusters. For these simulations, a spectral clustering algorithm [80] (based on [74, 104]) is used. Briefly, the K largest eigenvectors are computed (using MATLAB's `svds` command) for an $N \times N$ normalized distance matrix with diagonal elements set to zero and stacked into an $N \times K$ matrix whose rows are normalized (to have unit length) and then clustered using k-means. The cluster assignments of the N rows of the matrix of eigenvectors correspond to the assignments for the N data points, which is fed into k-medoids to refine the solution and identify locally-optimal exemplars. This formulation of spectral clustering is not invariant to the scale (*i.e.* units) of the pairwise similarities so the input pairwise distance is divided by a factor, σ, such that the normalized distance between points i and j is defined as $d_{ij} = -s(i,j)/\sigma$. For each number of exemplars, 36 normalization factors spanning four orders of magnitude were attempted, namely $\sigma \in \{0.1, 0.2, \ldots, 0.9, 1, 2, \ldots, 9, 10, 20, \ldots, 90, 100, 200, \ldots, 900\}$. Most trials require 10–20 seconds of CPU time; a few take longer than a minute but none longer than two minutes. Performance is decent for $K < 25$ clusters, but it dramatically falls off for more clusters. Note that spectral clustering algorithms operate without exemplars by partitioning along gaps in data (*i.e.*, minimizing 'cut' weights), which is an entirely different objective than exemplar-based clustering. The algorithm is included in the comparison for the sake of completeness.

For applications involving computer vision and this Olivetti faces dataset, see Section 5.1.

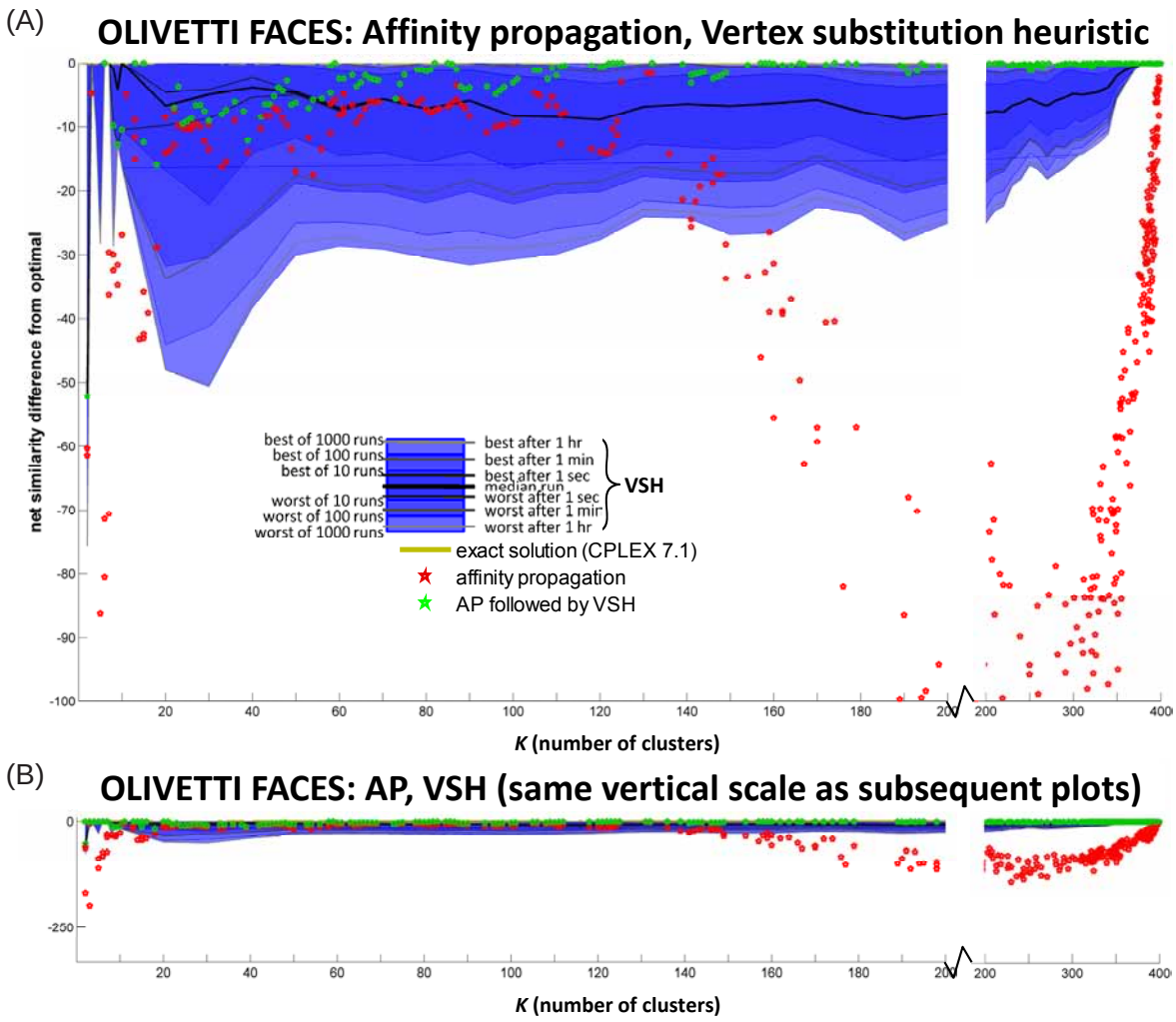


Figure 4.7: Olivetti faces and affinity propagation, vertex substitution heuristic. (A) shows 1000 runs of the vertex substitution heuristic alongside affinity propagation (red stars) and affinity propagation followed by VSH (green stars). Affinity propagation alone performs somewhat worse than the median performance of VSH, but affinity propagation followed by the vertex substitution heuristic performs significantly better than the median run of VSH alone. (B) shows the same plot with a compressed vertical scale directly comparable to subsequent plots.

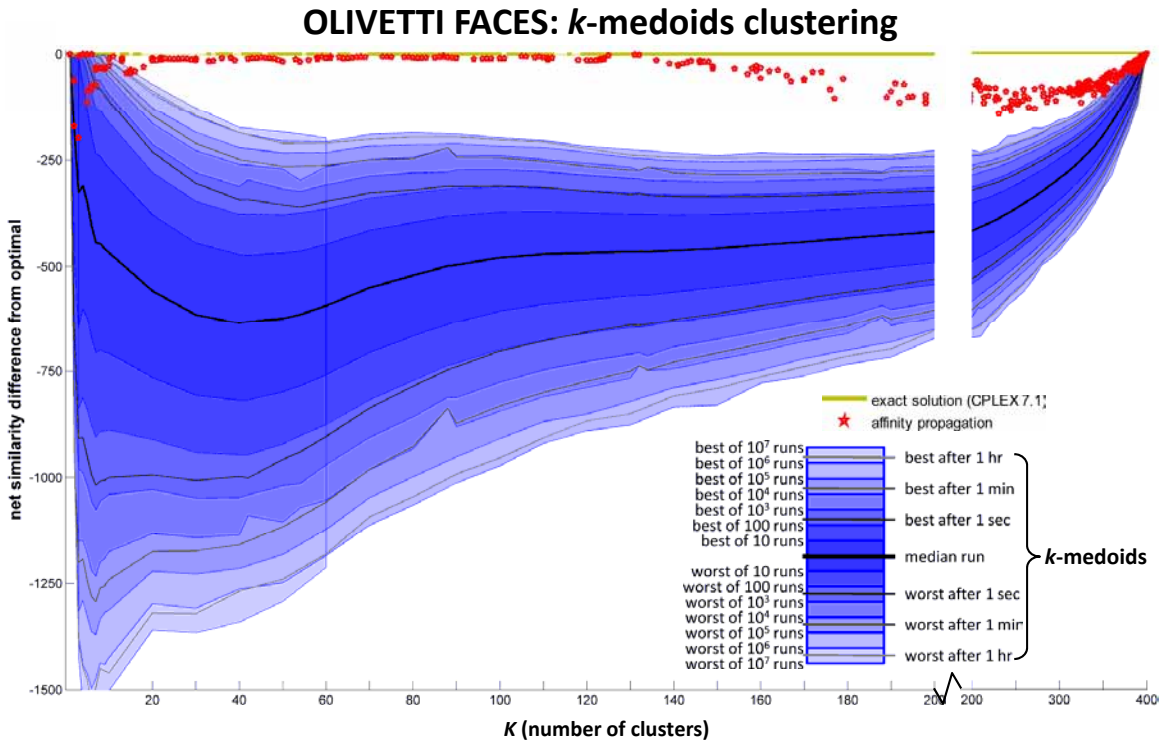


Figure 4.8: Olivetti faces and k-medoids clustering

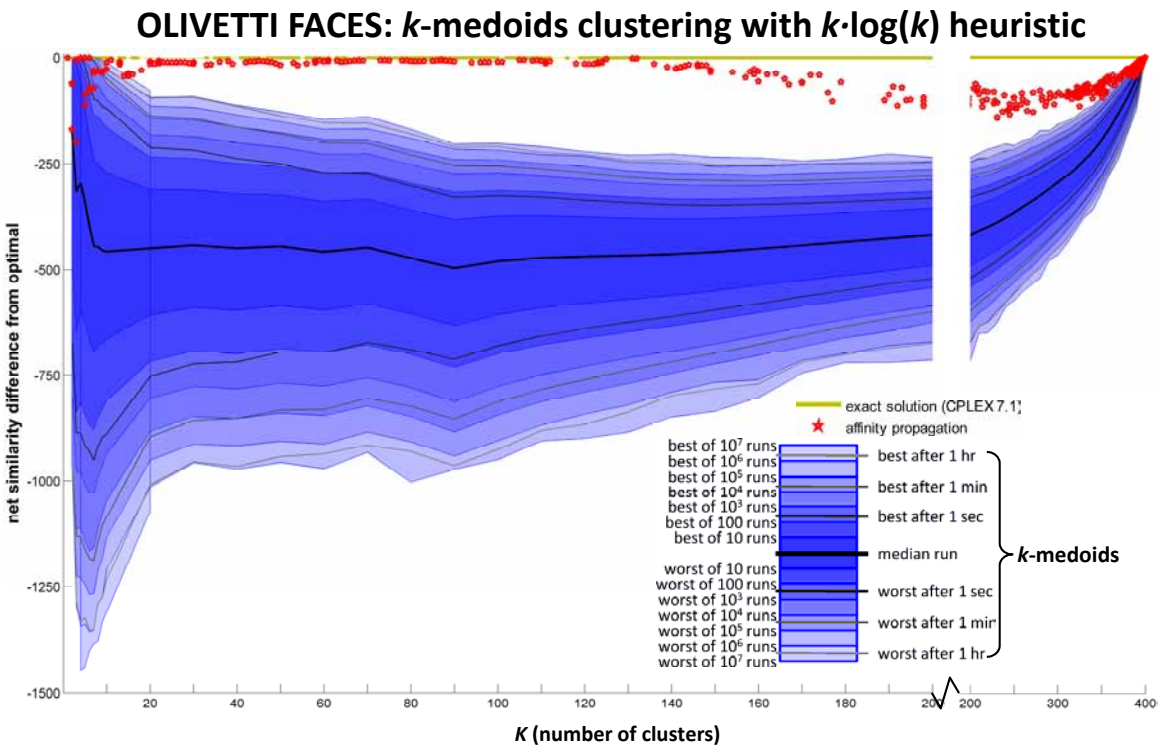


Figure 4.9: Olivetti faces and k-medoids clustering with $k \cdot \log(k)$ heuristic

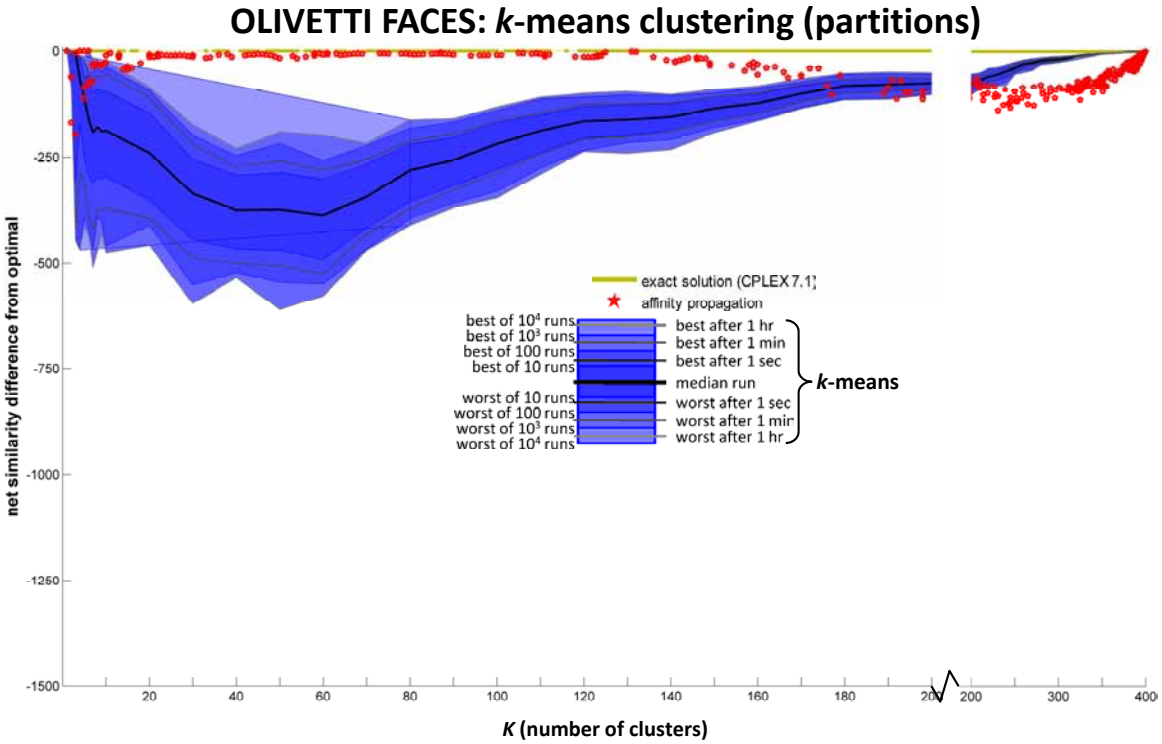


Figure 4.10: Olivetti faces and k-means clustering (by partitions)

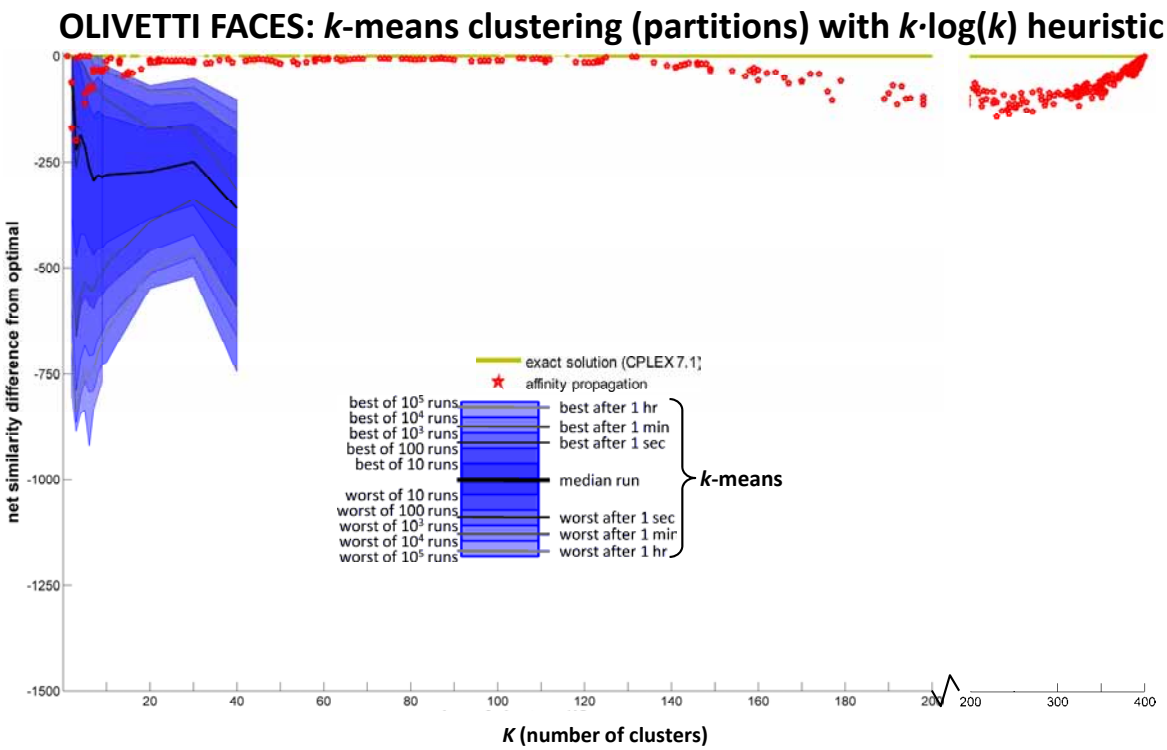


Figure 4.11: Olivetti faces and k-means clustering with $k \cdot \log(k)$ heuristic (by partitions)

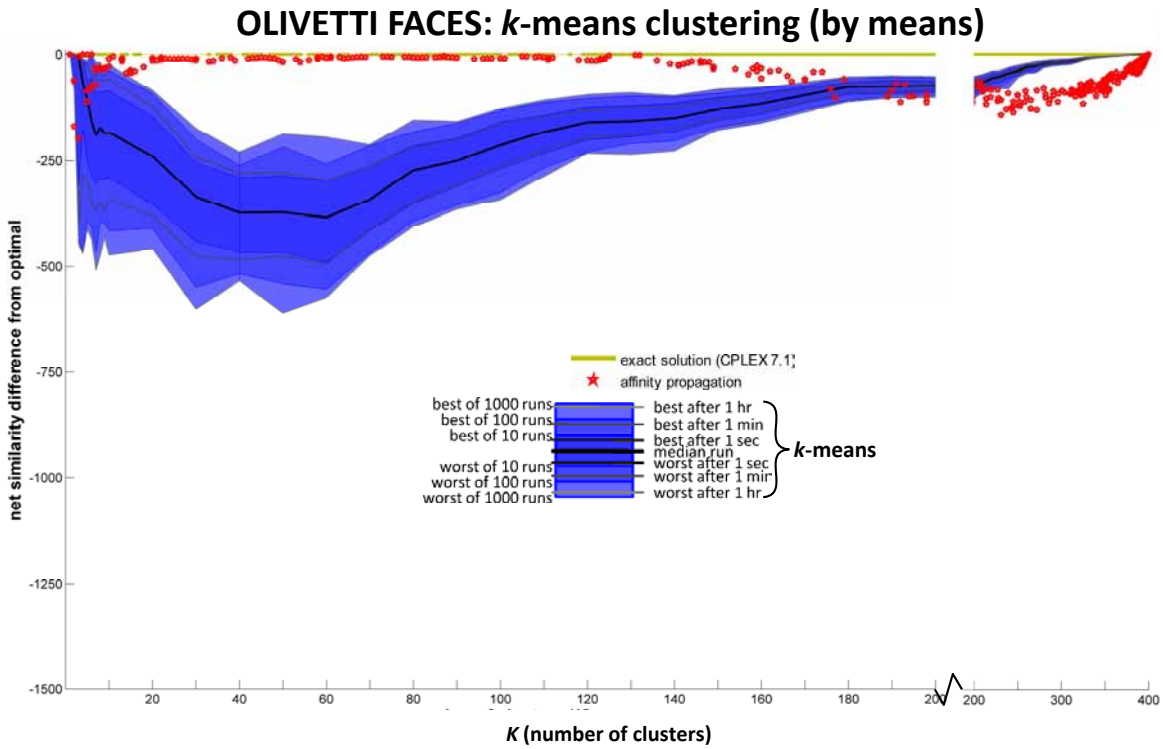


Figure 4.12: Olivetti faces and k-means clustering (by means)

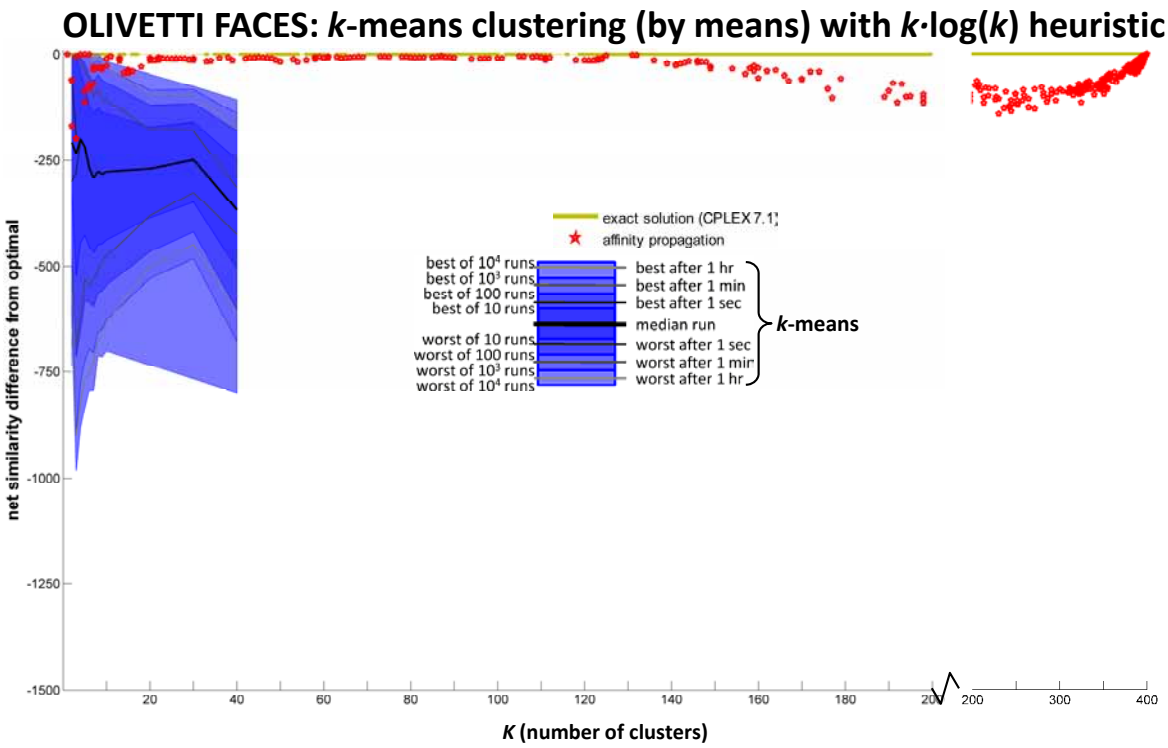


Figure 4.13: Olivetti faces and k-means clustering with $k \cdot \log(k)$ heuristic (by means)

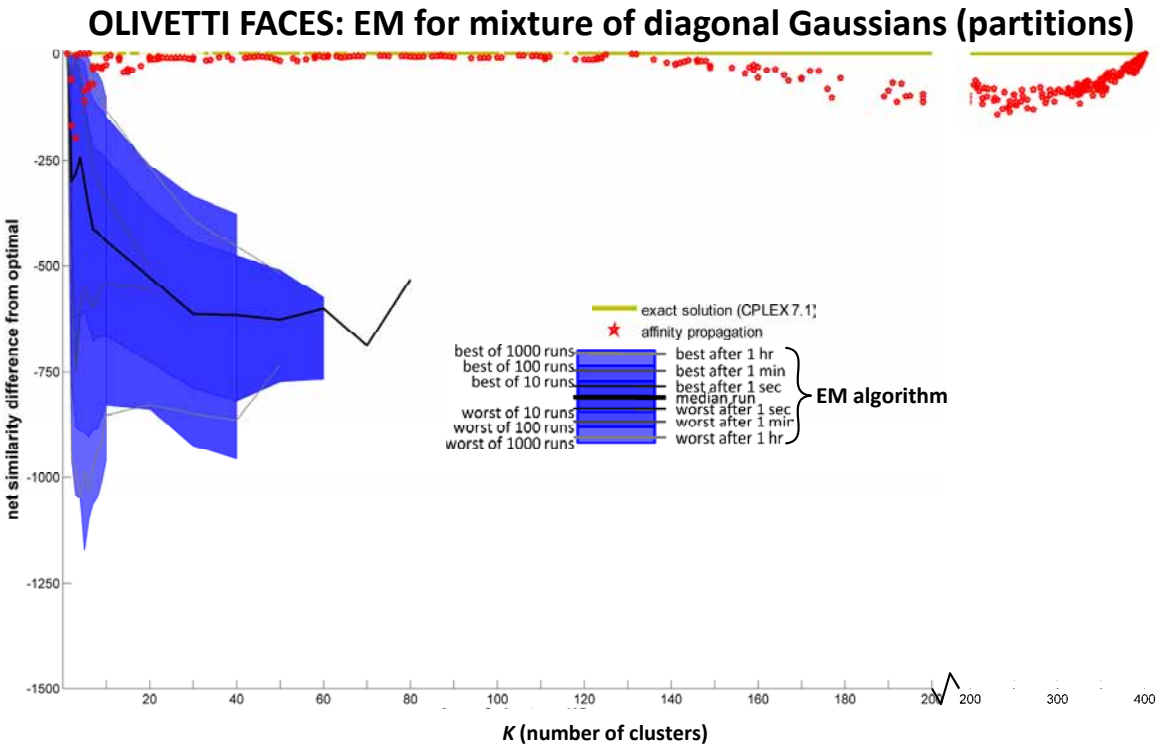


Figure 4.14: Olivetti faces and EM for mixture of diagonal Gaussians (by partitions)

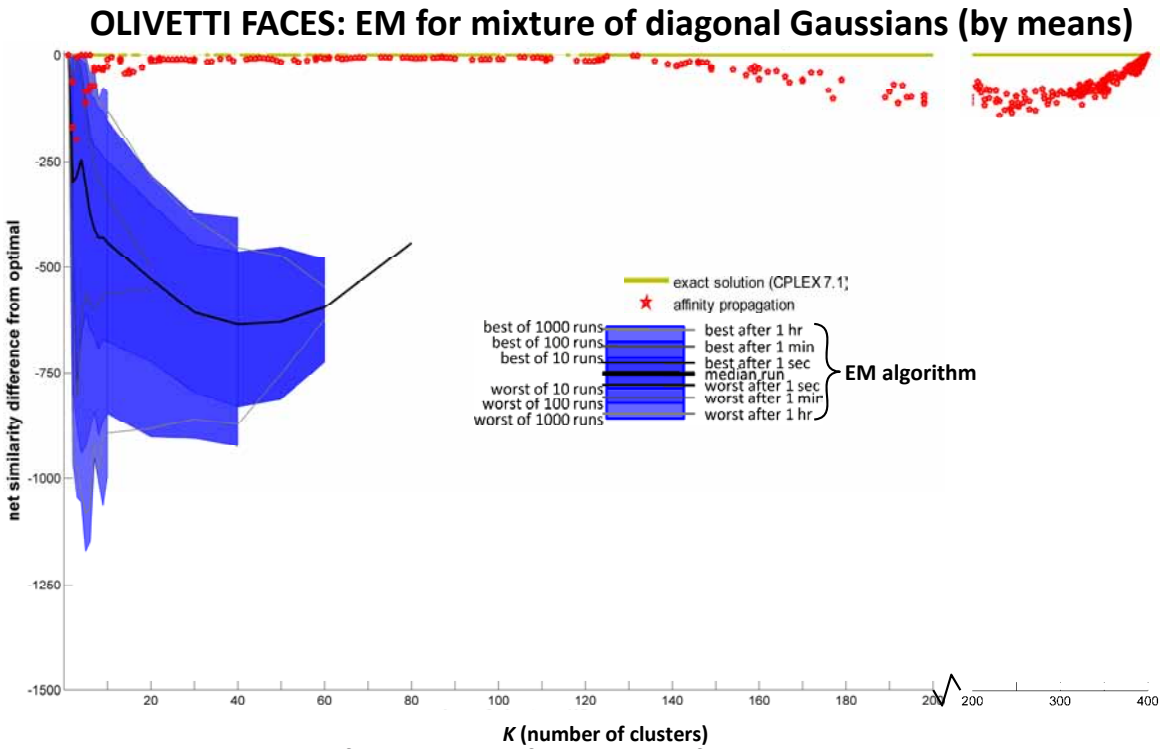


Figure 4.15: Olivetti faces and EM for mixture of diagonal Gaussians (by means)

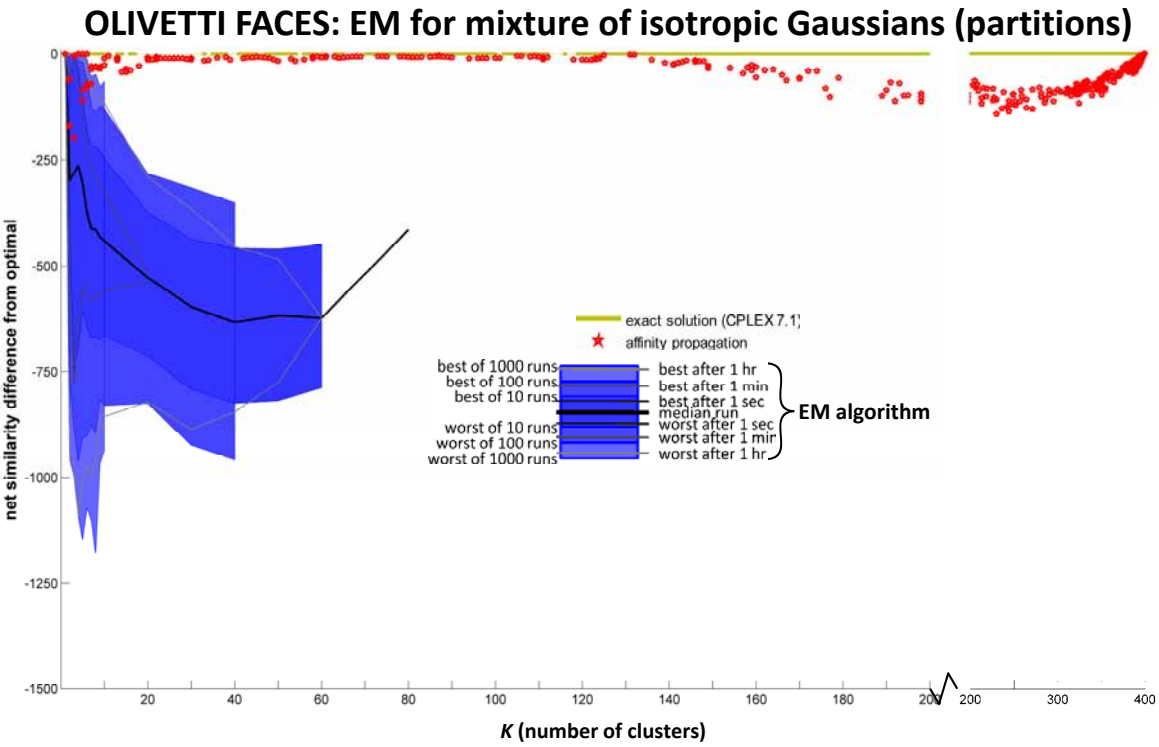


Figure 4.16: Olivetti faces and EM for mixture of isotropic Gaussians (by partitions)

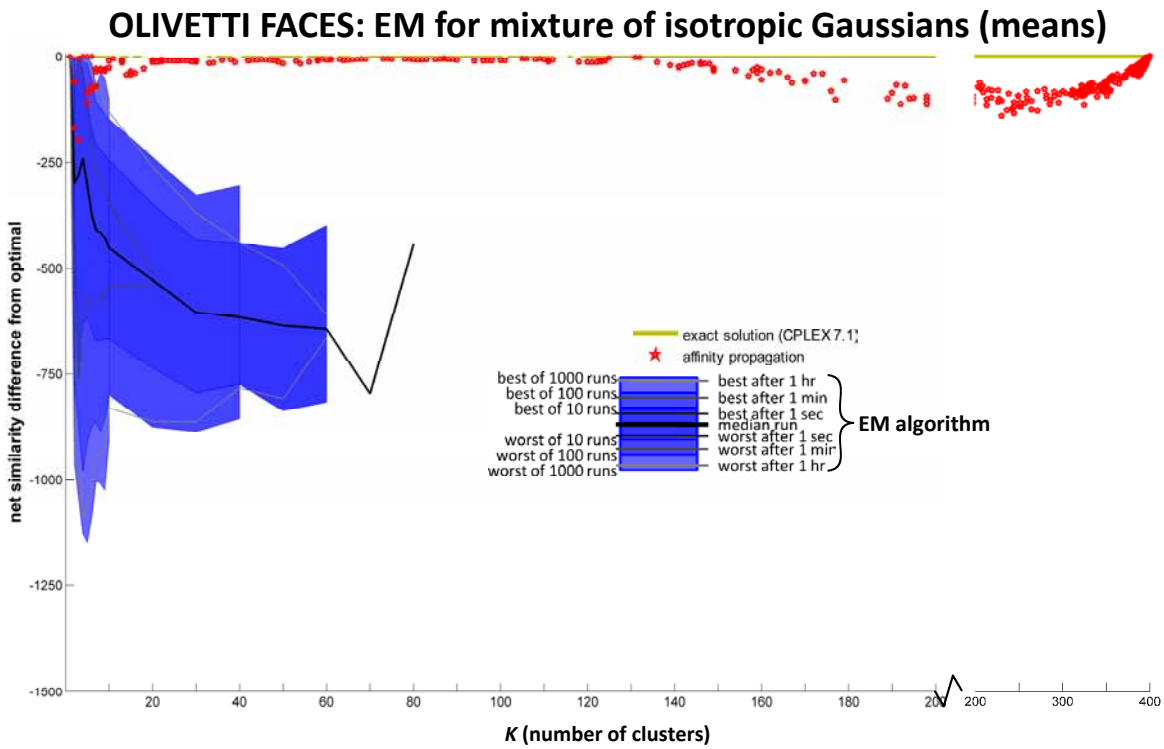


Figure 4.17: Olivetti faces and EM for mixture of isotropic Gaussians (by means)

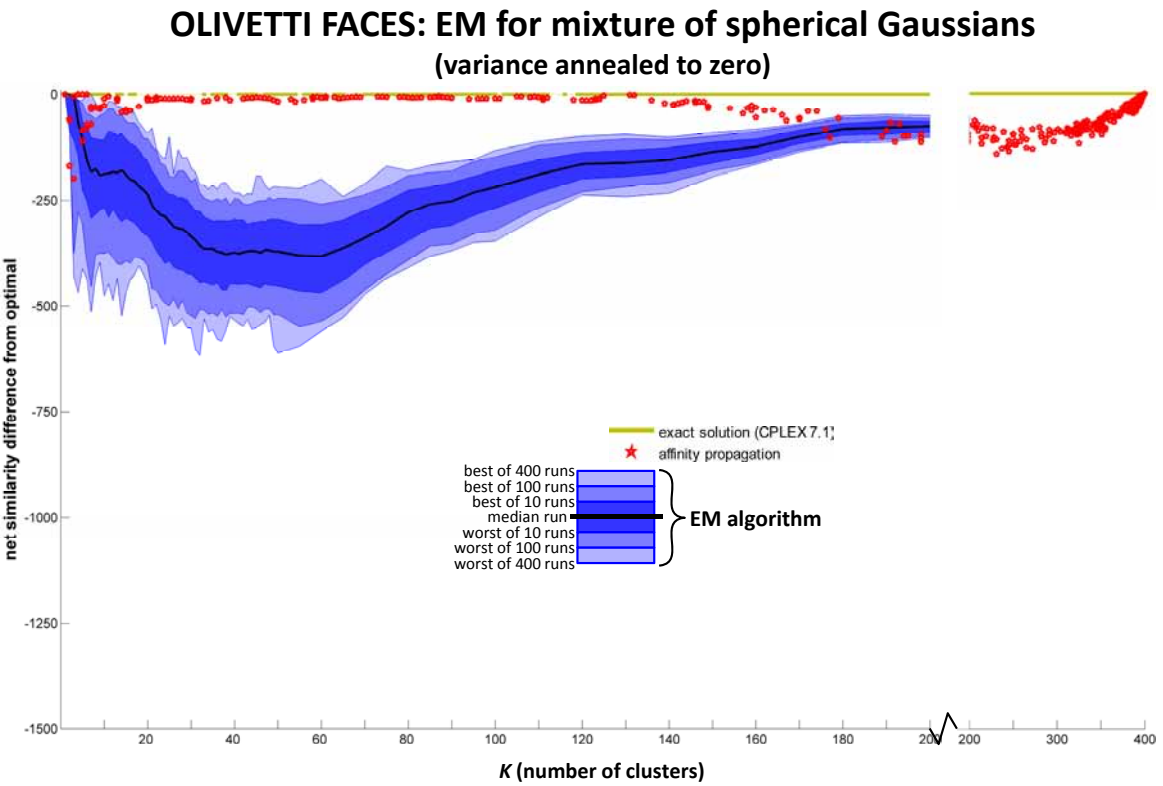


Figure 4.18: Olivetti faces and EM for mixture of spherical Gaussians (annealed variance)

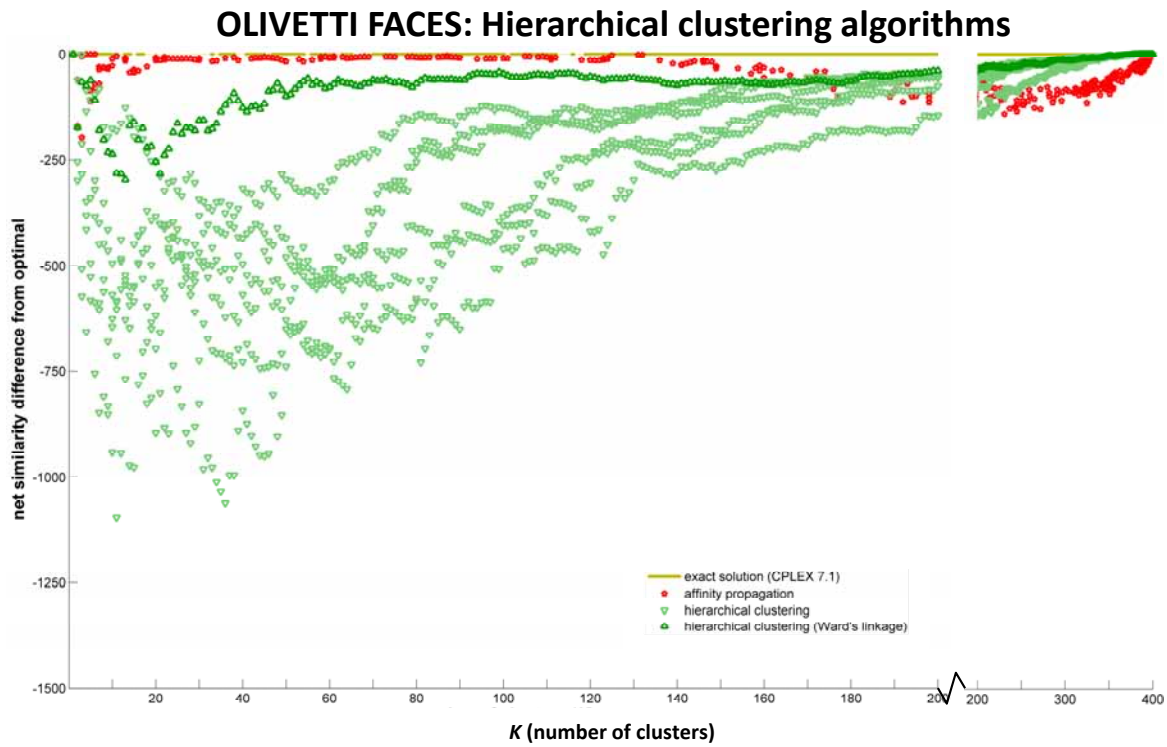


Figure 4.19: Olivetti faces and hierarchical agglomerative clustering algorithms

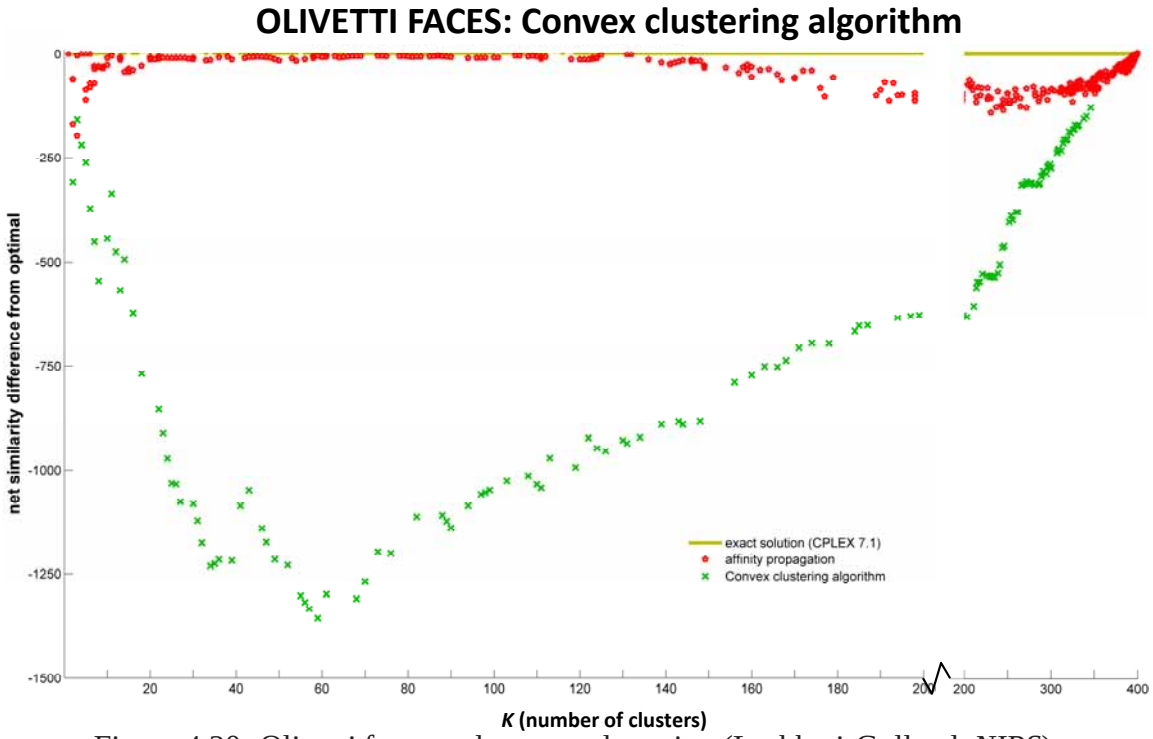


Figure 4.20: Olivetti faces and convex clustering (Lashkari-Golland, *NIPS*)

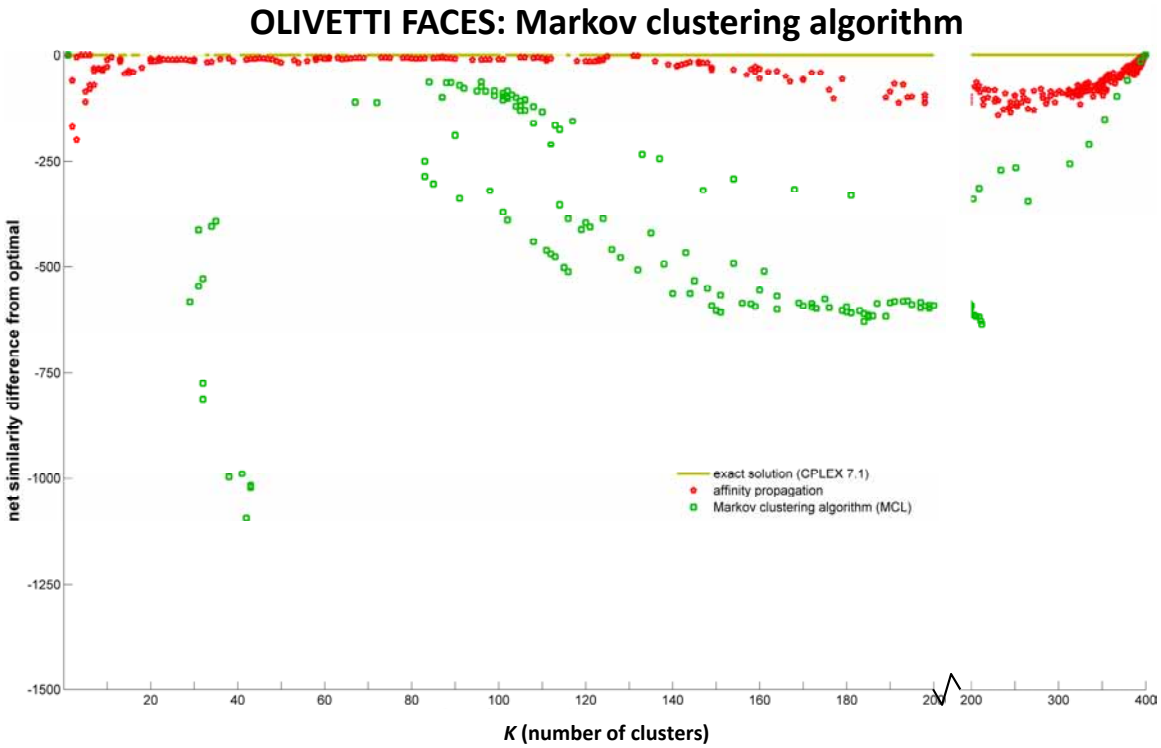


Figure 4.21: Olivetti faces and Markov clustering algorithm (van Dongen)

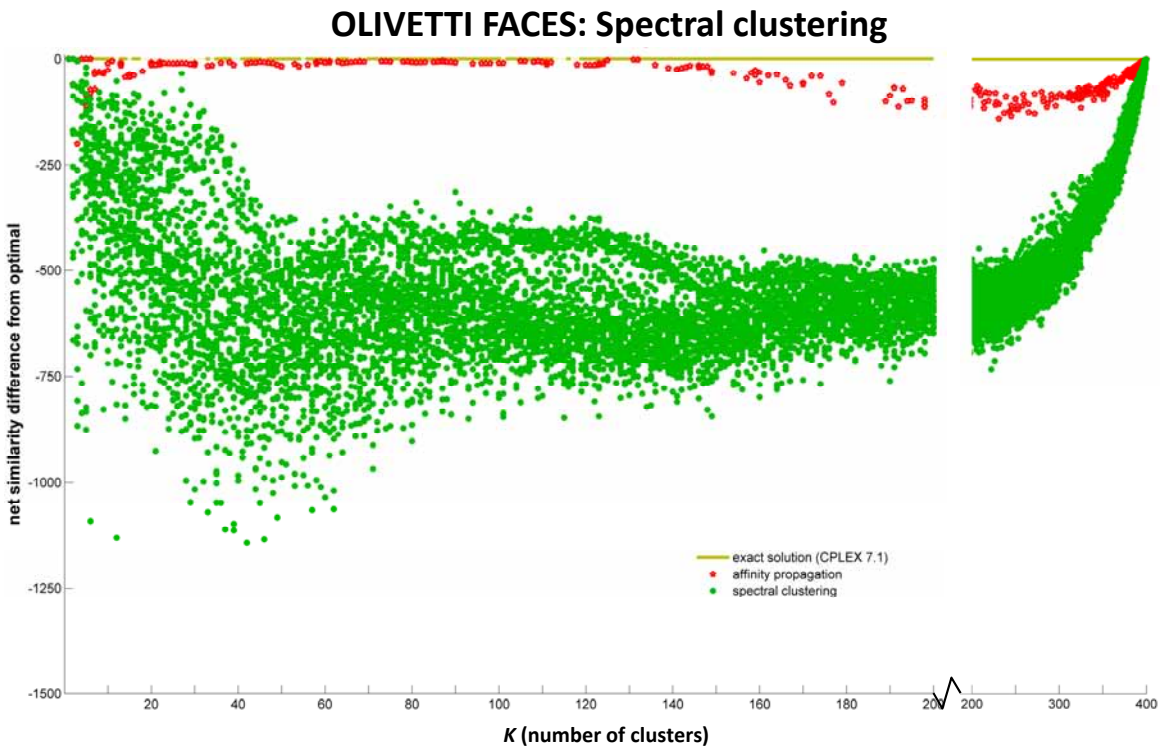


Figure 4.22: Olivetti faces and spectral clustering

4.1.4 Affinity Propagation and Mixture of Gaussians models

The previous section demonstrates that, for the Olivetti faces dataset, affinity propagation performs significantly better than parametric clustering methods (k-means, EM for mixture of Gaussians) on the task of exemplar-based clustering. A question that naturally arises is: how well does affinity propagation perform when the task is parametric clustering? For example, if cluster centers were parameterized by means and not constrained to be actual data points, means might better-describe high-dimensional Gaussian-distributed data, where the data would tend to lie near a hollow hypersphere surrounding the mean.

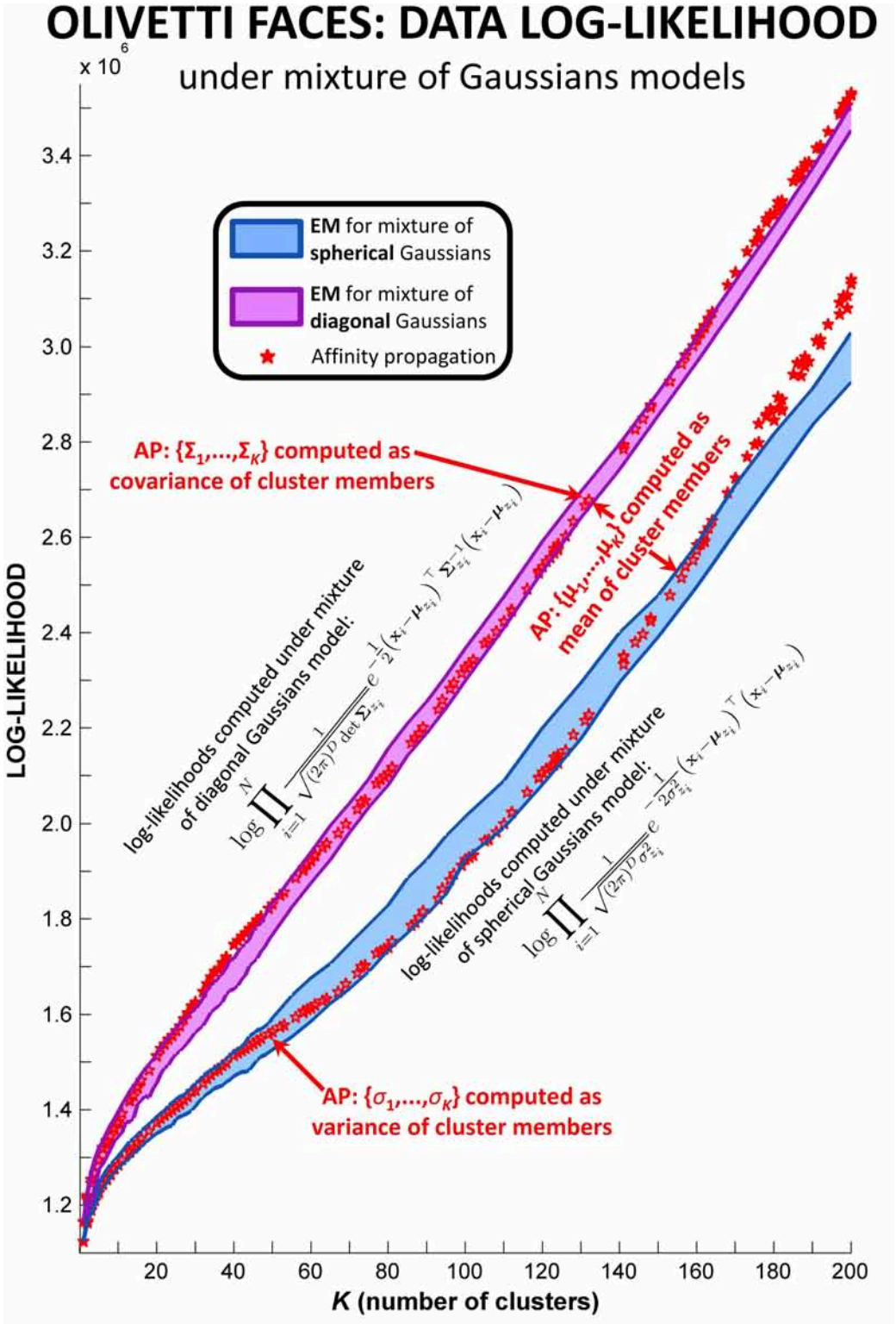


Figure 4.23: Olivetti faces are clustered using parametric clustering methods (EM for mixture of spherical, diagonal Gaussians) and data log-likelihoods are compared with corresponding mixture of Gaussians models learned from affinity propagation's clustering results.

For the Olivetti faces data, EM for a mixtures of diagonal and spherical (*i.e.* isotropic) Gaussians was run for a range of K-values and the resulting data log-likelihoods are shown in Figure 4.23. The plot shows the performance range using all possible (N) furthest-first traversals to initialize at each setting of K. Then, the previous results of affinity propagation are compared by computing means and covariances[7] from the clustering partition and evaluating the data likelihood under the apppropriate mixture model. All algorithms use the same lower bound for variances (10^{-3}), which is frequently invoked when computing variances from singleton clusters that consistly arise beyond $K > 150$ (due to the output partitioning of affinity propagation containing hard assignments).

Notice that for this dataset, affinity propagation does a decent job at finding Gaussian mixture models, even though it is optimizing an entirely different exemplar-based clustering cost function more comparable to a set of spherical Gaussians with a single (global) variance. For all values of K, affinity propagation falls within the performance range of furthest-first traversal initializations of the EM algorithm. Notably, for mixtures of diagonal Gaussians in the range $30 \leq K \leq 50$, affinity propagation's performance exceeds even the best inialization of EM.

4.2 Affinity Propagation and Large Datasets

The results presented in the previous section and in [10] indicate that for small datasets ($N < 1000$) such as the Olivetti faces, the vertex substitution heuristic frequently outperforms affinity propagation. However, because CPLEX optimization software can be used to find exact solutions for such small datasets, a question that naturally arises is: "how do affinity propagation and the vertex substitution heuristic compare for larger problems, where exact clustering is not practically feasible?". In this section, affinity propagation is comprehensively compared to the vertex substitution heuristic, k-medoids, k-means, EM for mixtures of Gaussians, hierarchical

[7] The covariance matrices for affinity propagation are computed to be of the same form (*e.g.* diagonal, isotropic) as the mixture of Gaussians models to which comparisons are made.

clustering, and spectral clustering on three datasets containing many thousands of data points.

In the case of larger datasets, comparison is less straightforward for two reasons:

- Affinity propagation (and CPLEX linear programming) use the input preference to find a set of exemplars to maximize the net similarity. Other algorithms explicitly require the number of exemplars as input and then attempt to maximize the similarity of the data to an exemplar set of that size. With the exception of the computationally-infeasible linear programming relaxation, the two optimizations are not interchangeable; many runs of affinity propagation could be required in a possibly-fruitless bisection search for a preference leading to a specified K-value. On the other hand, a similar search over K-values (potentially with many restarts) would be required for other algorithms (*e.g.* k-medoids) to minimize $\mathcal{S}$ if a preference is specified. This was not a problem with the smaller Olivetti dataset, where algorithms could be run for all possible K-values or many preferences (leading to most if not all K-values).

- The exact solution is unavailable as a baseline for comparison so solution quality comparisons over a range of K- or p-values cannot be in absolute terms. Differences between $\mathcal{S}_{\text{data}}$ or $\mathcal{S}$ vary across several orders of magnitude for different values of K or p (see Figure 4.3), so performance differences between algorithms without perfectly-matched K are not visible at a global scale without subtracting a nearby baseline such as the optimal value (as in Figures 4.7 to 4.22). For consistency, the figures in this section use an approximate baseline equal to the median performance of 100 runs of k-medoids clustering, which is easy enough to be computed for all [thousands of] possible values of K.

Another issue with larger datasets is that computation time is a more important variable in assessing algorithm performance and so plots need to reflect this extra dimension. Plotting in three dimensions—as illustrated in Figure 4.25—is cluttered and lacks perspective, so two-dimensional slices across notable points on the time (one minute, one hour, one day, one week)

axis are shown instead. Each plot contains a smooth curve with several dozen p- or K-values, where the number of clusters varies approximately from $K = 10$ to $K = N/2$, as shown on the horizontal axes. Slices along the K- or p-axes were shown in [35] but require the number of clusters to be consistent across all algorithms; this would be somewhat arbitrary, given that affinity propagation's solution only finds an approximation.

The datasets examined consist of measurements for 8124 mushrooms, 11,000 handwritten digits, and customer ratings for 17,770 movies from Netflix. Similarities range from negative squared Euclidean-distance in a 64-dimensional data space for the handwritten digits, to uniform-loss embeddable in 121-dimensional space for mushrooms, to extremely-high dimension as is the case for the sparse movie ratings data.

4.2.1 Mushroom data ($N = 8124$)

The mushroom dataset [90] is available at the UCI Machine Learning Respository[8]. It contains 22 discrete, non-ordinal attributes for each of 8124 mushrooms as illustrated in Figure 4.24. These attributes include color, shape, and surface characteristics of the mushrooms' cap, gill, and stalk—the stalk root attribute is ignored because it is more than 30% incomplete. The similarity function uses the uniform loss as shown in equation (4.1) which counts the number of matching attributes between each mushroom, meaning that similarities are integers in the range $0 \leq s(i,k) \leq 21$.

$$s(i,k) = \sum_{j=1}^{21} \left[\left(i^{\text{th}} \text{ mushroom's } j^{\text{th}} \text{ attribute} \right) = \left(k^{\text{th}} \text{ mushroom's } j^{\text{th}} \text{ attribute} \right) \right] \qquad (4.1)$$

Simulations were carried out for the same set of clustering algorithms as in Section 4.1 and are shown in Figures 4.26–4.27. Simulations were carried out using algorithms such as k-means (with and without the $k \cdot \log(k)$ heuristic, finding exemplars by initializing k-medoids

[8] Located at http://mlearn.ics.uci.edu/MLRepository.html

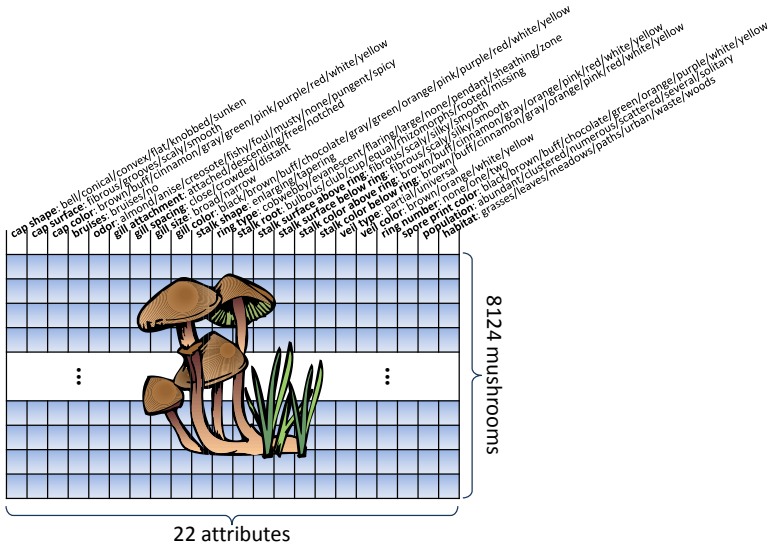


Figure 4.24: The mushrooms dataset contains 22 discrete, non-ordinal attributes for each of 8124 mushrooms, as illustrated above.

with the means and by the cluster partitions), EM for mixture of Gaussians (using diagonal and isotropic/spherical Gaussians, finding exemplars by means and by partitions), hierarchical clustering, and spectral clustering. These techniques rely on the data being embedded in a relatively low-dimensional vector space which is accomplished by a one-hot representation of each attribute; this expands the data to a set of 121-dimensional binary-valued vectors.

Hierarchical clustering required roughly an hour for each of the seven linkage methods to form the linkage tree structure; then each K-value required several seconds to find the exemplars using k-medoids initialized with the cluster partitioning. The results were not competitive and required several gigabytes of memory – more than other algorithms (*e.g.* affinity propagation required about 1.6 GB of RAM). As mentioned earlier, results for a range of computation times and K- or p-values are shown in the 3D plot of Figure 4.25.

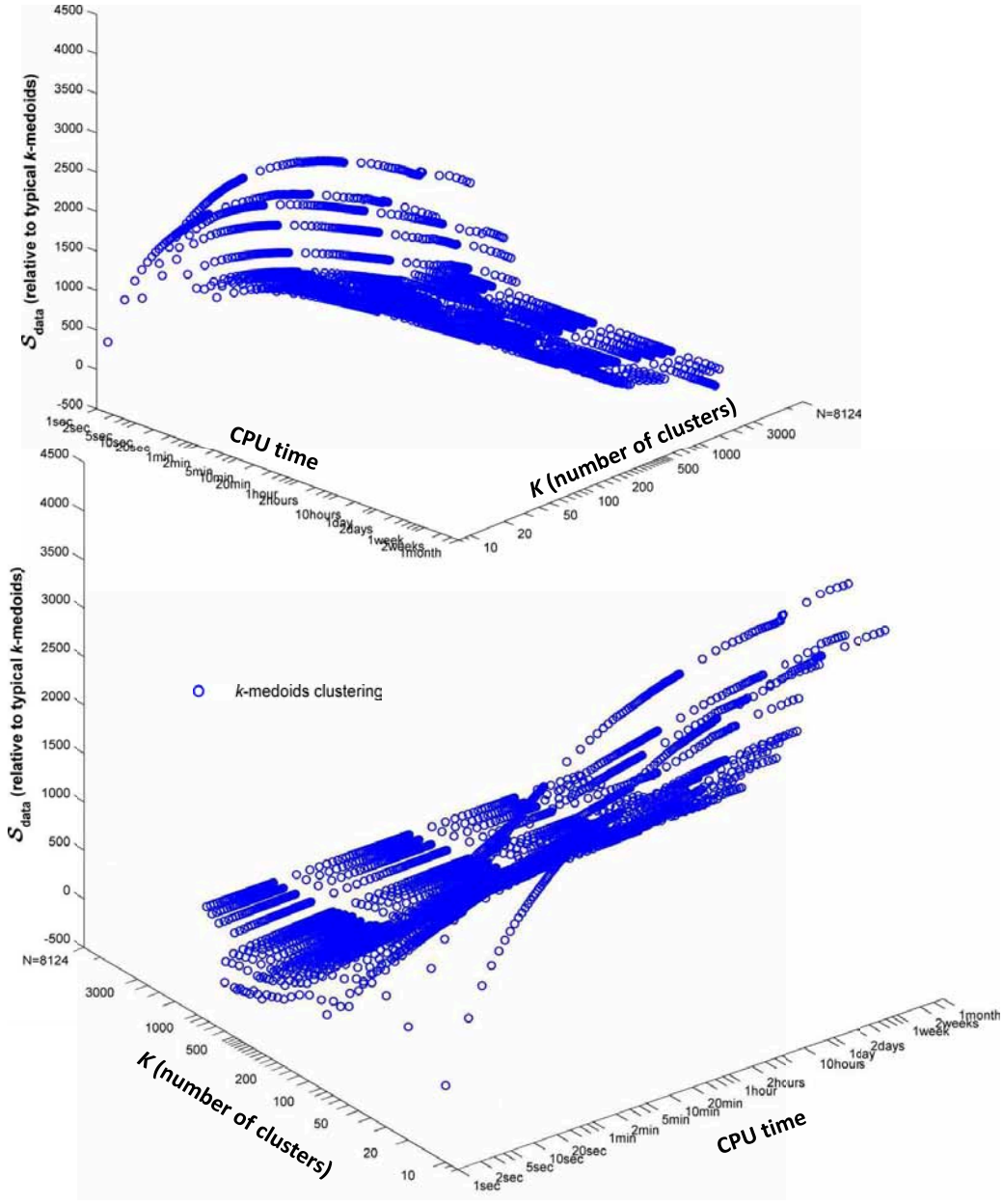
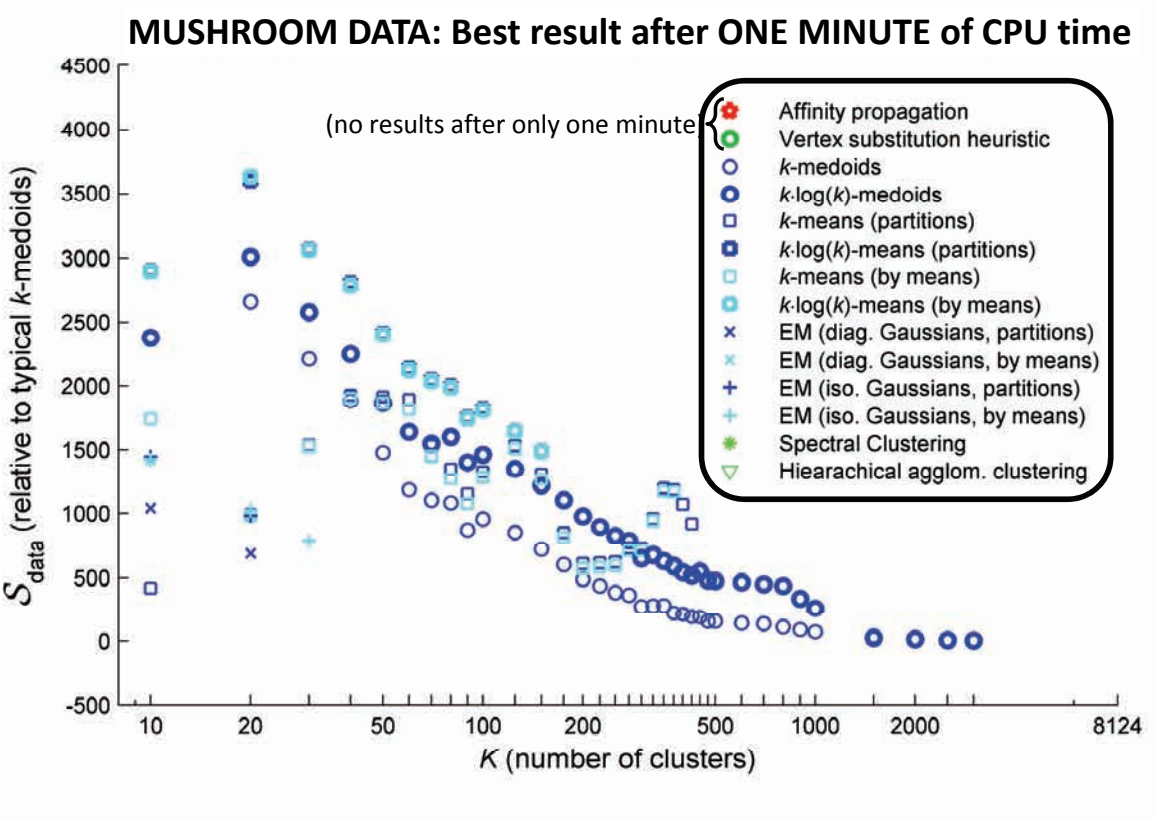


Figure 4.25: Results for k-medoids with a range of computation times and p- or K-values (preferences or number of exemplars) are shown in the 3D plot above. The plot is a useful illustration of the parameters involved in the algorithm and the issues at play in visualizing the results, but it is not informative for algorithm comparison (consider the appearance if additional algorithms were shown). For this, 2D plots that are cut along noteworthy slices of the CPU time-axis are shown in Figures 4.26–4.28.



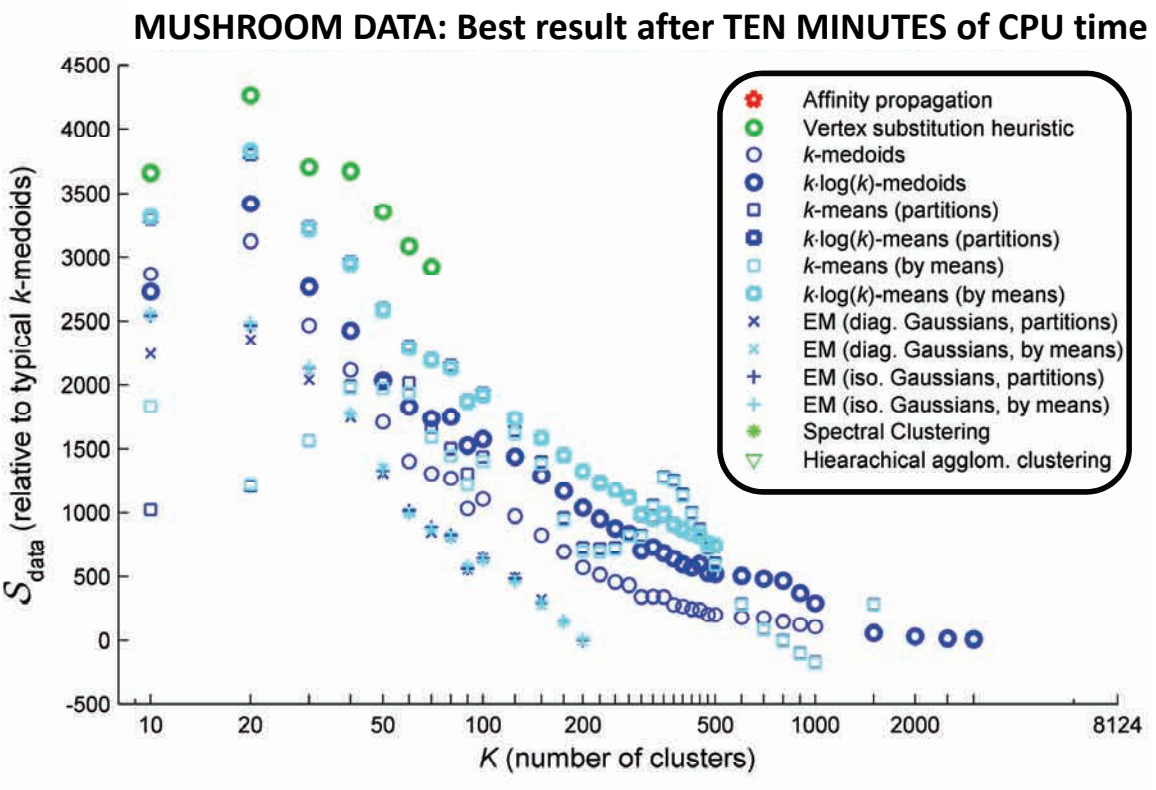


Figure 4.26: Performance comparison for mushrooms dataset after minutes of computation.

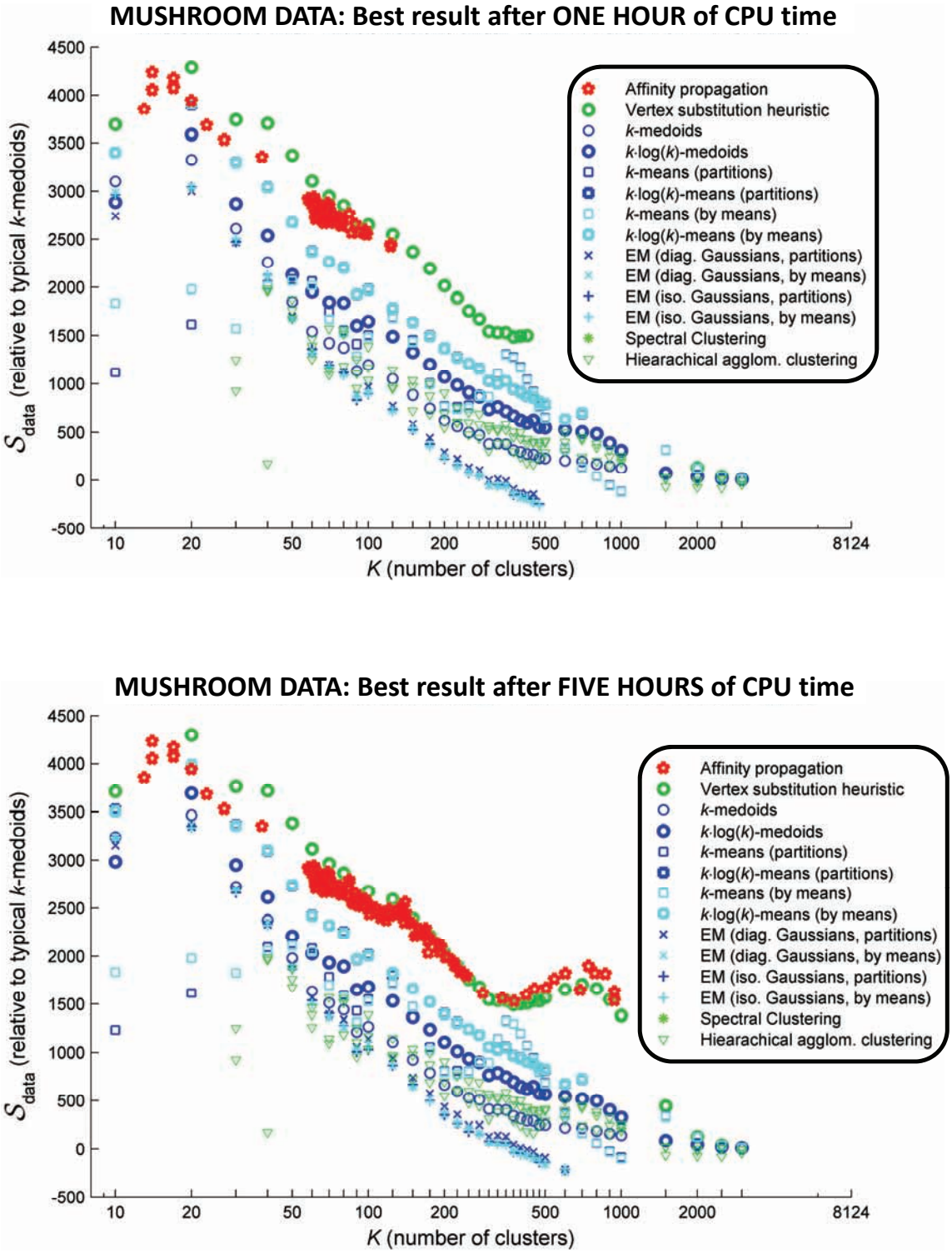


Figure 4.27: Performance comparison for mushrooms dataset after hours of computation.

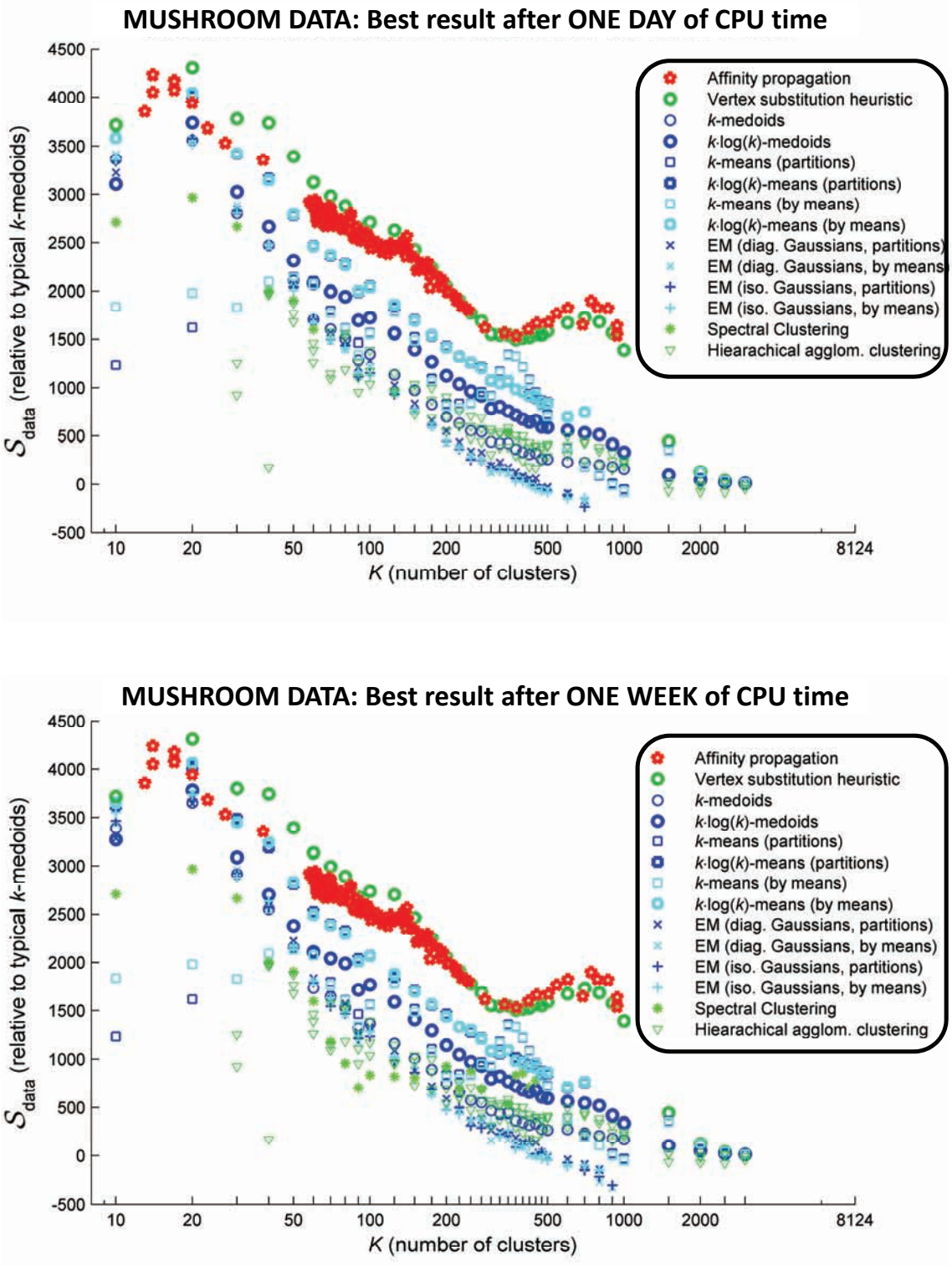


Figure 4.28: Performance comparison for mushrooms dataset after days of computation.

After a total of 3 years of single-CPU computation time spent on the mushrooms dataset, Figures 4.26–4.28 show that among the algorithms examined and under most circumsances, only the vertex substitution heuristic achieves results that are competitive with affinity propagation. Spectral clustering achieves fairly good solutions, but requires many days of computation time as it searches for appropriate values of the σ parameter.

Affinity propagation and the vertex substitution heuristic are examined more directly in Figure 4.29, which displays results on the same axes as Figures 4.26–4.28. Computation times are also shown, and affinity propagation usually requires 40–60 minutes whereas the vertex substitution heuristic requires several minutes for small $K < 50$ and up to two hours for larger $K > 500$.

4.2.2 USPS digits ($N = 11000$)

The next-largest dataset examined consists of 11,000 handwritten digits originating from United States Postal Service ZIP code scans from Buffalo, NY[9]. This is in the form of 1100 8×8 greyscale images of each handwritten digit '0' through '9', for a total of 11,000 images / data points—the dataset in its entirety is shown in Figure 4.30. The similarity between two images was set to the negative sum of squared pixel differences, which, like the Olivetti data, implies k-means and mixture of Gaussians parametric models *can* directly minimize the similarity (albeit with means instead of exemplar data points). Note that $K = 10$ is not necessarily the proper choice for clustering this data; even though a useful classifier might assign one of ten possible labels if it classifies each digit image according to its associated numeral, clustering as an earlier step in the analysis can very well find significantly more than ten digit exemplars.

[9] Available at http://www.cs.toronto.edu/ roweis/data.html

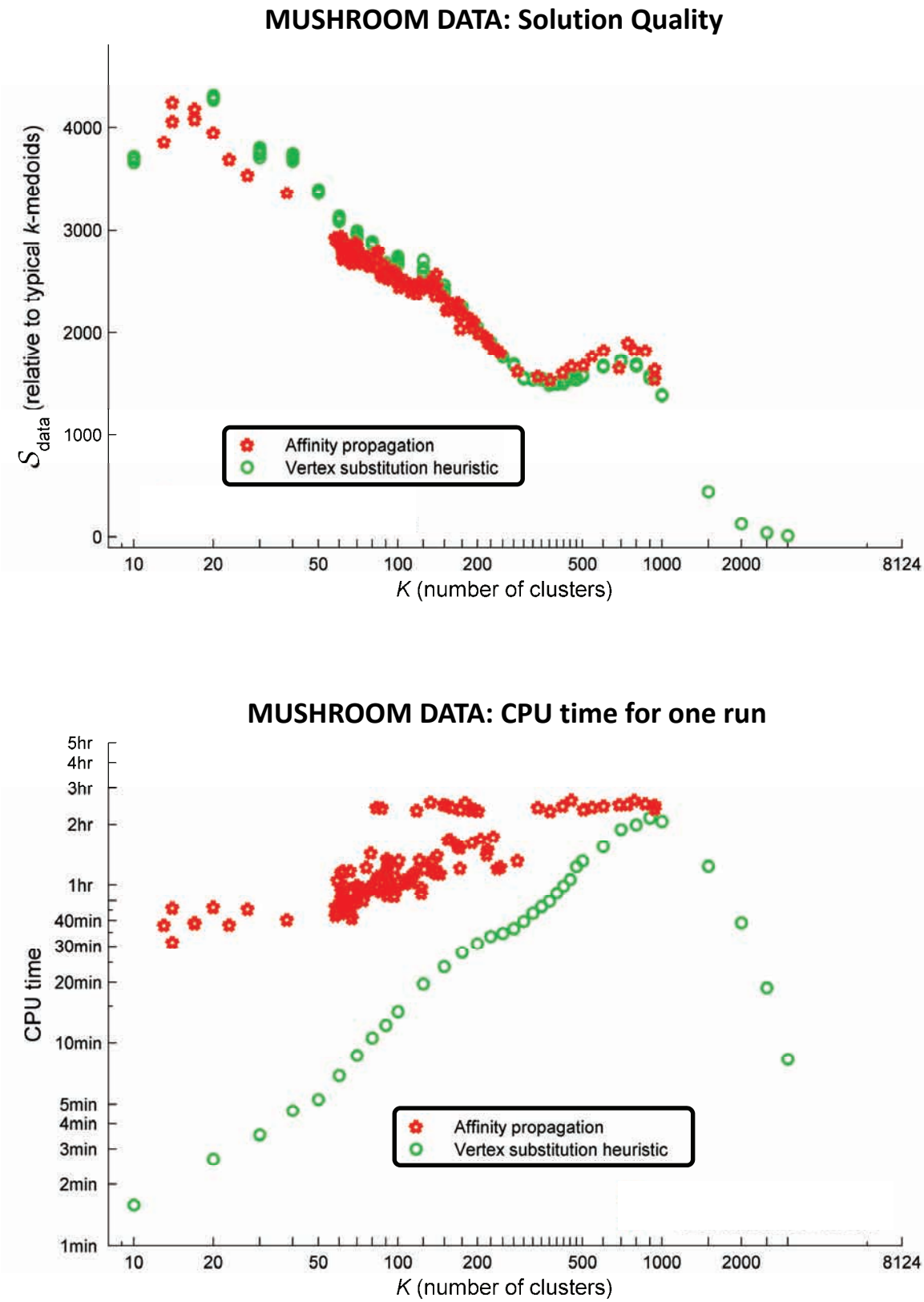


Figure 4.29: Performance comparison for mushrooms dataset between affinity propagation and the vertex substitution heuristic.

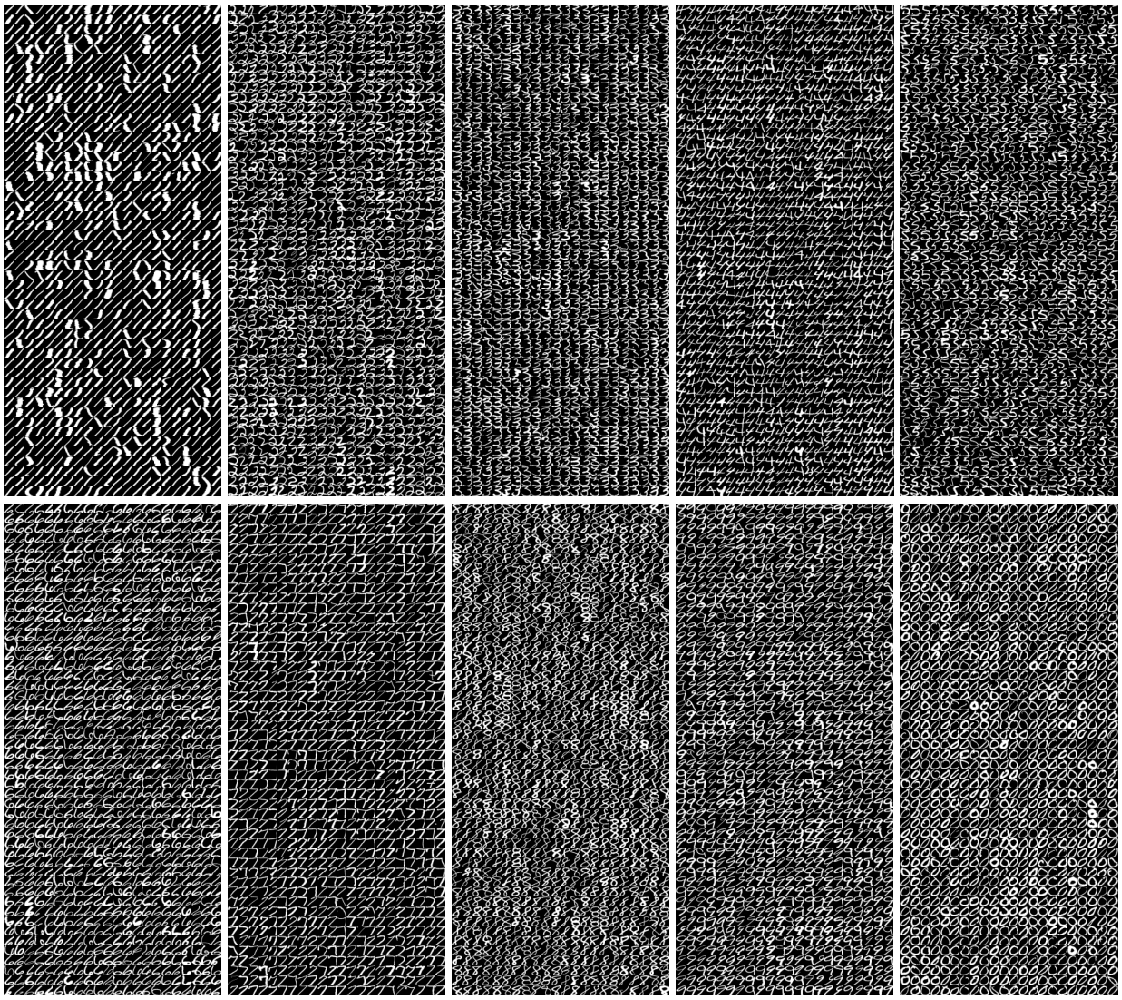
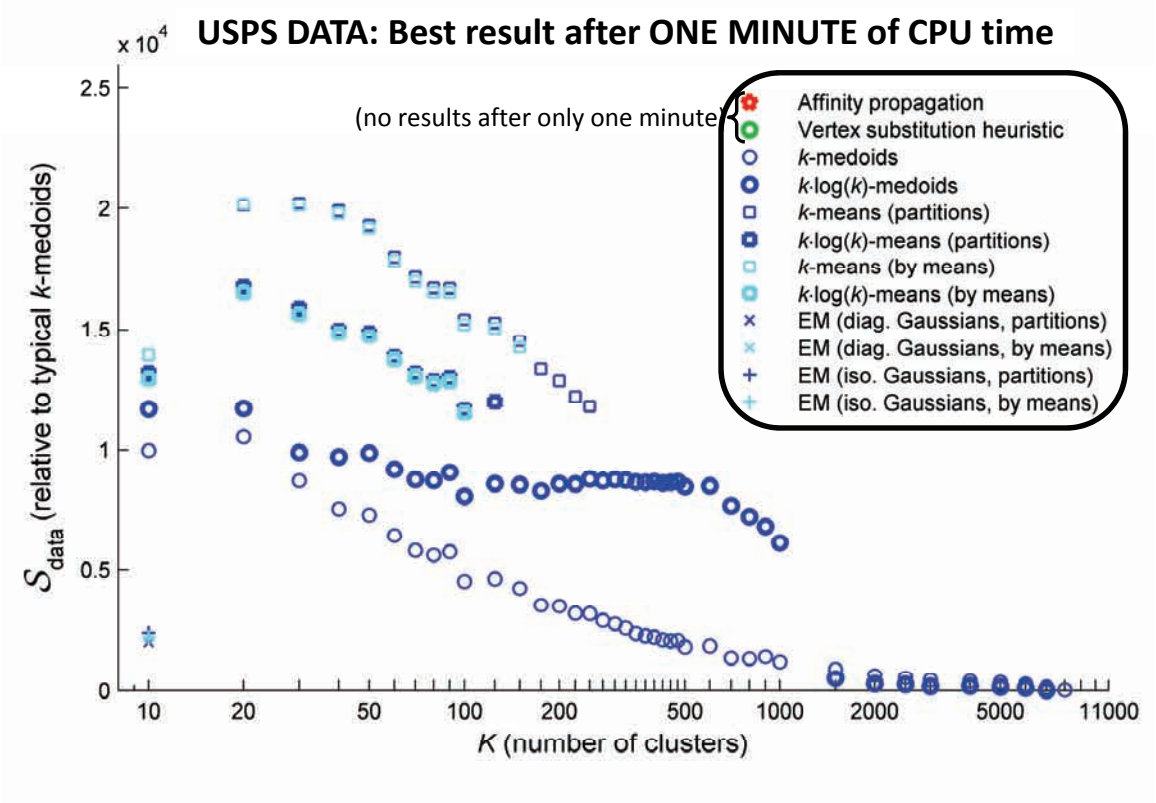


Figure 4.30: The USPS digits dataset consists of 11000 8×8 images of digits scanned from ZIP codes in the Buffalo, NY office, as shown above.



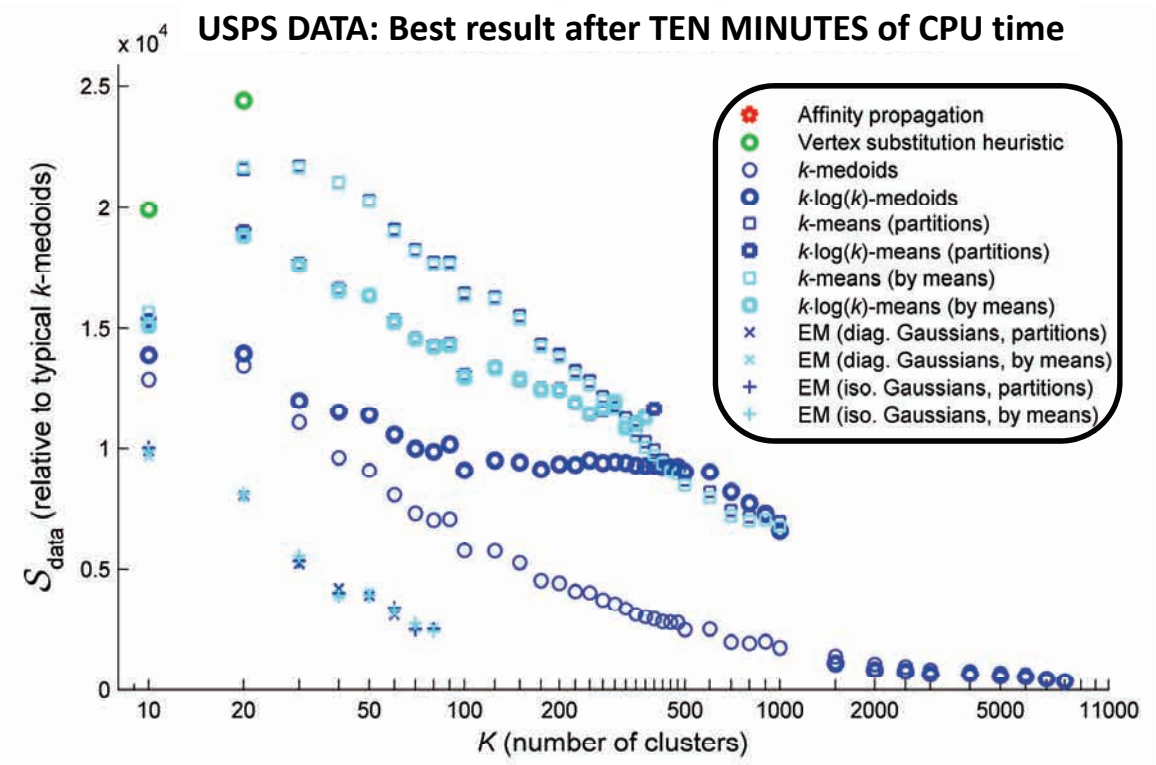


Figure 4.31: Performance comparison for USPS dataset after minutes of computation.

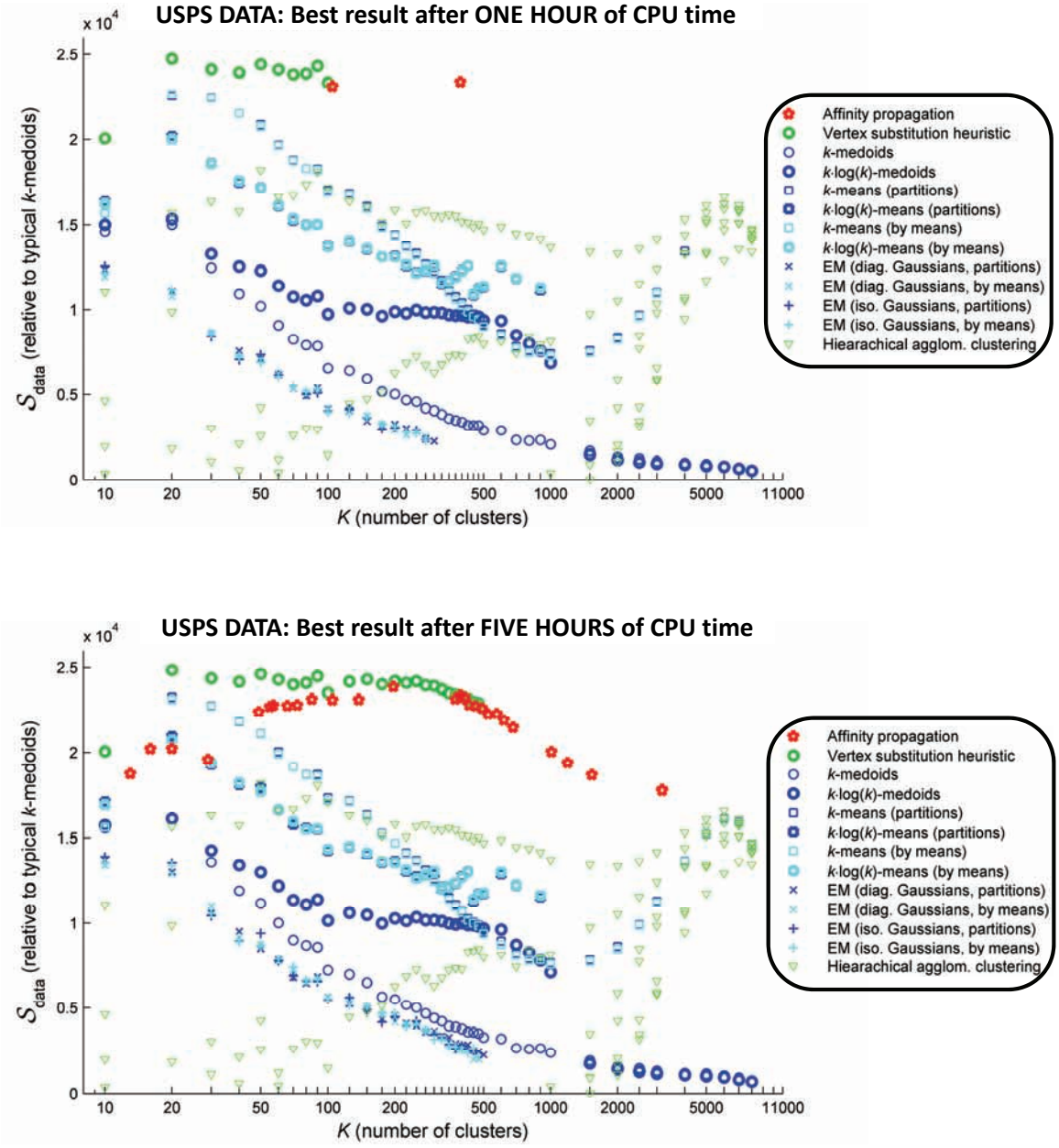


Figure 4.32: Performance comparison for USPS dataset after hours of computation.

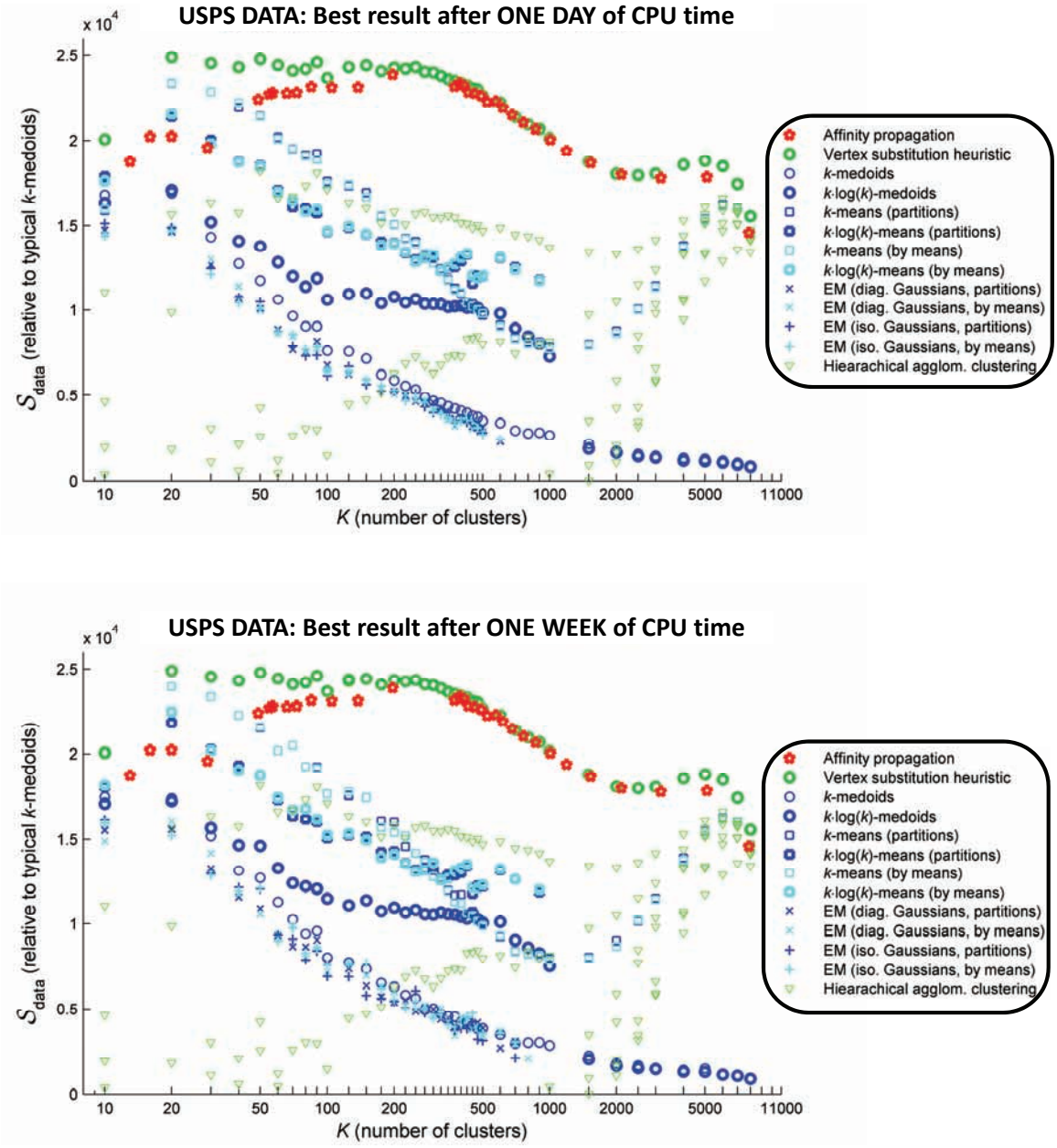


Figure 4.33: Performance comparison for USPS dataset after days of computation.

Simulations were carried out for the same set of clustering algorithms as in Section 4.2.1 and are shown in Figures 4.31–4.32, except for spectral clustering which did not produce results within a week of CPU time. Some techniques (k-means, EM) rely on the original data in the form of an 11000×256 data matrix rather than the similarity matrix.

Hierarchical clustering required somewhat less than an hour of computation time for each of the seven linkage methods to form the linkage tree structure; then each K-value required several seconds to find the exemplars using k-medoids initialized with the cluster partitioning. The results turn out to be competitive only for extremely large numbers of clusters *i.e.* $K > N/2$. After a total of ten years of single-CPU computation time spent analyzing the USPS dataset, the main competitor for the affinity propagation algorithm is the vertex substitution heuristic. After a few minutes of CPU time (before VSH and affinity propagation report results), the k-means algorithm achieves quite competitive results that are even better than affinity propagation for small $K < 50$; this is most likely because it is directly minimizing squared Euclidean distance which happens to be the definition of (negative) similarity for this problem.

As before, affinity propagation is compared directly with the vertex substitution heuristic in Figure 4.34. Computation times are shown, and affinity propagation requires several hours whereas one run of the vertex substitution heuristic requires a varying amount of time that depends on search space size; ranging from several minutes for low values of $K \leq 20$ to roughly a day for higher values of $K \geq 2000$. VSH outperforms affinity propagation in terms of both computation time and solution quality for $K \leq 60$; affinity propagation outperforms VSH in terms of speed (at times by a factor of 5), achieving solutions of similar quality beyond $K = 200$ or so.

4.2.3 Netflix movies ($N = 17770$)

The largest dataset explored is a collection of Netflix customer ratings for 17,770 movies, available at http://www.netflixprize.com and illustrated in Figure 4.35. The data consists of $\sim$100 million ratings from 480,189 Netflix customers for 17,770 movies, so the data could be

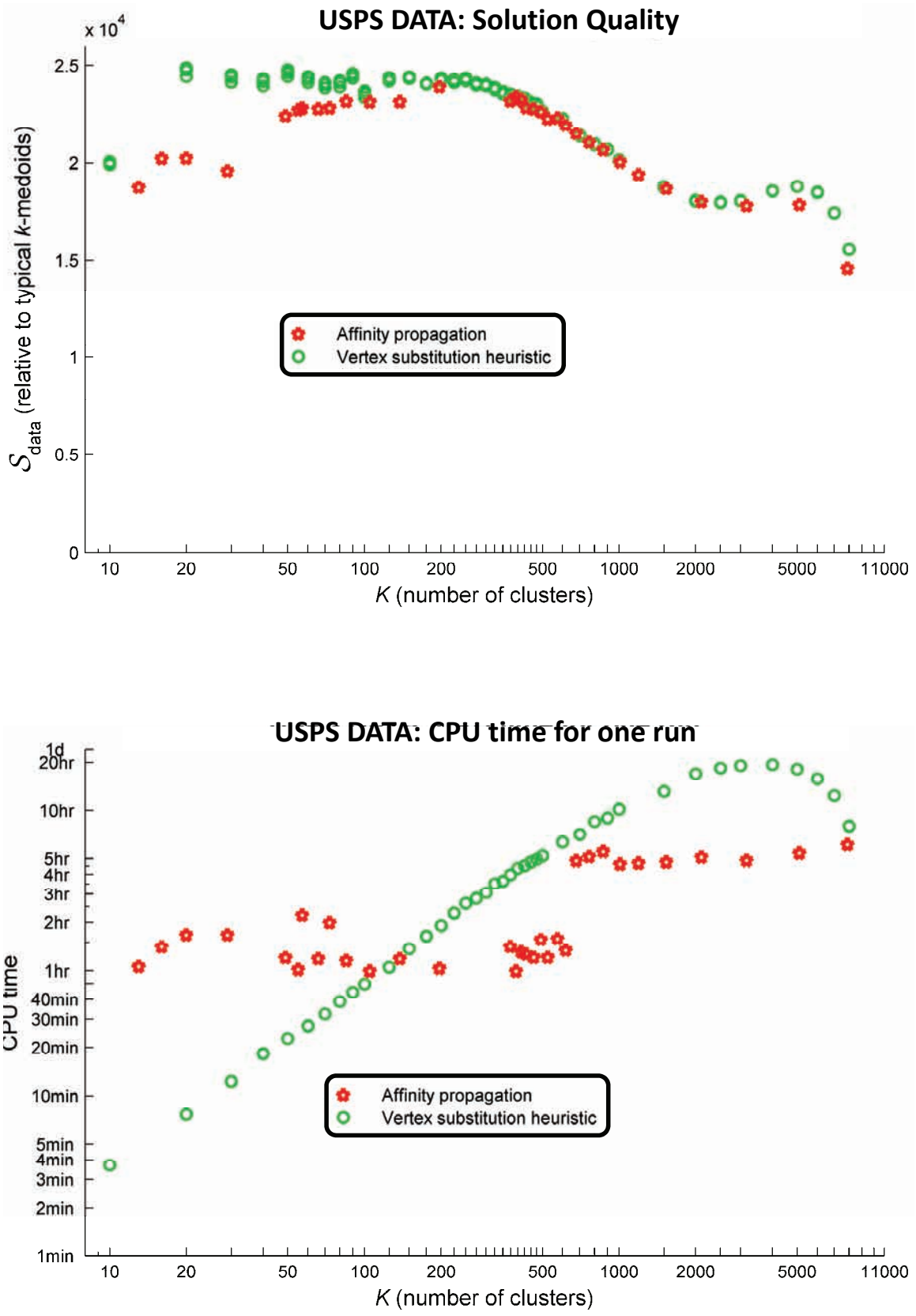


Figure 4.34: Performance comparison for USPS dataset between affinity propagation and the vertex substitution heuristic.

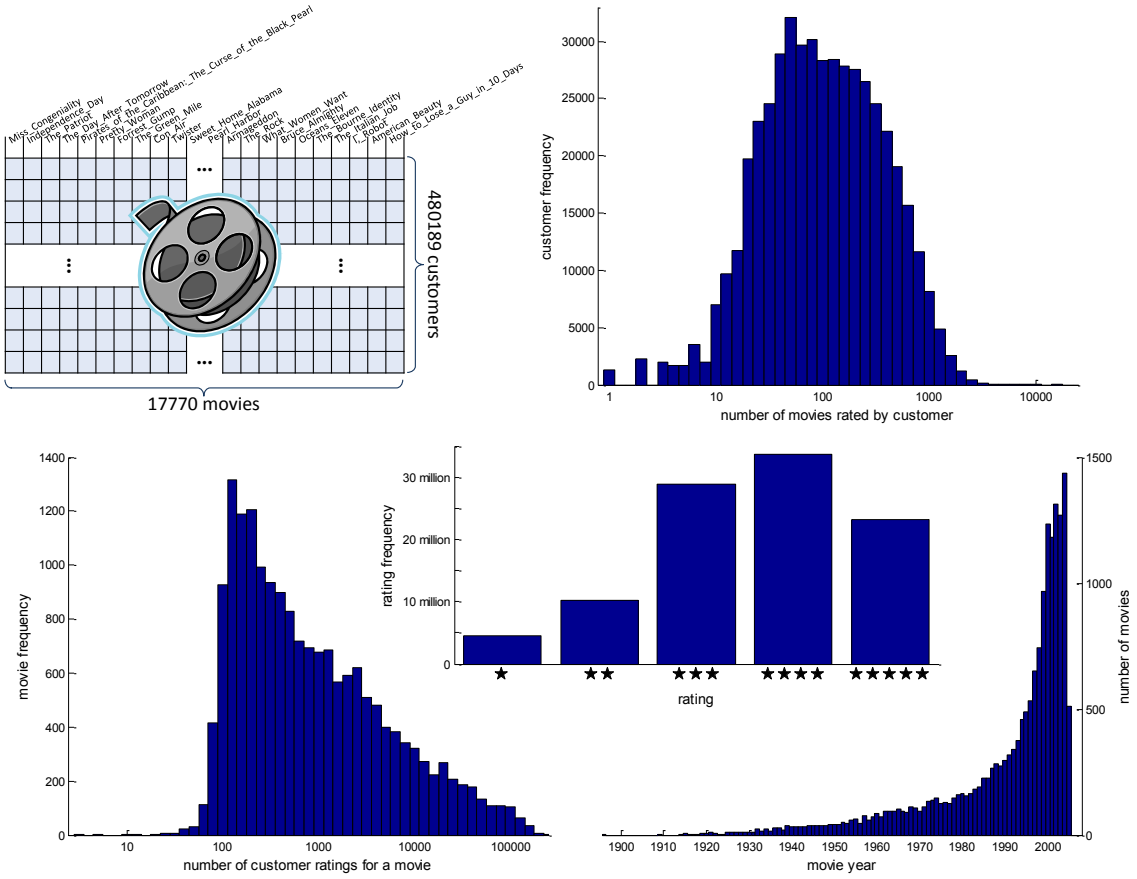


Figure 4.35: The Netflix dataset consists of ∼100 million ratings of 17,770 movies by 480,000 Netflix customers, as shown above. Most movies are rated by hundreds of customers, and likewise, most customers have rated hundreds of movies. Ratings are given in units of 'stars' ranging between $\star$ and $\star\star\star\star\star$, with the $\star\star\star\star$ rating being the most common.

represented in a 17770×480189 matrix that is ∼98.8% sparse. Typically, customers have rated roughly 100 movies and most movies have at least 100 customers ratings. Ratings are given as one through five 'stars', with $\star\star\star\star\star$ being the highest rating (used 23% of the time) and $\star$ being the lowest (used for 5% of ratings); ratings are most-often $\star\star\star$ or $\star\star\star\star$, with a frequency of roughly 30% each.

Movies are clustered using a similarity function that accounts for users giving similar ratings to similar movies; the similarity $s(i, k)$ between movie i and movie k is set to the negative mean-squared-difference between ratings for viewers common to both movies, denoted by set $\mathcal{C}_{ik}$. Ratings for customers who have viewed one movie but not the other are ignored; for

movies with very few or no common viewers, similarities are regularized as described in equation (4.2). Briefly, if there are fewer than five common user ratings then the expression is regularized by padding with enough simulated rating differences of 2 to equal five common user ratings (this is required for 5.3% of movie pairings). Movie pairs with no common users are assigned a similarity of $-\infty$ (necessary for only 0.2% of cases).

$$s(i,k) = \begin{cases} -\infty,\ \mathcal{C}_{ik} \equiv \emptyset \quad (0.2\%\ \text{of cases}) \\ -\frac{\sum_{c \in \mathcal{C}_{ik}} (x_{ic} - x_{kc})^2 + 2^2 \cdot (5 - |\mathcal{C}_{ik}|)}{5},\ 0 < |\mathcal{C}_{ik}| < 5 \quad (5.3\%\ \text{of cases}) \\ -\frac{\sum_{c \in \mathcal{C}_{ik}} (x_{ic} - x_{kc})^2}{|\mathcal{C}_{ik}|},\ |\mathcal{C}_{ik}| \geq 5 \quad (94.5\%\ \text{of cases}) \end{cases} \tag{4.2}$$

Simulations were carried out for clustering algorithms that take similarities as input—k-medoids (including the $k \cdot \log(k)$ heuristic), the vertex substitution heuristic, and affinity propagation. Because the dataset consists of 17770×17770 inputs requiring roughly 2.5 GB of memory (plus working space), the more memory-intensive algorithms such as hierarchical clustering and spectral clustering were not feasible. Simulation results from 8 years of single-node CPU time are shown in Figure 4.36–4.38, with identical horizontal and vertical scales for easy comparison.

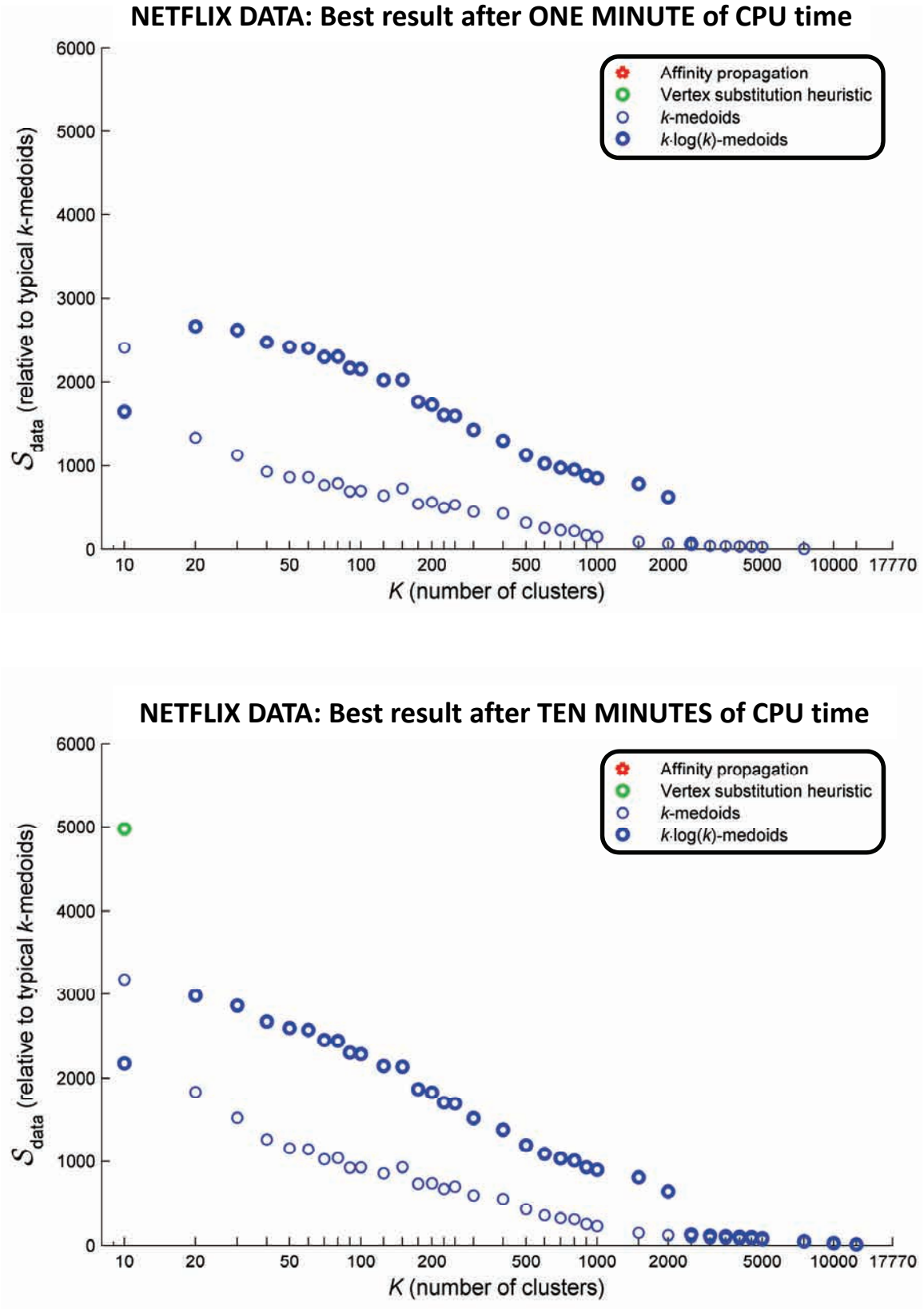


Figure 4.36: Performance comparison for Netflix dataset after minutes of computation.

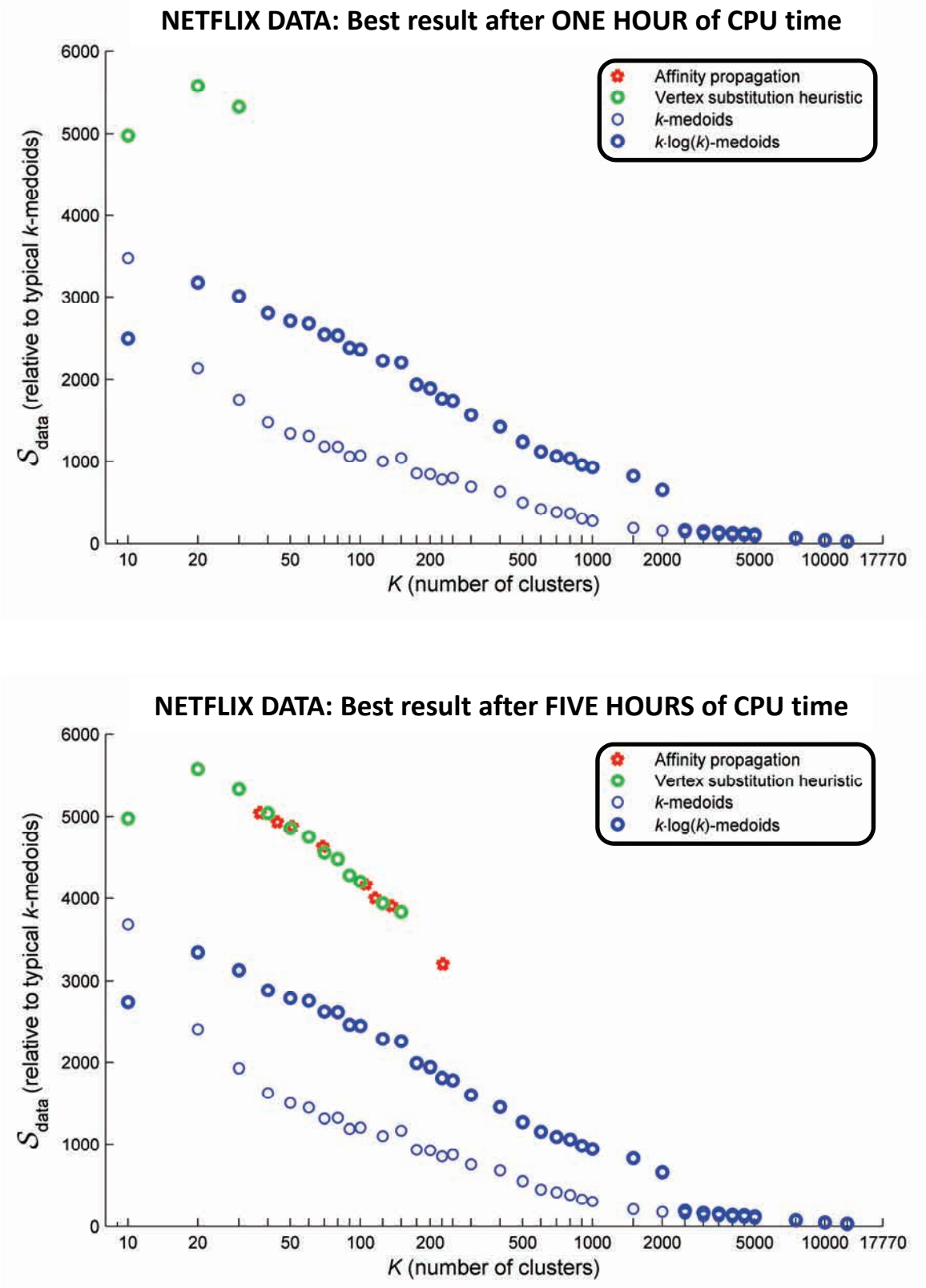


Figure 4.37: Performance comparison for Netflix dataset after hours of computation.

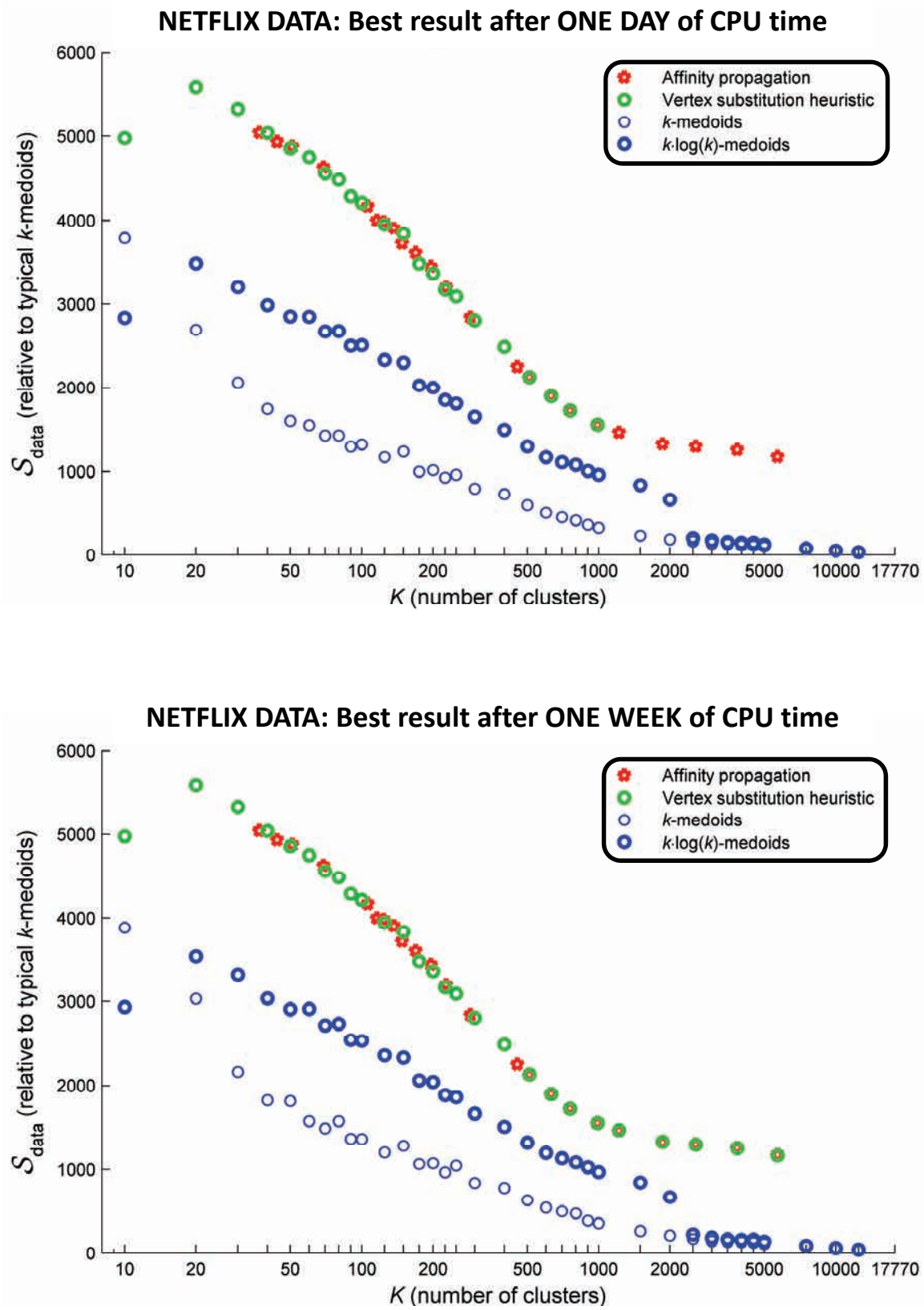


Figure 4.38: Performance comparison for Netflix dataset after days of computation.

As before, the only algorithm examined that was competitive with affinity propagation is the vertex substitution heuristic (with the exception of some regions of k-medoids reporting first results quicker than affinity propagation in the first few minutes). Another set of simulations was conducted for a range of preferences (K-values) and, additionally, for a range of N-values (number of data points) accomplished by clustering only the first N movies. Affinity propagation is compared directly with 20 random restarts of VSH[10], with results shown in Figure 4.39. The plots show that for larger datasets containing $N \geq 6000$ data points and finding $K \geq 50$ exemplars, affinity propagation clearly comes out on top in terms of both solution quality achieved and computation time. In terms of computation time, affinity propagation is much less sensitive to the number of exemplars, and for larger problems is faster than a single restart of VSH (by as much as a factor of ten for the full dataset and $K > 500$).

Varying the dataset size and number of clusters provides an opportunity to examine several algorithms' memory requirements. As shown in Figure 4.40, k-medoids has the lowest memory requirements, doing much of its computation in-place with negligble memory required beyond that for storing the input similarities. The vertex substitution heuristic has modest memory requirements that depend on the number of clusters (for keeping track of the swaps); for the full Netflix dataset it ranges from 102% of the input similarity capacity (similar to k-medoids) to 136% using the range of preferences from Figure 4.39. The affinity propagation algorithm has much greater memory requirements in keeping soft information; it requires roughly 310% of the input similarity capacity, mostly to keep equally-sized availabilities and responsibilities in memory.

[10] twenty restarts are recommended in [10] by operations research experts for a good solution.

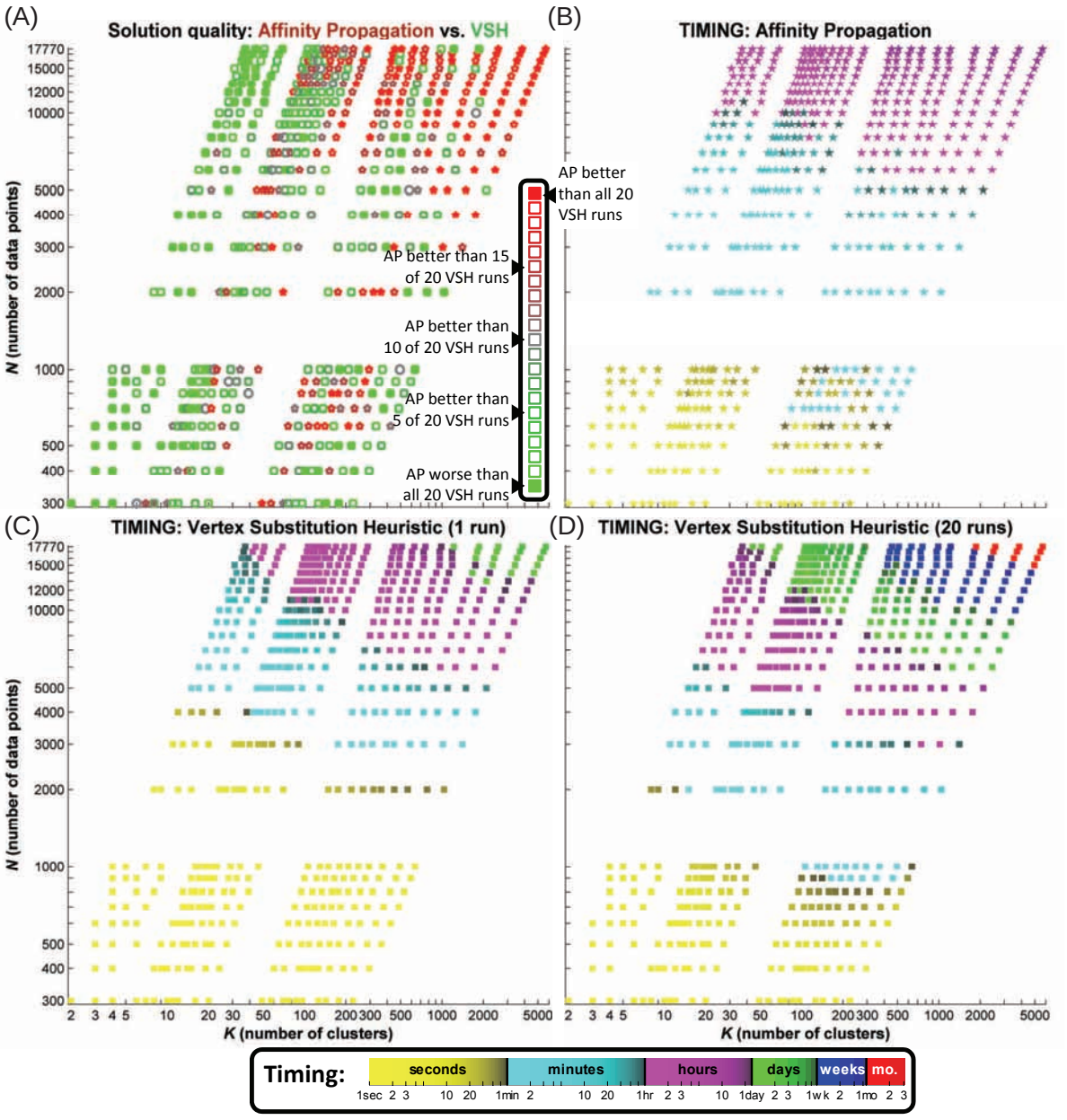


Figure 4.39: Performance comparison for Netflix dataset between affinity propagation and the vertex substitution heuristic, for a range of Netflix data sizes (N) and number of clusters (K). For large datasets ($N > 10000$) and non-trivial numbers of clusters ($K > 100$), (A) shows that affinity propagation consistently finds better solutions than 20 runs of the vertex substitution heuristic. The running times for the algorithms are shown for affinity propagation (B), one run of the vertex substitution heuristic (C), and 20 runs of the vertex substitution heuristic (D). Affinity propagation takes hours for the most complex problems, while one run of the vertex substitution heuristic can take days, and 20 runs can take weeks.

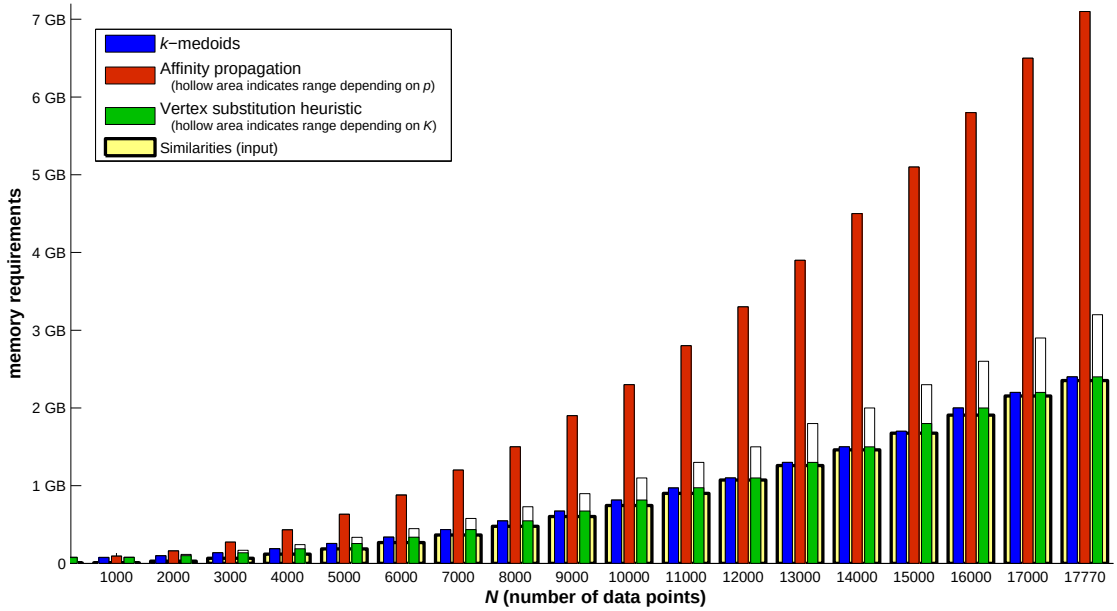


Figure 4.40: The memory requirements for k-medoids, affinity propagation, and the vertex substitution are shown for various subsets of the Netflix dataset. Memory requirements were tabulated from the resident process size (including code and data segments) using the UNIX `top` command and include roughly 72 MB overhead (insignificant considering the plot's scale) for MATLAB's toolbox caches, *etc.* The baseline input similarity memory requirements are shown as wide bars in the background; the k-medoids algorithm requires little beyond this. The vertex substitution heuristic requires between 2% and 36% more, depending on the value of K. Affinity propagation requires roughly 310% times the input similarities capacity, mostly for storing availabilities and responsibilities.

Chapter 5

Applications of Affinity Propagation

A major benefit to affinity propagation is that it clusters data without having a specific model of the data built into the method; this has lead to its use in a wide variety of problem domains using rich application-specific similarity models. This chapter briefly explores several results from the topics of computer vision and bioinformatics.

5.1 Affinity Propagation and Computer Vision: Image categorization

Many computer vision tasks either produce a clustering of input features as output or require it as a preprocessing step for subsequent analysis. Exemplars have been used with success in a variety of vision tasks, including image synthesis [27, 101], super-resolution [33, 92], image and video completion [52, 105], and combined tracking and object detection [40, 97].

The use of exemplars is attractive for several reasons. A relatively small number of representative exemplars can capture high-order statistics, since each exemplar can simultaneously express dependencies between a large number of image features. In contrast to general statistical methods for which many parameter configurations can correspond to unrealistic data, each exemplar is an image or an image fragment so it naturally corresponds to realistic image

data. For this reason, exemplars can be used to make realistic predictions for missing image data and avoid the blurring that often occurs when parametric methods are applied. Exemplars are represented efficiently as pointers into the training data (e.g., a subset of image features), so the number of bits of information needing to be specified during exemplar learning is quite small [48].

5.1.1 Augmenting the Olivetti dataset

The Olivetti dataset (§4.1) was modified for computer vision experiments as follows: to examine the effect of a wider range in image variation for each individual, the images of ten individuals were extracted, and for each of the resulting 100 images, three in-plane rotations and three scalings were applied[1], producing a dataset of 900 images. Initially, the similarity between image i and image k, $s(i,k)$ was set to the negative of the sum of squared pixel differences.

5.1.2 Performance on unsupervised image classification

In several vision tasks, such as image or video summarization, labels are unavailable and the goal is to detect meaningful image categories in an unsupervised fashion. Even in supervised tasks, it can be helpful to first perform unsupervised categorization of images or image parts so as to reduce the dimensionality of the input and simplify supervised learning. Here, the performance of affinity propagation is explored in terms of unsupervised classification error of the learned categories based on the true categories, where each learned category is associated with the true category that accounts for the largest number of data points in the learned category. The classification rate will approach 100% as the number of learned categories approaches the number of training cases, so classification rates are reported as a function of the number of learned categories.

[1]The rotations were $\{-10^\circ, 0^\circ, +10^\circ\}$ and the scaling factors were $\{0.9, 1, 1.1\}$.

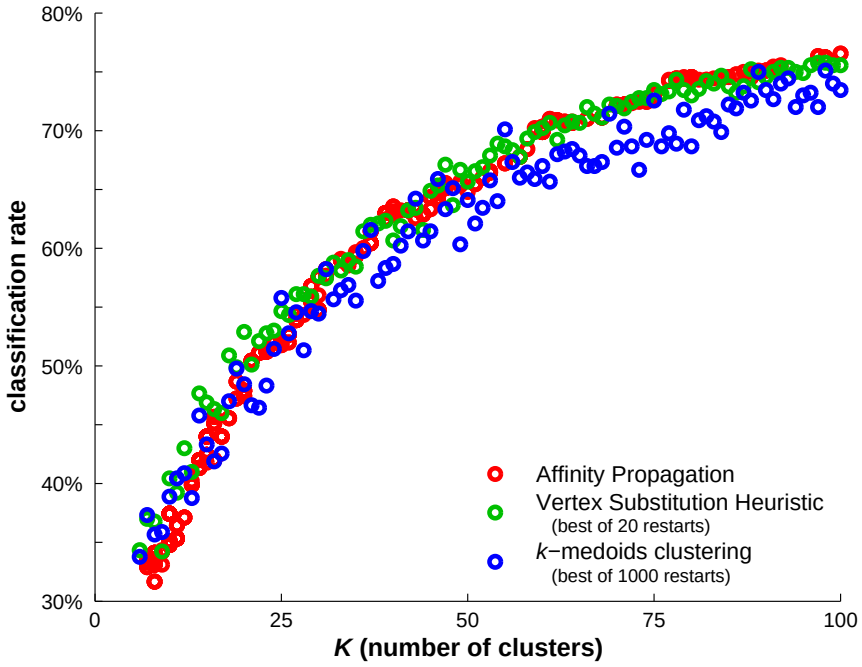


Figure 5.1: Performance on unsupervised classification error for the Olivetti face data. The classification rate (fraction of images correctly classified where the learned category is associated with the true category that accounts for the largest number of data points in the learned category) is plotted against the number of clusters or exemplars, K. Results for affinity propagation, best of 1000 runs of k-medoids clustering, and the vertex substitution heuristic are shown. For larger numbers of clusters (*e.g.*, $K > 25$), k-medoids clustering typically achieves a classification rate 3–5% worse than affinity propagation or the vertex substitution heuristic (which both achieve similar results).

Figure 5.1 plots the classification rate as a function of the number of clusters (K) for affinity propagation, the best of 1000 runs of k-medoids clustering, and the vertex subsitution heuristic on the faces dataset containing 900 images. Affinity propagation achieves similar results to the vertex substitution heuristic but typically achieves a classification rate 3–5% better than k-medoids clustering, for the same K.

5.1.3 Performance using non-metric similarities

In the context of comparing two face images, squared Euclidean distance ignores the fact that certain facial features may appear in different positions in each image. This section outlines a non-metric similarity function that can be tailored toward matching face images, and shows that it achieves higher classification rates.

Denoting the vector of pixel intensities for images i and k by $\mathbf{x}_i$ and $\mathbf{x}_k$, the previous section used the following definition of similarity:

$$s(i,k) = -\left\|\mathbf{x}_i - \mathbf{x}_k\right\|^2$$

Here, the similarity of image i to image k is computed by extracting a sub-image from the center of image i and finding its best match to all sub-images (not necessarily centered) in image k. Let $\mathbf{T}$ denote an operator that cuts a window of a fixed size out of the image it is operating on. There will be many operators corresponding to different possible positions from which the window may be extracted; let $\mathbf{T}_0$ denote the operator that cuts the window out of the center of the image. The non-metric similarity used here is given by:

$$s(i,k) = -\min_{\mathbf{T}} \left\|\mathbf{T}_0\mathbf{x}_i - \mathbf{T}\mathbf{x}_k\right\|^2$$

The original Olivetti images of size 64×64 are used here with a window size of 50×50. Figure 5.2(A) shows an example of an image $\mathbf{x}_i$ (upper left) and the windows that achieve the

minimum in the above expression for the other nine images in the same (true) category.

Figure 5.2(B) plots the classification rate against the number of clusters (K) for affinity propagation applied to this non-metric similarity function. Included for comparison is the plot obtained using k-medoids clustering and the vertex substitution heuristic applied to the same non-metric similarity function. Also included is the affinity propagation plot obtained using the negative squared Euclidean distance similarities described in the previous section (circles, as before). The non-metric similarity definition facilitates a significant increase in the classification rate and affinity propagation achieves similar classification rates to the vertex substitution heuristic and higher classification rates compared to k-medoids clustering.

5.2 Affinity Propagation and Sparsity: Exon Detection

An important problem in current genomics research is the discovery of genes and gene variants that are expressed as messenger RNAs (mRNAs) in normal tissues. In a recent study [37], DNA-based techniques were used to identify more than 800,000 possible exons ('putative exons') in the mouse genome. For each putative exon, an Agilent microarray probe matching a 60-base long DNA segment was used to measure the amount of corresponding mRNA that was present in each of twelve mouse tissues. Each twelve-dimensional vector, called an 'expression profile' for the DNA, can be viewed as a feature vector indicating the putative exon's function. Also, when nearby segments of DNA undergo coordinated transcription across multiple tissues, they are likely to come from transcribed regions of the same gene [36]. By grouping together feature vectors for nearby probes, we can detect genes and variations of genes.

Figure 5.3(A) shows a normalized subset of the data and gives three examples of groups of nearby feature vectors that are similar enough to provide evidence of gene units. The actual data is generally much noisier [36], and includes:

- Multiplicative noise, because exon probe sensitivity can vary by two orders of magnitude.
- Correlated additive noise, because a probe can cross-hybridize in a tissue-independent

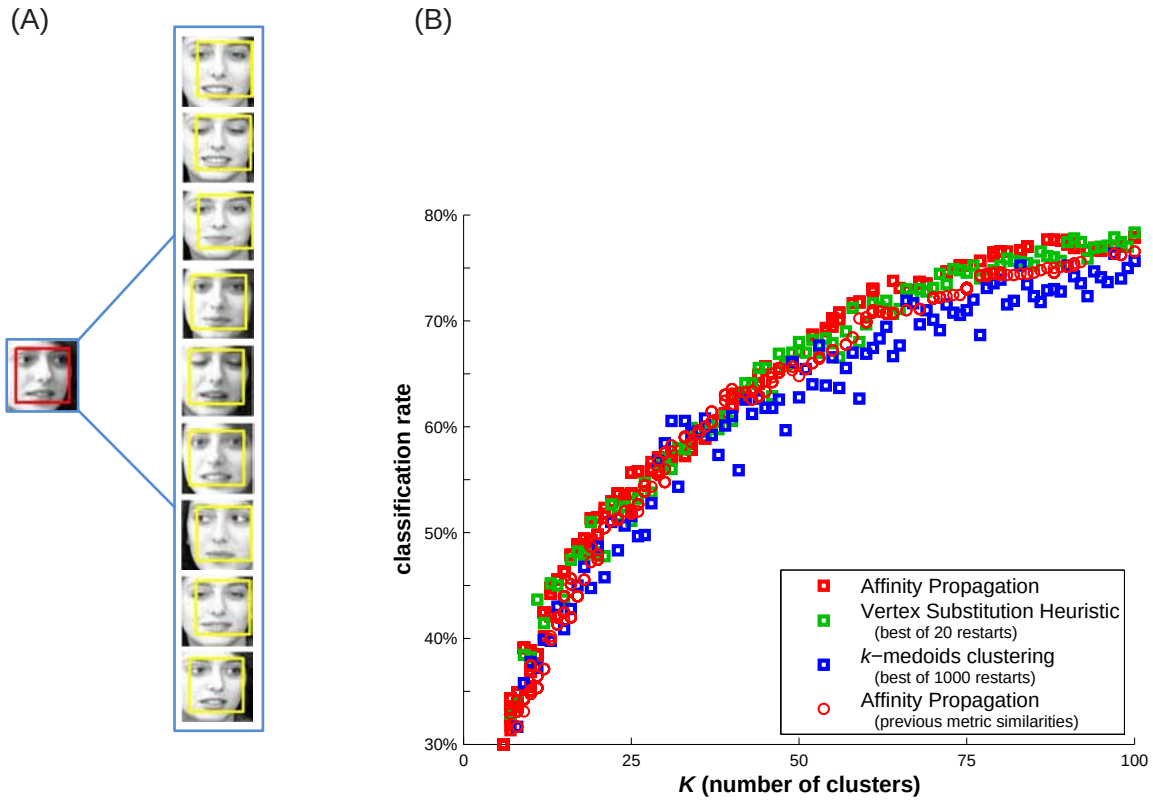
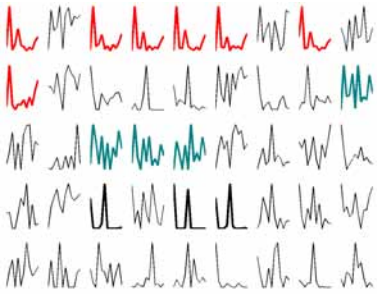


Figure 5.2: Unsupervised image categorization using non-metric similarities. (A) The similarity of an image (left) to each of the other images is determined by finding the best match (in terms of squared error) between a window centered in the first image and all possible equally-sized windows in the second image. (B) The classification rate is plotted against the number of exemplars (shown as squares) for affinity propagation, the best of 1000 runs of k-medoids clustering, and the vertex substitution heuristic using the non-metric similarity function. Also shown (as circles) is the plot for affinity propagation applied using the metric similarity function described previously. Again, affinity propagation and the vertex substitution heuristic achieve similar classification rates, which in turn are several percentage points better than affinity propagation applied to the previously-examined metric similarities.

(A)



(B)

		intron exemplar	DNA segments							
intron exemplar		$+\infty$ (must be exemplar)	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
DNA segments	$\log p_0(1)$	$p + \log p_0(1)$	$s(1,2)$	$s(1,3)$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$\log p_0(2)$	$s(2,1)$	$p + \log p_0(2)$	$s(2,3)$	$s(2,4)$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$\log p_0(3)$	$s(3,1)$	$s(3,2)$	$p + \log p_0(3)$	$s(3,4)$	$s(3,5)$	$-\infty$	$-\infty$	$-\infty$
	$\log p_0(4)$	$-\infty$	$s(4,2)$	$s(4,3)$	$p + \log p_0(4)$	$s(4,5)$	$s(4,6)$	$-\infty$	$-\infty$
	$\log p_0(5)$	$-\infty$	$-\infty$	$s(5,3)$	$s(5,4)$	$p + \log p_0(5)$	$s(5,6)$	$s(5,7)$	$-\infty$
	$\log p_0(6)$	$-\infty$	$-\infty$	$-\infty$	$s(6,4)$	$s(6,5)$	$p + \log p_0(6)$	$s(6,7)$	$s(6,8)$
	$\log p_0(7)$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$s(7,5)$	$s(7,6)$	$p + \log p_0(7)$	$s(7,8)$
	$\log p_0(8)$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$s(8,6)$	$s(8,7)$	$p + \log p_0(8)$

Figure 5.3: Exons can be detected by grouping together DNA segments with similar normalized expression patterns across tissues, shown in (A). A sparse matrix of similarities is constructed as shown in (B), where the zeroeth data point is an intron exemplar (background model) and off-diagonal similarity entries are computed as in equation (5.1). The probability of each expression profile under the background model is placed in the first column and along the diagonal (as preferences) along with a global preference additive term, p, used for controlling the false-positive rate.

manner to background mRNA sources.

- Spurious additive noise, due to a noise measurement procedure and biological effects such as alternative splicing.

To account for noise, false putative exons and the proximity of exons in the same gene, the input similarity $s(i,k)$ between DNA segment (data point) i and DNA segment k is:

$$e^{s(i,k)} = \beta e^{-\beta|i-k|} \cdot \max_{y,z,\sigma} \left\{ p(y,z,\sigma) \cdot \frac{e^{-\frac{1}{2\sigma^2}\sum_{m=1}^{12}\left[x_i^m-(y\cdot x_j^m+z)\right]^2}}{(\sqrt{2\pi\sigma^2})^{12}} \right\} \tag{5.1}$$

where x_i^m is the expression level for the m^{th} tissue in the i^{th} probe (in genomic order). The parameters y, z, and σ were assumed to be independent and uniformly distributed[2] so $p(y,z,\sigma) \propto 1$ over the domain. To account for the non-transcribed regions (*e.g.* containing introns), an additional artificial data point was included (which is indexed as data point '0') and the similarity of each other point to this 'non-transcribed exemplar' was determined by evaluating each point under a mixture of Gaussians for the entire dataset; this background model likelihood is referred to as $p_0(x_i)$ so $\forall i > 0 : \; s(i,0) = \log p_0(x_i)$. The preference for the artificial data point was set to $s(0,0) = +\infty$ to guarantee its selection as an exemplar (this was also facilitated by setting $\forall i > 0 : \; s(0,i) = -\infty$), whereas the preference for every other data point was set to the background model log-likelihood plus a shared preference constant, p, that was used to control the number of exemplars found and thus the sensitivity of the system.

A total of 75,066 DNA segments were all mined from the genome for mouse Chromosome 1, with a similarity matrix constructed as illustrated in Figure 5.3(B). Not all $(75066+1)^2 \approx 5.6$ billion possible similarities were used or even computed; the exponential $\beta e^{-\beta|i-k|}$ prior term and the assumption that genes on the same strand meant that similarities for $|i-k| > 100$ could be approximated[3] as $-\infty$. This reduces the problem size to approximately 15 million

[2] Based on the experimental procedure and a set of previously-annotated genes (RefSeq), they were estimated as $\beta = 0.05$, $y \in [0.025, 40]$, $z \in [-\max_{i,m} x_i^m, +\max_{i,m} x_i^m]$, and $\sigma \in (0, +\max_{i,m} x_i^m)$.

[3] According to ground truth (RefSeq), less than 1% of genes spanned a distance of more than 100 probes (none spanned more than 165). The sparseness constraint can accomodate a span of 200 if the exemplar is centered.

similarities and messages to exchange—99.73% sparse—or roughly the same difficulty as a non-sparse problem with $\sqrt{15 \times 10^6} \approx 3900$ data points. Figure 5.4 illustrates the identification of gene clusters and the assignment of some data points to the non-exon exemplar.

After clustering the 75067×75067 sparse matrix of similarities, DNA segments assigned to exemplars other than the non-transcribed exemplar were considered to be parts of genes. All DNA segments were separately mapped to the RefSeq database of annotated genes [85] to produce labels used for reporting true positive and false positive rates. These results are compared in Figure 5.5, where the true-positive (TP) rate is plotted against the false-positive (FP) rate for affinity propagation and k-medoids clustering. For each number of clusters, affinity propagation was run once and took roughly six minutes, whereas k-medoids clustering was run 10,000 times which required 208 hours. Affinity propagation achieves significantly higher TP rates, especially at low FP rates which are most useful to biologists. At a FP rate of 3%, affinity propagation achieved a TP rate of 39%, whereas the best k-medoids clustering result was 17%. For comparison, at the same FP rate, the engineering tool described in [36]—which integrates additional biological knowledge—achieved a TP rate of 43%.

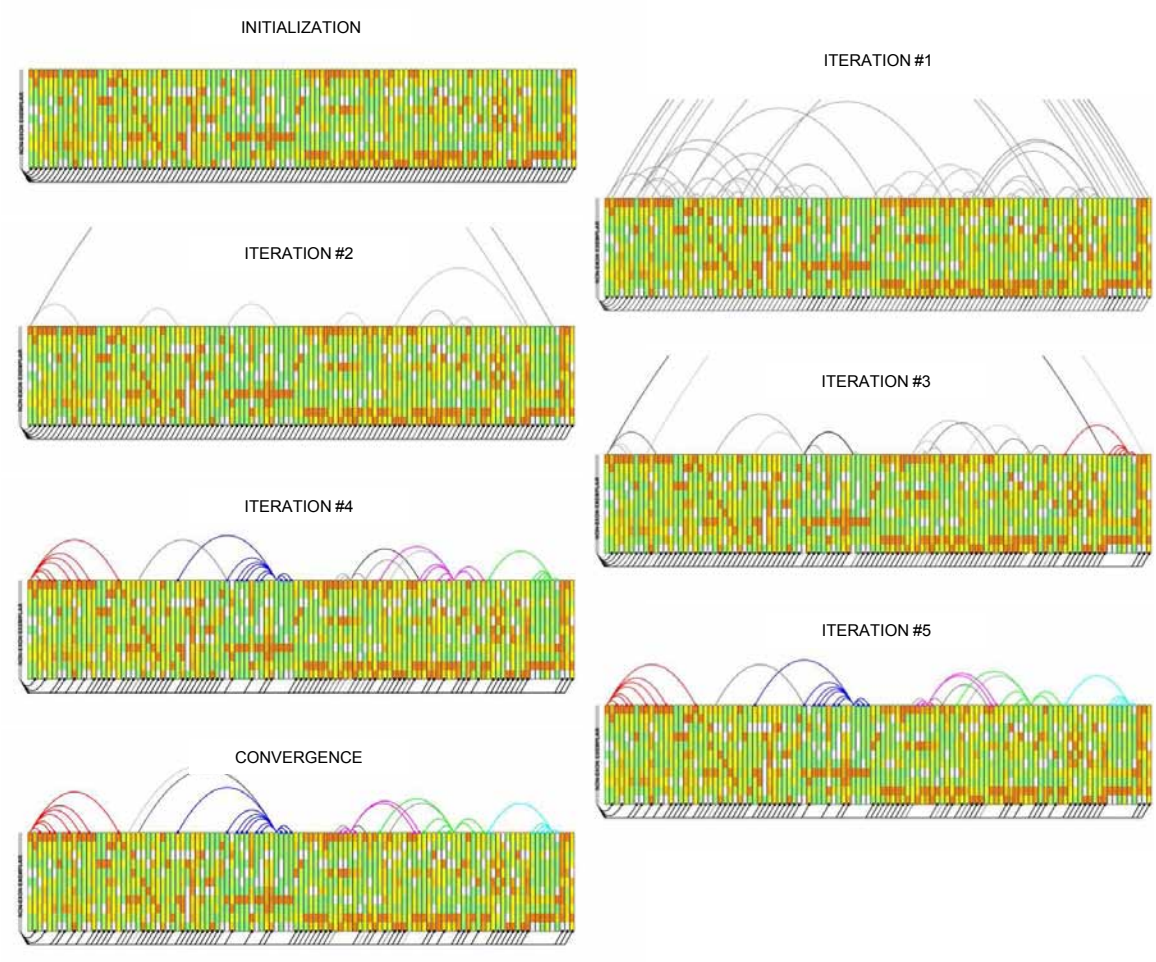


Figure 5.4: Affinity propagation was used to detect putative exons (data points) comprising genes from mouse chromosome 1. A small portion of the data and the emergence of clusters during each iteration of affinity propagation are shown. In each frame, the 100 boxes outlined in black correspond to 100 data points (from a total of 75,066 putative exons), and the 12 colored blocks in each box indicate the transcription levels of the corresponding DNA segment in 12 tissue samples. The grey profile on the far left corresponds to an artificial data point ($i = 0$) with infinite preference that is used to account for non-exon regions (*e.g.*, introns). Lines connecting data points indicate potential assignments, where gray lines indicate assignments that currently have weak evidence and dark-colored lines indicate assignments that currently have strong evidence.

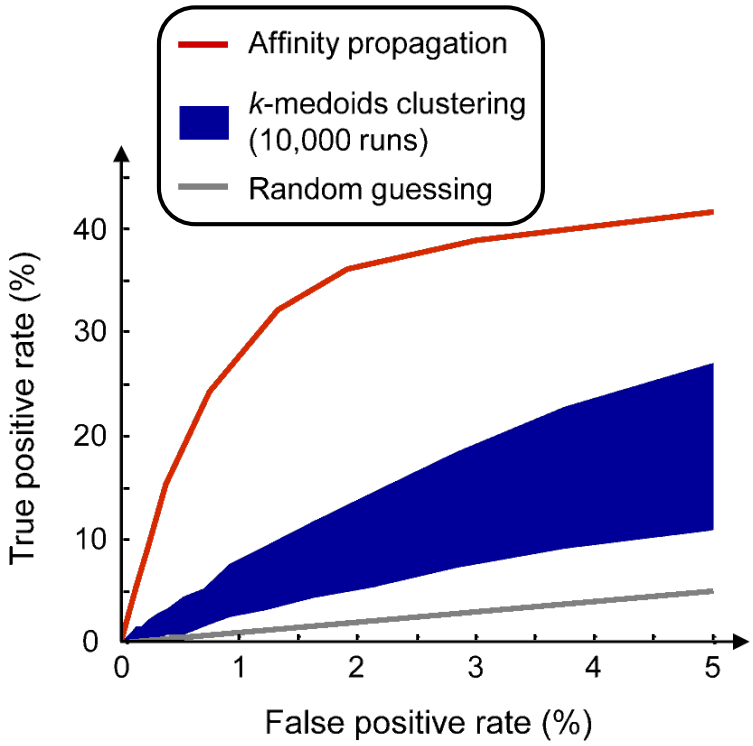


Figure 5.5: Affinity propagation was used to detect putative exons (data points) comprising genes from mouse chromosome 1. For each number of clusters, affinity propagation took six minutes, whereas 10,000 runs of k-medoids clustering took 208 hours. The false positive rate was adjusted via a constant added to the preference; this plot of the resulting true-positive rate versus false-positive rate for detecting exons (using labels from RefSeq [85]) shows that affinity propagation performs better at detecting biologically-verified exons than k-medoids clustering.

5.3 Treatment Portfolio Design via Affinity Propagation

A central question for any computational research collaborating with a biologist or medical researcher is in what form computational analyses should be communicated to the experimentalist or clinician. While application-specific predictions are often most appropriate, in many cases what is needed is a selection of potential options available to the biologist/medical researcher, so as to maximize the amount of information gleaned from an experiment (which can often be viewed as consisting of independently-assayed targets). If the number of options is not too large, these can be discussed and selected manually. On the other hand, if the number of possibilities is large, a computational approach may be needed to select the appropriate options. Affinity propagation has been shown [26] to be an effective approach to this task.

5.3.1 Treatment Portfolio Design

For concreteness, the possible set of options is referred to as 'treatments' and the assays used to measure the suitability of the treatments as 'targets'. Every treatment has a utility for each target and the goal of what is referred to as treatment portfolio design (TPD) is to select a subset of treatments (the portfolio) so as to maximize the net utility of the targets. The terms 'treatment', 'target', and 'utility' can take on quite different meanings, depending on the application. For example, treatments may correspond to queries, probes, or experimental procedures. Examples of targets include disease conditions, genes, and DNA binding events.

The input to TPD is a set of potential treatments or queries $\mathcal{T}$, a representative population of targets $\mathcal{R}$, and a utility function $u : \mathcal{T} \times \mathcal{R} \to \mathbb{R}$, where $u(T, R)$ is the utility of applying treatment $T \in \mathcal{T}$ to target $R \in \mathcal{R}$. This utility may be based on a variety of factors, including the benefit of the treatment, cost, time to application, time to response, estimated risk, *etc.* The goal of computational TPD is to select a subset of treatments $\mathcal{P} \subseteq \mathcal{T}$ (called the 'portfolio') so as to maximize their net utility for the target population. A defining aspect of the utility function is that it is additive (*i.e.*, the total or net utility is a sum of component utilities); for

portfolio $\mathcal{P}$, the net utility is:

$$\sum_{R \in \mathcal{R}} \max_{T \in \mathcal{P}} u(T, R) \ .$$

To account for the fact that some treatments are preferable to others regardless of their efficacy for the targets (*e.g.*, different setup costs), a treatment-specific cost function $c : \mathcal{T} \rightarrow \mathbb{R}$ can be used. The net utility, including the treatment cost is:

$$U(\mathcal{P}) = \sum_{R \in \mathcal{R}} \max_{T \in \mathcal{P}} u(T, R) - \sum_{T \in \mathcal{P}} c(T)$$

Provided with $\mathcal{T}$, $\mathcal{R}$, u and c, the computational task is to find $\max_{\mathcal{P} \subseteq \mathcal{T}} U(\mathcal{P})$. Note that the number of treatments in the portfolio will be determined by balancing the utility with the treatment cost.

In general, the treatment set does not equal the target set. Then, TPD can be viewed as a facility location problem with treatments serving as potential facilities (exemplars) and targets as customers. Affinity propagation can be adapted to address this: if point i is a target and point k is a treatment, then $s(i, k)$ can be set to the utility of that treatment for that target; if point k is a treatment, $s(k, k)$ can be set to the negative cost for that treatment.

One important difference, however, between the problem statements for exemplar-based clustering and TPD is the distinction between treatments and targets. The basic affinity propagation algorithm treats all points as potential exemplars and every non-exemplar point must be assigned to an exemplar. In TPD, only treatment can be selected as exemplars, and only targets have utilities for being assigned to exemplars (treatments). Treatments that are not selected for the portfolio (exemplar set) are neither exemplars nor assigned to another exemplar (treatment).

To allow some treatments to not be selected for the portfolio and also not be assigned to any other points, a special 'garbage collector' point is introduced and the similarities of treatments to this point are set to zero. So, unless there is a net benefit in utility minus cost when including a treatment in the portfolio (exemplar set), it will be assigned to the garbage collector point. In

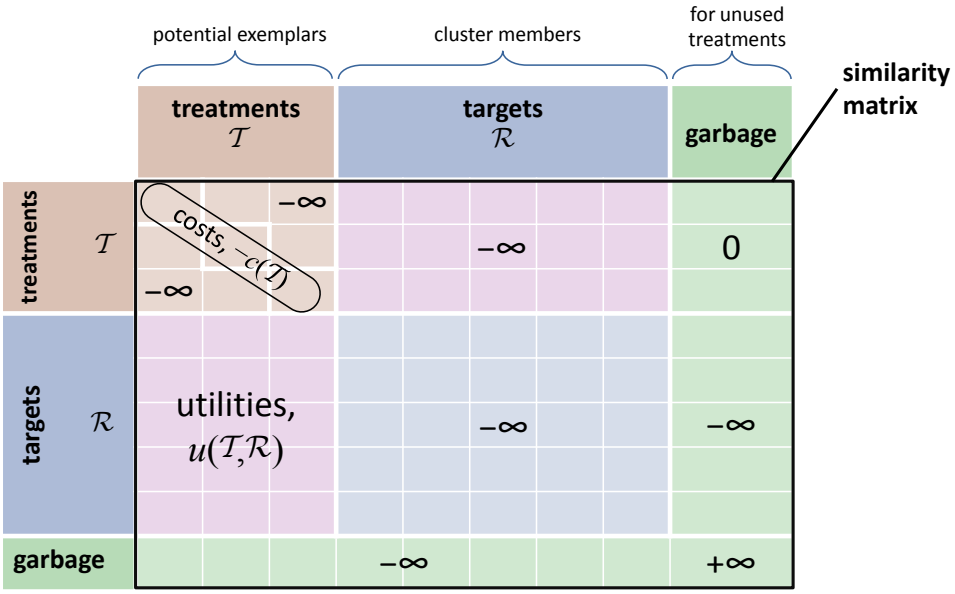


Figure 5.6: Treatment portfolio design can be rephrased in terms of similarities and preferences for affinity propagation. Constraints on similarities for treatment portfolio design (TPD) are outlined in equation (5.2).

summary, the following similarity constraints account for the bipartite structure of TPD:

$$\begin{gathered} s(\text{target, treatment})=u(\text{treatment, target}) \text{ and } s(\text{target, target}')=s(\text{target, garbage})=-\infty \\ s(\text{treatment, garbage})=0 \text{ and } s(\text{treatment, target})=s(\text{treatment, treatment}')=-\infty \\ s(\text{garbage, target})=s(\text{garbage, treatment})=-\infty \\ s(\text{treatment, treatment})=-c(\text{treatment}),\ s(\text{target, target})=-\infty \text{ and } s(\text{garbage, garbage})=+\infty \end{gathered} \tag{5.2}$$

The last constraints ensure that targets cannot be selected as exemplars and that the garbage collection point is always available as an exemplar. The specific form of similarities under these constraints is illustrated in Figure 5.6. Note that messages need only be exchanged between a treatment and target if the utility is not $-\infty$; this makes the algorithm $\mathcal{O}(|\mathcal{T}|\cdot|\mathcal{R}|)$ instead of $\mathcal{O}(|\mathcal{T}+\mathcal{R}|^2)$.

5.3.2 Application to HIV vaccine cocktail design

The issue of HIV vaccine cocktail design can be nicely posed as a TPD problem. The idea with this is to find a set of optimal HIV strains for the purpose of priming the immune sys-

(A) treatments, $\mathcal{T}$, are HIV strain sequences

(B) targets, $\mathcal{R}$, are short subsequences that correspond to epitopes

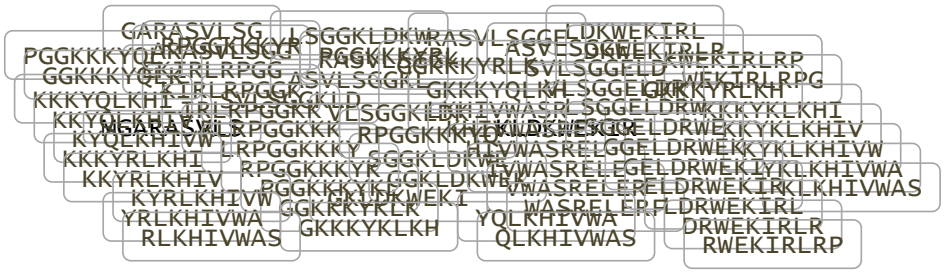


Figure 5.7: Treatment and target sets for HIV vaccine design. The treatments, $\mathcal{T}$, are thousands of HIV strain sequences that differ slightly from each other due to mutations (shown in bold). Sequences are shown as chains of amino acids (represented as text strings from an alphabet of 20 Latin letters). The targets, $\mathcal{R}$ are a set of short sequences that correspond to the epitopes to which immune systems respond. For this application, all possible (overlapping) 9-mers extracted from the HIV strain sequences are used.

tems of many patients. The treatments $\mathcal{T}$ are thousands of HIV strain sequences (available at www.hiv.lanl.gov). The targets $\mathcal{R}$ are a set of short sequences (patches, fragments) that correspond to the epitopes to which immune systems respond (all 9-mers are used). See Figure 5.7 for more details. The utility $u(T, R)$ of a strain T for a fragment R would ideally be set to its potential for immunological protection, but following the approaches in [30, 53, 54, 81], it is set to the frequency of the fragment in the database of HIV sequences if fragment R is present in strain T, and zero otherwise, as in equation (5.3).

$$u(T,R) = \begin{cases} \text{frequency of } R \text{ in HIV sequence database, if } T \text{ contains } R\,; \\ 0, \text{ otherwise}\,. \end{cases} \tag{5.3}$$

The net utility is also referred to as ‘coverage’.

Figure 5.8 shows aligned pieces of HIV’s *Gag* protein from several different strains, with two variable sites marked by arrows as well as known or predicted T-cell epitopes for the MHC

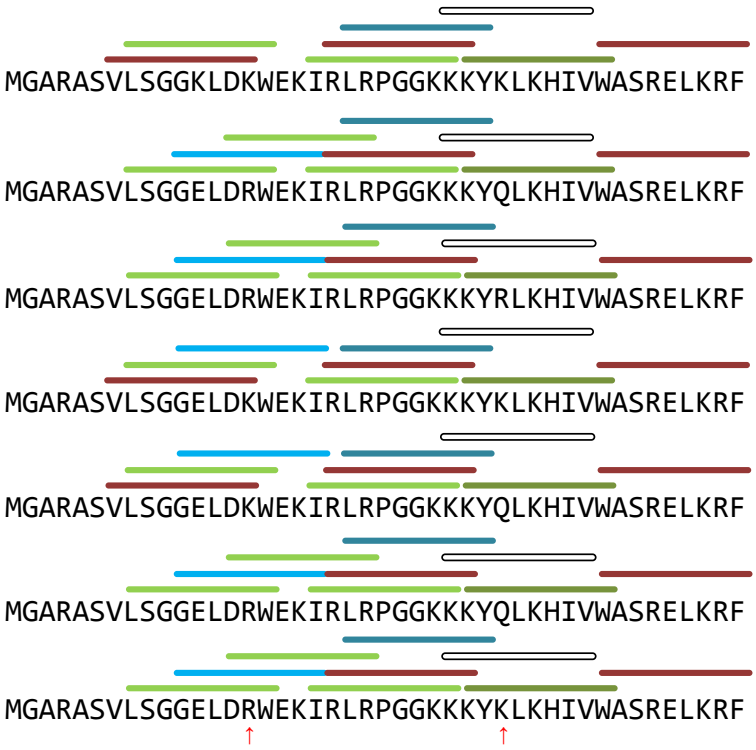


Figure 5.8: Fragments of Gag protein with epitopes recognized by several HIV-infected patients. Epitopes recognizable by a single patient are shown in a single color; mutations marked by red arrows escape MHC I binding.

molecules of five different patients taken from the Western Australia cohort [71]. Epitopes recognizable by a single patient are shown in a single color, and each patient is assigned a different color. Some mutations (marked by red arrows) 'escape' MHC I binding. For example, the red patient can react to the 9-mer epitope VLSGGKLDK in the first sequence, but not to VLSGGKLDR in the second. On the other hand, other mutations do not affect MHC binding, but may affect T-cell binding (a different set of T-cells will target different versions). The white patient could be immunized against three forms of the same epitope: KKYKLKHIV, KKYQLKHIV, KKYRLKHIV. In this small example, a vaccine can be designed consisting of the following segments which epitomizes (in an immunological sense) the seven strains shown in the figure: VLSGGKLDKWEKIRLRPGGKKKYKLKHIVWASRELERFLSGGKLDRW-EKIRLRKKYQLKHIVWKKKYRLKHIVW.

Much discussion among HIV vaccine experts has been focused on the need for constraining

vaccine constructs optimized for coverage to resemble naturally-occurring strains [30,81]. This is motivated by several pieces of evidence suggesting that deviation from naturally-occurring strains often reduces efficacy in animal models as well as in vaccine trials, both in terms of the cellular and antibody responses. Thus, [81] proposes enrichment of the vaccine with a sequence that sits in the center of the HIV phylogenetic tree, so that this single native-like (but still artificially-derived) strain is used to provide coverage of immune targets in as natural a way as possible, while the additional coverage is achieved with an epitome fragment(s). In contrast, in their recent paper [30], Fischer *et al.* avoid the use of fragments altogether and propose building the entire vaccine out of multiple strain-like constructs optimized by simulated strain recombination, dubbed 'mosaics'. A mosaic vaccine is therefore a cocktail of artificially-derived strains, not existent among the observed strains of the virus, but achievable by recombining the existing strains many times. These vaccine components resemble natural strains, but have higher 9-mer coverage than would be expected from a cocktail of natural strains. Mosaics can always achieve higher coverage than natural strains, so while they may not be viable as vaccines, they provide an upper bound on potential coverage.

As the dataset of known HIV sequences is constantly growing, the potential for achieving high coverage with a cocktail of true natural strains is growing as well. Newly-discovered strains differ from existing ones mostly by the combination of previously-seen mutations rather than by the presence of completely-new 9-mers. In fact, Fischer *et al.* have increased the Gag vaccine coverage with their use of mosaic by some 4–5% in comparison to natural strain cocktails. As the problem is $\mathcal{NP}$-hard, the natural strain cocktails (treatment portfolios) in their paper are found by a greedy technique analogous to the vertex substitution heuristic, which may further decrease the perceived potential of natural strain cocktails, especially for a larger number of components. For a large dataset consisting of 1755 Gag proteins from the LANL database, a Gag sequence consisting of the best four natural strains affinity propagation could find had only 3% lower coverage than the mosaic of the same size optimized on the same data (69% vs. 66%). Obviously, as the dataset grows, the computational burden for finding the

optimal cocktail grows exponentially, as is the case for the general TPD problem.

Furthermore, while potentially-important for the cellular arm of the immune system, the closeness of vaccine components to natural strains is even more important for properly presenting potential targets of the humoral (antibody) arm of the immune system. As opposed to the T-cell epitopes, antibody epitopes are found on the surface of the folded proteins. It has been shown that slight changes in HIV's *Env* protein can cause it to mis-fold, and so naturally-occurring HIV strains are more likely to function properly than artificially-derived Env proteins.

In these experiments, the TPD problem for Gag vaccine cocktail optimization is performed with larger cocktails, where the coverage approaches 80% or more and exhaustive search is computationally infeasible. Affinity propagation is used to find an approximate solution, and its achieved utility is compared with that of the greedy method and the mosaic upper bound [30]. Table 5.1 summarizes these results on 1755 strains.

Table 5.1: The utility ("epitope coverage") of vaccine portfolios found by affinity propagation and a greedy method, including an upper bound on utility (found using mosaics).

vaccine	Natural strains		Artificial mosaic strains
portfolio size	Affinity propagation	Greedy Method	(upper bound)
$K = 20$	77.54%	77.34%	80.84%
$K = 30$	80.92%	80.14%	82.74%
$K = 38$	82.13%	81.62%	83.64%
$K = 52$	84.19%	83.53%	84.83%

These results show that affinity propagation achieves higher coverage than the greedy method. Importantly, these results also suggest that the sacrifice in coverage necessary to satisfy the vaccine community's oft-emphasized need for natural components may in fact be bearable if large datasets and appropriate algorithms are used to optimize coverage.

Chapter 6

Conclusions and Future Directions

In conclusion, clustering data by identifying exemplar data points rather than parametric methods allows for rich domain-specific models that can achieve superior results, as explored in Chapter 5. Affinity propagation (see Section 3.2) is an innovative and readily-extensible clustering algorithm that identifies exemplars quickly and successfully. It consistently finds better solutions than standard exemplar-based clustering algorithms such as k-medoids, and achieves comparable or better results to workhorse algorithms such as the vertex substitution heuristic (VSH) in far less time for large datasets.

Specifically, the benchmarks in Section 4 show that for large datasets with thousands of data points, many restarts of k-medoids clustering (Section 2.4) will achieve mediocre results within a few minutes; allotting the algorithm more hours or days of CPU time will only yield slight improvements in solution quality. Affinity propagation requires more minutes (or, for the largest dataset examined, hours) than k-medoids but achieves vastly superior results. For large datasets, the vertex substitution heuristic with variable neighbor search (Section 2.5) achieves comparable or worse results than affinity propagation but requires days or weeks of CPU time. For small datasets with hundreds of data points, affinity propagation and the vertex substitution heuristic both achieve near-optimal results though for realistic problems (where the number of clusters is much less than half the number of points), VSH initialized with affinity propagation

seems to be the best balance between computation time and solution quality. In any case, optimal clusterings of these small datasets can be found with linear programming techniques in a matter of hours.

In the two years since its introduction, affinity propagation has spawned a growing volume of research, such as:

- Vector quantization codebook design (Jiang *et al.* in [51])
- Soft-constraint affinity propagation for gene expression data (Leone *et al.* in [68])
- Multiple view image segmentation (Xiao *et al.* in [108])
- Finding light sources using images (An *et al.* in [2])
- Image categorization and normalized mutual information analysis (Grira *et al.* in [44])
- Semi-supervised object classification (Fu *et al.* in [39])
- Image-audio dataset analysis (Zhang *et al.* in [113])
- Gene3D: Protein analysis (Yeats *et al.* in [110])
- Protein sequence clustering (Wittkop *et al.* in [107])
- Affinity propagation with isomap-based metrics (Baya *et al.* in [4])
- Data streaming and analysis of grid computing jobs (Zhang *et al.* in [114])
- Analysis of cuticular hydrocarbons (Kent *et al.* in [58])
- Analysis of brain tissue MRI data (Verma *et al.* in [100])
- Clustering speakers from audio data (Zhang *et al.* in [115])
- Color-based clustering for text detection in images (Yi *et al.* in [112])
- Analysis of hydrophobic-polar protein model (Santana *et al.* in [89])
- Face recognition with linear discriminant analysis (Du *et al.* in [24])
- Clustering text data (Kim *et al.* in [59])
- Adaptive extensions of affinity propagation (Wang *et al.* in [102])
- Knowledge discovery in medical data sources (Senf *et al.* in [91])
- Analysis of land-use and land-cover data (Cardille *et al.* in [12])
- Customer micro-targeting (Jiang *et al.* in [50])

An interesting and recent research thrust is Dirichlet process affinity propagation [95] which involves adapting the graphical model in Figure 3.5 to incorporate a Dirichlet prior over the size of clusters into the factor graph. This representation can then be viewed as maximum *a posteriori* inference of a Dirichlet mixture model where the means are constrained to be exemplars (co-located with data points) and variances are fixed.

The affinity propagation algorithm raises many new questions for further research:

The relationship between max-product belief propagation and linear programming relaxations is not well-understood but is beginning to be more widely investigated (*e.g.*, [94, 109]). In [88], a linear programming relaxation for the weighted matching problem is compared to max-product belief propagation with a proof that “if the [linear programming] relaxation is

tight, *i.e.*, if the unique solution is integral, then the max-sum algorithm converges and the resulting estimate is the optimal matching". Much theoretical work remains in analyzing affinity propagation but this suggests a starting approach.

Clustering is traditionally an unsupervised learning task, but there are many applications where some labeled (or at least partially-labeled) data is available for semi-supervised learning. The affinity propagation factor graph can easily be extended to incorporate additional pairwise constrains such as requiring points with the same label to appear in the same cluster with just an extra layer of function nodes. The model is flexible enough for information other than explicit constraints such as two points being in different clusters or even higher-order constraints (*e.g.*, two of three points must be in the same cluster). There may also be applications where n-wise similarities are useful, such as triple-wise similarities for finding collinear points (*e.g.*, data from astronomical tracking)

Finally, the paradigm shift of using pointers to exemplar data instead of problem-specific parameters may have wider applicability. A cluster is a simple structure, perhaps requiring only one exemplar to identify the location of its center. More complex structures such as d-dimensional subspaces could use $d+1$ data points to be specified, or clusters could have additional attributes such as a scale or shape—analogous to a Gaussian's covariance matrix—that could be specified by multiple exemplars.

Appendix

Appendix A

The Bethe free energy approximation

Using the notation from Section 2.6.3, a factor graph with N nodes representing variables $\mathbf{X} = \{X_1, X_2, \ldots, X_N\}$ and M nodes representing functions $\{f_1, f_2, \ldots, f_M\}$ describes a global function that can be factorized as $f(\mathbf{X}{=}\mathbf{x}) = \prod_{m=1}^{M} f_m\big(\mathbf{x}_{N(m)}\big)$. We let $N(n) \subseteq \{1, 2, \ldots, M\}$ represent the set of function nodes neighboring variable node n and $N(m) \subseteq \{1, 2, \ldots, N\}$ the set of variable nodes connected to function node m, such that $\mathbf{x}_{N(m)}$ is the argument of function f_m and is shorthand for the set $\{x_n\}_{n \in N(m)}$. The current beliefs or marginal of each variable node are referred to as $q_n(x_n)$ and the clique marginal of variable nodes connected to function node f_m are $q_m(\mathbf{x}_{N(m)})$.

Belief propagation updates are now shown to be equivalent to coordinate descent minimization performed on the Bethe free energy expression in equation (2.21), reproduced here:

$$\begin{aligned}\mathcal{F}_{Bethe} = \; & \sum_m \sum_{\mathbf{x}_{N(m)}} q_m(\mathbf{x}_{N(m)}) \cdot \log q_m(\mathbf{x}_{N(m)}) - \sum_m \sum_{\mathbf{x}_{N(m)}} q_m(\mathbf{x}_{N(m)}) \cdot \log f_m(\mathbf{x}_{N(m)}) \\ & - \sum_n (|N(n)|{-}1) \sum_{x_n} q_n(x_n) \cdot \log q_n(x_n)\end{aligned}$$

Constraints must be added to ensure that the q-distributions are valid probability distributions, *i.e.*, $\forall n : \sum_{x_n} q_n(x_n) = 1$ and $\forall m : \sum_{\mathbf{x}_{N(m)}} q_m(\mathbf{x}_{N(m)}) = 1$, and that the single-node marginals are consistent with clique marginals $\forall m, n \in N(m) :\; q_n(x_n) = \sum_{\mathbf{x}_{N(m)\setminus n}} q_m(\mathbf{x}_{N(m)})$. Incorporating these as Lagrange multipliers, the expression to minimize becomes:

$$\begin{aligned}\mathcal{L} = \; & \mathcal{F}_{\text{Bethe}} + \textstyle\sum_m \sum_{n\in N(m)} \sum_{x_n} \lambda_{mn}(x_n) \left[q_n(x_n) - \sum_{\mathbf{x}_{N(m)\setminus n}} q_m(\mathbf{x}_{N(m)}) \right] \\ & + \textstyle\sum_m \alpha_m \left[1 - \sum_{\mathbf{x}_{N(m)}} q_m(\mathbf{x}_{N(m)}) \right] + \sum_n \beta_n \left[1 - \sum_{x_n} q_n(x_n) \right]\end{aligned}$$

Taking partial derivatives with respect to the marginals and setting them to zero yields:

$$\begin{aligned}&\partial\mathcal{L}/\partial q_n(x_n)\colon -(|N(n)|-1) - (|N(n)|-1)\cdot \log q_n(x_n) + \textstyle\sum_{m\in N(n)} \lambda_{mn}(x_n) - \beta_n = 0\,, \\ &\partial\mathcal{L}/\partial q_m(\mathbf{x}_{N(m)})\colon 1 + \log q_m(\mathbf{x}_{N(m)}) - \log f_m(\mathbf{x}_{N(m)}) - \underbrace{\textstyle\sum_{n\in N(m)} \lambda_{mn}(x_n)}_{\substack{\text{because} \\ \sum_{n\in N(m)} \sum_{x_n} \lambda_{mn}(x_n) \sum_{\mathbf{x}_{N(m)\setminus n}} q_m(\mathbf{x}_{N(m)}) = \sum_{n\in N(m)} \sum_{\mathbf{x}_{N(m)}} \lambda_{mn}(x_n) q_m(\mathbf{x}_{N(m)})}} - \alpha_m = 0\,.\end{aligned} \tag{A.1}$$

Solving for the marginals yields an update equation in terms of the Lagrange multipliers α_m, β_n, and $\lambda_{mn}(x_n)$. The first two are constant with respect to x and can be dropped if unit normalization is performed after each update.

$$\begin{aligned}q_n(x_n) &= e^{\frac{1}{|N(n)|-1}\sum_{m\in N(n)} \lambda_{mn}(x_n) - \frac{\beta_n}{|N(n)|-1} - 1} \propto \prod_{m\in N(n)} e^{\lambda_{mn}(x_n)/(|N(m)|-1)}\,, \\ q_m(\mathbf{x}_{N(m)}) &= e^{\sum_{n\in N(m)} \lambda_{mn}(x_n) + \log f_m(\mathbf{x}_{N(m)}) + \alpha_m - 1} \propto f_m(\mathbf{x}_{N(m)}) \cdot \prod_{n\in N(m)} e^{\lambda_{mn}(x_n)}\,.\end{aligned} \tag{A.2}$$

The "message" from variable node X_n to function node f_m can be defined as $\nu_{n\to m}(x_n) = e^{\lambda_{mn}(x_n)}$, which leads to simple expressions for the marginals, $q_n(x_n) \propto \prod_{m\in N(n)} \nu_{n\to m'}^{1/(|N(m)|-1)}$ and $q_m(\mathbf{x}_{N(m)}) \propto f_m(\mathbf{x}_{N(m)}) \cdot \prod_{n\in N(m)} \nu_{n\to m}(x_n)$. These expressions can be substituted into the marginals' consistency constraint, $\sum_{\mathbf{x}_{N(m)\setminus n}} q_m(\mathbf{x}_{N(m)}) = q_n(x_n)$, to yield:

$$\begin{aligned}\sum_{\mathbf{x}_{N(m)\setminus n}} \left(f_m(\mathbf{x}_{N(m)}) \cdot \prod_{n'\in N(m)} \nu_{n'\to m}(x_{n'}) \right) \; & \propto \prod_{m'\in N(n)} \nu_{n\to m'}^{1/(|N(m')|-1)} \\ & \Downarrow \\ \sum_{\mathbf{x}_{N(m)\setminus n}} \left(f_m(\mathbf{x}_{N(m)}) \cdot \prod_{n'\in N(m)\setminus n} \nu_{n'\to m}(x_{n'}) \right) \; & \propto \; \nu_{n\to m}(x_n)^{-1} \cdot \prod_{m'\in N(n)} \nu_{n\to m'}^{1/(|N(m')|-1)}\end{aligned} \tag{A.3}$$

Finally, the "message" from function node f_m to variable node X_n is defined to be $\mu_{m\to n}(x_n) =$

$\sum_{\mathbf{x}_{N(m)\backslash n}} f_m(\mathbf{x}_{N(m)}) \cdot \prod_{n' \in N(m)\backslash n} \nu_{n' \to m}(x_{n'})$. In this case, the final expression can be manipulated from equation (A.3) to make the following simplifications:

$$\begin{aligned}
\mu_{m\to n}(x_n) &\propto \nu_{n\to m}(x_n)^{-1} \cdot \prod_{m' \in N(n)} \nu_{n\to m'}(x_n)^{1/(|N(m')|-1)} \\
&\Downarrow \\
\prod_{m\in N(n)} \mu_{m\to n}(x_n) &\propto \prod_{m\in N(n)} \left[\nu_{n\to m}(x_n)^{-1} \cdot \textstyle\prod_{m' \in N(n)} \nu_{n\to m'}(x_n)^{1/(|N(m')|-1)}\right] \\
&= \left[\prod_{m\in N(n)} \nu_{n\to m}(x_n)^{-1}\right] \cdot \left[\prod_{m\in N(n)} \nu_{n\to m}(x_n)^{|N(m)|/(|N(m)|-1)}\right] \\
&= \prod_{m\in N(n)} \nu_{n\to m}(x_n)^{-1+|N(m)|/(|N(m)|-1)} = \prod_{m\in N(n)} \nu_{n\to m}(x_n)^{1/(|N(m)|-1)} \propto q_n(x_n)
\end{aligned}$$

and

$$\begin{aligned}
\mu_{m\to n}(x_n) &\propto \nu_{n\to m}(x_n)^{-1} \cdot \prod_{m' \in N(n)} \nu_{n\to m'}(x_n)^{1/(|N(m')|-1)} \\
&\Downarrow \\
\prod_{m'\in N(n)\backslash m} \mu_{m'\to n}(x_n) &\propto \prod_{m'\in N(n)\backslash m} \left[\nu_{n\to m'}(x_n)^{-1} \cdot \textstyle\prod_{m'' \in N(n)} \nu_{n\to m''}(x_n)^{1/(|N(m'')|-1)}\right] \\
&= \left[\nu_{n\to m}(x_n) \cdot \prod_{m' \in N(n)} \nu_{n\to m'}(x_n)^{-1}\right] \cdot \left[\prod_{m' \in N(n)} \nu_{n\to m'}(x_n)^{\frac{|N(m')\backslash m|}{|N(m')|-1}}\right] \\
&= \nu_{n\to m}(x_n)\,.
\end{aligned}$$

To summarize, we find the now-familiar belief propagation update equations:

$$\begin{aligned}
\nu_{n\to m}(x_n) \propto \prod_{m'\in N(n)\backslash m} \mu_{m'\to n}(x_n) \quad \text{and} \quad \mu_{m\to n}(x_n) \propto \sum_{\mathbf{x}_{N(m)\backslash n}} f_m(\mathbf{x}_{N(m)}) \cdot \prod_{n'\in N(m)\backslash n} \nu_{n'\to m}(x_{n'})\,, \\
q_n(x_n) \propto \prod_{m\in N(n)} \mu_{m\to n}(x_n) \quad \text{and} \quad q_m(\mathbf{x}_{N(m)}) \propto f_m(\mathbf{x}_{N(m)}) \cdot \prod_{n\in N(m)} \nu_{n\to m}(x_n)\,.
\end{aligned} \tag{A.4}$$

Bibliography

[1] S. M. Aji and R. J. McEliece. The Generalized Distributive Law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[2] S. An, W. Liu, and S. Venkatesh. Acquiring Critical Light Points for Illumination Subspaces of Face Images by Affinity Propagation Clustering. In *Proc. Pacific-Rim Conference on Multimedia (PCM)*, 2007.

[3] A. Banerjee, S. Merugu, I.S. Dhillon, and J. Ghosh. Clustering with Bregman Divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.

[4] A.E. Baya and P.M. Granitto. ISOMAP based metrics for clustering. In *Revista Iberoamericana de Inteligencia Artificial*, number 37, pages 15–23, 2008.

[5] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes (1). *IEEE International Conference on Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record*, 2:1064–1070 vol.2, 23-26 May 1993.

[6] Julian Besag. On the statistical analysis of dirty pictures (with discussion). *Journal of the Royal Statistical Society, Series B*, 48(3):259–302, 1986.

[7] H. A. Bethe. Statistical Theory of Superlattices. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 150(871):552–575, 1935.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, August 2006.

[9] Y. Boykov, O. Veksler, and R. Zabih. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, November 2001.

[10] M. J. Brusco and H.-F. Kohn. Comment on ”Clustering by Passing Messages Between Data Points”. *Science*, 319(5864):726c–, 2008.

[11] G. C., M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23:163–177, 1977.

[12] J.A. Cardille and M. Lambois. Widespread human signature in representative U.S. landscapes. In *93rd Ecological Society of America Annual Meeting*, 2008.

[13] V. Cerny. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[14] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *J. Comput. Syst. Sci.*, 65(1):129–149, 2002.

[15] L. Cooper. Location-allocation problems. *Operations Research*, 11:331–343, 1963.

[16] L. Cooper. Heuristic Methods for Location-Allocation Problems. *SIAM Review*, 6(1):37–53, 1964.

[17] L. Cooper. Solutions of generalized locational equilibrium models. *Journal of Regional Science*, 7:1–17, 1967.

[18] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley-Interscience, August 1991.

[19] CPLEX Optimization Inc. CPLEX Linear Optimizer and Mixed Integer Optimizer. Suite 279, 930 Tahoe Blvd. Bldg 802, Incline Village, NV 89541.

[20] Michelangelo Merisi da Caravaggio. Vocazione di san Matteo (The Calling of St. Matthew). Hanging in Contarelli Chapel at San Luigi dei Francesi, Rome, Italy, 1599.

[21] S. Dasgupta and L. Schulman. A Probabilistic Analysis of EM for Mixtures of Separated, Spherical Gaussians. *J. Mach. Learn. Res.*, 8:203–226, 2007.

[22] S. Dasgupta and L. J. Schulman. A Two-Round Variant of EM for Gaussian Mixtures. In *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 152–159, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[23] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[24] C. Du, J. Yang, Q. Wu, and F. Li. Integrating affinity propagation clustering method with linear discriminant analysis for face recognition. *Optical Engineering*, 46(11):110501, 2007.

[25] D. Dueck and B. J. Frey. Non-metric affinity propagation for unsupervised image categorization. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, Oct. 2007.

[26] D. Dueck, B. J. Frey, N. Jojic, V. Jojic, G. Giaever, A. Emili, G. Musso, and R. Hegele. Constructing Treatment Portfolios Using Affinity Propagation. In *RECOMB*, pages 360–371, 2008.

[27] A. A. Efros and T. K. Leung. Texture Synthesis by Non-parametric Sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, Corfu, Greece, September 1999.

[28] S. Eilon and R. D. Galvão. Single and double vertex substitution in heuristic procedures for the p-median problem. *Management Science*, 24:1763–1766, 1978.

[29] E. Feldman, F. A. Lehrer, and T. L. Ray. Warehouse location under continuous economies of scale. *Management Science*, 12:670–684, 1966.

[30] W. Fischer, S. Perkins, J. Theiler, T. Bhattacharya, K. Yusim, R. Funkhouser, C. Kuiken, B. Haynes, N. L Letvin, B. D Walker, B. H Hahn, and B. T Korber.

[31] L. Ford and D. Fulkerson. Maximal Flow Through a Network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[32] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.

[33] W. T. Freeman and E. C. Pasztor. Learning to Estimate Scenes from Images. In *NIPS*, pages 775–781, 1998.

[34] B. Frey and D. Dueck. Mixture Modeling by Affinity Propagation. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 379–386. MIT Press, Cambridge, MA, 2006.

[35] B. J. Frey and D. Dueck. Response to Comment on "Clustering by Passing Messages Between Data Points". *Science*, 319(5864):726d–, 2008.

[36] B. J. Frey, N. Mohammad, Q. D. Morris, W. Zhang, M. D. Robinson, S. Mnaimneh, R. Chang, Q. Pan, E. Sat, J. Rossant, B. G. Bruneau, J. E. Aubin, B. J. Blencowe, T. R. Hughesverre, and F. Mitelman. Genome-wide analysis of mouse transcripts using exon microarrays and factor graphs. *Nature Genetics*, 37:991–996, 2005.

[37] B. J. Frey, Q. Morris, M. D. Robinson, and T. R. Hughes. Finding Novel Transcripts in High-Resolution Genome-Wide Microarray Data Using the GenRate Model. In *RECOMB*, pages 66–82, 2005.

[38] B.J. Frey and D. Dueck. Clustering by Passing Messages Between Data Points. *Science*, 315:972–976, 2007.

[39] Y. Fu, Z. Li, X. Zhou, and T.S. Huang. Laplacian Affinity Propagation for Semi-Supervised Object Classification. *IEEE International Conference on Image Processing, 2007. ICIP 2007.*, 1:I –189–I –192, Oct 2007.

[40] D. Gavrila and V. Philomin. Real-Time Object Detection for "Smart" Vehicles. In *ICCV*, pages 87–93, 1999.

[41] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6:721–741, 1984.

[42] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[43] D. Greig, B. Porteous, and A. Seheult. Exact maximum a-posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B*, 51(2):271–279, 1989.

[44] N. Grira and M.E. Houle. Best of both: a hybridized centroid-medoid clustering heuristic. In Zoubin Ghahramani, editor, *ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 313–320. ACM, 2007.

[45] S. L. Hakimi. Optimum Locations of Switching Centers and the Absolute Centers and Medians of a Graph. *Operations Research*, 12:450–459, 1964.

[46] S. L. Hakimi. Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph Theoretic Problems. *Operations Research*, 13:462–475, 1965.

[47] P. Hansen and N. Mladenovic. Variable neighborhood search for the p-median. *Location Science*, 5(4):207–226, 1997.

[48] G. E. Hinton and M. Revow. Using Pairs of Data-Points to Define Splits for Decision Trees. In D. S. Touretzky, M. Mozer, and M. E. Hasselmo, editors, *NIPS*, pages 507–513. MIT Press, 1995.

[49] D. S. Hochbaum and D. B. Shmoys. A Best Possible Heuristic for the k-Center Problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[50] T. Jiang and A. Tuzhilin. Dynamic Micro Targeting: Fitness-Based Approach to Predicting Individual Preferences. In *ICDM*, pages 173–182. IEEE Computer Society, 2007.

[51] W. Jiang, F. Ding, and Q.-L. Xiang. An Affinity Propagation Based method for Vector Quantization Codebook Design. *CoRR*, abs/0710.2037, 2007.

[52] N. Jojic, B. J. Frey, and A. Kannan. Epitomic analysis of appearance and shape. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 34, Washington, DC, USA, 2003. IEEE Computer Society.

[53] N. Jojic, V. Jojic, B. Frey, C. Meek, and D. Heckerman. Using "epitomes" to model genetic diversity: Rational design of HIV vaccine cocktails. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 587–594. MIT Press, Cambridge, MA, 2006.

[54] Vladimir Jojic. *Algorithms for rational vaccine design*. PhD thesis, 2007.

[55] L. G. Kachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[56] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. II: The p-medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560, December 1979.

[57] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, December 1984.

[58] C. Kent, R. Azanchi, B. Smith, A. Chu, and J. Levine. A Model-Based Analysis of Chemical and Temporal Patterns of Cuticular Hydrocarbons in Male Drosophila melanogaster. *PLoS ONE*, 2(9):e962, Sep 2007.

[59] J. Kim and H. Park. Sparse Nonnegative Matrix Factorization for Clustering. Technical Report GT-CSE-08-01, Georgia Institute of Technology, Computational Science and Engineering Technical Reports, 2008.

[60] R. Kindermann and J.L. Snell. *Markov Random Fields and Their Applications*. American Mathematical Society, Providence, RI, 1980.

[61] Benjamin King. Step-Wise Clustering Procedures. *Journal of the American Statistical Association*, 62(317):86–101, 1967.

[62] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[63] R. Koetter, B. J. Frey, and N. Petrovic. Unwrapping phase images by propagating probabilities across graphs. In *ICASSP '01: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 2001*, pages 1845–1848, Washington, DC, USA, 2001. IEEE Computer Society.

[64] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):147–159, 2004.

[65] F.R. Kschischang, B.J. Frey, and H.-A.Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEETIT: IEEE Transactions on Information Theory*, 47, 2001.

[66] A. A. Kuehn and M. J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9:643–666, 1963.

[67] D. Lashkari and P. Golland. Convex Clustering with Exemplar-Based Models. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 825–832. MIT Press, Cambridge, MA, 2008.

[68] M. Leone, Sumedha, and M. Weigt. Clustering by soft-constraint affinity propagation: Applications to gene-expression data. *Bioinformatics*, 23(20):2708–2715, October 2007.

[69] J.-H. Lin and J. S. Vitter. Approximation Algorithms for Geometric Median Problems. Technical Report CS-92-37, 1992.

[70] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathemtical Statistics and Probability*, pages 281–297, 1967.

[71] Simon A. Mallal. The Western Australian HIV Cohort Study, Perth, Australia. *Journal of Acquired Immune Deficiency Syndromes and Human Retrovirology*, 17:S23–S27, 1998.

[72] F. E. Maranzana. On the location of supply points to minimize transport costs. *Operations Research Quarterly*, 15:261–270, 1964.

[73] R. J. Mceliece, D. J. C. Mackay, and J.-F. Cheng. Turbo decoding as an instance of Pearl's "Belief Propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152, 1998.

[74] M. Meila and J. Shi. Learning Segmentation by Random Walks. In *NIPS*, pages 873–879, 2000.

[75] N. Metropolis, A.W. Rosenbluth, A.H. Teller, and E. Teller. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[76] M. Mézard, G. Parisi, and R. Zecchina. Analytic and Algorithmic Solution of Random Satisfiability Problems. *Science*, 297(5582):812–815, 2002.

[77] Marc Mézard. Where Are the Exemplars? *Science*, 315(5814):949–951, 2007.

[78] N. Mladenovic, J. Brimberg, P. Hansen, and J. A. Moreno-Pérez. The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179(3):927–939, 2007.

[79] K. P. Murphy, Y. Weiss, and M. Jordan. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

[80] A. Ng, M. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, number 14, pages 849–856, Cambridge, MA, 2002. MIT Press.

[81] D.C. Nickle, M. Rolland, M. A Jensen, S. L. Kosakovsky Pond, W. Deng, M. Seligman, D. Heckerman, J. I. Mullins, and N. Jojic. Coping with Viral Diversity in HIV Vaccine Design. *PLoS Comput. Biol.*, 3(4):e75, Apr 2007.

[82] S. Nowozin and G. Bakir. A Decoupled Approach to Exemplar-based Unsupervised Learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, 2008.

[83] Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine*, pages 329–334, August 1985.

[84] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[85] K. D. Pruitt, T. A. Tatusova, and D. R. Maglott. NCBI Reference Sequence Project: update and current status. *Nucleic Acids Research*, 31(1):34–37, 2003.

[86] C. S. ReVelle and R. Swain. Central facilities location. *Geographical Analysis*, 2:30–42, 1970.

[87] F. Samaria and F. Fallside. Face Identification and Feature Extraction Using Hidden Markov Models. In G. Vernazza, editor, *Image Processing: Theory and Applications*, pages 295–298. Elsevier, June 1993.

[88] S. Sanghavi, D. Malioutov, and A. Willsky. Linear programming analysis of loopy belief propagation for weighted matching. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1273–1280. MIT Press, Cambridge, MA, 2008.

[89] R. Santana, P. Larranaga, and J.A. Lozano. Learning factorizations in estimation of distribution algorithms using affinity propagation. Technical Report EHU-KZAA-IK-1/08, Department of Computer Science and Aritificial Intelligence, University of the Basque Country, 2008.

[90] J. C. Schlimmer. *Concept acquisition through representational adjustment*. PhD thesis, 1987.

[91] A.J. Senf, C. Leonard, and J. DeLeo. A Statistical Algorithm to Discover Knowledge in Medical Data Sources. In *ICMLA '07: Proceedings of the Sixth International Conference on Machine Learning and Applications*, pages 537–540, Washington, DC, USA, 2007. IEEE Computer Society.

[92] E. Shechtman, Y. Caspi, and M. Irani. Space-Time Super-Resolution. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(4):531–545, 2005.

[93] P. H. A. Sneath and R. R. Sokal. *Numerical taxonomy: the principles and practice of numerical classification*. Freeman, San Francisco, USA, 1973.

[94] D. Sontag, T. Meltzer, A. Globerson, T. Jaakkola, and Y. Weiss. Tightening LP Relaxations for MAP using message passing. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*. AUAI Press, Arlington, Virginia, 2008.

[95] D. Tarlow, R. Zemel, and B. Frey. Flexible Priors for Exemplar-based Clustering. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*. AUAI Press, Arlington, Virginia, 2008.

[96] M. B. Teitz and P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16(5):955–961, September–October 1968.

[97] K. Toyama and A. Blake. Probabilistic Tracking with Exemplars in a Metric Space. *Int. J. Comput. Vision*, 48(1):9–19, 2002.

[98] N. Ueda, R. Nakano, Z. Ghahramani, and G. E. Hinton. Split and Merge EM Algorithm for Improving Gaussian Mixture Density Estimates. *The Journal of VLSI Signal Processing*, pages 133–140, August 2000.

[99] S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, 2000.

[100] R. Verma and P. Wang. On Detecting Subtle Pathology via Tissue Clustering of Multiparametric Data using Affinity Propagation. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, Oct. 2007.

[101] T. Vetter and T. Poggio. Image Synthesis from a Single Example Image. In *ECCV '96: Proceedings of the 4th European Conference on Computer Vision-Volume I*, pages 652–659, London, UK, 1996. Springer-Verlag.

[102] K. Wang, J. Zhang, D. Li, X. Zhang, and T. Guo. Adaptive Affinity Propagation Clustering. *Acta Automatica Sinica*, 33(12).

[103] Joe H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[104] Yair Weiss. Segmentation Using Eigenvectors: A Unifying View. In *ICCV '99: Proceedings of the International Conference on Computer Vision*, volume 2, page 975, Washington, DC, USA, 1999. IEEE Computer Society.

[105] Y. Wexler, E. Shechtman, and M. Irani. Space-Time Video Completion. *CVPR*, 01:120–127, 2004.

[106] R. A. Whitaker. A fast algorithm for the greedy interchange for large-scale clustering and median location. *Canadian Journal of Operations Research and Information Processing*, 21:95–108, 1983.

[107] T. Wittkop, J. Baumbach, F. Lobo, and S. Rahmann. Large scale clustering of protein sequences with FORCE – A layout based heuristic for weighted cluster editing. *BMC Bioinformatics*, 8(1), 2007.

[108] J. Xiao, J. Wang, P. Tan, and L. Quan. Joint Affinity Propagation for Multiple View Segmentation. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–7, Oct. 2007.

[109] C. Yanover, T. Meltzer, and Yair Weiss. Linear Programming Relaxations and Belief Propagation — An Empirical Study. *Journal of Machine Learning Research*, 7:1887–1907, 2006.

[110] C. Yeats, J. Lees, A. Reid, P. Kellam, N. Martin, X. Liu, and C.A. Orengo. Gene3D: comprehensive structural and functional annotation of genomes. *Nucleic Acids Research*, 36:414–418, 2008.

[111] J.S. Yedidia, W.T. Freeman, and Y. Weiss. Generalized Belief Propagation. In *NIPS*, pages 689–695, 2000.

[112] J. Yi, Y. Peng, and J. Xiao. Color-based clustering for text detection and extraction in image. In *MULTIMEDIA '07: Proceedings of the 15th International Conference on Multimedia*, pages 847–850, New York, NY, USA, 2007. ACM.

[113] H. Zhang, Y. Zhuang, and F. Wu. Cross-modal correlation learning for clustering on image-audio dataset. In *MULTIMEDIA '07: Proceedings of the 15th International Conference on Multimedia*, pages 273–276, New York, NY, USA, 2007. ACM.

[114] X. Zhang, C. Furtlehner, and M. Sebag. Data Streaming with Affinity Propagation. 2008.

[115] X. Zhang, J. Gao, P. Lu, and Y. Yan. A novel speaker clustering algorithm via supervised affinity propagation. In *Proceedings of International Conference on Accoustics, Speech, and Signal Processing*, pages 4369–4372, 2008.