Supporting Online Material for

Clustering by Passing Messages Between Data Points

Brendan J. Frey* and Delbert Dueck

*To whom correspondence should be addressed. E-mail: frey@psi.toronto.edu

Published 11 January 2007 on *Science* Express
DOI: 10.1126/science.1136800

This PDF file includes:

MATLAB implementation of affinity propagation

Here, we provide a MATLAB implementation of affinity propagation that does not account for sparse similarities matrices. (See http://www.psi.toronto.edu/affinitypropagation for software that efficiently processes sparse similarity matrices).

The only input is the MATLAB matrix of similarities, `S`, where `S(i,k)` is the similarity $s(i,k)$. The preferences should be placed on the diagonal of this matrix; if appropriate preferences are not known, usually, a good choice for the preferences is to set them all equal to $\text{median}_{i,k:i\neq k}s(i,k)$. The following MATLAB code executes 100 iterations of affinity propagation. After execution, the combined evidence $r(i,k)+a(i,k)$ is stored in the $N\times N$ matrix `E`, the number of exemplars is stored in `K`, and the indices of the exemplars for the data points are stored in the N-vector `idx` (point `i` is assigned to the data point with index `idx(i)`.)

```
N=size(S,1); A=zeros(N,N); R=zeros(N,N); % Initialize messages
S=S+1e-12*randn(N,N)*(max(S(:))-min(S(:))); % Remove degeneracies
lam=0.5; % Set damping factor
for iter=1:100
    % Compute responsibilities
    Rold=R;
    AS=A+S; [Y,I]=max(AS,[],2);
    for i=1:N AS(i,I(i))=-realmax; end;
    [Y2,I2]=max(AS,[],2);
    R=S-repmat(Y,[1,N]);
    for i=1:N R(i,I(i))=S(i,I(i))-Y2(i); end;
    R=(1-lam)*R+lam*Rold; % Dampen responsibilities

    % Compute availabilities
    Aold=A;
    Rp=max(R,0); for k=1:N Rp(k,k)=R(k,k); end;
    A=repmat(sum(Rp,1),[N,1])-Rp;
    dA=diag(A); A=min(A,0); for k=1:N A(k,k)=dA(k); end;
    A=(1-lam)*A+lam*Aold; % Dampen availabilities
end;
E=R+A; % Pseudomarginals
I=find(diag(E)>0); K=length(I); % Indices of exemplars
[tmp c]=max(S(:,I),[],2); c(I)=1:K; idx=I(c); % Assignments
```

Mostly, the above code directly follows the updates in the main text. Implemented naively, the updates in the main text would use order N^3 scalar binary operations per iteration. However, if certain computations are re-used, only order N^2 scalar binary operations

are needed, and if only a subset of J similarities are provided, the sparse affinity propagation software available at http://www.psi.toronto.edu/affinitypropagation uses only order j operations.. In the above code, when computing the responsibility sent from i to k, the maximum value of $a(i,k')+s(i,k')$ w.r.t. k' *and* the next-to-maximum value are computed. Then, the maximum value of $a(i,k')+s(i,k')$ w.r.t. $k' \neq k$ needed in equation (1) of the main text can be found in a single operation, by checking to see if k gives the maximum (in which case the next-to-maximum value is used) or not (in which case the maximum value is used). The implementation provided above could be made more efficient and does not take into account sparse data networks, where many input similarities are $-\infty$.

Comparison to exact inference for model selection

For the data shown in Fig. 1C and available at *(S1)*, we ran affinity propagation using a preference (common for all data points) ranging from -200 to -0.1. In this case, the number of points is small enough that we were also able to find the solution that exactly minimizes the energy, for a given input preference. Fig. S1 plots the number of detected exemplars versus the input preference for affinity propagation and the exact method.

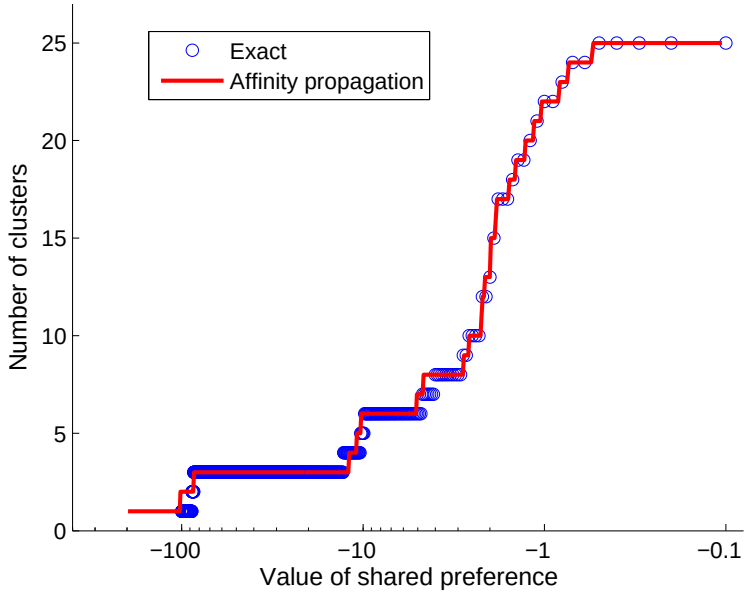

Figure S1: Comparison to exact inference

The number of exemplars detected by affinity propagation is nearly identical to the exact solution.

In a plot of the number of clusters versus the shared preference value, high rates of change correspond to the existence of multiple subsets of data that have approximately the same intra-subset similarities and approximately the same inter-subset similarities. When the preferences are below a certain value, these subsets are grouped together with other clusters. When the preferences rise above that value, it becomes beneficial (in energy) for the multiple subsets to simultaneously form distinct clusters. In a task where the data has a hierarchical similarity structure, different plateaus would correspond to the extraction of different levels of structure.

Detecting genes and comparing against REFSEQ annotations

It was previously shown that when nearby segments of DNA undergo coordinated transcription across multiple tissues, they are likely to come from transcribed regions of the same gene *(S2)*. In that work, $75,066$ DNA segments corresponding to possible transcribed regions of genes were mined from the genome for mouse Chromosome 1. All $75,066$ segments were indexed according to genomic order, and a matrix of similarities was constructed as described in *(S2)*, so that clusters of segments would correspond to predicted genes.

Briefly, the input similarity $s(i,k)$ between DNA segment (data point) i and DNA segment k measures similarity between the expression of the DNA segments across 12 tissues (as measured by a microarray) and proximity of the DNA segments in the genome. To account for non-transcribed regions, an additional artificial data point was included (index $75,067$) and the similarity of each other point to this 'non-transcribed exemplar' was determined using average statistics of the entire data set. The preference for this artificial data point was set to ∞ to ensure it was chosen as an exemplar, while the preference for every other data point was set by comparing its corresponding expression levels to the distribution of expression levels for the entire data set. After clustering the $75,067 \times 75,067$ sparse similarity matrix, 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 *(S2)*, to produce labels used to report true positive and false positive rates. These labels, along with the sparse similarity matrix (and sparse affinity propagation implementation), the pre-processed data, and the predictions made by the engineered gene discovery tool reported in *(S2)* are available at *(S1)*.

In addition to using both affinity propagation and over $100,000$ runs of K-centers clustering to detect transcribed regions (as described in the main text), we also used hierarchical agglomerative clustering, or HAC *(S4)*. We used the MATLAB 6.1 implementation of HAC with the 'single linkage' technique. This program could not be applied to the entire $75,067 \times 75,067$ similarity matrix, so the genome was divided into windows containing 600 DNA segments each, in steps of 400. We obtained results using HAC with a variety of linkage functions applied to the same similarity matrix as was used by affinity propagation (data not shown). We found we could obtain better results for HAC using a pair-wise linkage distance equal to one minus the Pearson correlation of the mRNA concentrations for the two query mRNAs (based on 12 conditions). HAC was applied (single linkage worked best) and a threshold was used cut the HAC tree. Points belonging to non-singleton clusters were labeled as being transcribed, and the central 400 labels in each window were kept before moving on to the next window. The threshold was varied to obtain different false detection rates. To prevent distant query mRNAs from being linked together, the linkage distance for training cases i and j was set to infinity if $|i - j| > d$. We let d range from 1 to 10 and found that $d = 3$ gave highest sensitivity. The results for HAC are reported in the main text.

Identifying representative sentences using affinity propagation

For each sentence in the main text of this manuscript, words delimited by spaces were extracted, punctuation was removed, and words with fewer than 5 characters were discarded. The similarity of sentence i to sentence k was set to the negative sum of the information-theoretic costs *(S5)* of encoding every word in sentence i using the words in sentence k and a dictionary of all words in the manuscript. For each word in sentence i, if the word matched a word in sentence k, the coding cost for the word was set to the negative logarithm of the number of words in sentence k (the cost of coding the index of the

matched word), and otherwise it was set to the negative logarithm of the number of words in the manuscript dictionary (the cost of coding the index of the word in the manuscript dictionary). A word was considered to match another word if either word was a substring of the other.

The preference for each sentence as an exemplar was set to the number of words in the sentence times the negative logarithm of the number of words in the manuscript dictionary (the cost of coding all words in the exemplar sentence using the manuscript dictionary), plus a number shared across all sentences that was adjusted to select the number of detected exemplars. This number was set to -90 to detect the four exemplar sentences reported in the main text. For different settings of this number, the following sentences were identified as exemplars.

-100

- Affinity propagation identifies exemplars by recursively sending real-valued messages between pairs of data points.
- The availability $a(i, k)$ is set to the self responsibility $r(k, k)$ plus the sum of the positive responsibilities candidate exemplar k receives from other points.
- For different numbers of clusters, the reconstruction errors achieved by affinity propagation and k-centers clustering are compared in Fig. 3B.

-90

- Affinity propagation identifies exemplars by recursively sending real-valued messages between pairs of data points.
- The number of detected exemplars (number of clusters) is influenced by the values of the input preferences, but also emerges from the message-passing procedure.
- The availability $a(i, k)$ is set to the self responsibility $r(k, k)$ plus the sum of the positive responsibilities candidate exemplar k receives from other points.

- For different numbers of clusters, the reconstruction errors achieved by affinity propagation and k-centers clustering are compared in Fig. 3B.

-80

- Affinity propagation identifies exemplars by recursively sending real-valued messages between pairs of data points.
- The number of detected exemplars (number of clusters) is influenced by the values of the input preferences, but also emerges from the message-passing procedure.
- The availability $a(i, k)$ is set to the self responsibility $r(k, k)$ plus the sum of the positive responsibilities candidate exemplar k receives from other points.
- Fig. 1A shows the dynamics of affinity propagation applied to 25 two-dimensional data points using negative squared error as the similarity.
- At a false positive rate of 3% affinity propagation achieved a true positive rate of 39% whereas the best k-centers clustering result was 17%.

-60

- Affinity propagation identifies exemplars by recursively sending real-valued messages between pairs of data points.
- The number of detected exemplars (number of clusters) is influenced by the values of the input preferences, but also emerges from the message-passing procedure.
- The availability $a(i, k)$ is set to the self responsibility $r(k, k)$ plus the sum of the positive responsibilities candidate exemplar k receives from other points.
- Fig. 1A shows the dynamics of affinity propagation applied to 25 two-dimensional data points using negative squared error as the similarity.
- At a false positive rate of 3% affinity propagation achieved a true positive rate of 39% whereas the best k-centers clustering result was 17%.

- In fact it can be applied to problems where the similarities are not symmetric and even to problems where the similarities do not satisfy the triangle inequality.

-50

- The most frequently used technique for learning exemplars is k-centers clustering which starts with an initial set of exemplars usually a randomly selected subset of the data points and iteratively refines this set so as to decrease the sum of squared errors in each iteration.

- Affinity propagation identifies exemplars by recursively sending real-valued messages between pairs of data points.

- The number of detected exemplars (number of clusters) is influenced by the values of the input preferences, but also emerges from the message-passing procedure.

- The availability $a(i, k)$ is set to the self responsibility $r(k, k)$ plus the sum of the positive responsibilities candidate exemplar k receives from other points.

- Fig. 1A shows the dynamics of affinity propagation applied to 25 two-dimensional data points using negative squared error as the similarity.

- Instead, the measure of similarity between putative exons was based on a cost function measuring their proximity in the genome and the degree of coordination of their transcription levels across the 12 tissues.

- At a false positive rate of 3% affinity propagation achieved a true positive rate of 39% whereas the best k-centers clustering result was 17%.

- In fact, it can be applied to problems where the similarities are not symmetric and even to problems where the similarities do not satisfy the triangle inequality.

Derivation of affinity propagation as the max-sum algorithm in a factor graph

As described in the main text, identifying exemplars can be viewed as searching over valid configurations of the labels $\mathbf{c} = (c_1, \ldots, c_N)$ so as to minimize the energy $E(\mathbf{c}) =$

$-\sum_{i=1}^{N} s(i, c_i)$. It is easier to think of *maximizing* the net similarity, $\mathcal{S}$, which is the negative energy plus a constraint function that enforces valid configurations:

$$\begin{aligned} \mathcal{S}(\mathbf{c}) &= -E(\mathbf{c}) + \textstyle\sum_{k=1}^{N} \delta_k(c_k) \\ &= \textstyle\sum_{i=1}^{N} s(i, c_i) + \sum_{k=1}^{N} \delta_k(c_k) \end{aligned} \qquad \text{where } \delta_k(\mathbf{c}) = \begin{cases} -\infty, \text{ if } c_k = k \text{ but } \exists i\text{: } c_i = i \\ 0, \text{otherwise} \end{cases} \tag{S1}$$

Here, $\delta_k(\mathbf{c})$ is a penalty term that equals $-\infty$ if some data point i has chosen k as its exemplar (*i.e.*, $c_i = k$), without k having been correctly labeled as an exemplar (*i.e.*, $c_k = k$).

This function (Eq. S1) can be represented using a factor graph *(S6)*. Each term in $\mathcal{S}(\mathbf{c})$ is represented by a 'function node' and each label c_i is represented by a 'variable node'. Edges exist only between function nodes and variable nodes, and a variable node is connected to a function node if and only if its corresponding term depends on the variable. So, the term $s(i, c_i)$ appearing in the above expression has a corresponding function node that is connected to the single variable c_i. The term $\delta_k(\mathbf{c})$ has a corresponding function node that is connected to all variables $c_1, \ldots, c_N$. In a factor graph, the 'global function' — in this case $\mathcal{S}(\mathbf{c})$ — is given by the sum of all the functions represented by function nodes. (Factor graphs are also used to represent a global function that is a product of simpler functions, but here we use the summation form.)

The max-sum algorithm (the log-domain version of the max-product algorithm) can be used to search over configurations of the labels in the factor graph that maximize $\mathcal{S}(\mathbf{c})$. This algorithm is identical to the sum-product algorithm (a.k.a. loopy belief propagation), except that it computes maximums of sums, instead of sums of products. These algorithms are provably exact for trees, but have been used to obtain record-breaking results for highly constrained search problems including error-correcting decoding *(S7,S8)*, random satisfiability *(S9)*, stereo vision *(S10)* and two-dimensional phase-unwrapping *(S11)*. The max-sum algorithm for the factor graph in Fig. S2A can be derived in a straightforward fashion (c.f. *(S6)*) and consists of sending messages from variables to functions and from functions to variables in a recursive fashion.

The message sent from c_i to $\delta_k(\mathbf{c})$ consists of N real numbers — one for each possible value, j, of c_i — and can be denoted $\rho_{i \to k}(j)$ (Fig. S2B). Later, we show that these

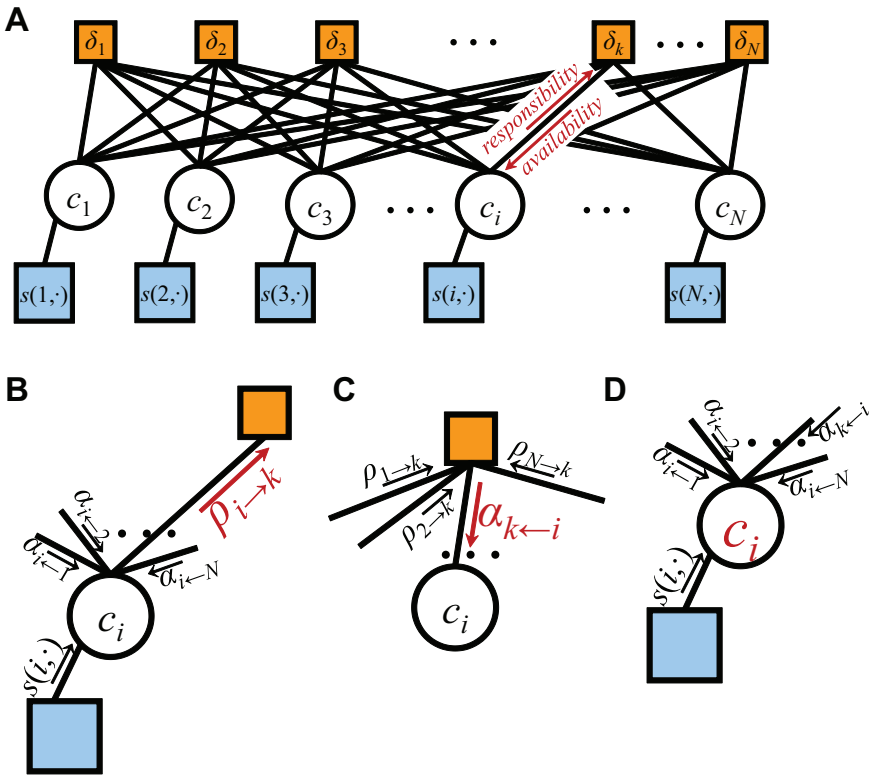

Figure S2: Factor Graph for Affinity Propagation

N numbers can be reduced to a single number, making affinity propagation efficient as a message-passing algorithm. The message sent from $\delta_k(\mathbf{c})$ to c_i also consists of N real numbers and can be denoted $\alpha_{i\leftarrow k}(j)$ (Fig. S2C). At any time, the value of c_i can be estimated by summing together all incoming availability and similarity messages (Fig. S2D).

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

$$\rho_{i\to k}\left(c_i\right) = s\left(i, c_i\right) + \sum\nolimits_{k':k'\neq k} \alpha_{i\leftarrow k'}\left(c_i\right) \tag{S2a}$$

Messages sent from functions to variables are computed by summing incoming messages and then maximizing over all variables except the variable the message is being sent to.

The message sent from function δ_k to variable c_i is:

$$\alpha_{i\leftarrow k}(c_i) = \overbrace{\max_{j_1,\ldots,j_{i-1},j_{i+1},\ldots,j_N} \left[\delta_k\left(j_1,\ldots,j_{i-1},c_i,j_{i+1},\ldots,j_N\right) + \sum\nolimits_{i'} \rho_{i'\to k}\left(j_{i'}\right)\right]}^{\text{best possible configuration satisfying } \delta_k \text{ given } c_i}$$

$$= \begin{cases} \overbrace{\sum_{i':i'\neq k} \max_{j'} \rho_{i'\to k}(j')}^{\text{best configuration with or without cluster } k}, \text{ for } c_i = k = i \\ \overbrace{\sum_{i':i'\neq k} \max_{j':j'\neq k} \rho_{i'\to k}(j')}^{\text{best configuration with no cluster } k}, \text{ for } c_i \neq k = i \\ \overbrace{\rho_{k\to k}\left(k\right)}^{k \text{ is an exemplar}} + \overbrace{\sum_{i':i'\notin\{i,k\}} \max_{j'} \rho_{i'\to k}\left(j'\right)}^{\text{best configuration of others}}, \text{ for } c_i = k \neq i \\ \max\left[\overbrace{\max_{j':j'\neq k} \rho_{k\to k}(j') + \sum_{i':i'\notin\{i,k\}} \max_{j':j'\neq k} \rho_{i'\to k}(j')}^{\text{best configuration with no cluster } k}, \overbrace{\rho_{k\to k}(k) + \sum_{i':i'\notin\{i,k\}} \max_{j'} \rho_{i'\to k}(j')}^{\text{best configuration with a cluster } k}\right], \text{ for } c_i \neq k \neq i \end{cases} \tag{S2b}$$

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

$$\rho_{i\to k}(c_i) = s\left(i,c_i\right) + \sum\nolimits_{k':k'\neq k} \tilde{\alpha}_{i\leftarrow k'}(c_i) + \sum\nolimits_{k':k'\neq k} \bar{\alpha}_{i\leftarrow k'} \tag{S3a}$$

$$\alpha_{i\leftarrow k}(c_i) = \begin{cases} \sum_{i':i'\neq k} \max_{j'} \tilde{\rho}_{i'\to k}(j') + \sum_{i':i'\neq k} \bar{\rho}_{i'\to k}, \text{ for } c_i = k = i \\ \sum_{i':i'\neq k} \max_{j':j'\neq k} \tilde{\rho}_{i'\to k}(j') + \sum_{i':i'\neq k} \bar{\rho}_{i'\to k}, \text{ for } c_i \neq k = i \\ \tilde{\rho}_{k\to k}(k) + \sum_{i':i'\notin\{i,k\}} \max_{j'} \tilde{\rho}_{i'\to k}(j') + \sum_{i':i'\neq i} \bar{\rho}_{i'\to k}, \text{ for } c_i = k \neq i \\ \max\left[\begin{array}{r} \max_{j':j'\neq k} \tilde{\rho}_{k\to k}(j') + \sum_{i':i'\notin\{i,k\}} \max_{j':j'\neq k} \tilde{\rho}_{i'\to k}(j') + \sum_{i':i'\neq i} \bar{\rho}_{i'\to k}, \\ \tilde{\rho}_{k\to k}(k) + \sum_{i':i'\notin\{i,k\}} \max_{j'} \tilde{\rho}_{i'\to k}(j') + \sum_{i':i'\neq i} \bar{\rho}_{i'\to k} \end{array}\right], \text{ for } c_i \neq k \neq i \end{cases} \tag{S3b}$$

For convenience, if we let $\bar{\rho}_{i\to k} = \max_{j:j\neq k} \rho_{i\to k}(j)$ then $\max_{j':j'\neq k} \tilde{\rho}_{i\to k}\left(j'\right) = 0$ and $\max_{j'} \tilde{\rho}_{i\to k}\left(j'\right) = \max\left(0, \tilde{\rho}_{i\to k}(k)\right)$. Also note that in the update for $\alpha_{i\leftarrow k}(c_i)$ (Eq. S3b), none of the expressions on the right depend directly on the value of c_i, rather only the choice

of expression depends on c_i. Consequently, in the N-vector of messages $\alpha_{i\leftarrow k}(c_i)$, there are only two values — one for $c_i = k$ and another for $c_i \neq k$. We set $\bar{\alpha}_{i\leftarrow k} = \alpha_{i\leftarrow k}(c_i\colon c_i \neq k)$ which will make $\tilde{\alpha}_{i\leftarrow k}(c_i)$ zero for all $c_i \neq k$. This also means that $\sum_{k':k'\neq k} \tilde{\alpha}_{i\leftarrow k'}(c_i) = \tilde{\alpha}_{i\leftarrow c_i}(c_i)$ for $c_i \neq k$ and the summation is zero for $c_i = k$, leading to further simplification:

$$\rho_{i\to k}(c_i) = \begin{cases} s(i,k) + \sum_{k':k'\neq k} \bar{\alpha}_{i\leftarrow k'}, \text{ for } c_i = k \\ s(i,c_i) + \tilde{\alpha}_{i\leftarrow c_i}(c_i) + \sum_{k':k'\neq k} \bar{\alpha}_{i\leftarrow k'}, \text{ for } c_i \neq k \end{cases} \tag{S4a}$$

$$\alpha_{i\leftarrow k}(c_i) = \begin{cases} \sum_{i':i'\neq k} \max(0, \tilde{\rho}_{i'\to k}(k)) + \sum_{i':i'\neq k} \bar{\rho}_{i'\to k}, \text{ for } c_i = k = i \\ \sum_{i':i'\neq k} \bar{\rho}_{i'\to k}, \text{ for } c_i \neq k = i \\ \tilde{\rho}_{k\to k}(k) + \sum_{i':i'\notin\{i,k\}} \max(0, \tilde{\rho}_{i'\to k}(k)) + \sum_{i':i'\neq i} \bar{\rho}_{i'\to k}, \text{ for } c_i = k \neq i \\ \max\Big[0, \tilde{\rho}_{k\to k}(k) + \sum_{i':i'\notin\{i,k\}} \max(0, \tilde{\rho}_{i'\to k}(k))\Big] + \sum_{i':i'\neq i} \bar{\rho}_{i'\to k}, \text{ for } c_i \neq k \neq i \end{cases} \tag{S4b}$$

Next, we 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:

$$\begin{aligned} \tilde{\rho}_{i\to k}(c_i{=}k) &= \rho_{i\to k}(c_i{=}k) - \bar{\rho}_{i\to k} = \rho_{i\to k}(k) - \max_{j:j\neq k} \rho_{i\to k}(j) \\ &= s(i,k) + \textstyle\sum_{k':k'\neq k} \bar{\alpha}_{i\leftarrow k'} - \max_{j:j\neq k} \Big[s(i,j) + \tilde{\alpha}_{i\leftarrow j}(j) - \sum_{k':k'\neq k} \bar{\alpha}_{i\leftarrow k'}\Big] \end{aligned} \tag{S5a}$$

$$\begin{aligned} \tilde{\alpha}_{i\leftarrow k}(c_i{=}k) &= \alpha_{i\leftarrow k}(c_i{=}k) - \bar{\alpha}_{i\leftarrow k} = \alpha_{i\leftarrow k}(k) - \alpha_{i\leftarrow k}(j\colon j\neq k) \\ &= \begin{cases} \sum_{i':i'\neq k} \max(0, \rho_{i'\to k}(k)) + \sum_{i':i'\neq k} \bar{\rho}_{i'\to k} - \sum_{i':i'\neq k} \bar{\rho}_{i'\to k}, \text{ for } k = i \\ \max\Big[0, \tilde{\rho}_{k\to k}(k) + \sum_{i':i'\notin\{i,k\}} \max(0, \tilde{\rho}_{i'\to k}(j'))\Big] + \sum_{i':i'\neq i} \bar{\rho}_{i'\to k} \\ \qquad -\tilde{\rho}_{k\to k}(k) + \sum_{i':i'\notin\{i,k\}} \max(0, \tilde{\rho}_{i'\to k}(j')) + \sum_{i':i'\neq i} \bar{\rho}_{i'\to k}, \text{ for } k \neq i \end{cases} \end{aligned} \tag{S5b}$$

$$\begin{aligned} \tilde{\alpha}_{i\leftarrow k}(c_i{=}k) &= \alpha_{i\leftarrow k}(c_i{=}k) - \bar{\alpha}_{i\leftarrow k} = \alpha_{i\leftarrow k}(k) - \alpha_{i\leftarrow k}(j\colon j\neq k) \\ &= \begin{cases} \sum_{i':i'\neq k} \max(0, \rho_{i'\to k}(k)) + \sum_{i':i'\neq k} \bar{\rho}_{i'\to k} - \sum_{i':i'\neq k} \bar{\rho}_{i'\to k}, \text{ for } k = i \\ \tilde{\rho}_{k\to k}(k) + \sum_{i':i'\notin\{i,k\}} \max(0, \tilde{\rho}_{i'\to k}(j')) + \sum_{i':i'\neq i} \bar{\rho}_{i'\to k} \\ \quad - \max\Big[0, \tilde{\rho}_{k\to k}(k) + \sum_{i':i'\notin\{i,k\}} \max(0, \tilde{\rho}_{i'\to k}(j'))\Big] - \sum_{i':i'\neq i} \bar{\rho}_{i'\to k}, \text{ for } k \neq i \end{cases} \end{aligned} \tag{S5b}$$

Noting that $\tilde{\rho}_{k\to i}(c_i)$ and $\tilde{\alpha}_{i\leftarrow k}(c_i)$ for $c_i \neq k$ are not used in the updates (particularly

because $\tilde{\alpha}_{i\leftarrow k}(c_i{\neq}k) = 0$), messages can be considered to be scalar with responsibilities defined as $r(i,k) = \tilde{\rho}_{i\rightarrow k}(k)$ and availabilities as $a(i,k) = \tilde{\alpha}_{i\leftarrow k}(k)$.

$$
\begin{aligned}
r(i,k) &= \tilde{\rho}_{i\rightarrow k}(c_i{=}k) = s(i,k) - \max_{j:j\neq k}\left[s(i,j) + a(i,j)\right] \\
a(i,k) &= \tilde{\alpha}_{i\leftarrow k}(c_i{=}k) = \begin{cases} \sum_{i':i'\neq k} \max(0, r(i',k)), \text{ for } k{=}i \\ \min\Big[0, r(k,k) + \sum_{i':i'\notin\{i,k\}} \max(0, r(i',k))\Big], \text{ for } k{\neq}i \end{cases}
\end{aligned}
\tag{S6}
$$

This is the form seen in the main text; the $\min[0,\cdot]$ in the availability update comes from the fact that $x - \max(0,x) = \min(0,x)$.

To estimate the value of a variable c_i after any iteration, we sum together all incoming messages to c_i and take the value, $\hat{c}_i$, that maximizes this:

$$
\begin{aligned}
\hat{c}_i &= \underset{j}{\operatorname{argmax}}\left[\textstyle\sum_k \alpha_{i\leftarrow k}(j) + s(i,j)\right] \\
&= \underset{j}{\operatorname{argmax}}\left[\textstyle\sum_k \tilde{\alpha}_{i\leftarrow k}(j) + \sum_k \bar{\alpha}_{i\leftarrow k} + s(i,j)\right] \\
&= \underset{j}{\operatorname{argmax}}\left[a(i,j) + s(i,j)\right]
\end{aligned}
\tag{S7}
$$

An alternative form for this that includes both availabilities and responsibilities can be obtained by including an additional term inside the $\operatorname{argmax}_j[\cdot]$ that leaves the result unchanged as follows:

$$
\begin{aligned}
\hat{c}_i &= \underset{j}{\operatorname{argmax}}\left[a(i,j) + \overbrace{s(i,j) - \max_{j':j'\neq j}\left[s(i,j') + a(i,j')\right]}^{r(i,j) \text{ from responsibility update equation}}\right] \\
&= \underset{j}{\operatorname{argmax}}\left[a(i,j) + r(i,j)\right]
\end{aligned}
\tag{S8}
$$

This is discussed further in the main text where availabilities are added to responsibilities to determine if a data point is an exemplar or not.

The sum-product algorithm on the affinity propagation factor graph

If we use data likelihoods, $S(i,k) = e^{s(i,k)}$, instead of log-domain similarities we can

derive an analogous set of update equations with the sum-product algorithm:

$$R(i,k) = S(i,k) \Big/ \sum_{k':k' \neq k} \left(S(i,k') \cdot \ A(i,k') \right) \tag{S9a}$$

$$A(i,k) = \begin{cases} \left[\prod_{j:j \notin \{i,k\}} \frac{1}{1+R(j,k)} + 1 \right]^{-1}, \text{ for } i \neq k \\ \prod_{j:j \neq k} \left[1 + R(j,k) \right], \text{ for } i = k \end{cases} \tag{S9b}$$

Here, we use $R(i,k) = e^{r(i,k)}$ to refer to an responsibility probability or proportion, and $A(i,k) = e^{a(i,k)}$ for an availability. These update equations are not as straightforward to implement in N^2 time due to numerical precision issues, and the algorithm is no longer invariant to arbitrary scaling of the S-matrix.

An alternative factor graph

The energy and constraint functions can be rephrased in terms of N^2 binary variables rather than N N-ary variables by considering the factor graph topology shown in Fig. S3.

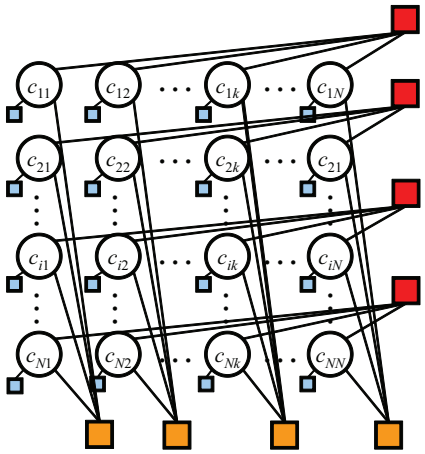

Figure S3: Alternative factor graph topology

Here, binary variable $c_{i,k}$ is one if $c_i = k$ and zero otherwise. There are functions similar to δ_k for each column, constraining that if other data points indicate k is their exemplar, point k needs to be labeled as such. In addition, there are constraints that exactly one variable in each row must be one and all others zero. Message-passing updates can also be

obtained from the max-sum algorithm in this factor graph.

References

S1. Software implementations of affinity propagation, along with the data sets and similarity matrices used to obtain the results described in this manuscript are available at http://www.psi.toronto.edu/ affinitypropagation.

S2. B. J. Frey, *et al.*, *Nature Gen.* **37**, 991 (2005).

S3. K. D. Pruitt, T. Tatusova, D. R. Maglott, *Nucleic Acids Res.* **31**, 34 (2003)

S4. R. R. Sokal, C. D. Michener, *Univ. Kans. Sci. Bull.* **38**, 1409 (1948)

S5. T. M. Cover, J. A. Thomas, *Elements of Information Theory*, (John Wiley & Sons, New York, NY, 1991).

S6. F. R. Kschischang, B. J. Frey, H.-A. Loeliger, *IEEE Trans. Inform. Theory* **47**, 1 (2001).

S7. D. J. C. MacKay, *IEEE Trans. Info. Theory* **45**, 399 (1999).

S8. C. Berrou, A. Glavieux, *IEEE Trans. Comm.* **44**, 1261 (1996).

S9. M. Mézard, G. Parisi, R. Zecchina, *Science* **297**, 812 (2002).

S10. T. Meltzer, C. Yanover, Y. Weiss, *Proc. ICCV*, 428 (2005).

S11. B. Frey, R. Koetter, N. Petrovic, *NIPS* **15**, 737 (2001).