A Binary Variable Model for Affinity Propagation

Inmar E. Givoni and Brendan J. Frey

January 9, 2009

Abstract

Affinity propagation (AP) was recently introduced as an unsupervised learning algorithm for exemplar-based clustering. We present a derivation of AP that is much simpler than the original one and is based on a quite different graphical model. The new model allows easy derivations of message updates for extensions and modifications of the standard AP algorithm. We demonstrate this by adjusting the new AP model to represent the capacitated clustering problem. For those wishing to investigate or extend the graphical model of the AP algorithm, we suggest using this new formulation since it allows for simpler and more intuitive model manipulation.

1 Introduction

Affinity propagation (Frey & Dueck, 2007) is a clustering algorithm that, given a set of similarities between pairs of data points, exchanges messages between data points so as to find a subset of exemplar points that best describe the data. AP associates each data point with one exemplar, resulting in a partitioning of the whole data set into clusters. The goal of AP is to minimize the overall sum of similarities between data points and their exemplars.

AP was originally derived as an instance of the max-product (belief-propagation) algorithm in a loopy factor-graph (Kschischang, Frey, & Loeliger, 2001; Pearl, 1988). The simple scalar messages updates for affinity propagation were obtained by Frey & Dueck (2007) only after cumbersome manipulations of the standard max-product message updates that reduced N-ary messages to binary ones. Any modifications to the algorithm such as introducing useful additional constraints via function nodes, or modifying the functional form of the factors require re-deriving the message updates, subject to these modifications. Obtaining the simplified messages entails further manipulations of the update equations.

We describe an alternative yet equivalent formulation for the AP factor graph, which allows simpler derivations of the AP message updates. Further, in our experience, this model simplifies message derivation for modifications to the AP model. We demonstrate the ease of use of the new model by showing how AP can be easily extended to solve a capacitated clustering problem (Mulvey & Beck, 1984) where each cluster has an upper limit L on the number of points it can contain.

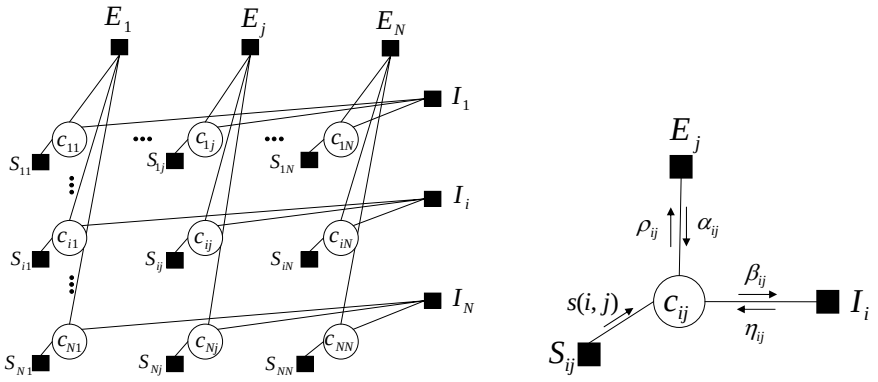
2 A binary model for affinity propagation

Let $\{c_{ij}\}_{j=1}^{N}$ be N binary variables associated with data point i ($i \in 1, \ldots, N$), s.t. $c_{ij} = 1$ iff the exemplar for point i is point j. In this notation, $c_{jj} = 1$ indicates that j is an exemplar. All assignments to exemplars and all exemplar choices can be described by the set of N^2 binary variables $\{c_{ij}\}_{i=1,j=1}^{N}$.

Each data point in affinity propagation clustering is assigned to a single exemplar. Therefore, the first constraint that must be accounted for in the binary variable formulation is that $\sum_{j=1}^{N} c_{ij} = 1$. We refer to this as the '1-of-N' constraint. An additional constraint from the original AP formulation is the 'exemplar consistency' constraint stating that node i may only choose j as its exemplar if j chose itself as an exemplar. Fig. 1a shows a factor graph for affinity propagation that uses the above variables and constraints. The 1-of-N constraints are introduced via the I function nodes; in every row i of the grid, exactly one $c_{ij}, j \in \{1, \ldots, N\}$ variable must be set to 1. The E function nodes enforce the exemplar consistency constraints; in every column j, the set of $c_{ij}, i \in \{1, , \ldots, N\}$ variables set to 1 indicates all the points who have chosen point j as their exemplar. For these points to be able to choose point j as an exemplar, j must also choose itself as an exemplar (c_{jj} must also equal 1).

Since we intend to use the max-sum (log-domain max-product) formulation, where local functions are added to form the global objective function to maximize, the I and E functions can be naturally defined as follows:

$$I_i(c_{i1}, \ldots, c_{iN}) = \begin{cases} -\infty & \text{if } \sum_j c_{ij} \neq 1, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$



a b

Figure 1: A binary variable model for affinity propagation

$$E_j(c_{1j}, \ldots, \ldots, c_{Nj}) = \begin{cases} -\infty & \text{if } c_{jj} = 0 \text{ and } \exists i \neq j \text{ s.t. } c_{ij} = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The S_{ij} function nodes incorporate the user-defined input similarities $s(i,j)$ between data points and their potential exemplars.

$$S_{ij}(c_{ij}) = \begin{cases} s(i,j) & \text{if } c_{ij} = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The graphical model in Fig. 1a together with (1)-(3) result in the following max-sum objective function:

$$\mathcal{S}(c_{11}, \ldots, c_{NN}) = \sum_{i,j} S_{ij}(c_{ij}) + \sum_i I_i(c_{i1}, \ldots, c_{iN}) + \sum_j E_j(c_{1j}, \ldots, c_{Nj})$$

There are 5 message types passed between variable nodes and function nodes when executing the max-sum algorithm on the graph. They are annotated in Fig. 1b. Since each c_{ij} variable is binary, it may appear we need

to send two-valued messages between nodes. In practice, for any message we only need to propagate the *difference* between the message values for its two possible settings. The original two-valued message can be recovered from this scalar up to an additive constant, which is not important because adding a constant to all values of a message does not alter the output of the algorithm. We can also think of these scalar messages as pseudo log-likelihood ratios. When the algorithm terminates, we recover the estimated MAP settings for any hidden variable by adding all its incoming messages together. To make decisions, hidden variables corresponding to positive merged messages are set to one, and all the others are set to zero.

We now derive the scalar message updates in the binary variable AP model. Recall the max-sum message update rules (Bishop, 2006):

$$\mu_{x\to f}(x) = \sum_{\{l|f_l\in \mathrm{ne}(x)\backslash f\}} \mu_{f_l\to x}(x), \tag{4}$$

$$\mu_{f\to x}(x) = \max_{x_1,\ldots,x_M} [f(x,x_1,\ldots,x_m) + \sum_{\{m|x_m\in \mathrm{ne}(f)\backslash x\}} \mu_{x_m\to f}(x_m)] \tag{5}$$

where the notation $\mathrm{ne}(x)\backslash f$ is used to indicate the set of variable node x's neighbors excluding function node f, and $\mathrm{ne}(f)\backslash x$ is used to indicate the set of function node f's neighbors excluding variable node x.

We denote by $\beta_{ij}(m)$ the message value for setting the hidden variable c_{ij} to m, $m \in \{0,1\}$:

$$\beta_{ij}(m) = \mu_{c_{ij}\to I_i}(m)$$

The scalar message difference $\beta_{ij}(1) - \beta_{ij}(0)$ is denoted by β_{ij}. Similar notation is used for α, ρ and η. In what follows, for each message we cal-

culate its value for each setting of the binary variable, and then take the difference. Note that the scalar message sent from the S_{ij} function is always the similarity between i and j since $S_{ij}(1) - S_{ij}(0) = s(i,j)$.

We begin by calculating the messages sent from the variable nodes to the function nodes, using (4). The message sent from a c_{ij} variable node to its I_i function node is simply the sum of all incoming messages to the c_{ij} variable node, except for the message *arriving* from the I_i node itself. These incoming messages are the S_{ij} and α_j messages.

For $c_{ij} = 1$:

$$\begin{aligned} \beta_{ij}(1) &= \mu_{c_{ij} \to I_i}(1) = \sum_{\{l | f_l \in \mathrm{ne}(c_{ij}) \setminus I_i\}} \mu_{f_l \to c_{ij}}(1) \\ &= S_{ij}(1) + \alpha_{ij}(1) \end{aligned} \tag{6}$$

Similarly, for $c_{ij} = 0$:

$$\beta_{ij}(0) = S_{ij}(0) + \alpha_{ij}(0) \tag{7}$$

Taking the difference between $\beta_{ij}(1)$ and $\beta_{ij}(0)$ we obtain:

$$\begin{aligned} \beta_{ij} &= \beta_{ij}(1) - \beta_{ij}(0) \\ &= \big(S_{ij}(1) - S_{ij}(0)\big) + \big(\alpha_{ij}(1) - \alpha_{ij}(0)\big) \\ &= s(i,j) + \alpha_{ij} \end{aligned} \tag{8}$$

Using a similar set of equations, we obtain the ρ_{ij} messages that are sent to the E_j function nodes. These can be expressed as the sum of all incoming

messages to the c_{ij} variable node, except for the message arriving from the E_j node:

$$\begin{aligned} \rho_{ij}(1) &= S_{ij}(1) + \eta_{ij}(1) \\ \rho_{ij}(0) &= S_{ij}(0) + \eta_{ij}(0) \\ \rho_{ij} &= \rho_{ij}(1) - \rho_{ij}(0) = s(i,j) + \eta_{ij} \end{aligned} \tag{9}$$

We now turn our attention to the messages sent from the function nodes to the variable nodes, which are calculated using (5). When calculating a η_{ij} message sent from function node I_i to the c_{ij} variable node, we fix the value of c_{ij} to 1 or 0, and we find an optimal assignment of the hidden variables c_{ik}, $k \neq j$. Any assignment that violates the I_i function '1-of-N' constraint results in that function evaluating to $-\infty$, and such an assignment is trivially non-optimal.

When $c_{ij} = 1$ we get:

$$\begin{aligned} \eta_{ij}(1) &= \mu_{I_i \to c_{ij}}(1) \\ &= \max_{c_{ik}, k \neq j} [I_i(c_{i1}, \ldots, c_{ij} = 1, \ldots, c_{iN}) + \sum_{\{t | c_{it} \in \mathrm{ne}(I_i) \setminus c_{ij}\}} \mu_{c_{it} \to I_i}(c_{it})] \\ &= \max_{c_{ik}, k \neq j} [I_i(c_{i1}, \ldots, c_{ij} = 1, \ldots, c_{iN}) + \sum_{t \neq j} \beta_{it}(c_{it})] \\ &= \sum_{t \neq j} \beta_{it}(0). \end{aligned} \tag{10}$$

The final equality holds since $c_{ij} = 1$ indicates point i has chosen point j as its exemplar, and therefore no other point can be i's exemplar. Consequently, the only assignment of the c_{ik}, $k \neq j$ variables that satisfies the '1-of-N'

constraint is setting them all to zero. Being the only satisfying assignment, it is also the optimal one, and the I_i function evaluates to its maximal value of 0.

When $c_{ij} = 0$ we get:

$$\begin{aligned} \eta_{ij}(0) &= \max_{c_{ik}, k \neq j} [I_i(c_{i1}, \ldots, c_{ij} = 0, \ldots, c_{iN}) + \sum_{t \neq j} \beta_{it}(c_{it})] \\ &= \max_{k \neq j} [\beta_{ik}(1) + \sum_{t \notin \{k,j\}} \beta_{it}(0)]. \end{aligned} \tag{11}$$

In the above, when c_{ij} is set to 0 there is more flexibility in choosing an optimal solution. Still, exactly one of the c_{ik}, $k \neq j$ variables must be set to one resulting in a maximization over $N - 1$ possible solutions.

Taking the difference between (10) and (11) we get

$$\begin{aligned} \eta_{ij} &= \eta_{ij}(1) - \eta_{ij}(0) \\ &= -\max_{k \neq j} [\beta_{ik}(1) + \sum_{t \notin \{k,j\}} \beta_{it}(0) - \sum_{t \neq j} \beta_{it}(0)] \\ &= -\max_{k \neq j} [\beta_{ik}(1) - \beta_{ik}(0)] = -\max_{k \neq j} \beta_{ik}. \end{aligned} \tag{12}$$

We turn to the α_{ij} messages, where we must distinguish between the cases $i = j$ and $i \neq j$, and we first describe the case $i = j$. Setting $c_{jj} = 1$ (j has chosen itself as an exemplar) is enough to guarantee that any setting of the other variables c_{kj}, $k \neq j$ will be valid (the other points can either chose j as an exemplar, or not) and the maximum can be taken over all the configurations with the exemplar consistency constraint trivially satisfied. On the other hand, when $c_{jj} = 0$ (j is not an exemplar), the only valid configuration satisfying the exemplar consistency constraint is for all the

other variables $c_{kj}, k \neq j$ to also equal 0 (none of them has chosen j as an exemplar). The message updates for these two setting are therefore:

$$\alpha_{jj}(1) = \sum_{k \neq j} \max_{c_{kj}} \rho_{kj}(c_{kj}), \tag{13}$$

and

$$\alpha_{jj}(0) = \sum_{k \neq j} \rho_{kj}(0). \tag{14}$$

The difference between (13) and (14) is

$$\alpha_{jj} = \sum_{k \neq j} \max[\rho_{kj}, 0]. \tag{15}$$

If $i \neq j$, for $c_{ij} = 1$ we get

$$\begin{aligned} \alpha_{ij}(1) &= \max_{c_{kj}, k \neq i} [E_j(c_{1j}, \ldots, c_{ij} = 1, \ldots, c_{Nj}) + \sum_{k \neq i} \rho_{kj}(c_{kj})] \\ &= \rho_{jj}(1) + \sum_{k \notin \{i,j\}} \max_{c_{kj}} \rho_{kj}(c_{kj}) \end{aligned} \tag{16}$$

The above follows because when $c_{ij} = 1$ (i has chosen j as its exemplar) the only valid configuration for c_{jj} is to equal 1 as well (j has chosen itself as an exemplar) due to the exemplar consistency constraint. Once c_{ij} and c_{jj} are set to 1, the configuration of the other hidden variables c_{kj}, $k \notin \{i, j\}$ is unconstrained (they can either choose j as an exemplar or not) and the maximum is taken over all other configurations.

For $c_{ij} = 0$ we get

$$\alpha_{ij}(0) = \max \left[\rho_{jj}(1) + \sum_{k \notin \{i,j\}} \max_{c_{kj}} \rho_{kj}(c_{kj}) \, , \, \sum_{k \neq i} \rho_{kj}(0) \right] \tag{17}$$

In the above case, there are two distinct types of valid configurations, depending on whether c_{jj} is set to 0 or to 1, and the optimal configuration is therefore the maximization over these two settings. If $c_{jj} = 1$ (j has chosen itself as an exemplar), the configuration of the other hidden variables c_{kj}, $k \notin \{i, j\}$, is unconstrained and the maximum is taken over all other configurations. Alternatively, if $c_{jj} = 0$ (j is not an exemplar), all other c_{kj}, $k \neq i$ variables must also be 0 (no other point may choose j as an exemplar).

Taking the difference between (16) and (17) gives

$$\begin{aligned} \alpha_{ij} &= \min\left[0, \rho_{jj}(1) + \sum_{k \notin \{i,j\}} \max_{c_{kj}} \rho_{kj}(c_{kj}) - \sum_{k \neq i} \rho_{kj}(0)\right] \\ &= \min\left[0, \rho_{jj} + \sum_{k \notin \{i,j\}} \max[\rho_{kj}, 0]\right], \end{aligned} \tag{18}$$

where we have used the fact that $x - \max[x, y] = \min[0, x - y]$, and $\max[x, y] - y = \max[x - y, 0]$.

To summarize, the message update equations are:

$$\beta_{ij} = s(i,j) + \alpha_{ij} \qquad \eta_{ij} = -\max_{k \neq j} \beta_{ik}$$

$$\rho_{ij} = s(i,j) + \eta_{ij}$$

$$\alpha_{ij} = \begin{cases} \displaystyle\sum_{k \neq j} \max[\rho_{kj}, 0] & i = j \\ \displaystyle\min\left[0, \rho_{jj} + \sum_{k \notin \{i,j\}} \max[\rho_{kj}, 0]\right] & i \neq j \end{cases}$$

Note that we can express ρ directly in terms of α:

$$\begin{aligned}\rho_{ij} &= s(i,j) + \eta_{ij} \\ &= s(i,j) - \max_{k \neq j} \beta_{ik} \\ &= s(i,j) - \max_{k \neq j} \big(s(i,k) + \alpha_{ik} \big).\end{aligned} \tag{19}$$

The α_{ij} messages are identical to the AP 'availability' messages $a(i,j)$, and the ρ_{ij} messages are identical to the AP 'responsibility' messages $r(i,j)$; thus we have recovered the original affinity propagation updates.

3 Deriving message updates for capacitated affinity propagation

In order to demonstrate the usefulness of the new derivation, we augmented the standard AP model with the constraint that each cluster has an upper limit L on the number of points it can contain. We denote this variant of AP as Capacitated AP, or CAP. This model can be used to solve a special case of the NP-hard problem known as the capacitated clustering problem (CCP) (Mulvey & Beck, 1984), where L is an integer, and the demands, or weights of the points, are equal and fixed to 1.

The graphical model representing CAP is the same as the one in Fig. 1a, where the only modification is in the definition of the E_j function nodes that now have an additional setting for which the function takes on a $-\infty$ value:

$$E_j(c_{1j},\ldots,\ldots,c_{Nj}) = \begin{cases} -\infty & \text{if } (c_{jj}=0 \text{ and } \exists i \neq j \text{ s.t. } c_{ij}=1) \text{ or } (\sum_i c_{ij} > L), \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

Deriving the new message updates requires re-deriving (13)-(18). As before, when calculating the message sent to variable node c_{ij} from function node E_j, we fix the value of c_{ij} to either 0 or 1, and we find the configuration of the hidden variables $c_{kj}, k \neq i$ that maximizes the function node value plus the sum of all incoming messages to the function node E_j, *except* the message sent from the variable node c_{ij} (*i.e.* we compute $\max_{c_{kj},k\neq i}[E_j(c_{1j},\ldots,c_{ij}=m,\ldots,c_{Nj}) + \sum_{k\neq i}\rho_{kj}(c_{kj})]$).

Therefore, given the new limit L on the number of points that can be assigned to an exemplar, the only difference in the logic used to derive the updates for (13)-(18) is with regard to the elements over which the max operation is taken in (13) and (16). In (13), we maximize over all $k \neq j$, hence we maximize over $N-1$ messages. When taking into account the limit L, we need to maximize only over the $L-1$ largest messages since the $N-(L-1)$ variables corresponding to the rest of the messages must be set to 0 in order to satisfy the capacity constraint. Similarly in (16) we maximize over all $k \notin \{i,j\}$, hence over $N-2$ messages while in the new update we need to maximize only over the $L-2$ largest messages.

Let $\mathcal{L}_j$ be the set of the $L-1$ largest ρ_{kj} messages sent from variable nodes c_{kj} to function node E_j , where $k \neq j$, and let $\mathcal{L}_{ij}$ be the set of $L-2$ largest ρ_{kj} messages, where $k \notin \{i,j\}$. The new update messages are:

$$\alpha_{ij} = \begin{cases} \sum_{\mathcal{L}_j} \max[\rho_{kj}, 0] & i = j \\ \min\left[0, \rho_{jj} + \sum_{\mathcal{L}_{ij}} \max[\rho_{kj}, 0]\right] & i \neq j \end{cases}$$

Where all other messages remain the same.

Since the sets $\mathcal{L}_j$ and $\mathcal{L}_{ij}$ do not need to be sorted, we can retain the $\mathcal{O}(N)$ runtime of calculating each α_{ij} message by using an optimized algorithm based on quicksort (?, ?)

We note that in order to derive the CAP message updates using the original N-ary variable model (Frey & Dueck, 2007), we would need to propagate the changes through the process of reducing N-ary messages to binary ones[1]. Without good familiarity with the reduction process, it may not even be immediately clear that a given modification to the model can still result in scalar messages. The binary model introduced here greatly simplifies the process of deriving the message updates.

Extending the CAP algorithm to the case where each potential exemplar j is associated with a different limit L_j is trivial and only requires selecting the appropriate largest $L_j - 1$ and $L_j - 2$ messages for each particular j. Furthermore, the general capacitated clustering problem with unrestricted demands and non-integer L_j's can also be represented by the same model with a slightly different E_j function. The solution, however, relies on solving a knapsack problem as an intermediate step. Since AP can be easily transformed to represent the facility location problem (Dueck et al., 2008) the

[1]See Supporting Online Material for (Frey & Dueck, 2007), equations (S2a)-(S8) at http://www.sciencemag.org/cgi/content/full/1136800/DC1

CAP formulation can also be transformed to give an algorithm for solving the well-known capacitated facility location problem (Sridharan, 1995). Finally, the hard limit on L used here can be replaced with a cost that increases as cluster size increases. A particular choice of this cost function that stems from an underlying Dirichlet process prior was used in (Tarlow, Zemel, & Frey, 2008).

4 Experimental validation

We compare the CAP algorithm to capacitated k-medoids (CKM) - a variant of k-medoids (KM),(Bishop, 2006) that was adapted to greedily account for the cluster size limit[2]. The measure we use to compare the algorithms is the total similarity: the sum of similarities between all non-exemplar points to their exemplar. We generated 100 different sets of 2D points sampled from a uniform distribution, where each set size is N=500. In order to make sure we choose a difficult capacity limit L (one that is not trivially satisfied by a standard clustering algorithm) we used the following procedure: for each data set we first ran regular AP with a range of preferences to obtain a range of different k values (recall that AP does not take as input a number k of clusters to output; the number of clusters is controlled by a tuneable 'preference' parameter). Then we ran standard KM for every obtained value of k. For each k value, We compared the total similarity obtained by AP

[2]We iterate between assigning as the exemplar of each cluster the point that has the maximal similarity to all other points in that cluster, and re-assigning all non-exemplar points to exemplars greedily (from largest to smallest similarity) while making sure the limit constraint for each exemplar is satisfied, until convergence.

to that of KM and chose the largest cluster size $\hat{L}$ of the method with the better total similarity. We decreased $\hat{L}$ by one to obtain L, a cap size which is not trivially satisfied by either KM or AP.

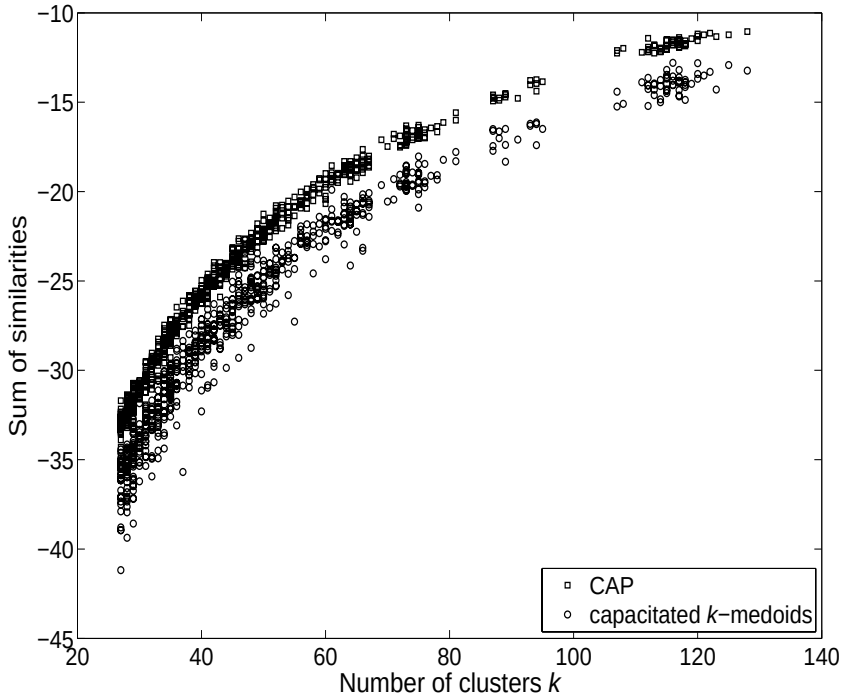


Figure 2: A comparison of CAP and capacitated k-medoids as measured in terms of sum of similarities of points to their exemplars

We ran CAP with the same preferences given to AP and ran CKM with the corresponding k value of each preference. We ran CKM with 1000 different random restarts of which the best run was taken. The average total similarity of CAP was -24.35 while that of the CKM was -27.21. In all 730 experiments CAP outperformed CKM. Fig. 2 shows the breakdown of exper-

imental results according to the number of clusters k.

5 Conclusion

We introduced an alternative derivation of the AP algorithm, stemming from an alternative, yet equivalent, graphical model. We believe this formulation will allow simpler derivation of extensions to the original model as was demonstrated in this work for the capacitated clustering problem.

References

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms, second edition*. The MIT Press.

Dueck, D., Frey, B. J., Jojic, N., Jojic, V., Giaever, G., Emili, A., et al. (2008). Constructing treatment portfolios using affinity propagation. In M. Vingron & L. Wong (Eds.), *Recomb* (Vol. 4955, pp. 360–371). Springer.

Frey, B., & Dueck, D. (2007). Clustering by passing messages between data points. *Science*, *305*(5814), 972 – 976.

Kschischang, F., Frey, B., & Loeliger, H.-A. (2001). Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, *47*(2), 498 – 519.

Mulvey, J. M., & Beck, M. P. (1984, December). Solving capacitated clus-

tering problems. *European Journal of Operational Research*, *18*(3), 339-348.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems : Networks of plausible inference*. San-Francisco, CA: Morgan Kaufmann.

Sridharan, R. (1995). The capacitated plant location problem. *European Journal of Operational Research*, *87*, 203 – 213.

Tarlow, D., Zemel, R., & Frey, B. (2008). Flexible priors for exemplar-based clustering. In *Uncertainty in artificial intelligence (UAI).*

Appendix

Here we provide a more detailed description of the algorithm and experimental procedure. We adopted the convergence criteria used in (Tarlow et al., 2008) with a damping factor of 0.95 and set the maximal number of iterations to 5000. Once the algorithm terminates we use the solution as a starting point for the CKM algorithm and if its output outperforms that of CAP, or if the solution of CAP was not valid, we use the CKM solution. The reason for this is that when CAP does not converge, we found that is oscillates in regions of the solution space that have high objective function values. The results obtained by the simple post-processing step we take here serve to ensure a valid solution, which, as we find in section 4, outperforms multiple random restarts of CKM.

We use the same method for deciding on the final exemplar set $\mathcal{J}$ as used by regular AP (Frey & Dueck, 2007), but for determining the assignment of every point $i \notin \mathcal{J}$ to its appropriate exemplar we compute $\arg\max_{j \in \mathcal{J}} \alpha_{ij} +$

ρ_{ij}.

Our initial data is comprised of 100 different data sets, and for each data set we run the comparison for 10 different values of k based on the results obtained by running regular AP (see section 4). However, in 27% of the runs (270 out of 1000), the solution found by CAP has more than k clusters. We do not include these experiments in the comparison since although they are valid solutions, we wanted to be able to compare results across the different algorithms operating in the same regime.

The average and median run times of CAP are 140.1 and 53.7 seconds per run, respectively while those of CKM were 281.2 and 198.9 seconds per run, respectively (a complete CKM run consists of 1000 reruns with different random starting points.) The code was implemented in matlab and run on a Linux-based machine with 12 GB of RAM, and two dual-core AMD Opteron 2220 CPUs running at 2.8 GHz.